Animated interactive data visualization using animint2 in R

1 Contents and motivation

This manual explains how to design and create interactive data visualizations using the R package `animint2`.

1.1 Contents

The chapters of this manual are organized as follows.

1.1.1 The `animint2` extensions to the grammar of graphics

The first seven chapters should be read sequentially, since they give a step by step guide to interactive data visualization using `animint2`.

This chapter gives an overview of data analysis and visualization. It provides motivation and a theoretical foundation for the other chapters, and should be especially useful for readers who are completely new to data analysis. It introduces the method of data visualization prototyping using sketches, without introducing R code.

Starting with chapter 2, we will show how plot sketches can be translated into R code. Chapter 2 explains the basics of plotting using ggplots and `animint2`, and should be most useful for readers who have never used `ggplot2`. It explains how standard ggplots can be rendered on web pages using `animint2`.

Chapter 3 introduces showSelected, one of the two main keywords that `animint2` introduces for interactive data visualization design. Chapter 3 begins by explaining selection variables, which provide the mechanism of interaction in `animint2`. Chapter 3 then explains how the showSelected keyword makes it possible to plot data subsets. Chapter 3 also explains how to use smooth transitions and animation.

Chapter 4 introduces clickSelects, the other main keyword that `animint2` introduces for interactive data visualization design. The clickSelects keyword makes it possible for the user to change a selection variable by directly clicking on a plot element.

Chapter 5 explains several different ways to share your interactive data visualizations on the web.

Chapter 6 covers some other features of `animint2`, including how to specify hyperlinks, tooltips, data-driven selector variable names.

Chapter 7 covers the limitations of the current implementation of the `animint2` R package, and explains workarounds for some common issues. It also includes some ideas for improvements, for those who would like to contribute to `animint2`.

1.1.2 Examples

The remaining chapters can be read in any order, since each chapter explains how to make data visualizations for a particular data set.

Chapter 8 explains how to create a multi-panel interactive World Bank data visualization.

Chapter 9 shows a visualization of data from cyclists in Montreal.

Chapter 10 explains how to create an interactive re-design of the nearest neighbors data visualization from the Elements of Statistical Learning book by Hastie et al.

Chapter 11 shows a data visualization that explains the Lasso, a machine learning model for regularized regression.

Chapter 12 shows a data visualization that explains support vector machines (SVM), a machine learning model for binary classification.

Chapter 13 explains how to create an interactive visualization that explains the Poisson regression model.

Chapter 14 shows an example of how to create data-driven selectors using named clickSelects/showSelects in an interactive visualization of a peak detection model.

Chapter 15 explains how to create an interactive visualization of the Newton root-finding algorithm.

Chapter 16 explains how to create an interactive visualization of an optimal changepoint detection model.

Chapter 17 explains how to create an interactive visualization of the k-means clustering algorithm.

Chapter 18 explains how to create an interactive visualization of the gradient descent algorithm for learning neural network weight matrices.

1.1.3 Appendices

Useful idioms contains detailed explanations of several R code idioms that are used throughout this manual.

The contributing guide contains instructions about how you can contribute improvments to this manual.

1.2 Motivation

The purpose of this manual is to explain the usage of `animint2`, an R package for interactive data visualization. This introductory chapter answers the following questions:

- What are data, and how are they analyzed?
- What is data visualization, and when is it useful for data analysis?
- What is interactive data visualization, and when is it useful?

This introductory chapter uses the following outline:

- What is data?

- Small data – data visualization is not necessary.
- Medium data – static data visualization is sufficient.
- Large data – interactive data visualization is useful.

1.2.1 What is data analysis?

Data are any pieces of information that are systematically recorded, either on paper or on a computer. Anybody can create data, just by systematically writing things down. Typically, data are created in order to help answer a specific question, and are organized into tables with rows for observations and columns for variables or different types of information. The word “data” is the plural form of “datum,” which we use to refer to one observation/row of a data table. We use the term “data set” to refer to a subset of observations/rows, or the entire data table.

There are many examples of data that could be created to answer questions based on everyday experiences:

- How does the weather this year compare to previous years? Have we had more or less rain than usual? To answer these questions, we could create a data table with column for measurements of different weather conditions: temperature, rainfall, etc. There should also be columns for the date and time of each observation, and a row for each observation.
- How is this new diet affecting me? If you are trying a new diet, you may want to record what you eat and how you feel after each meal. In that case you could make a table with a row for each meal and four columns: date, time, what you ate, and how you felt after.
- Does this new lung cancer treatment work better than the old treatment? A doctor who conducts the clinical trial would randomly assign patients to receive either the new or old treatment. The doctor would then create a data table with a row for each patient, and several columns: years the patient has smoked, treatment type (new or old), patient age at treatment, patient age at death.

We define “data analysis” as the process of answering these questions by converting the raw data table into other, more comprehensible forms. One highly effective class of methods for data analysis is called “data visualization,” which seeks to provide answers to questions by converting a data set into an informative picture. The term “data visualization” refers to both the picture itself (also known as a plot, chart, figure, graph, graphic, or data viz), and the process of creating the picture.

There are many different ways to perform data analysis, and data sets of different sizes should be analyzed using different techniques. There are many different ways to characterize the size of data sets, and every author uses a slightly different definition. In this manual we will use a classification of data sets into three sizes: small, medium, and large. We begin by discussing small data sets, for which data visualization is not necessary.

1.2.2 Small data analysis without visualization

In this section, we will discuss “small data,” which are small enough such that data analysis can be done by simply looking at the entire data table. For small data sets, there is no need to use data visualization. Instead, the data can simply be presented for visual inspection in a table.

As a concrete example, consider the famous tea tasting experiment proposed by Ronald Fisher. A Lady claimed that she could taste the difference when milk is added to the teacup before or after the tea. Fisher asked the question, can the Lady really taste the difference between the two types of tea?

To answer that question, Fisher prepared four cups of tea with milk added after, and four cups of tea with milk added before. Fisher then placed the cups in a random order, had the Lady taste all eight cups of tea, and asked her to identify the four in which milk was added after the tea. According to `help(fisher.test)` in R, the Lady correctly identified three of the four cups in which milk was added after the tea. Fisher than wrote down the following data: the total number of cups (8), the total number of cups with milk added (4), and the total number of cups that the Lady correctly identified (3). In R, this data set can be viewed by printing a contingency table of count data:

```
      Truth
Guess  Milk Tea
  Milk    3   1
  Tea     1   3
```

In this case, the data set is small enough such that Fisher's question can be answered by simply looking at the data table itself. If the Lady had been able to correctly identify all four cups, then that would have been a very convincing demonstration of her ability. However, she was apparently only able to correctly identify three out of the four cups, which is less convincing.

The main topic of this manual is data visualization, which is not necessary for such small data sets. Instead, we will focus on data sets that are too big to be analyzed by manual visual inspection of the data table.

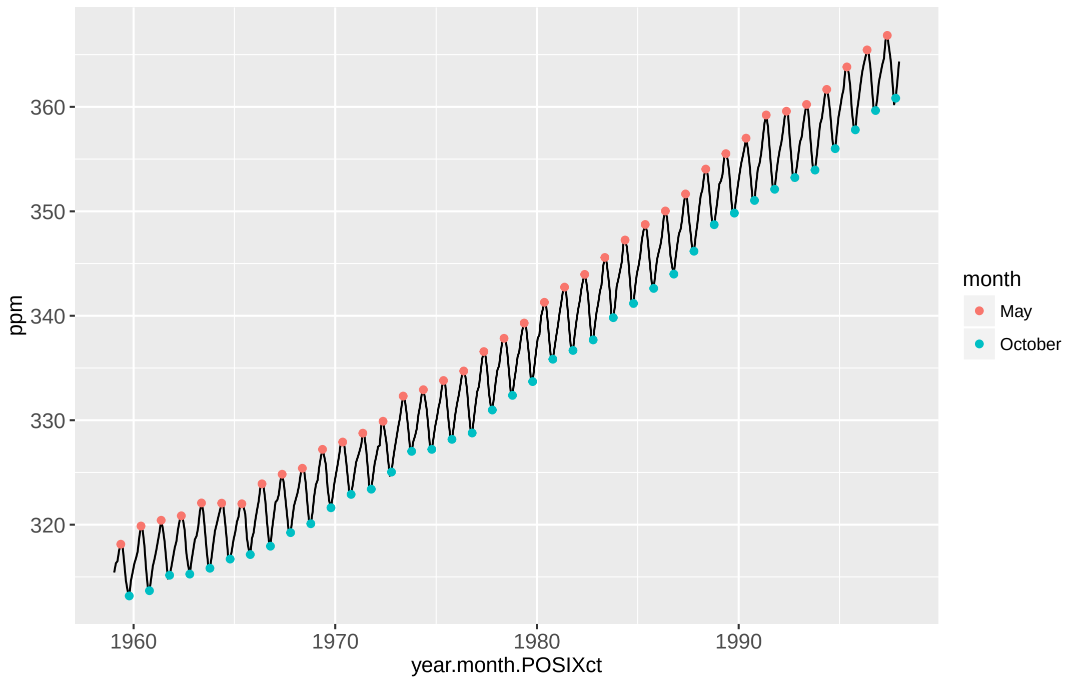
1.2.3 Medium data analysis with static data visualizations

For medium sized data sets, simply inspecting the data table is no longer sufficient to answer the questions posed during data analysis. Medium data are big enough such that we need to use visualization to understand the data.

For example, consider the following data on atmospheric carbon dioxide (CO2) concentrations, recorded monthly between 1959 and 1997.

```
     year.int month.int    month year.month.POSIXct    ppm
  1:     1959         1  January         1959-01-15 315.42
  2:     1959         2 February         1959-02-15 316.31
 ---
467:     1997        11 November         1997-11-15 362.49
468:     1997        12 December         1997-12-15 364.34
```

Printing these data on the R command line shows that there are 468 rows/observations total. This is not a huge number of observations, but it is already big enough so that answering questions is not easy by simple visual inspection of the data table. Instead, we will create a static data visualization:



The static data visualization shows that CO2 concentrations increased over the second half of the twentieth century. This particular data visualization is called a Keeling Curve. It is named after Charles David Keeling, the pioneering scientist who collected the first frequent regular data on atmospheric CO2. The general increasing trend can be explained by considering the chemical process of combustion, which converts oxygen to CO2. Keeling noted that "the observed rate of increase is nearly that to be expected from the combustion of fossil fuel" (REF: The Concentration and Isotopic Abundances of Carbon Dioxide in the Atmosphere, Keeling 1960).

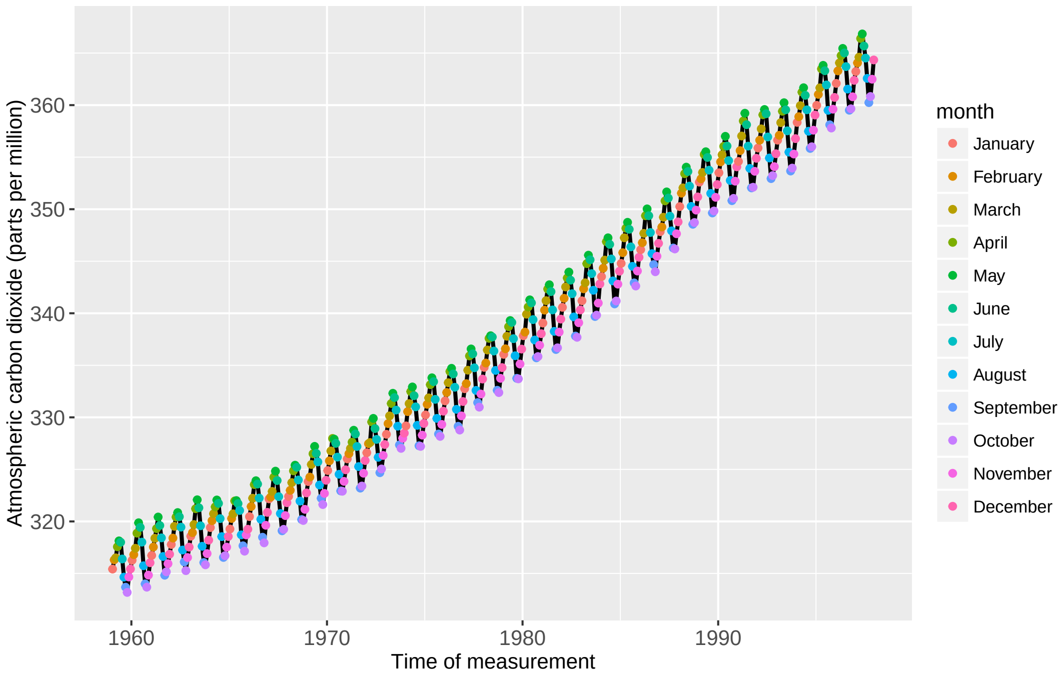
The data visualization also reveals the interesting seasonal trend that attains a local maximum each May and a local minimum each October. This seasonal trend can be explained by considering the forests in the Northern Hemisphere. The leaves on the trees in these forests perform photosynthesis, the chemical conversion of CO2 to oxygen. During the winter months there are no leaves on the trees, so CO2 accumulates in the atmosphere until it peaks in May of each year. When the leaves come back each year, they perform photosynthesis throughout Spring and Summer, which causes the atmospheric CO2 concentration to drop until it reaches its yearly minimum in October.

We say that this data visualization is "static" or "non-interactive" because the reader can view it but can not change what is displayed. That is fine for medium sized data sets, in which we can see all the details of the data set. However, as we discuss in the following section, static data visualization is not sufficient to show all the details in larger data sets.

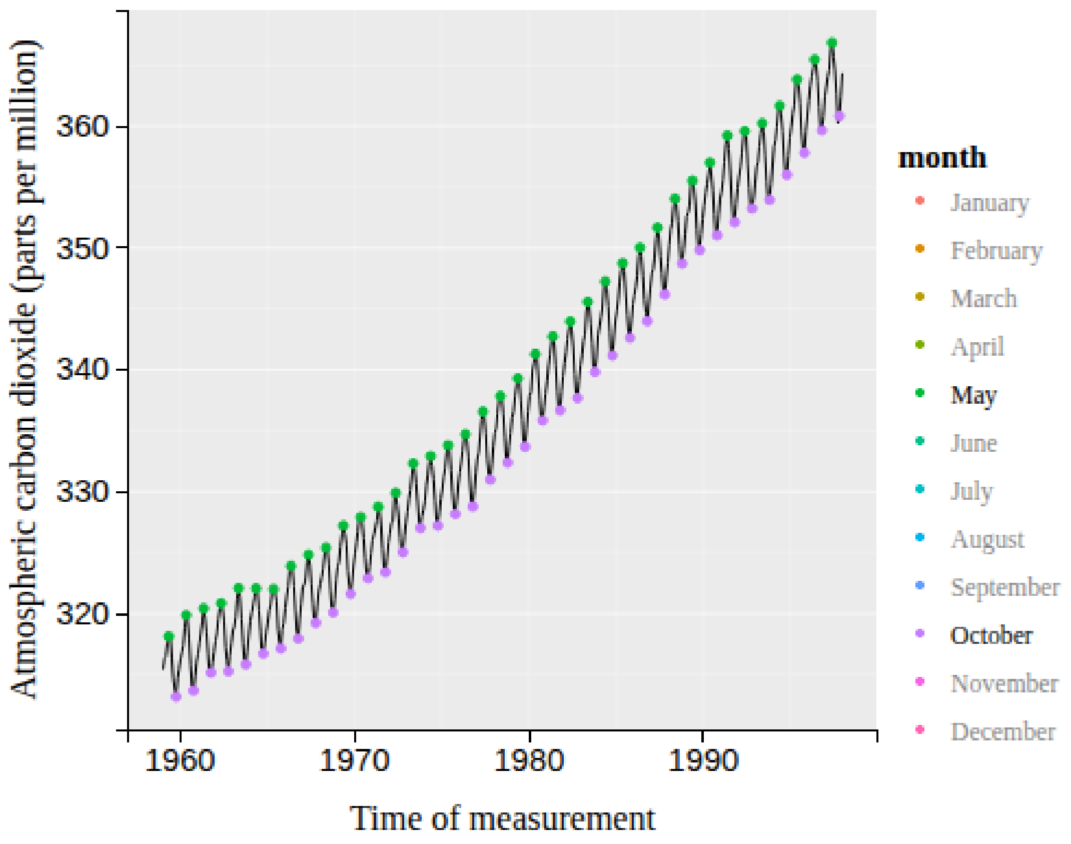
1.2.4 Large data analysis with interactive data visualization

Some data sets are so large that it is not possible or desirable to plot all of the data at once in a static data visualization. For such "large data" sets, traditional approaches to data analysis include summarizing the data, and then visualizing the summary. However, the summary can be misleading, because it does not show all the details of the original data. In such situations, "interactive data visualization" becomes useful.

First, let us consider a slightly more complicated form of the Keeling Curve data visualization.



The plot above shows colored points for each month of the year, rather than for only May and September, the yearly local minima that we wanted to emphasize. Since it lacks this emphasis, this static plot is not as informative as the previous plot. This is an example where it is not desirable to plot all of the data at once. We can solve this problem using the following interactive plot.



In the plot above, the default emphasis is May and October, but the user can click the legend to update the emphasis. This simple example illustrates the main idea of interactive data visualization using animint. There are many choices that must be made to show details of big data sets. For example, the choice of which months to emphasize in the plot above. Rather than fixing such choices in a static plot, the goal of interactive data visualization is to allow the reader to see what the plot looks like when different choices are made. In the example above, we used an interactive legend which allows the user to select different months and see what the plot looks like after changing the selection.

The example above also provides a good example of `clickSelects` and `showSelected`, the two keywords that animint introduces to allow interaction. Without going into too many details, the plot above uses `clickSelects=month` for the interactive legend, meaning that clicking the legend should change the selected months. Furthermore, we used `showSelected=month` for the points, meaning that we should only plot the set of points which corresponds to the currently selected months. In Chapters 3-4, we will explain how to design data visualizations by writing R code using these two new keywords.

1.3 Chapter summary and exercises

This chapter explained some basic facts about data, and gave definitions of different sizes of data: small, medium, and large.

- Based on the definitions introduced in this chapter, what is the difference between small and medium data?
- What is the difference between medium and large data?

Next, Chapter 2 explains how the grammar of graphics can be used to create data visualizations.

Part I

animint2 basics

2

Grammar of graphics

This chapter explains the grammar of graphics, which is a powerful model for describing a large class of data visualizations. After reading this chapter, you will be able to

- State the advantages of the grammar of graphics relative to previous plotting systems
- Install the `animint2` R package
- Translate plot sketches into ggplot code in R
- Render ggplots on web pages using `animint2`
- Create multi-layer ggplots
- Create multi-panel ggplots

2.1 History and purpose of the grammar of graphics

Most computer systems for data analysis provide functions for creating plots to visualize patterns in data. The oldest systems provide very general functions for drawing basic plot components such as lines and points (e.g. the `graphics` and `grid` packages in R). If you use one of these general systems, then it is your job to put the components together to form a meaningful, interpretable plot. The advantage of general systems is that they impose few limitations on what kinds of plots can be created. The disadvantage is that general systems typically do not provide functions for automating common plotting tasks (axes, panels, legends).

To overcome the disadvantages of these general plotting systems, charting packages such as `lattice` were developed (Sarkar, 2008). Such packages have several pre-defined chart types, and provide a dedicated function for creating each chart type. For example, `lattice` provides the `bwplot` function for making box and whisker plots. The advantage of such systems is that they provide a column name specification interface that simplifies creation of entire plots, including legends and panels. Crucially, this column name specification interface allows for rapid experimentation with different plot designs (for example, exchanging the variables used for legends and panels). The disadvantage is the set of pre-defined chart types, which means that it is not easy to create more complex graphics (with several layers of geoms super-imposed on top of each other, each with its own data).

Newer plotting systems based on the grammar of graphics are situated between these two extremes. Wilkinson proposed the grammar of graphics in order to describe and create a large class of plots (Wilkinson, 2005). Wickham later implemented several ideas from the grammar of graphics in the `ggplot2` R package (Wickham, 2009). The `ggplot2` package has several advantages with respect to previous plotting systems.

- Like general plotting systems, and unlike `lattice`, `ggplot2` imposes few limitations on the types of plots that can be created (there are no pre-defined chart types). So it is

possible to create complex graphics, with different layers plotted on top of each other, each with its own data.
- Unlike general plotting systems, and like `lattice`, `ggplot2` simplifies creation of legends and panels via a column name specification interface (and so makes it easy to rapidly experiment with using different variables in different plot designs).
- Since `ggplot2` is based on the grammar of graphics, an explicit mapping of data variables to visual properties is required. Later in this chapter, we will explain how this mapping allows sketches of plot ideas to be directly translated into R code.

Finally, all of the previously discussed plotting systems are intended for creating *static* graphics, which can be viewed equally well on a computer screen or on paper. However, the main topic of this manual is `animint2`, an R package for *interactive* graphics. In contrast to static graphics, interactive graphics are best viewed on a computer with a mouse and keyboard that can be used to interact with the plot.

Since many concepts from static graphics are also useful in interactive graphics, the `animint2` package is implemented as an extension/fork of `ggplot2`. In this chapter we will introduce the main features of `ggplot2` which will also be useful for interactive plot design in later chapters.

In 2013, we created the `animint` package, which depends on the `ggplot2` package. However during 2014-2017, the `ggplot2` package introduced many changes that were incompatible with the interactive grammar of `animint`. Therefore in 2018 we created the `animint2` package which copies/forks the relevant parts of the `ggplot2` package. Now `animint2` can be used without having `ggplot2` installed. In fact, it is recommended to use `library(animint2)` without attaching `ggplot2`. However it is fine to use `animint2` along with packages that import/load `ggplot2`. For an example, see Chapter 16, which uses the `penaltyLearning` package (which imports `ggplot2`).

2.2 Installing and attaching `animint2`

To install the most recent release of `animint2` from CRAN,

```r
if(!requireNamespace("animint2"))install.packages("animint2")
```

```
Loading required namespace: animint2
```

To install an even more recent development version of `animint2` from GitHub,

```r
if(!requireNamespace("animint2")){
  if(!requireNamespace("remotes"))install.packages("remotes")
  remotes::install_github("tdhock/animint2")
}
```

Once you have installed `animint2`, you can load and attach all of its exported functions via:

```r
library(animint2)
```

2.3 Translating plot sketches into ggplots

This section explains how to translate a plot sketch into R code. We use a data set from the World Bank as an example. We begin by loading it and examining its column names.

```
data(WorldBank, package="animint2")
names(WorldBank)
```

```
 [1] "iso2c"                         "country"
 [3] "year"                          "fertility.rate"
 [5] "life.expectancy"               "population"
 [7] "GDP.per.capita.Current.USD"    "15.to.25.yr.female.literacy"
 [9] "iso3c"                         "region"
[11] "capital"                       "longitude"
[13] "latitude"                      "income"
[15] "lending"
```

We see 15 column names above. The `WorldBank` data set consist of measures such as fertility rate and life expectancy for each country over the period 1960-2010. To simplify the output below, we compute an abbreviated `Region` column, and consider only the subset of data which are relevant for the data visualizations below.

```
WorldBank$Region <- sub(
  " (all income levels)", "", WorldBank$region, fixed=TRUE)
world_bank <- subset(
  WorldBank,
  is.finite(fertility.rate) & is.finite(life.expectancy),
  select=c(Region,country,year,fertility.rate,life.expectancy))
tail(world_bank)
```

```
                  Region  country year fertility.rate life.expectancy
13032 Sub-Saharan Africa Zimbabwe 2006          3.551        44.70178
13033 Sub-Saharan Africa Zimbabwe 2007          3.491        45.79707
13034 Sub-Saharan Africa Zimbabwe 2008          3.428        47.07061
13035 Sub-Saharan Africa Zimbabwe 2009          3.360        48.45049
13036 Sub-Saharan Africa Zimbabwe 2010          3.290        49.86088
13037 Sub-Saharan Africa Zimbabwe 2011          3.219        51.23644
```

```
dim(world_bank)
```

```
[1] 9852    5
```

The code above prints the last few rows, and the dimension of the data table (9852 rows and 5 columns).

Suppose that we are interested to see if there is any relationship between life expectancy and fertility rate. We could fix one year, then use those two data variables in a scatterplot. Consider the figure below which sketches the main components of that data visualization.

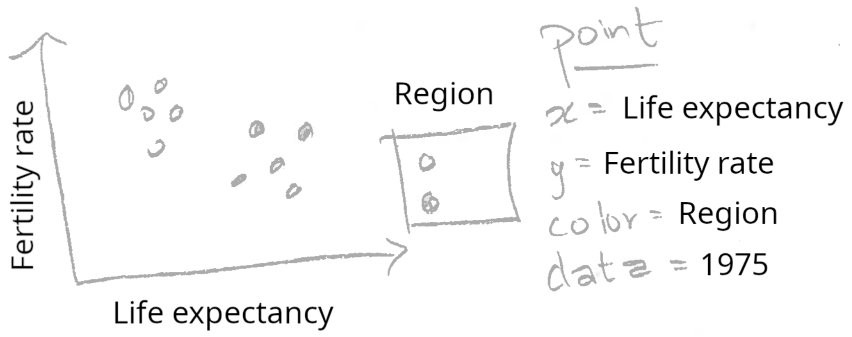


Figure 2.1: Sketch of scatter plot based on World Bank data

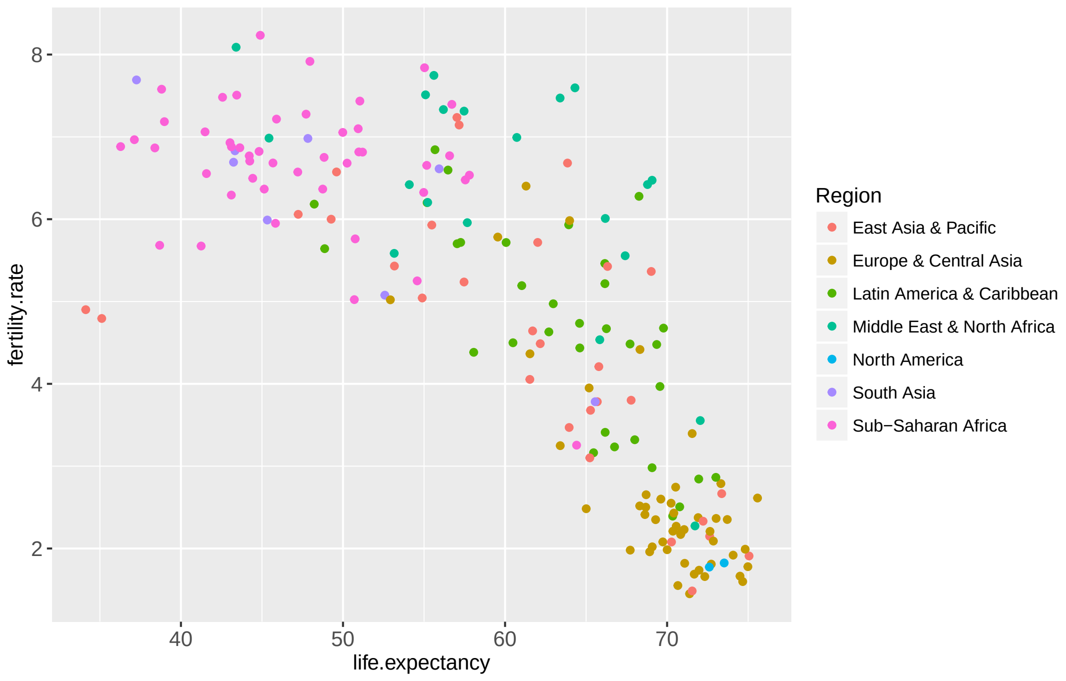
The sketch above shows life expectancy on the horizontal (x) axis, fertility rate on the vertical (y) axis, and a legend for the region. These elements of the sketch can be directly translated into R code using the following method. First, we need to construct a data table that has one row for every country in 1975, and columns named `life.expectancy`, `fertility.rate`, and `region`. The `world_bank` data already has these columns, so all we need to do is consider the subset for the year 1975:

```
world_bank_1975 <- subset(world_bank, year==1975)
tail(world_bank_1975)
```

```
                          Region      country year fertility.rate
11623        East Asia & Pacific      Vanuatu 1975          5.929
11676        East Asia & Pacific        Samoa 1975          5.237
12524 Middle East & North Africa  Yemen, Rep. 1975          8.089
12683         Sub-Saharan Africa South Africa 1975          5.251
12895         Sub-Saharan Africa       Zambia 1975          7.435
13001         Sub-Saharan Africa     Zimbabwe 1975          7.395
      life.expectancy
11623        55.47998
11676        57.46951
12524        43.40459
12683        54.57920
12895        51.04137
13001        56.71702
```

The code above prints the data for 1975, which clearly has the appropriate columns, and one row for each country. The next step is to use the notes in the sketch to code a ggplot with a corresponding `aes` or aesthetic mapping of data variables to visual properties:

```
scatter <- ggplot()+
  geom_point(
    mapping=aes(x=life.expectancy, y=fertility.rate, color=Region),
    data=world_bank_1975)
scatter
```

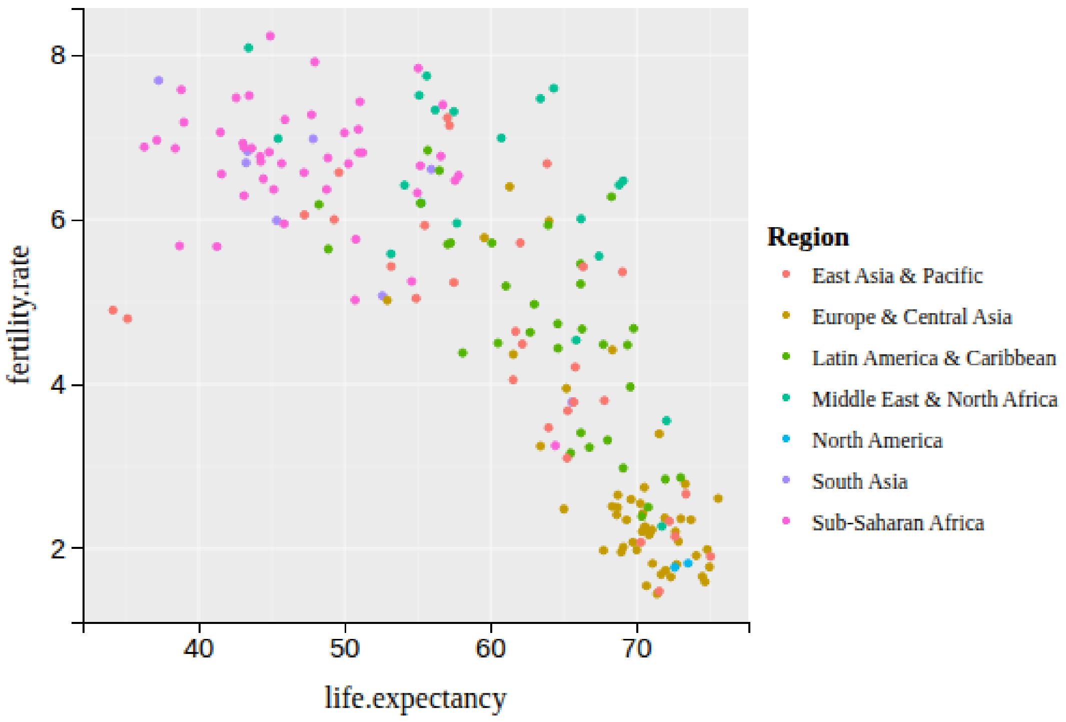


The `aes` function is called with names for visual properties (`x`, `y`, `color`) and values for the corresponding data variables (`life.expectancy`, `fertility.rate`, `region`). This mapping is applied to the variables in the `world_bank_1975` data table, in order to create the visual properties of the `geom_point`. The ggplot was saved as the `scatter` object, which when printed on the R command line shows the plot on a graphics device. Note that we automatically have a `region` color legend.

2.4 Rendering ggplots on web pages using animint

This section explains how the `animint2` package can be used to render ggplots on web pages. The ggplot from the previous section can be rendered with `animint2`, by using the `animint` function.

```
animint(scatter)
```



If, when you run the code above, the animint does not render in your web browser for some reason (for example if you see a blank web page), then please consult our wiki FAQ which will help you find a solution. Internally, the `animint` function creates a list of class animint, and then R runs the `print.animint` function via the S3 object system. The `animint2` package implements a compiler that takes the list as input, and outputs a web page with a data visualization. The compiler is the `animint2dir` function, which compiles the animint `scatter.viz` list to a directory of data and code files that can be rendered in a web browser. It is activated automatically by the `print.animint` function.

When viewed in a web browser, the animint plot should look mostly the same as static versions produced by standard R graphics devices. One difference is that the region legend is interactive: clicking a legend entry will hide or show the points of that color.

Exercise: try changing the `aes` mapping of the ggplot, and then making a new animint. Quantitative variables like `population` are best shown using the `x`/`y` axes or point `size`. Qualitative variables like `lending` are best shown using point `color` or `fill`.

2.5 Multi-layer data visualization (multiple geoms)

Multi-layer data visualization is useful when you want to display several different geoms or data sets in the same plot. For example, consider the following sketch which adds a `geom_path` to the previous data visualization.

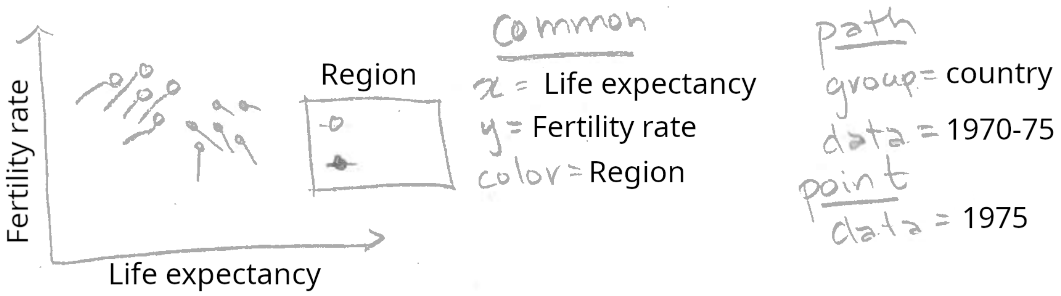
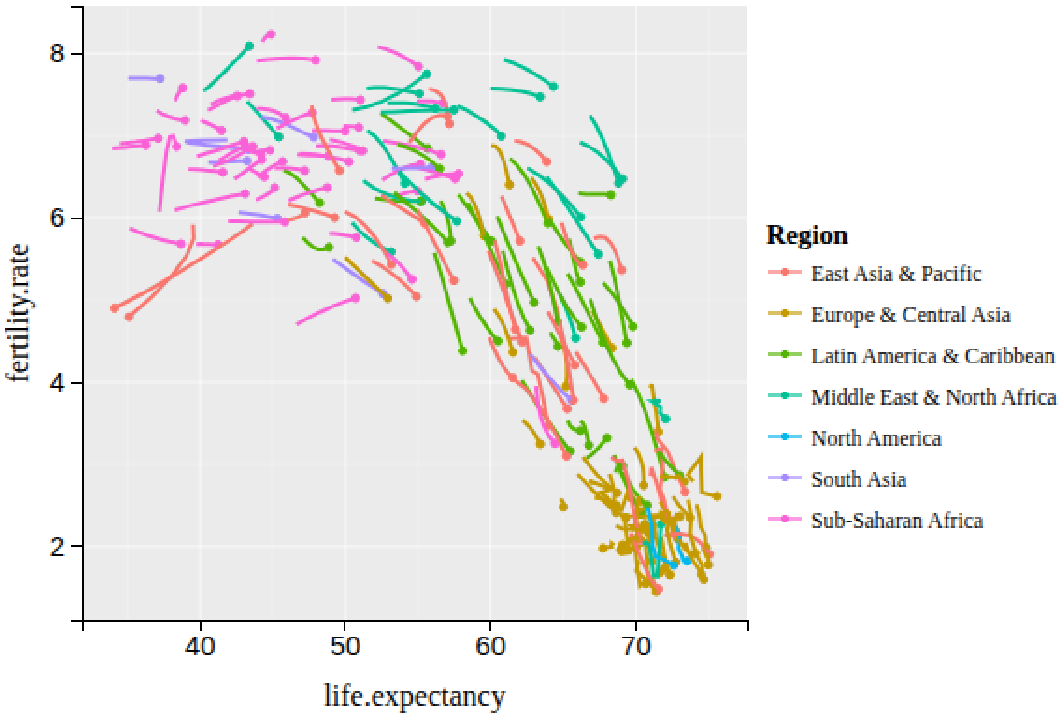


Figure 2.2: Sketch of multi-layer visualization using World Bank data

Note how the sketch above includes two different geoms (point and path). The two geoms share a common definition of the `x`, `y`, and `color` aesthetics, but have different data sets. Below we translate this sketch into R code.

```
world_bank_before_1975 <- subset(world_bank, 1970 <= year & year <= 1975)
two.layers <- scatter+
  geom_path(aes(
    x=life.expectancy,
    y=fertility.rate,
    color=Region,
    group=country),
    data=world_bank_before_1975)
(viz.two.layers <- animint(two.layers))
```



Note that we save the return value of the `animint` function to the `viz.two.layers` object (which is also printed due to the parentheses). In this manual we will often use variable names that start with `viz` to denote animint data visualization objects, which are in fact lists of ggplots and options.

The plot above shows a data visualization with 2 geoms/layers:

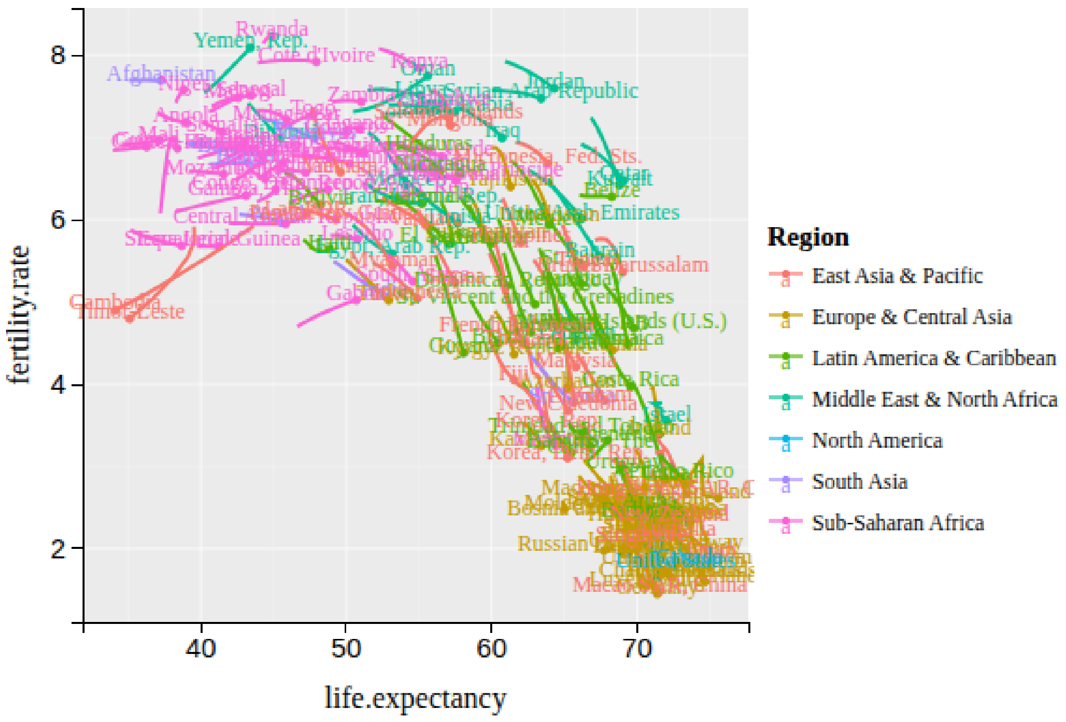
- the `geom_point` shows the life expectancy, fertility rate, and region of all countries in 1975.
- the `geom_path` shows the same variables for the previous 5 years.

The addition of the `geom_path` shows how the countries changed over time. In particular, it shows that most countries moved to the right and down, meaning higher life expectancy and lower fertility rate. However, there are some exceptions. For example, the two East Asian countries in the bottom left suffered a decrease in life expectancy over this period. And there are some countries which showed an increased fertility rate.

Exercise: try changing the `region` legend to an `income` legend. Hint: you need to use the same `aes(color=income)` specification for all geoms, and you will need to use the original `WorldBank` data with all columns (not `world_bank` which has a limited number of columns). You may want to use `scale_color_manual` with a sequential color palette, see `RColorBrewer::display.brewer.all(type="seq")` and read the appendix for more details.

Can we add the names of the countries to the data viz? Below, we add another layer with a text label for each country's name.

```
three.layers <- two.layers+
  geom_text(aes(
    x=life.expectancy,
    y=fertility.rate,
    color=Region,
    label=country),
    data=world_bank_1975)
animint(three.layers)
```



This data viz is not so easy to read, since there are so many overlapping text labels. The interactive region legend helps a little, by allowing the user to hide data from selected regions. However, it would be even better if the user could show and hide the text for individual countries. That type of interaction can be achieved using the `showSelected` and `clickSelects` parameters, which we explain in Chapters 3-4.

Exercise: Re-make this data visualization using `aes(tooltip)`, which is a new feature in `animint2` (not present in `ggplot2`), and is discussed in Chapter 5. Set `aes(tooltip=country)` so that the country name will be visible when you hover the cursor over the corresponding geom.

Next, we move on to discuss a major strength of animint: data visualization with multiple linked plots.

2.6 Multi-plot data visualization

Multi-plot data visualization is useful when you want to show some related data sets using more than one aesthetic mapping. In interactive data visualization, one plot is often used to display a summary, and another plot is used to display details. For example, consider a data visualization with two plots: a time series with World Bank data from 1960-2010 (summary), and a scatterplot with data from 1975 (details). We sketch the time series plot below.

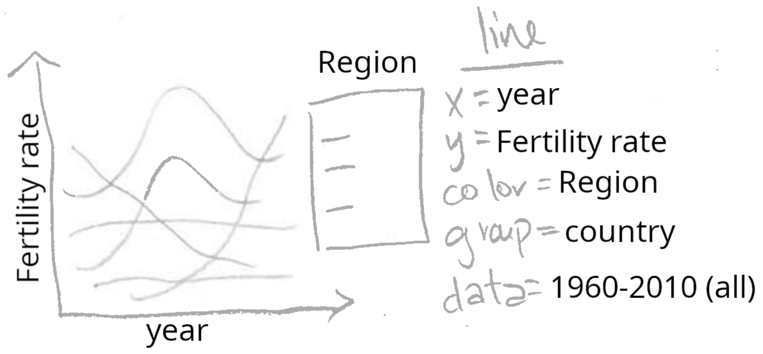


Figure 2.3: Sketch of visualization with two plots based on World Bank data

Note how the sketch above can be directly translated into the R code below. We copy the existing viz list (`viz.two.layers`) to a new list (`viz.two.plots`), then we assign a ggplot to a new element named `timeSeries`.

```r
viz.two.plots <- viz.two.layers
viz.two.plots$timeSeries <- ggplot()+
  geom_line(aes(
    x=year,
    y=fertility.rate,
    color=Region,
    group=country),
    data=world_bank)
```

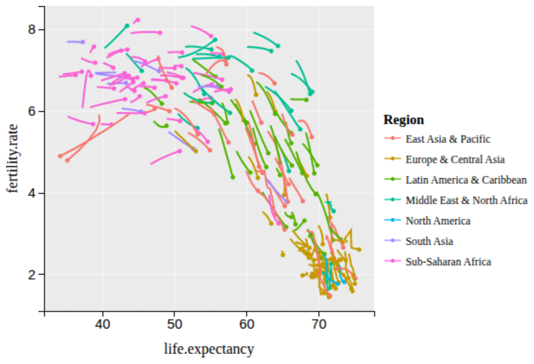
That results in a named list of two elements (both elements are ggplots with class `gganimint`).

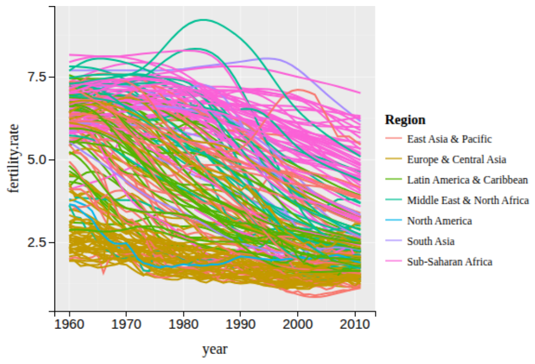
```r
summary(viz.two.plots)
```

```
           Length Class     Mode
plot1      9      gganimint list
timeSeries 9      gganimint list
```

This data visualization list can be printed/rendered by typing its name. Since the list contains two ggplots, `animint2` renders the data viz as two linked plots.

```r
viz.two.plots
```





The data visualization above contains two ggplots, which each map different data variables to the horizontal `x` axis. The time series uses `aes(x=year)`, and shows a summary of fertility rate values over all years. The scatterplot uses `aes(x=life.expectancy)`, and shows details of the relationship between fertility rate and life expectancy during 1975.

Try clicking a legend entry in either the scatterplot or the time series above. You should see the data and legends in both plots update simultaneously. Since `aes(color=Region)` was specified in both plots, animint creates a single shared selector variable called `Region`. Clicking either legend has the effect of updating the set of selected regions, and so animint updates the legends and data in both plots accordingly. This is the main mechanism that animint uses to create interactive data visualizations with linked plots, and will be discussed in more detail in the next two chapters.

Exercise: use animint to create a data viz with three plots, by creating a list with three ggplots. For example, you could add a time series of another data variable such as `life.expectancy` or `population`.

Note that both ggplots map the fertility rate variable to the y axis. However, since they are separate plots, the ranges of their y axes are computed separately. That means that even when the two plots are rendered side-by-side, the two y axis are not exactly aligned. That is a problem since it would make it easier to decode the data visualization if each unit of vertical space was used to show the same amount of fertility rate. To achieve that effect, we use facets in the next section.

2.7 Multi-panel data visualization (facets)

Panels or facets are sub-plots that show related data visualizations. One of the main strengths of ggplots is that different kinds of multi-panel plots are relatively easy to create. Multi-panel data visualization is useful for two different purposes:

- You want to align the axes of several related plots containing different geoms. This facilitates comparison between several different geoms, and is a technique that is also useful for interactive data visualization.
- You want to divide the data from one geom into several panels. This facilitates comparison between data subsets, and is less useful for interactive data visualization (interactivity can often be used instead, to achieve the same effect of comparing data subsets).

2.7.1 Different geoms in each panel (aligned axes)

We begin by explaining the how facets are useful to align the axes of related plots. Consider the sketch below which contains a plot with two panels.

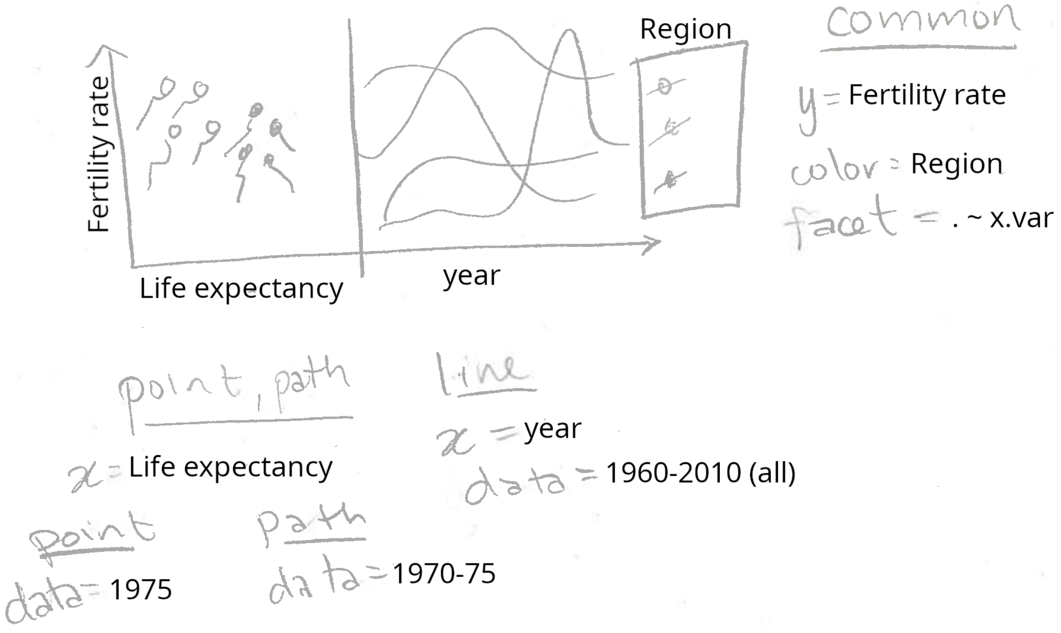
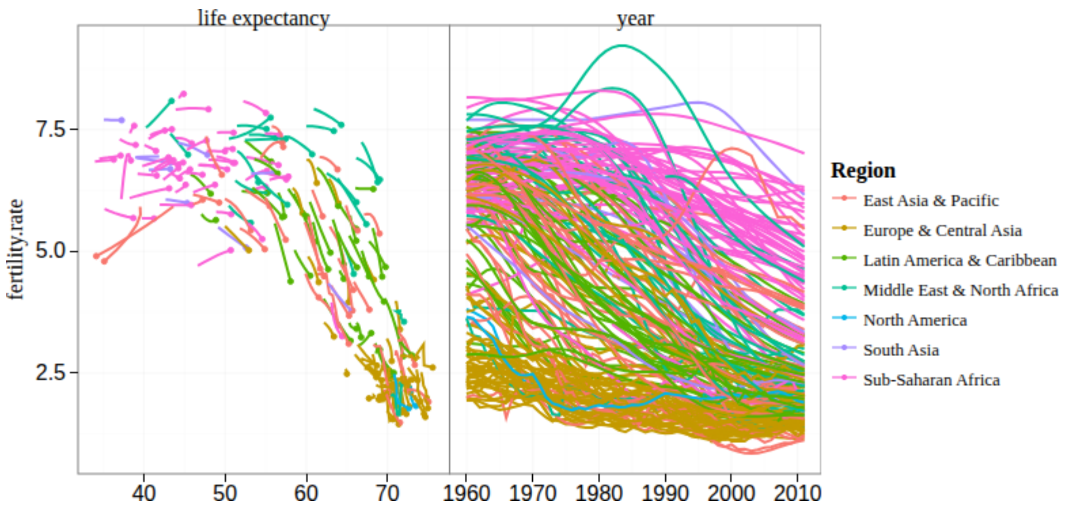


Figure 2.4: Sketch of multi-panel visualization of World Bank data using different geoms in each panel (aligned axes)

Note that the two panels plot different geoms using a panel-specific aesthetic mapping. The point and path in the left panel have `x=life.expectancy`, and the line in the right panel has `x=year`. Also note that we specified `facet=x.var`, so we need to add a variable called `x.var` to each of the three data sets. We translate this sketch to the R code below.

```r
add.x.var <- function(df, x.var){
  data.frame(df, x.var=factor(x.var, c("life expectancy", "year")))
}
(viz.aligned <- animint(
  scatter=ggplot()+
    theme_bw()+
    theme_animint(width=600)+
    theme(panel.margin=grid::unit(0, "lines"))+
    geom_point(aes(
      x=life.expectancy, y=fertility.rate, color=Region),
      data=add.x.var(world_bank_1975, "life expectancy"))+
    geom_path(aes(
      x=life.expectancy, y=fertility.rate, color=Region,
      group=country),
      data=add.x.var(world_bank_before_1975, "life expectancy"))+
    geom_line(aes(
      x=year, y=fertility.rate, color=Region, group=country),
      data=add.x.var(world_bank, "year"))+
    xlab("")+
    facet_grid(. ~ x.var, scales="free")))
```



The data visualization above contains a single ggplot with two panels and three layers. The left panel shows the `geom_point` and `geom_path`, and the right panel shows the `geom_line`. The panels have a shared axis for fertility rate, which ensures that the lines in the time series panel can be directly compared with the points and paths in the scatterplot panel.

Note that we used the `add.x.var` function to add a `x.var` variable to each data set, and then we used that variable in `facet_grid(scales="free")`. We call this the addColumn then facet idiom, which is generally useful for creating a multi-panel data visualization with aligned axes. In particular, if we wanted to change the order of the panels in the data visualization, we would only need to edit the order of the factor levels in the definition of `add.x.var`.

Also note that `theme_bw` means to use black panel borders and white panel backgrounds, and `panel.margin=0` means to use no space between panels. Eliminating the space between panels means that more space will be used for the panels, which serves to emphasize the data. We call this the Space saving facets idiom, which is generally useful in any ggplot with facets.

2.7.2 Same geoms in each panel (compare data subsets)

The second reason for using plots with multiple panels in a data visualization is to compare subsets of observations. This facilitates comparison between data subsets, and can be used in at least two different situations:

- One geom's data set has too many observations to display informatively in one panel.
- You want to compare different subsets of data that is plotted for one geom.

For example, consider the sketch below.

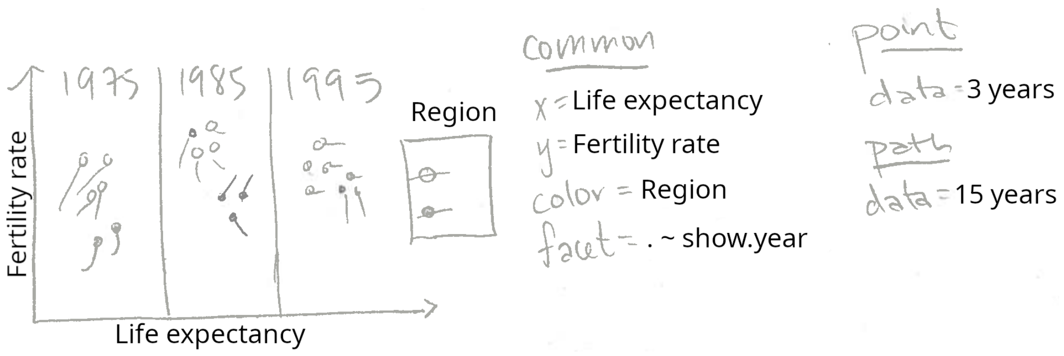


Figure 2.5: Sketch of multi-panel visualization with same geoms in each panel, to compare World Bank data in different years

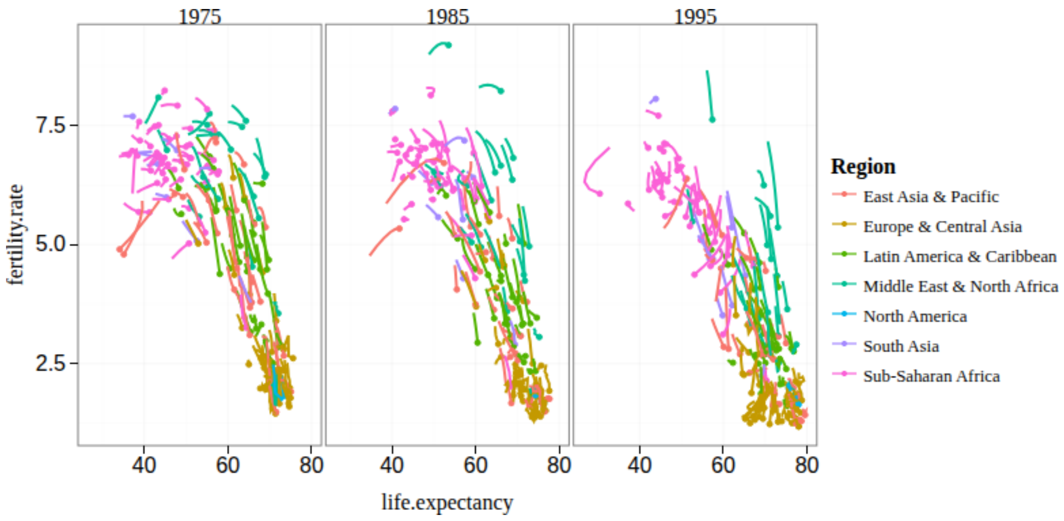
Note that the three panels plot the same two geoms (point and path). Since `facet=show.year`, and there are three panels shown, we will need to create data tables which have three values for the `show.year` variable. The `geom_point` has data for just 3 years, and the `geom_path` has data for 15 years (but 3 values of `show.year`). The code below creates these two data sets for three years of the `world_bank` data set.

```r
show.point.list <- list()
show.path.list <- list()
for(show.year in c(1975, 1985, 1995)){
  show.point.list[[paste(show.year)]] <- data.frame(
    show.year, subset(world_bank, year==show.year))
  show.path.list[[paste(show.year)]] <- data.frame(
    show.year, subset(
      world_bank, show.year - 5 <= year & year <= show.year))
}
show.point <- do.call(rbind, show.point.list)
show.path <- do.call(rbind, show.path.list)
```

We used a for loop over three values of `show.year`, the variable which we will use later in `facet_grid`. For each value of `show.year`, we store a data subset as a named element of a list. After the for loop, we use `do.call` with `rbind` to combine the data subsets. This is an example of the list of data tables idiom, which is generally useful for interactive data visualization.

Below, we facet on the `show.year` variable to create a data visualization with three panels.

```r
animint(
  scatter=ggplot()+
    geom_point(aes(
      x=life.expectancy, y=fertility.rate, color=Region),
      data=show.point)+
    geom_path(aes(
      x=life.expectancy, y=fertility.rate, color=Region,
      group=country),
      data=show.path)+
    facet_grid(. ~ show.year)+
    theme_bw()+
    theme_animint(width=600))
```



The data visualization above contains a single ggplot with three panels. It shows more of the `world_bank` data set than the previous visualizations which showed only the data from 1975. However, it still only shows a relatively small data subset. You may be tempted to try using a panel to display every year (not just 1975, 1985, and 1995). However, beware that this type of multi-panel data visualization is especially useful if there are only a few data subsets. With more than about 10 panels, it becomes difficult to see all the data at once, and thus difficult to make meaningful comparisons.

Instead of showing all of the data at once, we can instead create an animated data visualization that shows the viewer different data subsets over time. In the next chapter, we will show how the new `showSelected` keyword can be used to achieve animation, and reveal more details of this data set.

2.8 Chapter summary and exercises

This chapter presented the basics of static data visualization using `ggplot2`. We showed how animint can be used to render a list of ggplots in a web browser. We explained two features of ggplot2 that make it ideal for data visualization: multi-layer and multi-panel graphics.

Exercises:

- What are the three main advantages of `ggplot2` relative to previous plotting systems such as `grid` and `lattice`?
- What is the purpose of multi-layer graphics?
- Create a version of `viz.two.layers` with `aes(tooltip)` computed based on the min/max values of the data shown by the `geom_path`. Hint: for each country in `world_bank_before_1975`, compute a text string to use for `aes(tooltip)`. One way to

do this is via `data.table(world_bank_before_1975)[, .(tooltip=sprintf(...)), by=country]`.

- What are the two different reasons for creating multi-panel graphics? Which of these two types is more useful with interactivity?
- Let us define "A < B" to mean that "one B can contain several A." Which of the following statements is true?
  - ggplot < panel
  - panel < ggplot
  - ggplot < animint
  - animint < ggplot
  - layer < panel
  - panel < layer
  - layer < ggplot
  - ggplot < layer
- In the `viz.aligned` facets, why is it important to use the `scales="free"` argument?
- In `viz.aligned` we showed a ggplot with a scatterplot panel on the left and a time series panel on the right. Make another version of the data visualization with the time series panel on the left and the scatterplot panel on the right.
- In `viz.aligned` the scatterplot displays fertility rate and life expectancy, but the time series displays only fertility rate. Make another version of the data visualization that shows both time series. Hint: use both horizontal and vertical panels in `facet_grid`.
- Use `aes(size=population)` in the scatterplot to show the population of each country. Hint: `scale_size_animint(pixel.range=c(5, 10)` means that circles with a radius of 5/10 pixels should be used represent the minimum/maximum population.
- Create a multi-panel data visualization that shows each year of the `world_bank` data set in a separate panel. What are the limitations of using static graphics to visualize these data?
- Create `viz.aligned` using a plotting system that is not based on the grammar of graphics. For example, you can use functions from the `graphics` package in R (`plot`, `points`, `lines`, etc), or `matplotlib` in Python. What are some advantages of `ggplot2` and animint?

Next, Chapter 3 explains the `showSelected` keyword, which indicates a variable to use for subsetting the data before plotting.

3

The `showSelected` keyword

This chapter explains showSelected, one of the two main keywords that animint introduces for interactive data visualization. After reading this chapter, you will be able to

- Use the showSelected keyword in your plot sketches to specify geoms for which only a subset of data should be plotted at any time.
- Use selection menus in animint to change the subset of plotted data.
- Specify smooth transitions between data subsets using the duration option and key aesthetic.
- Create animated data visualizations using the time option.

3.1 Sketching with showSelected

In this section, we will explain how the showSelected keyword can be used in plot sketches. The showSelected keyword specifies a variable to use for subsetting the data before plotting. Each geom in a data visualization has its own data set, and its own definition of showSelected variables. That means different geoms can specify different data sets and showSelected keywords to show different data subsets.

In fact, we have already used the showSelected keyword, which was automatically created by the interactive legends that we created in the previous two chapters. For example, consider the sketch below of the Keeling Curve data visualization from Chapter 1.

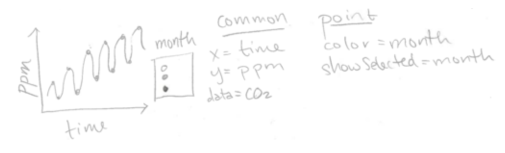


Figure 3.1: CO2 data viz

The sketch above includes `showSelected=month` for the `geom_point`, meaning that it should show the subset of data for the selected months. In contrast, since the `geom_line` does not include `showSelected` keywords, it always shows the entire data set (regardless of the selected months).

As another example, consider the sketch below of the first WorldBank data visualization from Chapter 2.

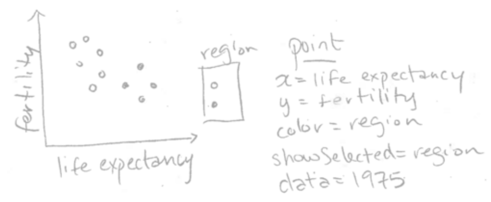


Figure 3.2: WorldBank data viz with showSelected

The sketch above specifies `showSelected=region` for the `geom_point`, meaning that it should show the subset of data for the selected regions.

Note that the code we used in chapter 2 did not explicitly specify `showSelected=region`. Instead, we specified `aes(color=region)`, and animint automatically assigned a showSelected keyword. In general, animint will assign a showSelected keyword for each variable that is used in a categorical legend.

However, the showSelected keyword is not limited to use with categorical legends. You can use showSelected keywords for any data variables you like, by explicitly specifying the variable names in the showSelected argument of the geom.

Each variable that is used with showSelected is treated by animint as a *selection variable*. For example, the Keeling Curve data viz has one selection variable (month), and so does the WorldBank data viz (region). For each selection variable, animint keeps track of the currently selected values. When the selection changes, animint updates the subset of data that is shown.

Each of the data visualizations sketched above has only one selection variable. However, a data visualization can have any number of selection variables. In the next section, we will explore a visualization of the World Bank data that has selection variables for `region` and `year`.

3.2 Selecting data subsets using menus

Consider the following sketch which adds a `showSelected` variable, and changes the data set.

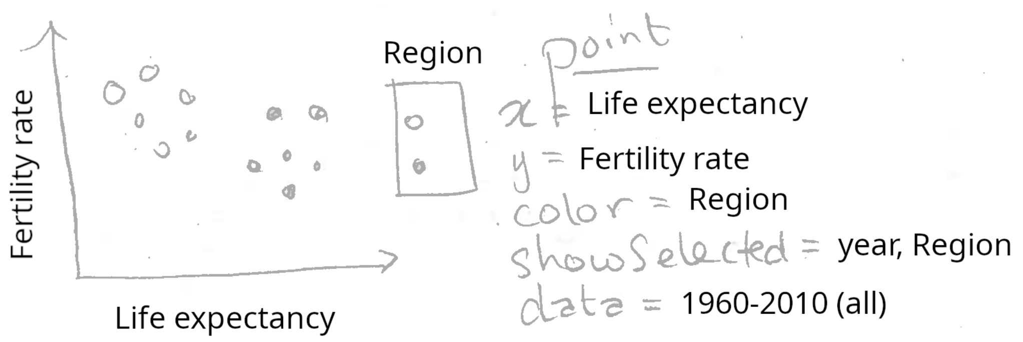
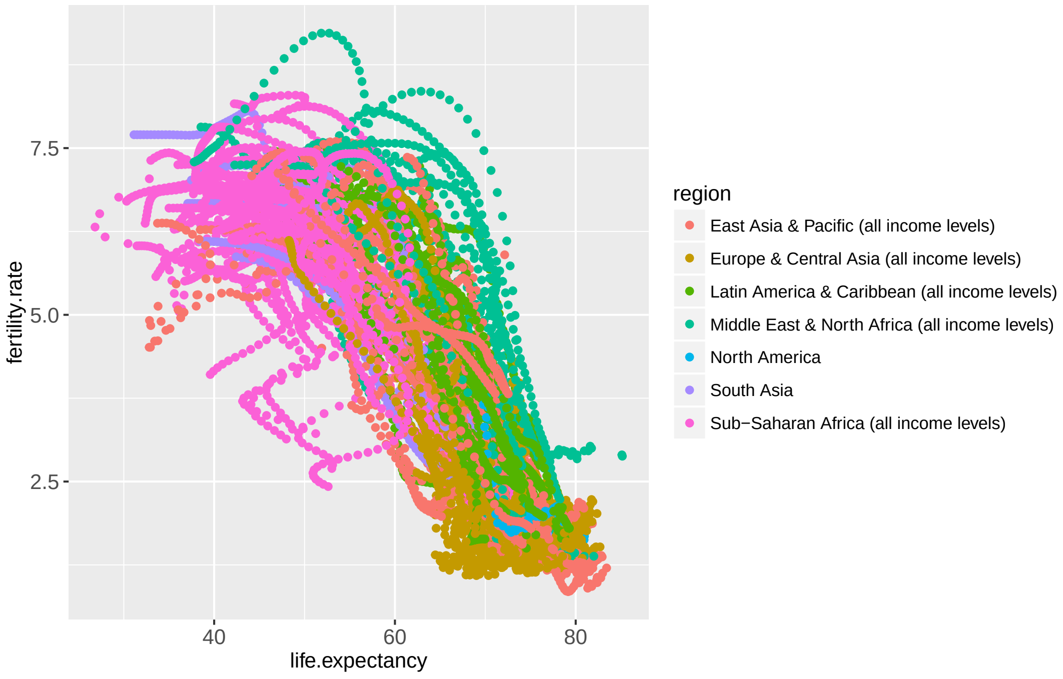


Figure 3.3: WorldBank data viz with showSelected

Note that there are two `showSelected` variables, `region` and `year`. Also note that the data is specified as all years (but only one will be shown at a time due to `showSelected=year`). Below, we translate this sketch into R code.

```
library(animint2)
data(WorldBank)
scatter <- ggplot()+
  geom_point(aes(
    x=life.expectancy, y=fertility.rate, color=region),
    showSelected="year",
    data=WorldBank)
scatter
```

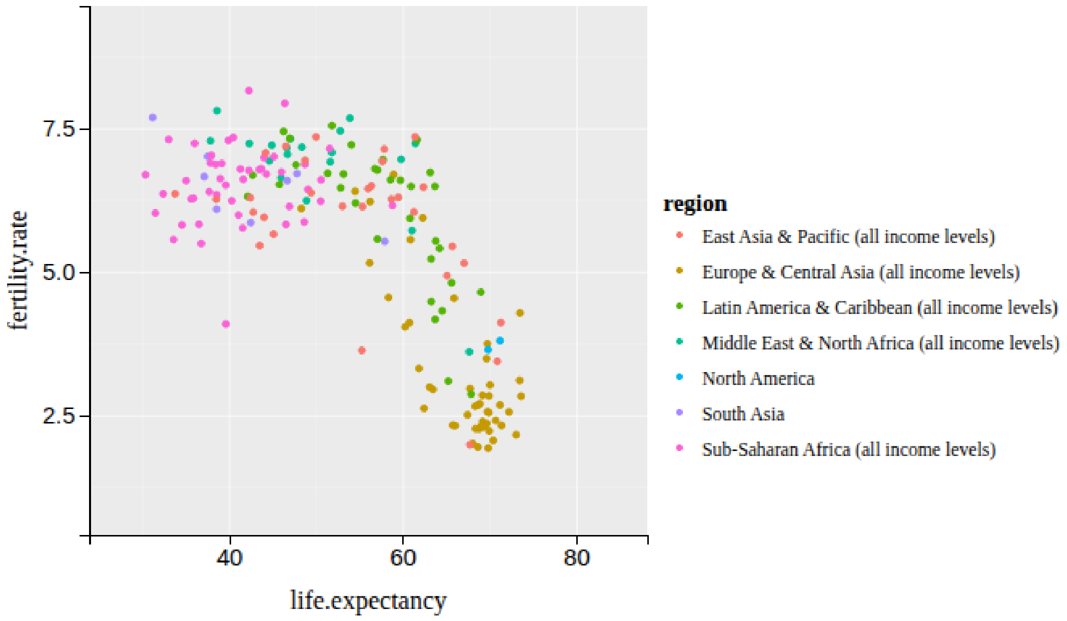
```
Warning: Removed 1490 rows containing missing values (geom_point).
```



Note that the ggplot above contains the `showSelected` argument, one of the two main features introduced in `animint2`. The `showSelected` keyword is ignored when rendering the plot using the usual R graphics devices, which produce a scatterplot with one point for every country and year. Note that since `color=region` was specified, animint also automatically uses `region` as a `showSelected` variable.

In constrast, rendering the same ggplot using animint yields the interactive data visualization below.

```
animint(scatter)
```



Note that the data viz above has two selection variables: region and year. Each variable has a menu at the bottom of the data viz that can be used to change the current selection. In this data viz, these selection menus are shown by default. They can be hidden by clicking the "Hide selection menus" button, and shown again by clicking the "Show selection menus" button.

Discrete legend variables such as region default to multiple selection, so several values are selected and shown at once. **Try** changing the selected region in the interactive legend and the selection menu. When you change the selection using either method, both the interactive legend and the selection menu should update to reflect the current selection.

We use the terms "direct manipulation" and "indirect manipulation" to describe these different ways of changing the selection. Direct manipulation typically involves clicking on the objects that you want to change, and is usually easier to understand. In contrast, indirect manipulation techniques such as menus are typically more complicated to understand. In the animint above, you can change the value of the `region` variable using either the legend or the menu. Using the legend is a more direct manipulation technique, since the legend is drawn closer to the plotted data points that will be updated.

Other selection variables such as year default to single selection, so only one value is selected and shown at any time. **Try** changing the selected value of the year variable using the selection menu. You should see the points in the scatterplot immediately update to show the fertility rate and life expectancy of all the countries in the year that you selected.

Multi-layer exercise: Add another geom to this interactive scatterplot. As in Chapter 2, you can use a `geom_text` to show the name of each country (easy), or a `geom_text` to show the selected year (medium), or a `geom_path` to show the previous 5 years of data (hard). Hint: make sure to specify `showSelected=year` for all geoms.

Multi-plot exercise: Add a time series plot to the data viz above. As in Chapter 2, you can use a `geom_line` to show the fertility rate for each country over all years. Add a `geom_vline` with `showSelected=year` to highlight the currently selected year.

3.3 Transitions: the duration option and key aesthetic

You may have noticed that there are buttons at the bottom of each data visualization created by animint. Try clicking the “Show animation controls” button above. This table contains a row for each selection variable. The text boxes show the number of milliseconds that are used for transition durations after updating each selection variable. The default transition duration for each selection variable is 0, meaning data will be immediately placed at their new positions after updating each variable.

To illustrate the significance of transition durations, **try** changing the transition duration of the year variable to 2000. Then, change the selected value of the year variable. You should see the data points move slowly to their new positions, over a duration of 2 seconds.

Some transitions result in points moving only a little bit, to nearby positions (e.g. 1979-1980). Other transitions result in points moving a lot more, to far away locations (e.g. 1980-1981). Why is that?

Smooth transitions only make sense for data points that exist both before and after changing the selection. In the R code below we compute a table of counts of data points that can be plotted in each of these three years.

```
three.years <- subset(WorldBank, 1979 <= year & year <= 1981)
can.plot <- with(three.years, {
  (!is.na(life.expectancy)) & (!is.na(fertility.rate))
})
table(three.years$year, can.plot)
```

```
      can.plot
       FALSE TRUE
  1979    27  187
  1980    27  187
  1981    26  188
```

It is clear from the table above that there are 187 points that can be plotted in 1979 and 1980. However, in 1981 there is one more data point, corresponding to a country for which we did not have data in 1980. Below we show the data for that country, Kosovo.

```
subset(three.years, country=="Kosovo")
```

```
     iso2c country year fertility.rate life.expectancy population
5850    KV  Kosovo 1979             NA              NA    1491000
5851    KV  Kosovo 1980             NA              NA    1521000
5852    KV  Kosovo 1981         4.5758        65.93268    1552000
     GDP.per.capita.Current.USD 15.to.25.yr.female.literacy iso3c
5850                         NA                          NA   KSV
5851                         NA                          NA   KSV
5852                         NA                          NA   KSV
                                         region  capital longitude latitude
5850 Europe & Central Asia (all income levels) Pristina    20.926   42.565
5851 Europe & Central Asia (all income levels) Pristina    20.926   42.565
```

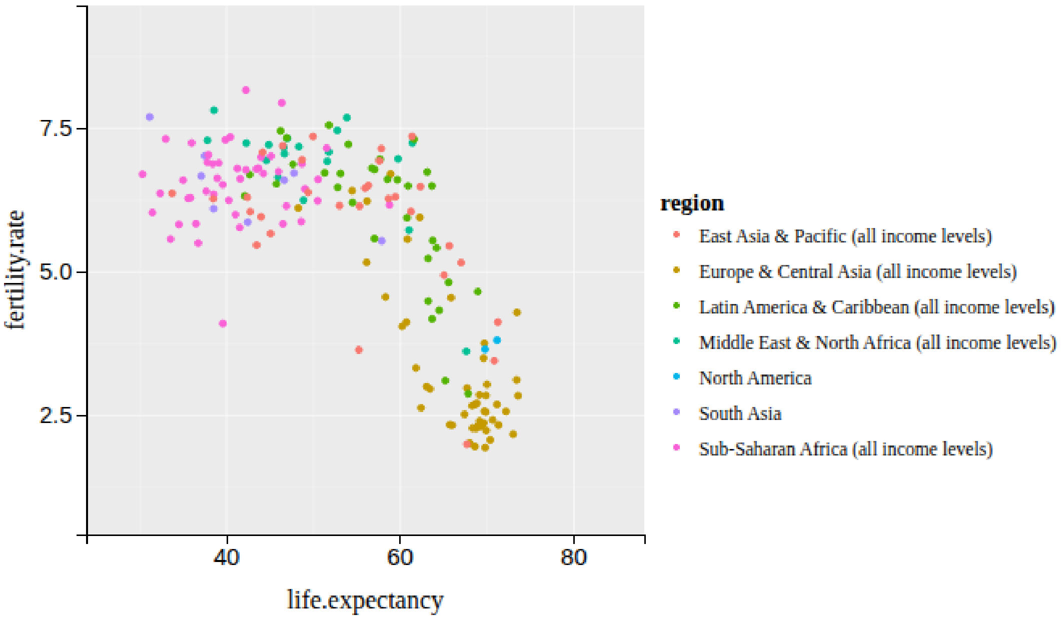
```
5852 Europe & Central Asia (all income levels) Pristina    20.926   42.565
                  income lending
5850 Lower middle income     IDA
5851 Lower middle income     IDA
5852 Lower middle income     IDA
```

Indeed, the table above shows that fertility rate and life expectancy are missing for Kosovo during 1979-1980. Thus it does not make sense to do a smooth transition for countries such as Kosovo which would not be plotted either before or after the transition. How to specify that in the data visualization? In the code below, we use `aes(key=country)` to specify that the `country` variable should be used to match data points before and after changing the selection.

```
scatter.key <- ggplot()+
  geom_point(aes(
    x=life.expectancy, y=fertility.rate, color=region,
    key=country),
    showSelected="year",
    data=WorldBank)
```

The `key` aesthetic in the ggplot above is only meaningful for interactive data visualization, so it ignored when rendering with the usual R graphics devices. However, if we render this ggplot using `animint2`, the `country` variable will be used to make sure transtion durations are meaningful. To specify a default transition duration for the `year` variable, we use the `duration` option in the data viz below.

```
(viz.duration <- animint(scatter.key, duration=list(year=2000)))
```



The `duration` option must be a named list. Each name should be a selection variable, and each value should specify the number of milliseconds to use for a transition duration when the selected value of that variable is changed.

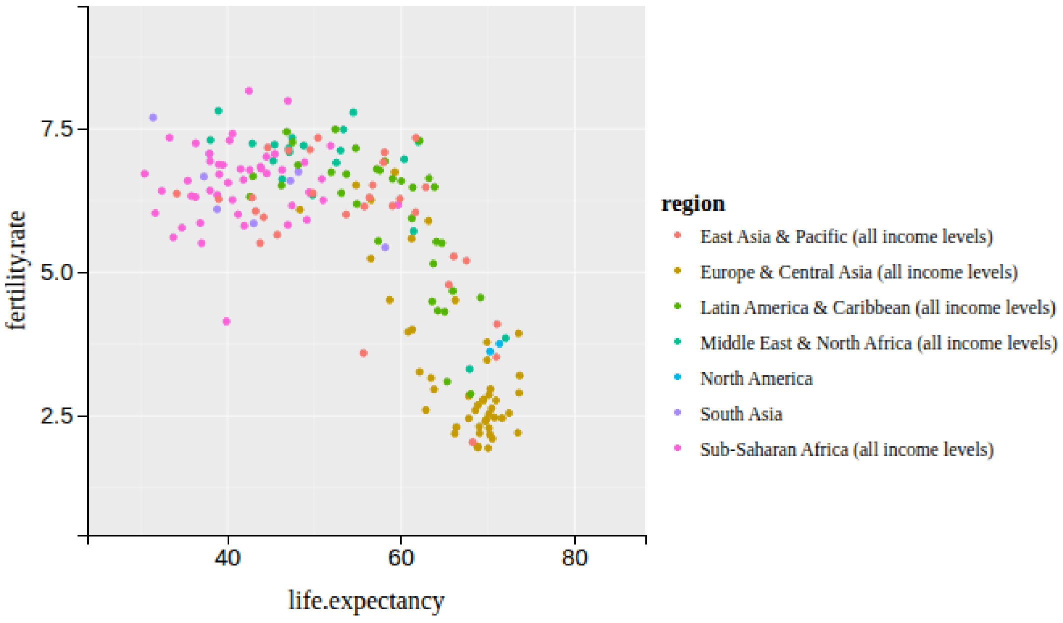
If you click "Show animation controls" in the data viz above, you will see that the text box for the year variable is 2000, as specified in the R code. If you change the selection from 1980 to 1981, you should see a proper transition.

In general the `key` aesthetic should be specified for all geoms that use `showSelected` with a variable that appears in the `duration` option. In this example, we used the `duration` option to specify a smooth transition for the `year` variable. Since we use `showSelected=year` in the `geom_point`, we also specified the `key` aesthetic for this geom.

3.4 Animation: the time option

The `time` option is used to specify a variable to use for animation. The code below specifies `year` as the variable to animate over time, with an update every 2000 milliseconds.

```
viz.duration.time <- viz.duration
viz.duration.time$time <- list(variable="year", ms=2000)
viz.duration.time
```



The data visualization above is animated, because the selected year advances every two seconds.

Exercise: make an animated data visualization that does NOT use smooth transitions. Hint: make a list of ggplots that has the `time` option but no `duration` option.

3.5 Chapter summary and exercises

This chapter explained the `showSelected` keyword, selection menus, transition durations, and animation.

Exercises:

- Make an improved version of `viz.aligned` from the previous chapter. Instead of fixing the year at 1975, use `showSelected=year` so that the user can select a year. Add geoms that show the selected year: a `geom_text` on the scatterplot, and a `geom_vline` on the time series.
- Translate one of the animation package examples to an animint. Hint: in the code for the animation package there is always a for loop over the time variable. Instead of calling a plotting function inside the for loop, use the list of data tables idiom to store the data that should be plotted. Then use those data along with `showSelected` to create ggplots, and render them using animint.

Next, Chapter 4 explains the `clickSelects` keyword, which indicates a geom that can be clicked to update a selection variable.

4

The `clickSelects` *keyword*

This chapter explains clickSelects, one of the two main keywords that animint introduces for interactive data visualization. The clickSelects keyword specifies a geom for which clicking updates a selection variable. Each geom in a data visualization has its own data set, and its own definition of the clickSelects keyword. So clicking on different geoms can change different selection variables.

After reading this chapter, you will be able to

- Understand how interactive legends implicitly use clickSelects.
- Use the clickSelects keyword in your plot sketches.
- Translate your plot sketches with clickSelects into R code.
- Use the `selector.types` option to specify multiple selection variables.

4.1 Interactive legends implicitly use clickSelects

In this section, we will explain how the clickSelects keyword is implicitly used in interactive legends. If you have read the previous chapters, you have already implicitly used clickSelects, which was automatically created for the interactive legends in the previous chapters. For example, consider the sketch of the World Bank data viz from the last chapter.

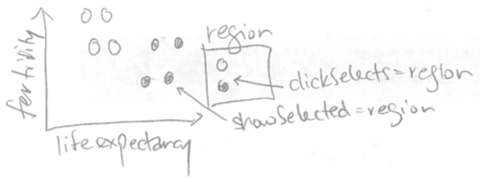


Figure 4.1: legend clickSelects

Since the legend has `clickSelects=region`, clicking an entry of that legend updates the `region` selection variable. Note that animint automatically makes every discrete legend interactive, so you do not need to explicitly specify `clickSelects=region` for the legend. In fact, when we specified `color=region` for the `geom_point`, animint2 does two things automatically:

- `showSelected=region` is assigned to the same `geom_point`.
- `clickSelects=region` is assigned to the color legend.

Note that `clickSelects` keywords are not limited to interactive legends. Each geom has its own `clickSelects` variable, which determines which selection variable is updated after clicking that geom. In the next section we will give several examples of how `clickSelects` can be used in combination with `showSelected` to create interactive data visualizations.

4.2 Use clickSelects to identify points on a scatterplot

The goal of this section is to create the following visualization of the World Bank data.

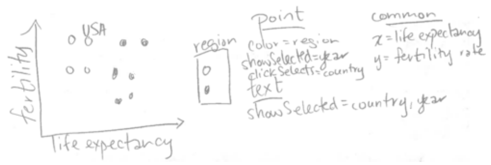
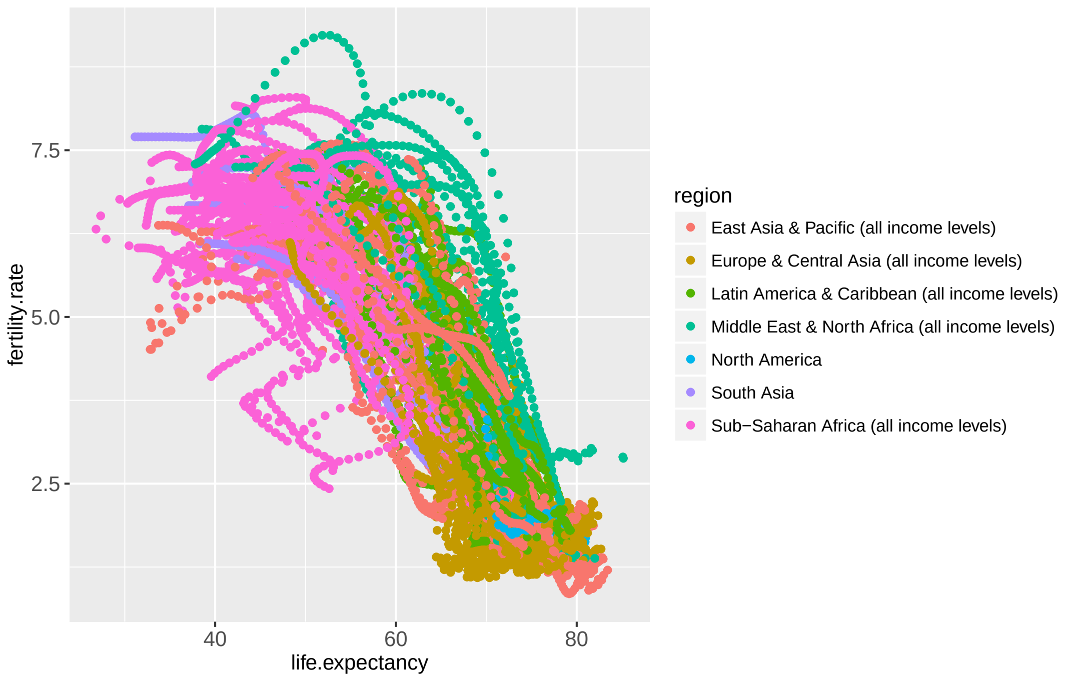


Figure 4.2: World Bank viz text

To start, consider the following R code which generates a scatterplot of the World Bank data:

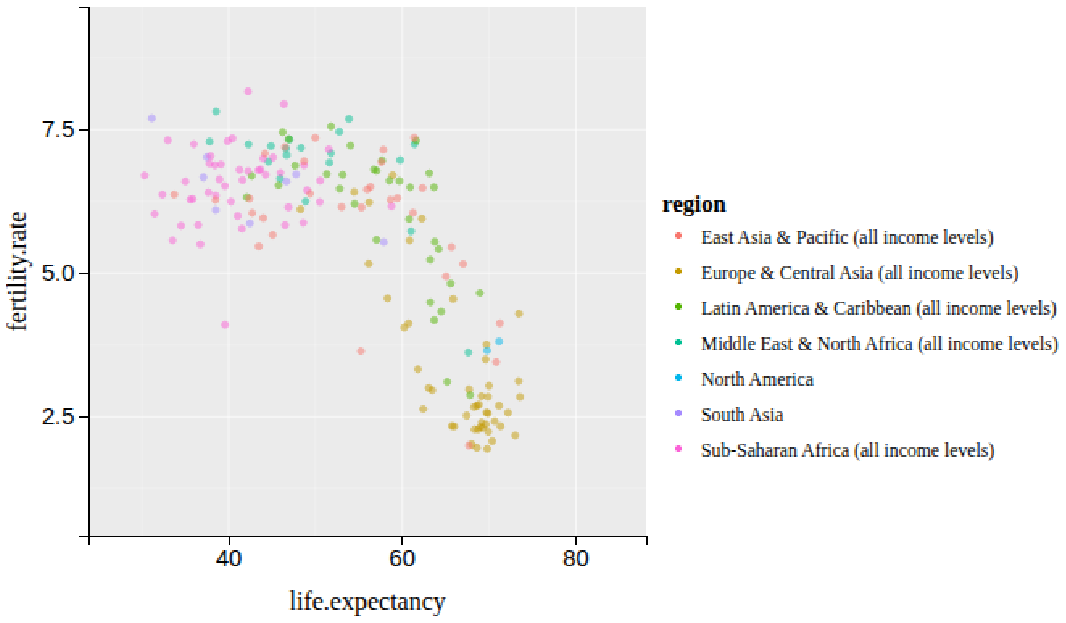
```r
library(animint2)
data(WorldBank)
scatter <- ggplot()+
  geom_point(aes(
    x=life.expectancy, y=fertility.rate, color=region,
    key=country),
    showSelected="year",
    clickSelects="country",
    data=WorldBank)
scatter
```

```
Warning: Removed 1490 rows containing missing values (geom_point).
```



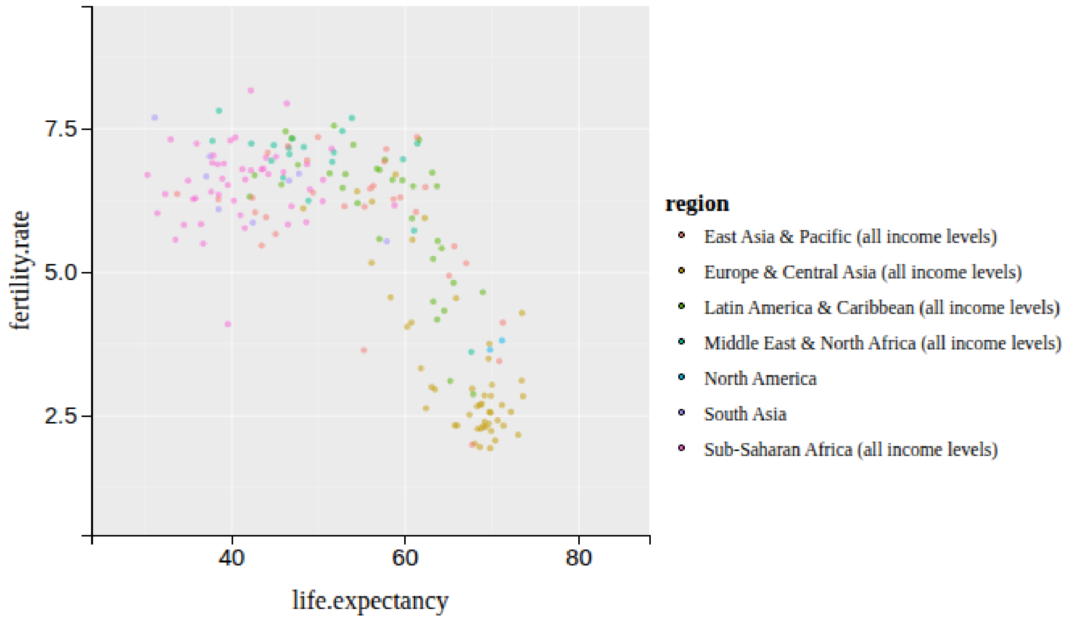
Note that the plot above is not interactive, because it is rendered using the traditional R graphics device. In contrast, rendering the same ggplot using `animint2` results in the following interactive plot:

```
(viz.scatter <- animint(
  scatter=scatter,
  duration=list(year=2000)))
```



Try clicking data points in the scatterplot above. You should see the value of the `country` selection menu change after clicking a data point. You should also see that the data point for the selected country is darker than the others. This serves to highlight the current selection, and is performed automatically for each geom with `clickSelects`. By default the selected point has alpha=1 (fully opaque, no transparency), and the other points have alpha=0.5 (semi-transparent). These defaults can be customized; for example in the code below a black outline is used to highlight the current selection.

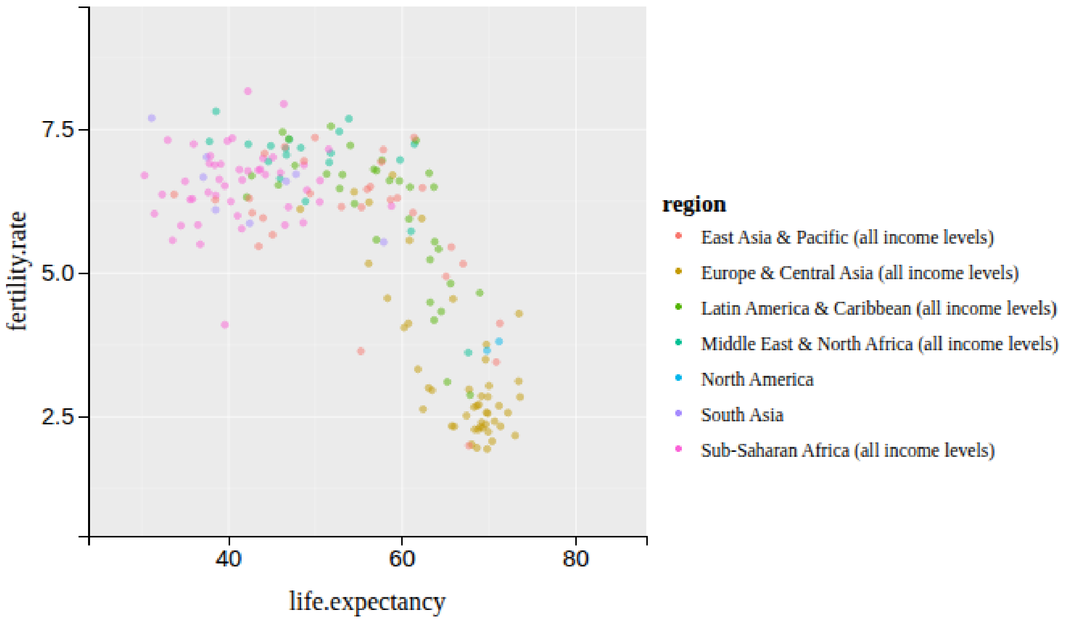
```
animint(
  ggplot()+
    geom_point(aes(
      x=life.expectancy, y=fertility.rate, fill=region,
      key=country),
      shape=21,
      color="black",
      color_off=NA,
      showSelected="year",
      clickSelects="country",
      data=WorldBank))
```



The data visualization above shows the currently selected country name in the selection menu, but it would be better to show it as a text label on the scatterplot. We can do that by adding a `geom_text` layer with two showSelected variables:

```
viz.text <- viz.scatter
viz.text$scatter <- scatter+
  geom_text(aes(
    x=life.expectancy, y=fertility.rate, label=country,
    key=country),
    showSelected=c("year", "country"),
    data=WorldBank)
```

```
viz.text
```



After clicking a data point in the scatterplot above, you should see a text label with the country name appear. Furthermore, **try** changing the year using the selection menu. You should see the text label move in a smooth transition along with the corresponding data point.

The data visualization above contains more than one geom, each with different interactive features. **Try** clicking the "Start Tour" button at the bottom of the data visualization, which will show what interactive features are available for the first geom in the data visualization. Clicking Next will show information for the next geom, and clicking Done or the grey background will end the Tour. The "Start Tour" feature can be useful for new users of your data visualization to discover what interactive features are present in each geom. The information displayed during the tour can be customized, by specifying the `help` and `title` params of each geom.

As explained in the last chapter, any variable specified using the `showSelected` argument of a geom is treated as an interactive variable. In the example above, we specified two `showSelected` variables for the `geom_text`. This means to only draw a text label for the rows of the `WorldBank` data set that match the current values of both selection variables. Since each combination of `country` and `year` has one row in these data, only one text label will be shown at a time.

Try clicking the legend entry that corresponds to the region for the currently selected country (e.g. if Canada is selected, try clicking the North America legend entry). You should see the point disappear, but the text stay displayed.

Exercise: how can you get the text to disappear along with the point? Hint: you need to add a keyword to the `geom_text`.

In the last chapter, we introduced the terms "direct manipulation" and "indirect manipulation" to describe interactions with legends and menus. In the data viz above, we can change the value of the `country` selection variable by either clicking a data point (direct manipulation) or using the selection menu (indirect manipulation). Both techniques are useful, but for different purposes:

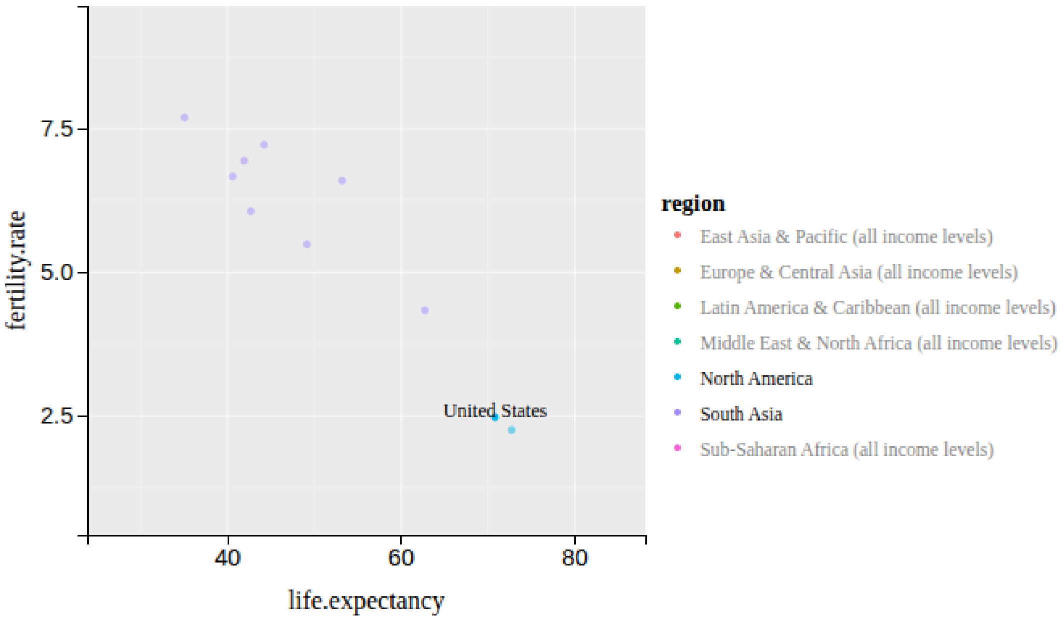
- Direct manipulation by clicking data points is useful to find the names of countries with extreme values of fertility rate and life expectancy. For example, for the year 1960, clicking the point at the bottom left of the plot reveals the country name Gabon.
- Indirect manipulation using menus is useful to see the plotted position of a country of interest. For example, it would be difficult to find France by clicking all the different points, but it is simple to find France by typing its name in the selection menu.

Note that when the data viz above is first rendered, the selected country is Andorra and the selected year is 1960. Since the data for Andorra is missing in 1960, there is no text label drawn at first. To change the first selection, you can specify the `first` option, as explained in the next section.

4.3 The first option

To specify the selection that should be shown when the data viz is first rendered, use the `first` option. It should be a named list with entries for each selection variable. For example, the code below specifies 1970 as the first year, United States as the first country, and North America and South Asia as the first regions.

```r
viz.first <- viz.text
viz.first$first <- list(
  year=1970,
  country="United States",
  region=c("North America", "South Asia"))
viz.first
```



Note that in the data viz above, there is only one country selected at a time. In the next section, we will explain how the `selector.types` option can be used to change `country` to a multiple selection variable.

4.4 The selector.types option

In this section our goal is to produce a slightly more complicated version of the scatterplot in the last section. The sketch below has only one difference with respect to the sketch from the last section: text labels are shown for more than one country.

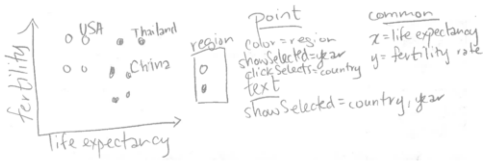
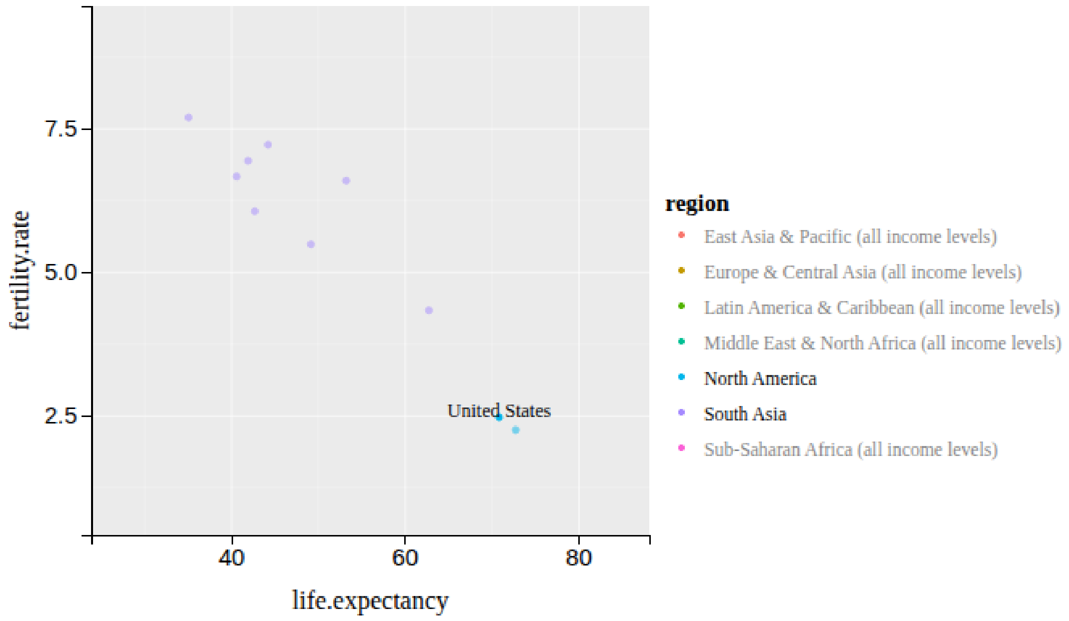


Figure 4.3: World Bank viz text

In animint, each selection variable has a type, either single or multiple. Single selection means that only one value can be selected at a time. Multiple selection means that any number of values can be selected at a time. In the plots in the last section, multiple selection was used for the `region` variable but not for the `year` and `country` variables. Why is that?

By default, animint assigns multiple selection to all variables that appear in interactive discrete legends, and single selection to other variables. However, single or multiple selection can be specified by using the `selector.types` option. In the R code below, we use the `selector.types` option to specify that `country` should be treated as a multiple selection variable.

```r
viz.multiple <- viz.first
viz.multiple$selector.types <- list(country="multiple")
viz.multiple
```



When the data viz above is first rendered, it shows data points from the year 1970, for each country in North America and South Asia. It also shows a text label for the United States.

You may have noticed that it is easy to add countries to the current selection, by clicking data points. Normally, clicking a selected data point will remove that country from the current selection. However, in this particular data viz, it is not so easy to remove them, since the text labels are rendered on top of the data points.

Exercise: Re-make the data viz above so that clicking a text label removes that country from the selection set. Hint: you need to add a `clickSelects` keyword to the `geom_text`.

Note that in the data viz above, the year variable can only be changed via the selection menu.

In the next section, we will add a facet with a geom that can be directly clicked to change the year variable.

4.5 Selecting a year on a time series plot

The goal of this section is to add a time series plot that can be clicked to change the selected year.

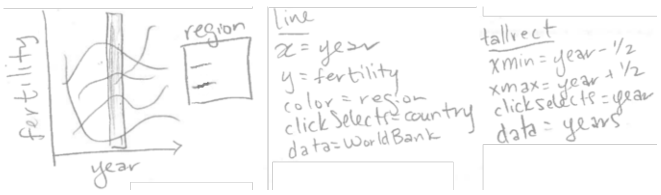


Figure 4.4: World Bank time series

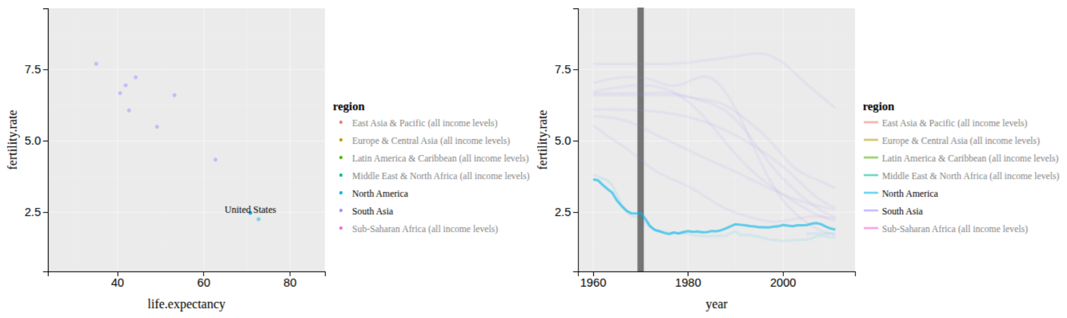
Note that the sketch above includes `geom_tallrect`, a new geom introduced in animint. It is “tall” because it occupies the entire vertical space of the plot, and thus only requires definition of its horizontal limits via the `xmin` and `xmax` aesthetics. Specifying `clickSelects=year` means that we want to be able to draw one tallrect for each year, and click a tallrect to change the selected year. Thus we need to create a new data set called `years` with one row for each unique year of the `WorldBank` data.

```r
years <- data.frame(year=unique(WorldBank$year))
head(years)
```

```
  year
1 1960
2 1961
3 1962
4 1963
5 1964
6 1965
```

Next, we add the time series ggplot to the existing data viz.

```r
viz.timeSeries <- viz.multiple
viz.timeSeries$timeSeries <- ggplot()+
  geom_tallrect(aes(
    xmin=year-0.5, xmax=year+0.5),
    clickSelects="year",
    alpha=0.5,
    data=years)+
  geom_line(aes(
    x=year, y=fertility.rate, group=country, color=region),
    clickSelects="country",
    size=3,
    alpha=0.6,
    data=WorldBank)
viz.timeSeries
```



Try clicking the background of the time series in the data viz above. You should see the data points and text labels move in a smooth transition to their places at the newly selected year.

Animation exercise: make the data viz animated by specifying the `time` option, as explained in Chapter 3.

Multi-layer exercise: add a `geom_text` that shows the current year on the scatterplot. Add a `geom_path` that shows data for the previous 5 years.

4.6 Selecting a year on a time series facet

The goal of this section is to add a facet with a time series plot that can be clicked to change the selected year.

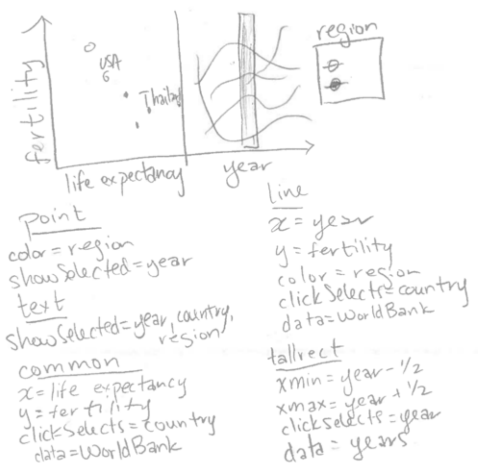
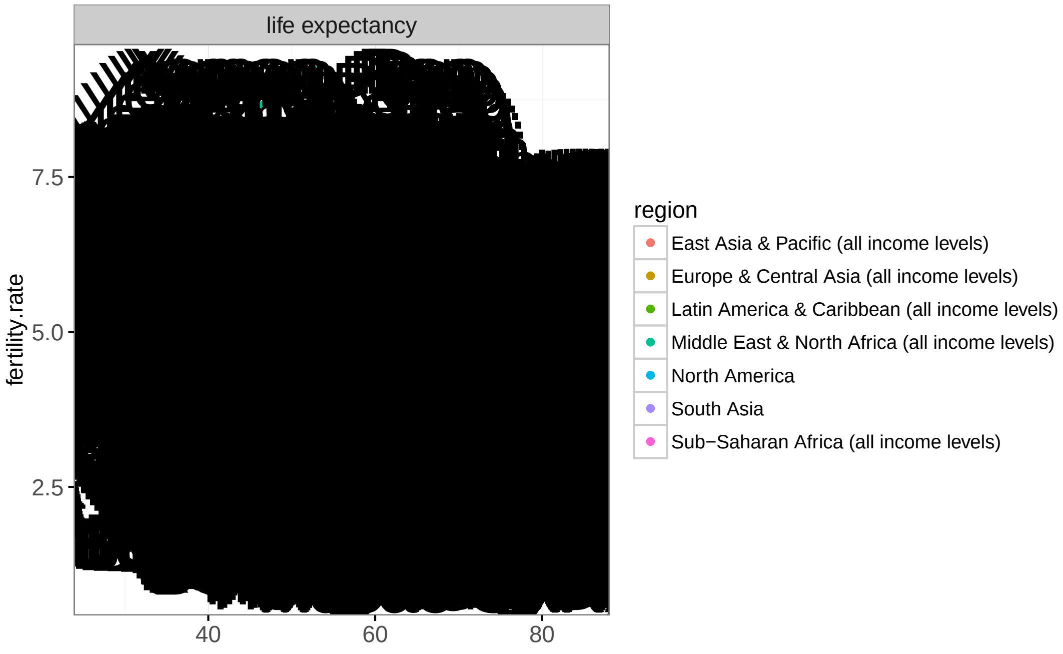


Figure 4.5: World Bank scatter facet

First, we re-create the scatterplot from the previous section using the addColumn then facet idiom, which is useful for creating ggplots with aligned axes.

```r
add.x.var <- function(df, x.var){
  data.frame(df, x.var=factor(x.var, c("life expectancy", "year")))
}
scatterFacet <- ggplot()+
  geom_point(aes(
    x=life.expectancy, y=fertility.rate, color=region,
    key=country),
    showSelected="year",
    clickSelects="country",
    data=add.x.var(WorldBank, "life expectancy"))+
  geom_text(aes(
    x=life.expectancy, y=fertility.rate, label=country,
    key=country),
    clickSelects="country",
    showSelected=c("year", "country", "region"),
    data=add.x.var(WorldBank, "life expectancy"))+
  facet_grid(. ~ x.var, scales="free")+
  xlab("")+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "lines"))
  scatterFacet
```

```
Warning: Removed 1490 rows containing missing values (geom_point).

Warning: Removed 1490 rows containing missing values (geom_text).
```

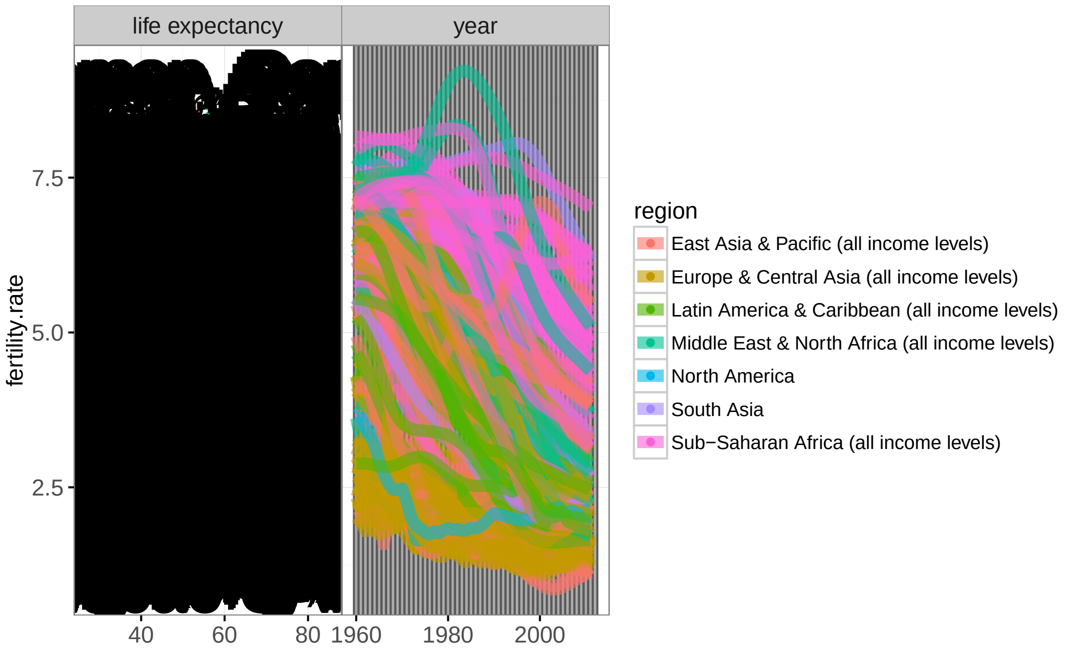


Note that the ggplot above uses the same `aes` definitions as the scatterplot from the previous section. The only difference is that we have used an augmented `WorldBank` data set with an additional `x.var` variable that we use with `facet_grid`. Below, we add geoms for a time series plot that is aligned on the fertility rate axis.

```
  scatterTS <- scatterFacet+
    geom_tallrect(aes(
      xmin=year-0.5, xmax=year+0.5),
      clickSelects="year",
      alpha=0.5,
      data=add.x.var(years, "year"))+
    geom_line(aes(
      x=year, y=fertility.rate, group=country, color=region),
      clickSelects="country",
      size=3,
      alpha=0.6,
      data=add.x.var(WorldBank, "year"))
  scatterTS
```

```
Warning: Removed 1490 rows containing missing values (geom_point).

Warning: Removed 1490 rows containing missing values (geom_text).
```

```
Warning: Removed 759 rows containing missing values (geom_path).
```



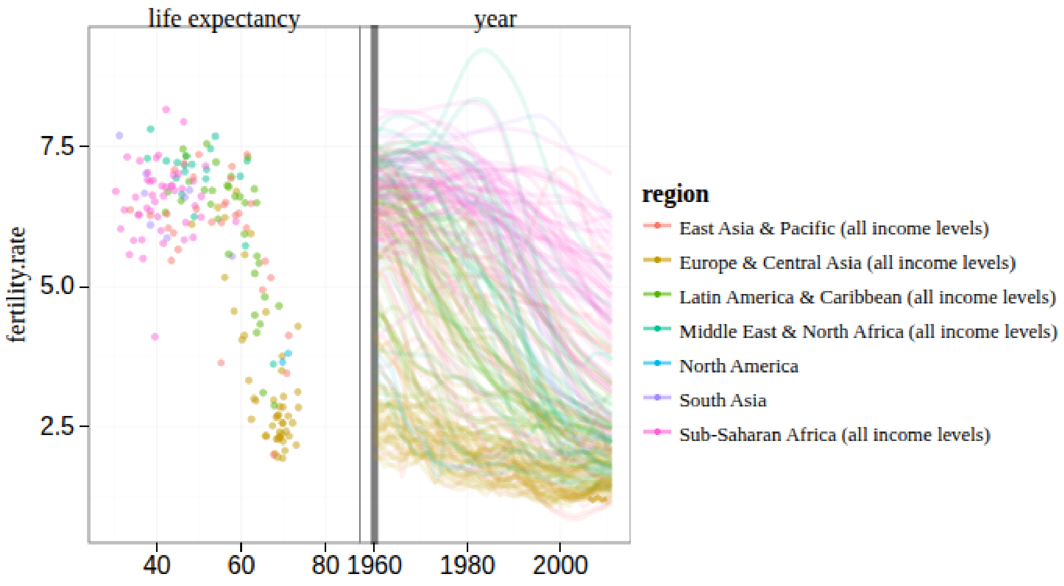
The two geoms defined above occupy a new facet for the `"year"` value of the `x.var` variable (defined by the `add.x.var` function). Since these two geoms have different definitions of `clickSelects`, clicking each geom will update the plot in a different way. Note that for the `geom_line` we specify `size=3`, which means a line stroke width of 3 pixels. In general it is a good idea to increase the size of geoms with `clickSelects`, to make them easier to click.

Also note that we specified `alpha=0.5` for the `geom_tallrect` and `alpha=0.6` for the `geom_line`. Since both of these geoms define `clickSelects`, some plotted lines and tallrect will be selected, and others will not be selected. The alpha values in R code specify the opacity of the selected objects, and other objects will have an `alpha` opacity which is 0.5 less than that value. In the example above, the un-selected lines will have `alpha=0.1`, and the un-selected tallrects will have `alpha=0` (completely transparent).

As of September 2023, it is also possible to specify the opacity, fill, and color for objects which are not currently selected (`alpha_off`, `fill_off`, `color_off`). Users can specify these parameters in the geom (not aes) to freely create a different appearance for selected and un-selected items, instead of being forced to rely on the behavior described above. For more information, see the discussion and example in Chapter 6, section Specifying how selection state is displayed.

Finally, we use the R code below to render the new aligned scatterplot and time series using animint.

```r
(viz.facets <- animint(scatterTS))
```



The interactive data viz above contains a new panel with lines that show a fertility rate time series over all years. Since we specified `clickSelects=country` for the `geom_line`, clicking a line updates the set of selected countries. Since we specified `clickSelects=year` for the `geom_tallrect`, clicking on a tallrect updates the selected year.

Exercise: add `time`, `duration`, `first`, and `selector.types` options to the data viz above.

4.7 Chapter summary and exercises

This chapter explained `clickSelects`, one of the two main keywords that animint introduces for interactive data visualization design. We used the World Bank data set to show how `clickSelects` can be used to specify different interactions for each of the plotted geoms. We explained how the `first` option can be used to specify the selected values that are used when the animint is first rendered. We also explained how the `selector.types` option can be used to specify multiple selection variables.

Exercises:

- So far we have seen three different ways to change selection variables: (1) interactive legends, (2) selection menus, and (3) clicking data with `clickSelects`. Order these three techniques in terms from most to least direct manipulation. Which technique is preferable in what circumstances?
- When `geom_point(clickSelects=something, alpha=0.75)` is rendered with the usual R graphics device, how much opacity/transparency is present for all data points? When `animint2` renders the same geom, some points will be selected and others not. What is the opacity/transparency of selected points? What is the opacity/transparency of points which are not selected?
- Add `aes(size=population)` to the points in the World Bank scatterplot. Is the size legend interactive? Why?
- Add a `geom_text` to the World Bank scatterplot that shows the selected year.
- Add a `geom_text` to the World Bank time series to show the names of the selected

countries.
- Add a `geom_path` to the World Bank scatterplot to show data for the last 5 years.
- Use the `time` option to make an animated version of `viz.facets`.
- Use `help` and `title` params for each geom in `viz.facets`, to make a Tour which is more informative about what is displayed in each geom.

Next, Chapter 5 explains several different methods for publishing and sharing animints on the web.

5 Sharing

This chapter explains several methods for sharing your interactive data visualizations on the web. After reading this chapter, you will be able to view animints

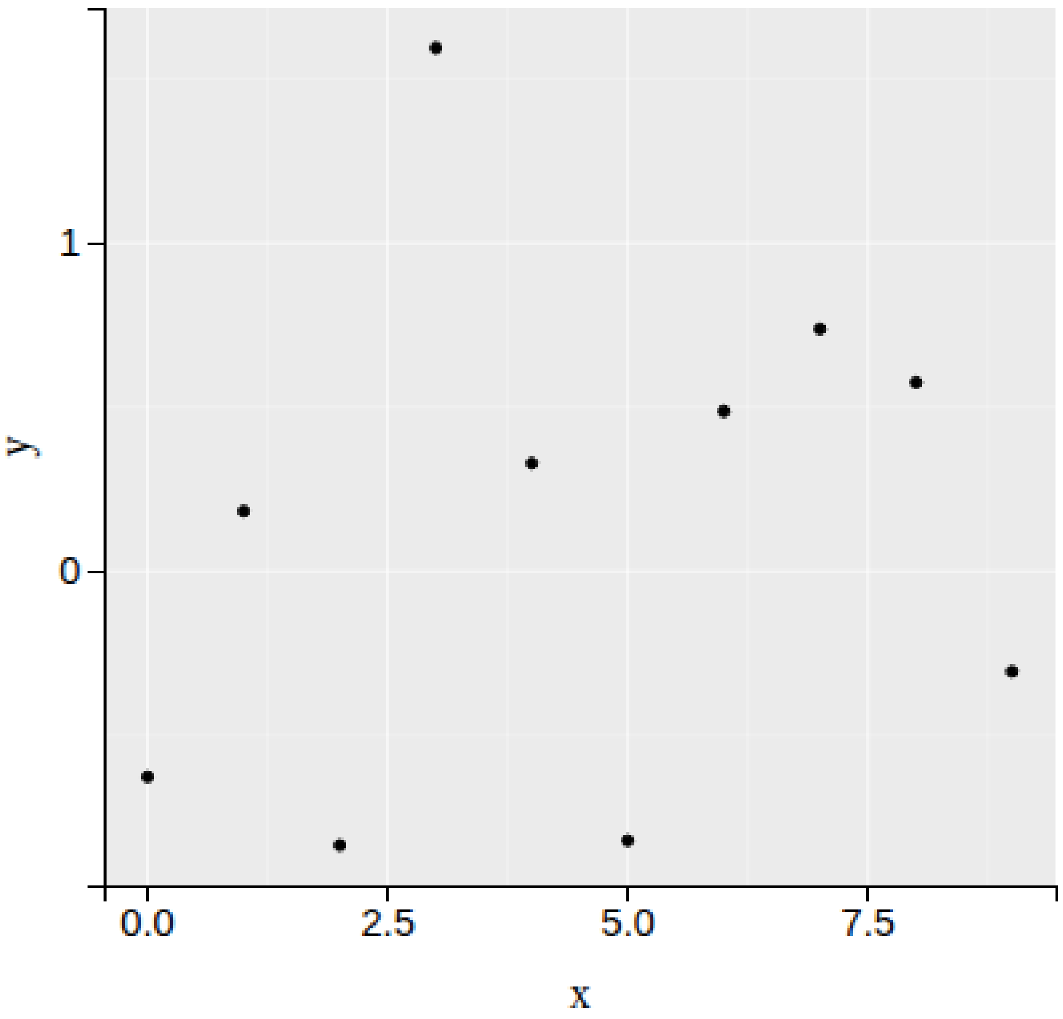
- from a local directory on your personal computer.
- in R Markdown documents.
- using any web server, including Netlify Drop.
- published using GitHub pages, and organized into a group called a gallery.

5.1 Compile to a local directory

When experimenting with different interactive data visualization designs, it is useful to preview them on your personal computer before publishing on the web. This section discusses two methods for compiling animints to a local directory.

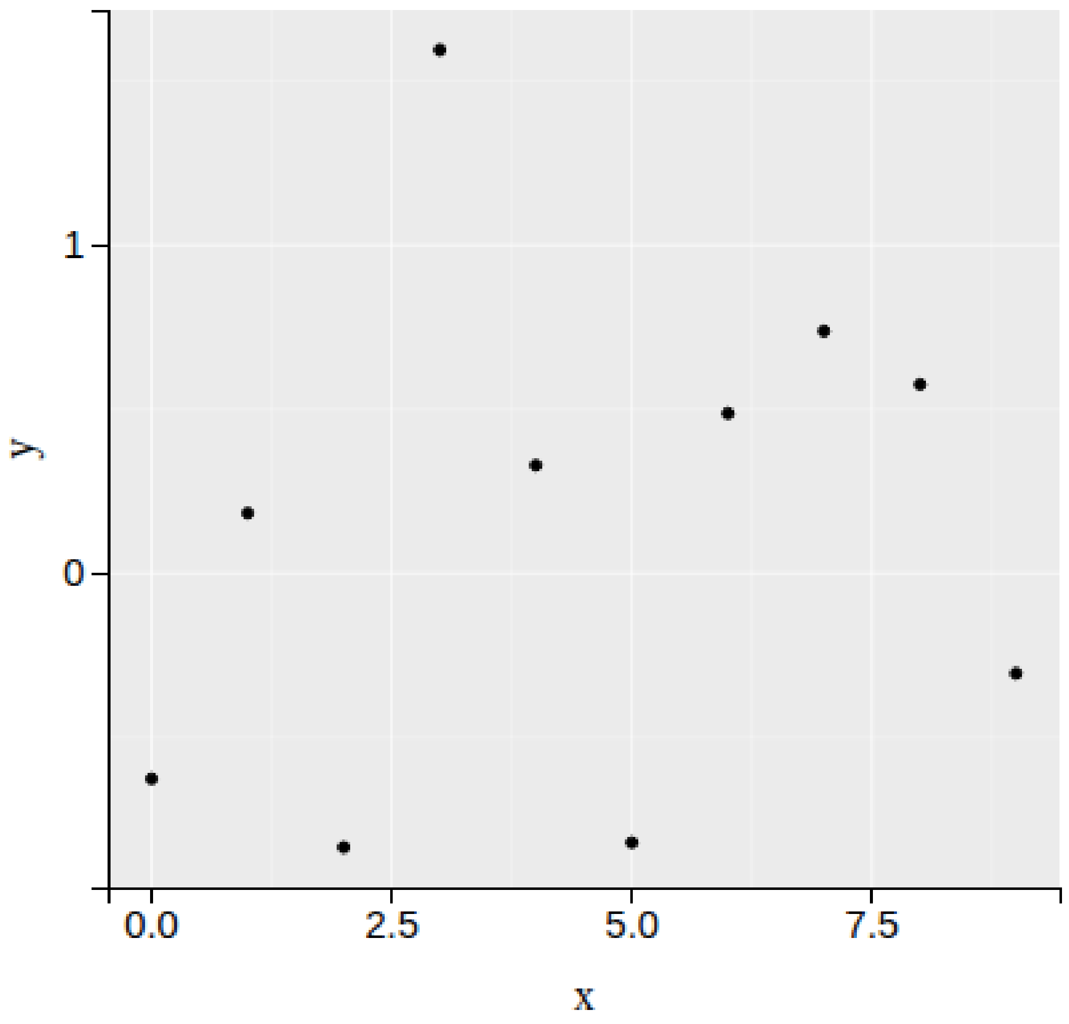
So far in previous chapters we have only discussed one method for creating interactive data visualizations. If `viz` is an animint (list of ggplots and options with class animint), then printing it on the R command line compiles that animint to a temporary directory, using code like this,

```r
set.seed(1)
ten.points <- data.frame(x=0:9, y=rnorm(10))
library(animint2)
animint(
  point=ggplot()+
    geom_point(aes(
      x, y),
      data=ten.points)
)
```



The code above saves the animint to a new temporary directory. Rather than saving each animint to a separate temporary directory, you can specify an output directory via the `out.dir` argument to `animint`. If you want to save the animint in the `"Ch05-sharing-ten-points"` directory, use:

```
animint(
  point=ggplot()+
    geom_point(aes(
      x, y),
      data=ten.points),
  out.dir="Ch05-sharing-ten-points"
)
```



If the parent of `out.dir` does not exist, that is an error (you can use `dir.create` to create the parent if necessary). If `out.dir` does not exist, then it will be created. If `out.dir` does exist (and contains a file named `animint.js`), then any files in that directory will be overwritten. To view the data viz, navigate to `Ch05-sharing-ten-points/index.html` in a web browser (which should be done automatically / by default). If the web page is blank, you may need to configure your browser to allow execution of local JavaScript code, as explained on our FAQ.

Internally, R calls the `print.animint` S3 method, which calls `animint2dir` to compile it to a new temporary directory on your personal computer. Generally we advise to avoid calling `animint2dir` directly, but it is useful if you want to avoid opening lots of similar browser windows when repeatedly revising and rendering an animint. You can prevent the default behavior of opening a browser window via:

```
viz <- animint(
  point=ggplot()+
    geom_point(aes(
      x, y),
      data=ten.points))
animint2dir(
  viz,
  out.dir="Ch05-sharing-ten-points-again",
  open.browser=FALSE)
```

5.2 Publish in R Markdown

To include an animint in an R Markdown document, use `animint(...)` inside of an R code chunk. R will run the `knit_print.animint` S3 method, which compiles the animint to a local directory, named based on the name of the R code chunk. For example a code chunk named `viz-facets` will be saved in the directory `vizfacets`. Make sure to put each animint in its own code chunk (do not put two animints in the same code chunk).

5.3 Publish on a web server

Since animints are just directories with HTML, TSV, and JavaScript files, you can publish them on any web server by simply copying the directory to that server.

For example I executed the World Bank data viz example code to create the `WorldBank-facets` directory on my personal computer.[^1] I copied that directory to my lab web server using `rsync -r WorldBank-facets/ monsoon.hpc.nau.edu:genomic-ml/WorldBank-facets/` and so I can view it on my university web server.[^2]

[^1] https://github.com/animint/animint2/blob/master/inst/examples/WorldBank-facets.R

[^2] https://rcdata.nau.edu/genomic-ml/WorldBank-facets/

If you don't have access to a personal/lab web server, try using one of the methods described below, which are free for anyone.

5.4 Publish on Netlify Drop

Netlify Drop is for hosting static web sites. To publish your data viz there, simply drag a directory to that web page (it can be a directory resulting from `animint2dir`, or from `rmarkdown::render` if your animint is inside Rmd, as described above). After the upload completes, you will be provided a link which can be used to view the files in that directory. No registration/login is required, but if you do not register an account, your data viz will be deleted after one hour. You can register for a free account if you want your data viz to be available longer. Another limitation is 54,000 files per directory, as mentioned in the Deploy Overview docs.

5.5 Publish on GitHub Pages

GitHub Pages is a service that provides static web site hosting, and can be used to publish animints. To publish an animint on GitHub Pages, you need a GitHub account, and the packages `gert` (for running git from R), `gh` (for using the GitHub API from R), and `gitcreds` (for interacting with the git credential store, easy authentication when pushing to GitHub). First, install those packages. If you don't have a GitHub account, you can sign up for free. Then make sure to tell R what name/email to use for git commits:

```
gert::git_config_global_set("user.name", "<your_full_name>")
gert::git_config_global_set("user.email", "<your_email>")
```

You can then use `animint2pages(viz, "new_repo")` function to publish your data `viz` to the `gh-pages` branch of `your_github_username/new_repo` (note that `your_github_username` is not specified in code, because `gitcreds` will get that information from the git credential store). It should print a message which tells you the URL/link where your data viz will be accessible. It takes a few minutes (usually not more than five) from the time you run animint2pages, to the time the data viz is published for viewing on GitHub Pages.

If you want to update an animint that has already been published on GitHub Pages, you can simply run `animint2pages(updated_viz, "existing_repo")`, which will update the `gh-pages` branch of the specified repository.

Beware that GitHub Pages imposes a limit of 100MB per file. For most animints, this limit should not be a problem. If your data viz contains a TSV file over 100MB, consider using the `chunk_vars` option to break that TSV file into several smaller files.

5.6 Organizing animints in a gallery

A gallery is a collection of meta-data about animints that have been published to GitHub Pages. For example, the main animint gallery is https://animint.github.io/gallery/ which is a web page that has links to various animints, organized into a table.

- There is one row for each animint.
- The first column `viz.link` shows a screenshot of the animint, which links to the animint web page.
- The second column `title` shows the name of the data viz (taken from the `title` option of the animint).
- The third column shows `links` to
  - the GitHub `repo` which hosts the data viz.
  - the R `source` code used to create that animint (taken from the `source` option of the animint).
  - optionally, a `video` that explains typical interactions with the animint.

A gallery is defined as a github repo that should have two source files in the `gh-pages` branch:

- `repos.txt` (list of github repositories which contain animints, one owner/repo per line), and
- `index.Rmd` (source for web page with links to animints).

To add a new animint to the gallery which is published in the `gh-pages` branch of `your_github_username/gallery_repo`, you can use the following method:

- Create a new animint in R code, and make sure to define the `source` and `title` options.
- use `animint2pages(viz, "viz_repo")` to publish that animint to the `gh-pages` branch of `your_github_username/viz_repo`.
- take a screenshot of that animint, and commit/push that screenshot as a file named `Capture.PNG` (case-sensitive), in the `gh-pages` branch of `your_github_username/viz_repo`.
- add `your_github_username/viz_repo` to the `repos.txt` file in the `gh-pages` branch of `your_github_username/gallery_repo`.
- run the R code `animint2::update_gallery("path/to/gallery_repo")` (note that a clone of the gallery repo must be present on the system where you run this function, and the GitHub remote must be named `origin`). It will read `gallery_repo/repos.txt`, read meta-data (`title`, `source`, `Capture.PNG`) from each repo which is not already present in `gallery_repo/meta.csv`, write updated meta-data files to the gallery, render `gallery_repo/index.Rmd` to `gallery_repo/index.html`, commit, and push to `origin`.
- the updated gallery should be viewable on the web, at https://your_github_username.github.io/gallery_repo after a few minutes (usually no more than five).

5.7 Chapter summary and exercises

This chapter explained how to share animints on the web.

Exercises:

- Create an animint using the options mentioned in this chapter: `out.dir` (name of directory to save animint on your computer), `source` (link to R source code used to create animint), `title` (description of animint).
- Use Netlify Drop to publish that animint on the web.
- Use `animint2pages` to publish that animint to a new GitHub repository. Create a screenshot and save it as `Capture.PNG` in the `gh-pages` branch of that repository. Add that repository to the main animint gallery repos.txt by submitting a Pull Request on GitHub.
- Create your own animint gallery repository, and add two or more of your own animints to that gallery.

Next, Chapter 6 explains the different options that can be used to customize an animint.

6 New features

This chapter gives a complete list of new features in `animint2`, beyond what is present in `ggplot2`. After reading this chapter, you will understand how to customize your `animint2` graphics via

- the `href`, `tooltip`, `id` aesthetics for observation-specific characteristics;
- named elements of `clickSelects` and `showSelected` for specifying several selection variables at once;
- new geoms: `tallrect`, `widerect`, `label_aligned`;
- the `chunk_vars` geom-specific option;
- the `color_off`, `fill_off`, and `alpha_off` geom parameters for specifying how selection state is displayed;
- the `help` and `title` geom-specific parameters, which can be set to text strings that will be shown in the guided tour;
- plot-specific options for legends, HTML table layout, and height/width;
- global options such as `duration`, `title`, and `source`.

6.1 Observation-specific options (new aesthetics)

This section explains the new aesthetics that are recognized by `animint2`.

6.1.1 Review of previously introduced aesthetics

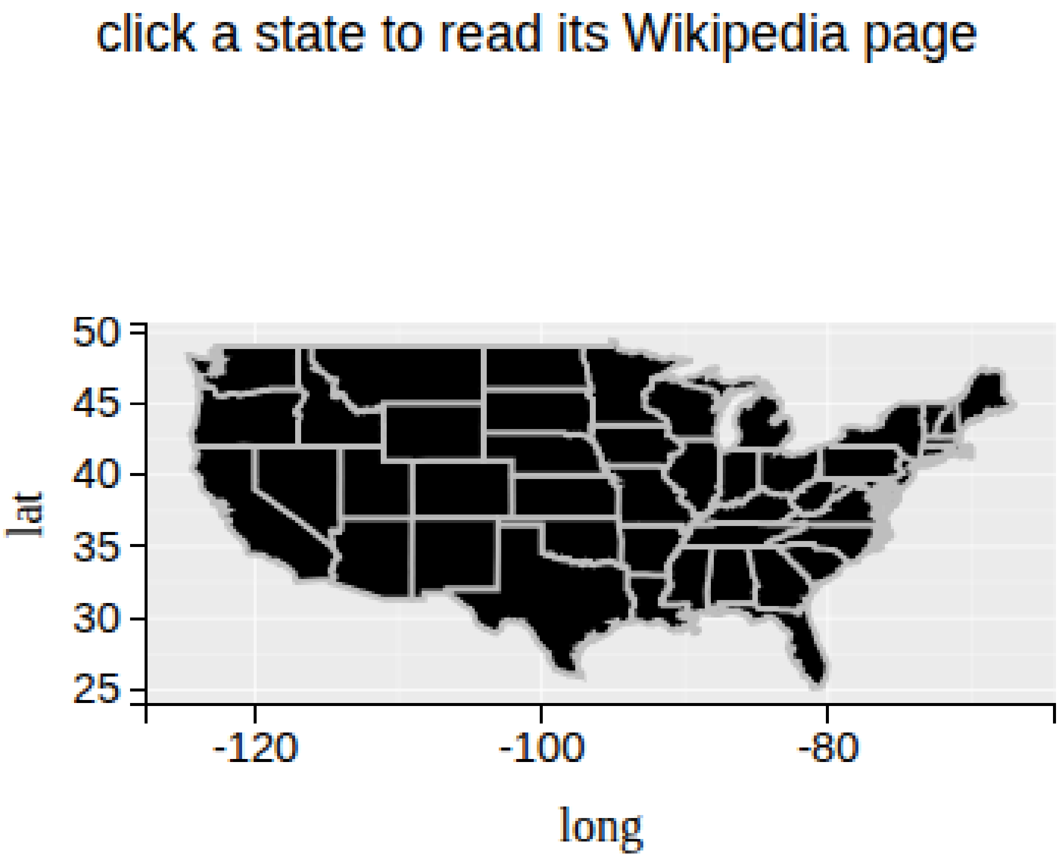
First we discuss the new aesthetics that we have already introduced in previous chapters.

Chapter 3 also introduced `aes(key)` to designate a variable to use for smooth transitions that are interpretable.

6.1.2 Hyperlinks using `aes(href)`

The code below uses `animint2` to draw a map of the United States.

```
library(animint2)
USpolygons <- map_data("state")
animint(
  map=ggplot()+
    ggtitle("click a state to read its Wikipedia page")+
    coord_equal()+
    geom_polygon(aes(
      x=long, y=lat, group=group,
      href=paste0("http://en.wikipedia.org/wiki/", region)),
      data=USpolygons, fill="black", colour="grey"))
```

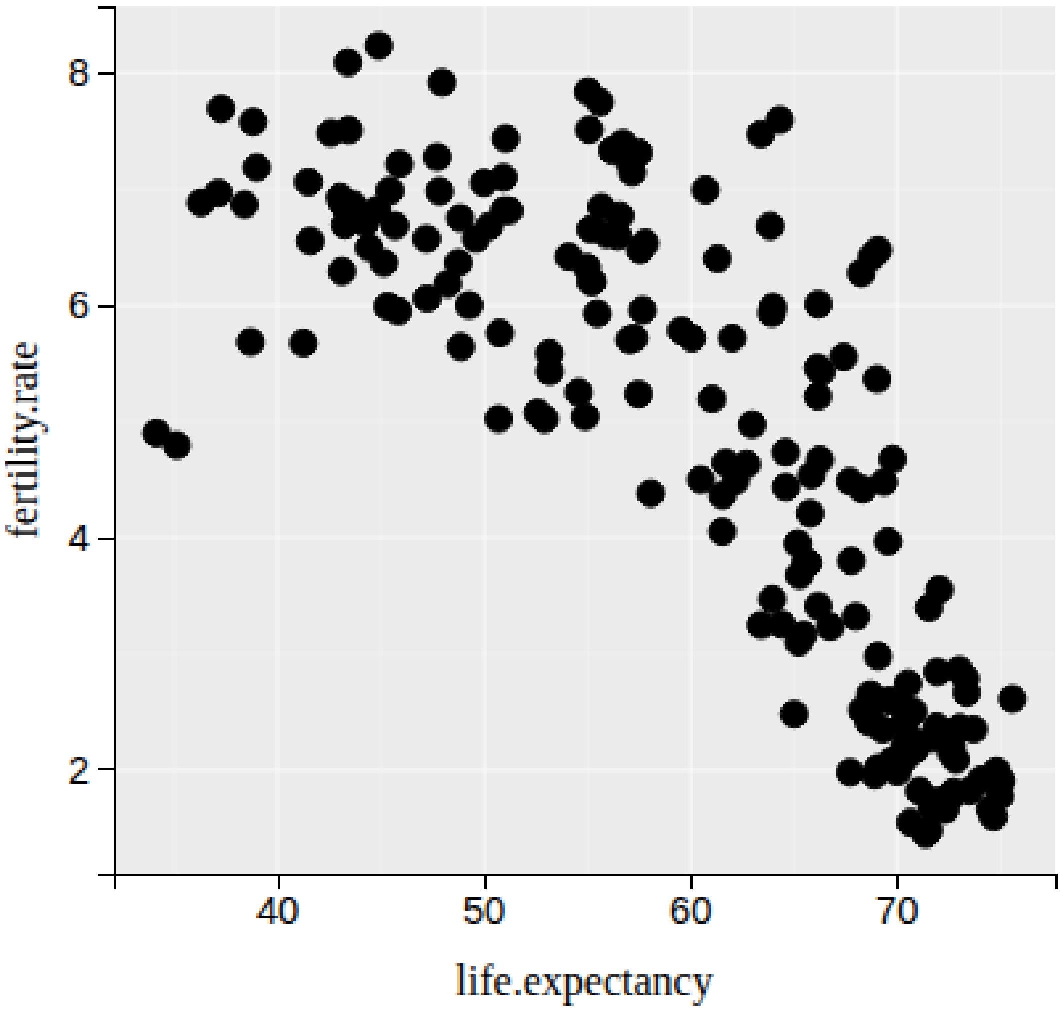


Try clicking a state in the data viz above. You should see the corresponding wikipedia page open in a new tab.

6.1.3 Tooltips using `aes(tooltip)`

Tooltips are little windows of text information that appear when you hover the cursor over something on the screen. In `animint2` you can use `aes(tooltip)` to designate the observation-specific message that appears. For example we use it to display the population and country name in the scatterplot of the World Bank data below.

```
data(WorldBank)
WorldBank1975 <- subset(WorldBank, year == 1975)
animint(
  scatter=ggplot()+
    geom_point(aes(
      x=life.expectancy, y=fertility.rate,
      tooltip=paste(country, "population =", population)),
      size=5,
        data=WorldBank1975))
```



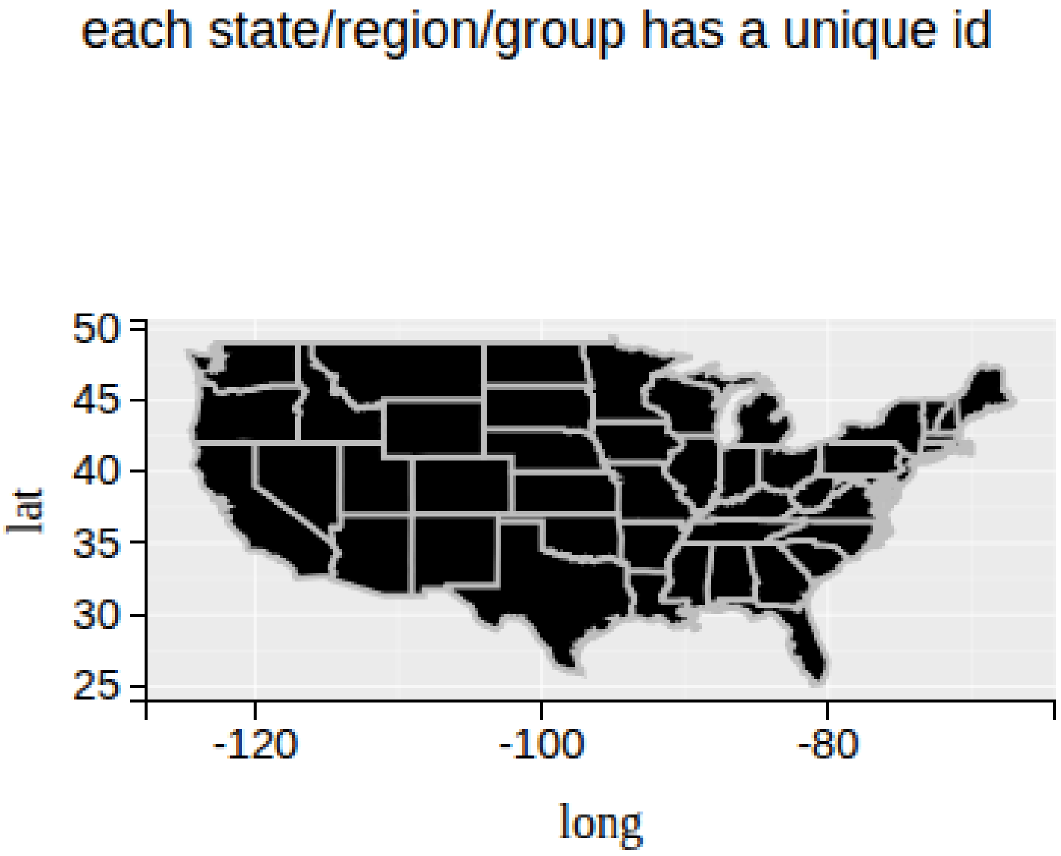
Try hovering the cursor over one of the data points. You should see a small box appear with the country name and population for that data point.

Note that a tooltip of the form "variable value" is specified by default for each geom with `aes(clickSelects)`. For example a geom with `aes(clickSelects=year)` shows the default tooltip "year 1984" for an observation with year 1984. You can change this default by explicitly specifying `aes(tooltip)`.

6.1.4 HTML id attribute using `aes(id)`

Since everything plotted by `animint2` is rendered as an SVG element in a web page, you may want to specify a HTML id attribute using `aes(id)` as below.

```
animint(
  map=ggplot()+
    ggtitle("each state/region/group has a unique id")+
    coord_equal()+
    geom_polygon(aes(
      x=long, y=lat, group=group,
      id=gsub(" ", "_", paste(region, group))),
      data=USpolygons, fill="black", colour="grey"))
```



Note how `gsub` is used to convert spaces to underscores, since a well-defined id must not include spaces. Note also that `paste` is used to add a group number, since there may be more than one polygon per state/region, and each id must be unique on a web page. The `animint2` developers use this feature for testing the animint JavaScript renderer code.

6.1.5 Data-driven selector names using named clickSelects and showSelected

Chapter 3 introduced `showSelected` for designating a geom which shows only the selected subset of its data.

Chapter 4 introduced `clickSelects` to designate a geom which can be clicked to change a selection variable.

Usually selector names are defined in `showSelected` or `clickSelects`. For example, `showSelected=c("year", "country")` means to create two selection variables (named `year` and `country`). However, that method becomes inconvenient if you have many selectors in your data viz. To illustrate we consider the following theoretical example (the code in this section is not directly executable). Say you want to use 20 different selector variable names, `selector1value` … `selector20value`. The usual way to define your data viz would be

```
viz <- list(
  points=ggplot()+
    geom_point(clickSelects="selector1value", data=data1)+
    ...
    geom_point(clickSelects="selector20value", data=data20)
)
```

However that method is bad since it violates the DRY principle (Don't Repeat Yourself). Another way to do that would be to use a for loop:

```
viz <- list(points=ggplot())
for(selector.name in paste0("selector", 1:20, "value")){
  data.for.selector <- all.data.list[[selector.name]]
  viz$points <- viz$points +
    geom_point(clickSelects=selector.name, data=data.for.selector)
}
```

That method is bad since it is slow to construct `viz`, and the compiled viz potentially takes up a lot of disk space since there is at least one TSV file created for each `geom_point`. The preferable method is to use a named character vector for `clickSelects`. The names should be used to indicate the column that contains the selector variable name. For example:

```
viz <- list(
  points=ggplot()+
    geom_point(
      clickSelects=c(selector.name="selector.value"),
      data=all.data)
)
```

The `animint2dir()` compiler looks through the data.frame `all.data` and create selectors for each of the distinct values of `all.data$selector.name`. Clicking one of the data points updates the corresponding selector with the value indicated in `all.data$selector.value`.

You can similarly use one geom with a named `showSelected` instead of a bunch of different geoms with `showSelected`.

This feature is useful not only to avoid repetition in the definition of the data viz, but also because they are more computationally efficient. For a detailed example with timings and disk space measurements, see Chapter 14.

6.2 New geoms

There are several new geoms in `animint2`, with respect to the geoms that are provided in `ggplot2`.

6.2.1 Tall and wide rectangles for selection

Tall and wide rectangles are special cases of `geom_rect` which by default cover the entire Y/X range.

- `geom_tallrect()` covers the entire Y range, and was first introduced in Chapter 4.
- `geom_widerect()` covers the entire X range, and is discussed in Chapter 8.

Both are useful in combination with `clickSelects`, so we also provide helper functions for this common use case:

- `make_tallrect(data, "x_variable")` creates a rectangle for each unique value of the column with name `x_variable` in `data`.
- `make_widerect(data, "y_variable")` creates a rectangle for each unique value of the column with name `y_variable` in `data`.

Both of these are capable of showing smooth transitions for the selected rectangle, as in the WorldBank data visualization with two time series facets.

- a expanded data set with `N^2` rows is computed for `N` unique values to select.
- there are `N` sets of rows, each with a different `showSelected` value.
- in each `showSelected` subset, the selected value has a special `aes(key)` value.
- so the result is a smooth transition: when we change the selection, we see the selected/special row moving to its new position (and all of the others are invisible).

For more discussion about this technique, see Chapter 11.

6.2.2 Aligned labels for avoiding overlapping text

When using multiple selection, it is often useful to display text labels to indicate the items in the selection set. When labels are aligned (with the same X or Y value), then `geom_label_aligned()` can be used to adjust positions, to avoid overlapping labels. For example, the WorldBank data visualization shows a text label before the first year of each selected country's time series. For more discussion about this technique, see Chapter 8.

6.3 Geom options

In `animint2`, there are several options for customization at the geom level: `chunk_vars` is used to specify how to split data sets for storage on disk, and `*_off` parameters are used to specify how a clickSelects geom should be displayed when it is not selected. Additionally, `help` and `title` may be specified, to add information to the guided tour.

6.3.1 The `chunk_vars` geom-specific compilation option

The `chunk_vars` option defines the selection variables that are used to split the data set into separate chunks (TSV files) to download. There is one TSV file created for each combination of values of the `chunk_vars` variables. More selection variables specified in `chunk_vars` means to split the data set into more TSV files, each of a smaller size.

The `chunk_vars` option should be specified as an argument to a `geom_*` function, and its value should be a character vector of selection variable names. When `chunk_vars=character(0)`, a character vector of length zero, all of the data is stored in a single TSV file. When `chunk_vars` is set to all of the `showSelected` variable names, then a TSV file is created for each combination of values of those variables.

In general the `animint2dir()` compiler chooses a sensible default for `chunk_vars`, but you may want to specify `chunk_vars` if the data viz is loading slowly, or taking up too much space on disk. If the data viz is loading slowly, you should add selection variables to `chunk_vars` to reduce the size of the first TSV file to download. If the data viz takes up too much space on disk, you should remove selection variables from `chunk_vars` to decrease the number of TSV files. Lots of small TSV files can take more disk space than a single TSV file because some filesystems store a constant amount of metadata for every file.

To illustrate the usage of `chunk_vars`, consider the following visualization of the `breakpoints` data set.

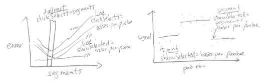
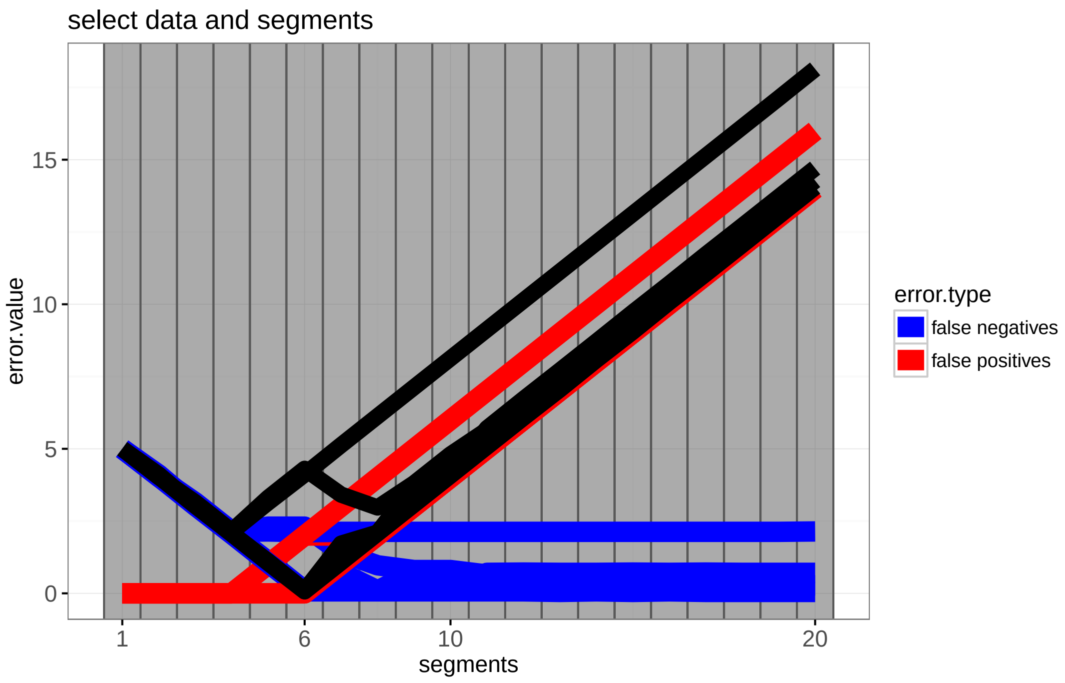


Figure 6.1: Breakpoints data viz

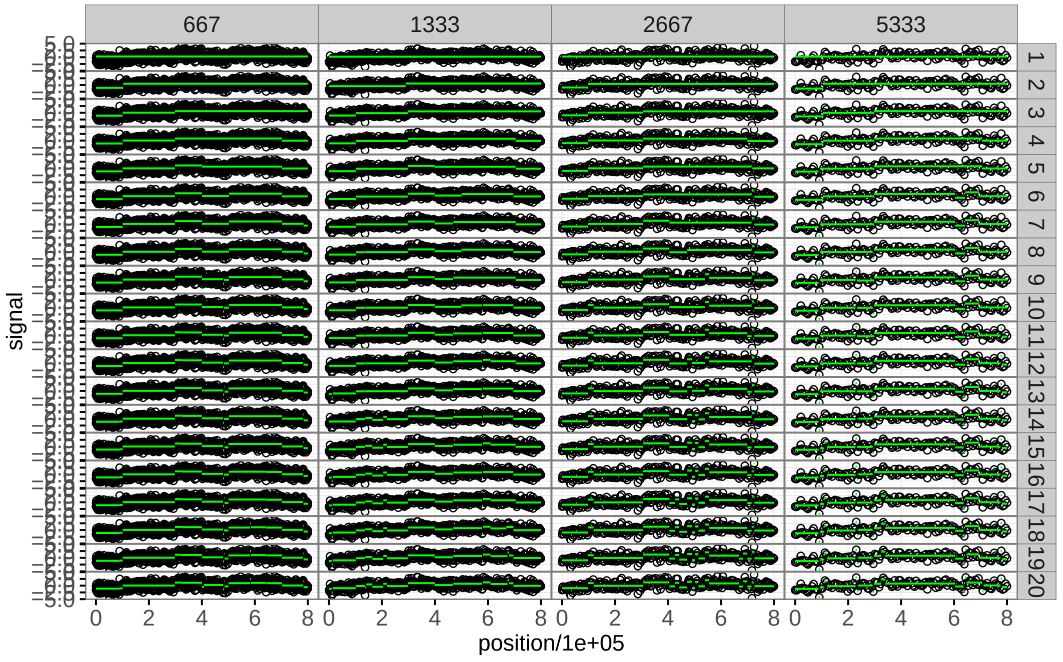
The sketch above consists of two plots. We begin by creating the plot of error curves on the left.

```r
data(breakpoints)
only.error <- subset(breakpoints$error, type=="E")
only.segments <- subset(only.error,bases.per.probe==bases.per.probe[1])
library(data.table)
fp.fn.names <- rbind(
  data.table(error.type="false positives", type="FP"),
  data.table(error.type="false negatives", type=c("I", "FN")))
error.dt <- data.table(breakpoints$error)
error.type.dt <- error.dt[fp.fn.names, on=list(type)]
fp.fn.dt <- error.type.dt[, list(
  error.value=sum(error)
), by=.(error.type, segments, bases.per.probe)]
errorPlot <- ggplot()+
  ggtitle("select data and segments")+
  theme_bw()+
  geom_tallrect(aes(
    xmin=segments-0.5, xmax=segments+0.5),
    clickSelects="segments",
    data=only.segments,
    alpha=1/2)+
  geom_line(aes(
    segments, error.value, color=error.type,
    group=paste(bases.per.probe, error.type)),
    showSelected="bases.per.probe",
    data=fp.fn.dt,
    size=5)+
  scale_color_manual(values=c(
    "false positives"="red", "false negatives"="blue"))+
  geom_line(aes(
    segments, error, group=bases.per.probe),
    clickSelects="bases.per.probe",
    data=only.error,
    size=4)+
  scale_x_continuous(breaks=c(1, 6, 10, 20))
errorPlot
```



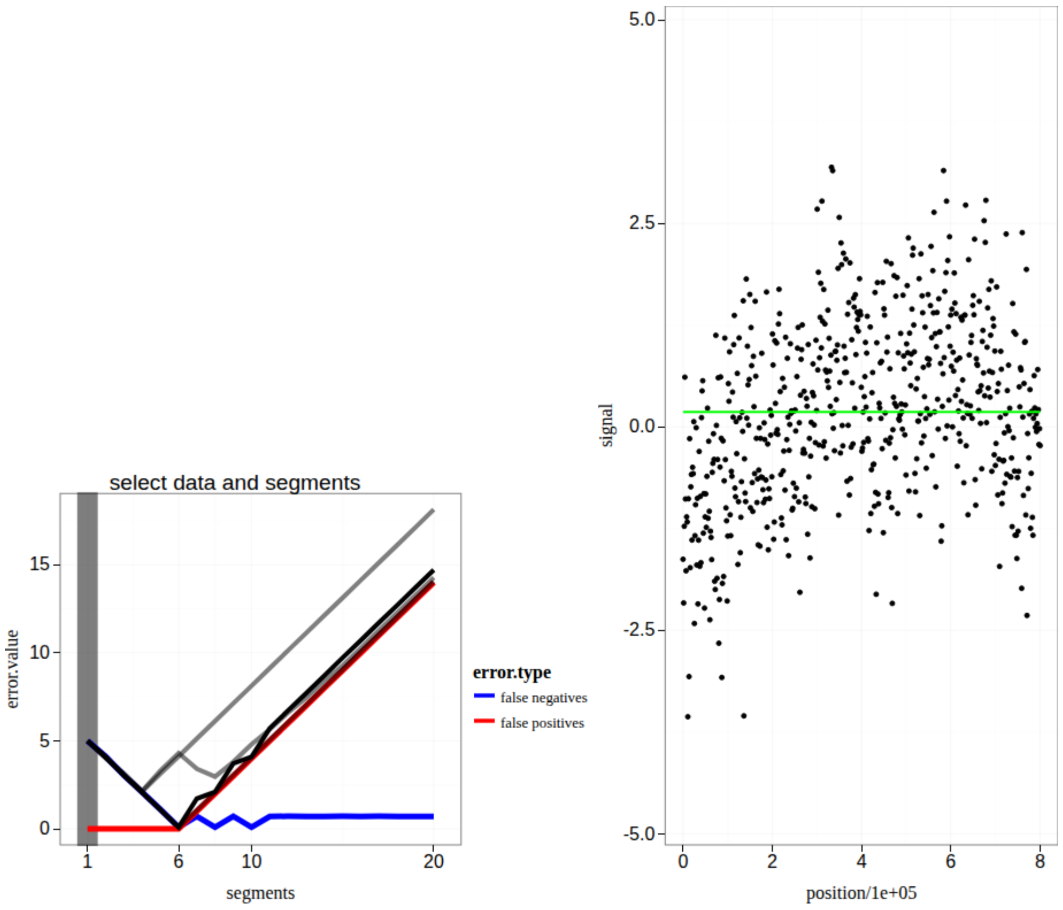
The plot above includes a `geom_tallrect` with `clickSelects=segments` and a `geom_line` with `clickSelects=bases.per.probe`. It will be used to select the data and model in the plot below.

```
signalPlot <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  theme_animint(height=800)+
  geom_point(aes(
    position/1e5, signal),
    showSelected="bases.per.probe",
    shape=1,
    data=breakpoints$signals)+
  geom_segment(aes(
    first.base/1e5, mean, xend=last.base/1e5, yend=mean),
    showSelected=c("segments", "bases.per.probe"),
    color="green",
    data=breakpoints$segments)
signalPlot+facet_grid(segments ~ bases.per.probe)
```



The non-interactive plot above has 80 facets, one for each combination of the two `showSelected` variables, `bases.per.probe` and `segments`. Below we make an interactive version in which only one of these facets will be shown.

```
(viz.chunk.vars <- animint(
  errorPlot,
  signal=signalPlot+
    geom_vline(aes(
      xintercept=base/1e5),
      showSelected=c("segments", "bases.per.probe"),
      color="green",
      chunk_vars=character(),
      linetype="dashed",
      data=breakpoints$breaks)))
```



Click the "Show download status table" button, and you should see counts of chunks (TSV files). Note that `geom6_vline_signal` has only 1 chunk, since `chunk_vars=character()` is specified for the `geom_vline` in the R code above. If another value of `chunk_vars` was specified, it would create a different number of TSV files, but the appearance of the data viz should be the same.

Below we use the `du` command line program to determine the disk usage of the data viz for different choices of `chunk_vars`.

```r
tsvSizes <- function(segment.chunk.vars){
  viz <- list(
    error=errorPlot,
    signal=signalPlot+
      geom_vline(aes(
        xintercept=base/1e5),
        showSelected=c("segments", "bases.per.probe"),
        color="green",
        chunk_vars=segment.chunk.vars,
        linetype="dashed",
        data=breakpoints$breaks)
  )
```

```r
    info <- animint2dir(viz, open.browser=FALSE)
    cmd <- paste("du -ks", info$out.dir)
    kb.dt <- fread(cmd=cmd)
    setnames(kb.dt, c("kb", "dir"))
    tsv.vec <- Sys.glob(paste0(info$out.dir, "/*.tsv"))
    is.geom6 <- grepl("geom6", tsv.vec)
    data.frame(
      kb=kb.dt$kb,
      geom6.tsv=sum(is.geom6),
      other.tsv=sum(!is.geom6))
  }
  chunk_vars_list <- list(
    neither=c(),
    bases.per.probe=c("bases.per.probe"),
    segments=c("segments"),
    both=c("segments", "bases.per.probe"))
  sizes.list <- lapply(chunk_vars_list, tsvSizes)
  (sizes <- do.call(rbind, sizes.list))
```

```
                  kb geom6.tsv other.tsv
neither          840         1        12
bases.per.probe  844         5        12
segments         900        19        12
both            1128        76        12
```

The table above includes counts of kilobytes for the data viz, along with counts of TSV files for `geom6_vline_signal` and the other geoms. Note how the choice of `chunk_vars` affects the number of TSV files and the disk space usage. Since `chunk_vars` was only specified for `geom6_vline_signal`, the number of TSV files for the other geoms does not change. When both `segments` and `bases.per.probe` are specified for `chunk_vars`, there are 76 TSV files for `geom6_vline_signal`, and the data viz takes 1128 kilobytes. In contrast, `chunk_vars=character()` produces only one TSV file for `geom6_vline_signal`, and the data viz uses 840 kilobytes.

In conclusion, the geom-specific `chunk_vars` option defines the number of TSV files created for each geom. When deciding the value of `chunk_vars`, you should consider both disk usage and loading time. A few large files take up less disk space but are slower to download than many small files.

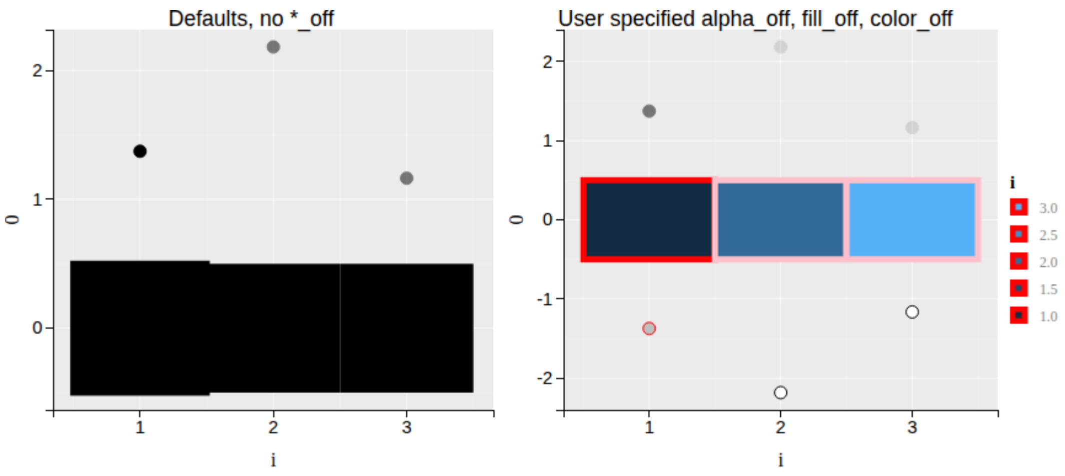
6.3.2 Specifying how selection state is displayed

Different geoms in `animint2` have sensible defaults for displaying selection state. In particular,

- when there is a `rect` or `tile` with `clickSelects`, we use black color/border to show items which are selected, and transparent for items which are not selected.
- for any other geom with `clickSelects`, we use full opacity `alpha` to show items which are selected, and `alpha-0.5` opacity to show items which are not selected.

The defaults explained above are illustrated in the first plot below. Those defaults may be customized by using the `alpha_off`, `fill_off`, and `color_off` geom parameters as in the code below,

```
N <- 3
set.seed(1)
demo_df <- data.frame(i=1:N, num=rnorm(N,2))
animint(
  defaults=ggplot()+
    ggtitle("Defaults, no *_off")+
    geom_tile(aes(
      i, 0),
      size=5,
      clickSelects="i",
      data=demo_df)+
    geom_point(aes(
      i, num),
      size=5,
      clickSelects="i",
      data=demo_df),
  off=ggplot()+
    ggtitle("User specified alpha_off, fill_off, color_off")+
    geom_tile(aes(
      i, 0, fill=i),
      clickSelects="i",
      color="red",
      color_off="pink",
      size=5,
      data=demo_df)+
    geom_point(aes(
      i, num),
      size=5,
      alpha=0.5,
      alpha_off=0.1,
      clickSelects="i",
      data=demo_df)+
    geom_point(aes(
      i, -num),
      size=5,
      alpha=1,
      alpha_off=1,
      color="red",
      color_off="black",
      fill="grey",
      fill_off="white",
      clickSelects="i",
      data=demo_df))
```

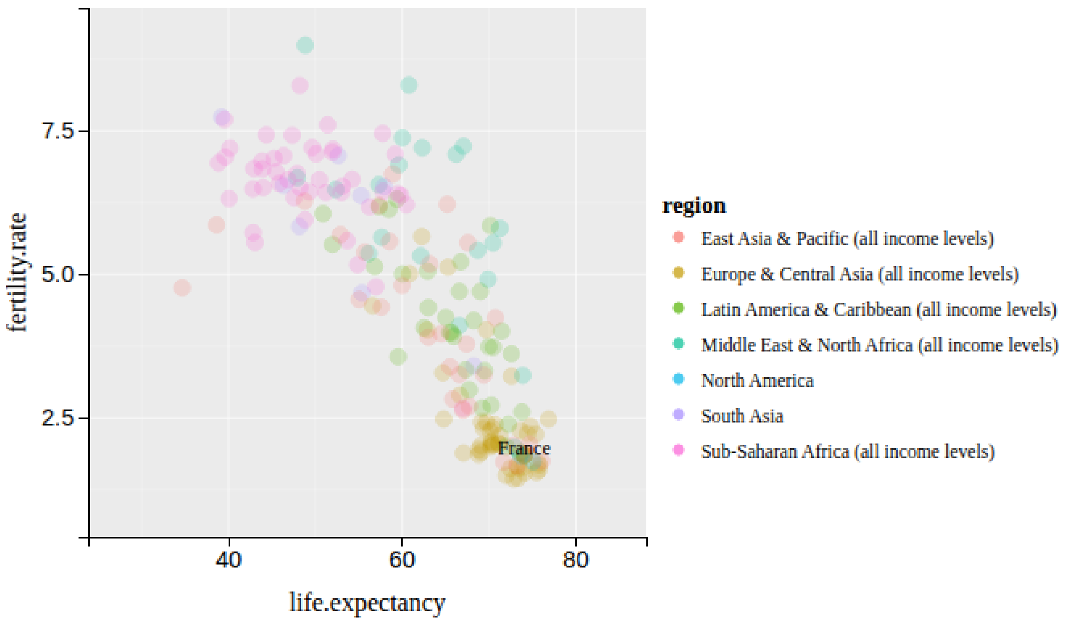


Note that when using any one of these visual properties in the `aes` mapping, it should not be specified as a geom parameter. For example in the tile above, we used `aes(fill)`, so `fill` and `fill_off` should not be specified as parameters for that geom (in order to make it clear that fill is used for displaying data values, not selection state).

6.3.3 Specifying guided tour text

Since Jan 2025, `animint2` supports a guided tour, which displays information about possible interactions with each geom. To customize what is displayed for each geom, you can specify the `help` and `title` parameters, as in the code below.

```r
animint(
  scatter=ggplot()+
    geom_point(aes(
      x=life.expectancy, y=fertility.rate, color=region),
      size=5,
      showSelected="year",
      clickSelects="country",
      help="One point drawn for each country in the selected year",
      alpha=0.7,
      data=WorldBank)+
    geom_text(aes(
      x=life.expectancy, y=fertility.rate, label=country),
      data=WorldBank,
      title="Selected country",
      showSelected=c("year","country")),
  first=list(
    country="France",
    year=1980))
```



In the code above, we specify `help` for `geom_point`, which controls the sub-text which is displayed for that geom, after clicking the “Start Tour” button at the bottom of the data visualization. After clicking the “Next” button, we can see the `title` that was specified in the code, shown at the top of the tour window, for the `geom_text`. This mechanism can be used to provide extra helpful information for the users of your data visualization, so they can more easily understand what is displayed, and what interactions are possible.

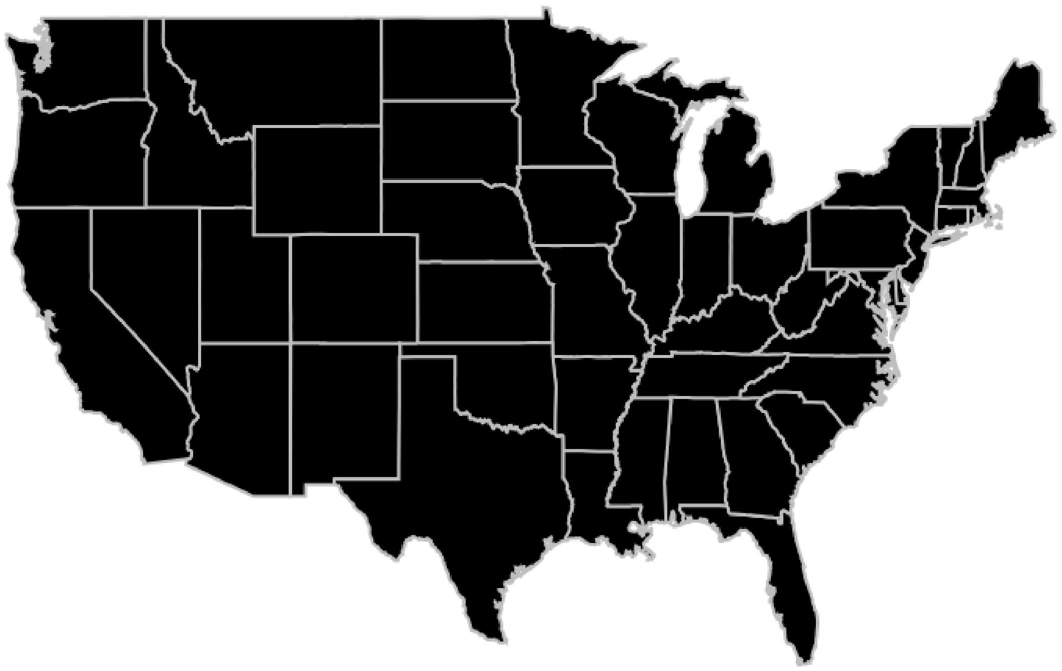
6.4 Plot-specific options

This section discusses options which are specific to one ggplot of a data viz. The `theme_animint` function is used to attach `animint2` options to ggplot objects.

6.4.1 Plot height and width

The `width` and `height` options are for specifying the dimensions (in pixels) of a ggplot rendered by `animint2`. For example, consider the following re-design of the plot of the United States:

```
animint(
  map=ggplot()+
    theme_animint(width=750, height=500)+
    theme(
      axis.line=element_blank(),
      axis.text=element_blank(),
      axis.ticks=element_blank(),
      axis.title=element_blank(),
      panel.border=element_blank(),
      panel.background=element_blank(),
      panel.grid.major=element_blank(),
      panel.grid.minor=element_blank())+
    geom_polygon(aes(
      x=long, y=lat, group=group),
      data=USpolygons, fill="black", colour="grey"))
```



Note that the plot above was rendered with a width of 750 pixels and a height of 500 pixels, due to the `theme_animint` options. If either of these options is not specified for any ggplot, then `animint2` uses a default of 400 pixels.

Also note that `theme` was used to specify several blank elements. This has the effect of removing the axes and background, and is generally useful for rendering maps.

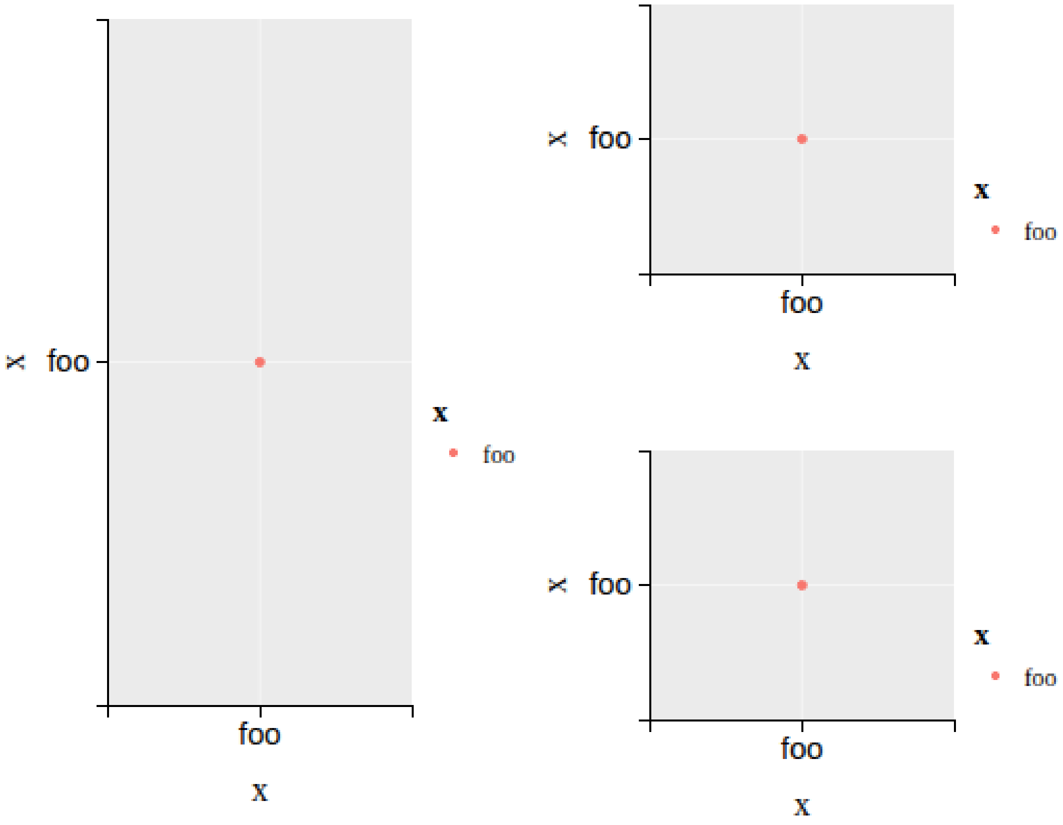
6.4.2 HTML table layout

Since summer 2025, `animint2` supports plot layout in an HTML table. A typical example is when you want to display three linked plots, with one plot to the right of two other plots, as in this visualization of cross-validation for change-point model selection. To specify the HTML table layout, we use three new arguments in `theme_animint()`:

- `rowspan=2` means the plot takes up two rows in the table (default 1).
- `colspan=2` means the plot takes up two columns in the table (default 1).
- `last_in_row=TRUE` means the plot is last in the current row, so the next plot will appear on the next row.
- if none of these are specified in any plots, then we use the classic layout (no HTML table, each plot appears one after the other, and is wrapped to the next row if there is not enough horizontal space).

For example, consider the demo below:

```
df <- data.frame(x="foo")
animint(
  left=ggplot()+
    theme_animint(width=200, rowspan=2)+
    geom_point(aes(x, x, color=x), data=df),
  topRight=ggplot()+
    theme_animint(width=200, height=200, last_in_row=TRUE)+
    geom_point(aes(x, x, color=x), data=df),
  bottomRight=ggplot()+
    theme_animint(width=200, height=200)+
    geom_point(aes(x, x, color=x), data=df))
```

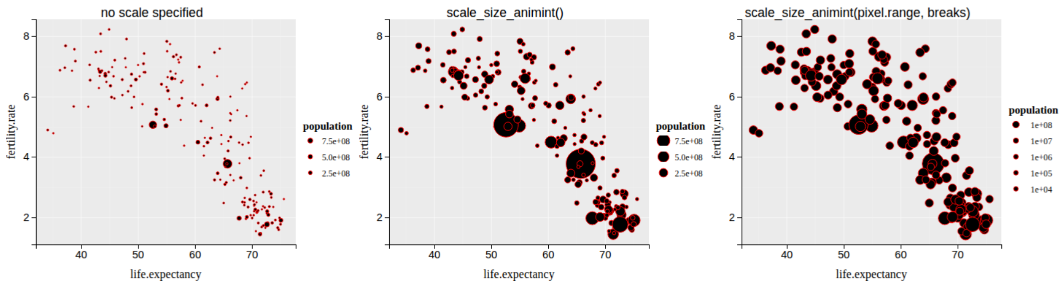


The visualization above shows one plot on the left, with two plots on the right, as expected.

6.4.3 Size scale in pixels

The `scale_size_animint()` scale should be used in all ggplots where you specify `aes(size)`. To see why, consider the following examples.

```
scatter1975 <- ggplot()+
  geom_point(
    aes(x=life.expectancy, y=fertility.rate, size=population),
    WorldBank1975,
    shape=21,
    color="red",
    fill="black")
(viz.scale.size <- animint(
  ggplotDefault=scatter1975+
    ggtitle("no scale specified"),
  animintDefault=scatter1975+
    ggtitle("scale_size_animint()")+
    scale_size_animint(),
  animintOptions=scatter1975+
    ggtitle("scale_size_animint(pixel.range, breaks)")+
    scale_size_animint(pixel.range=c(5, 15), breaks=10^(10:1))))
```

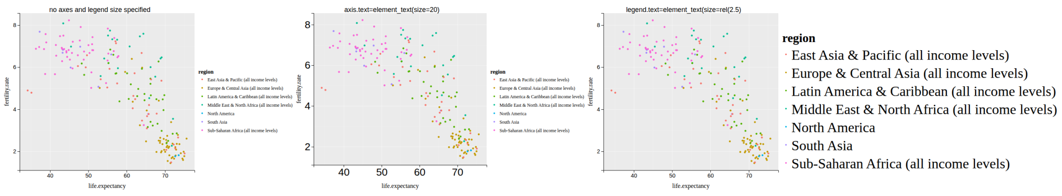


The first ggplot above has no scale specified, so it uses the default ggplot2 scale, which has two problems. The first problem is that it seems that all countries have about the same size except the two really big countries. That problem can be fixed by simply adding `scale_size_animint()` to the ggplot, which results in the second plot above. However, a second problem is that the legend entries do not show the full range of the data. That problem is fixed in the third plot above, by manually specifying the `breaks` to use for legend entries. Note that the `pixel.range` argument can also be used to specify the radius of the largest and smallest circles.

6.4.4 Axes and legend text size

The syntax of defining axes and legend text size(in pixels) is almost the same as `ggplot2`. Inside `theme`, you can use numbers directly to change the font size, or you can use `rel()` to define the relative size.

```
  scatter1975 <- ggplot()+
    geom_point(aes(
      x=life.expectancy, y=fertility.rate, color=region),
      data=WorldBank1975)
  (viz.text.size <- animint(
    animintDefault=scatter1975+
      theme_animint(width=500, height=500)+
      ggtitle("no axes and legend size specified"),
    animintAxesOptions=scatter1975+
      theme_animint(width=500, height=500)+
      theme(axis.text=element_text(size=20))+
      ggtitle("axis.text=element_text(size=20)"),
    animintLegendOptions=scatter1975+
      theme_animint(width=500, height=500)+
      theme(
        legend.title=element_text(size=24),
        legend.text=element_text(size=rel(2.5)))+
      ggtitle("legend.text=element_text(size=rel(2.5)"))
```



This allows you to change the font size while changing the size of the plot to make it look more coherent.

Note that the default font size in `animint2` is 11px for the axes and 16px for the legend.

6.5 Global data viz options

Global data viz options are any named elements of the `viz` list that are not ggplots.

6.5.1 Review of previously introduced global options

Chapter 3 introduced the `duration` option for specifying the duration of smooth transitions.

Chapter 3 introduced the `time` option for specifying a selection variable which is automatically updated (animation).

Chapter 4 introduced the `first` option for specifying the selection when the data viz is first rendered.

Chapter 4 introduced the `selector.types` option for specifying multiple selection variables.

6.5.2 Web page title with the title option

The `title` option should be a character string, and will be used to set the `<title>` element of the web page. It does not make sense to use the `title` option in an Rmd document such as this page. A title can and should be used with `animint2dir()`, as in the code below.

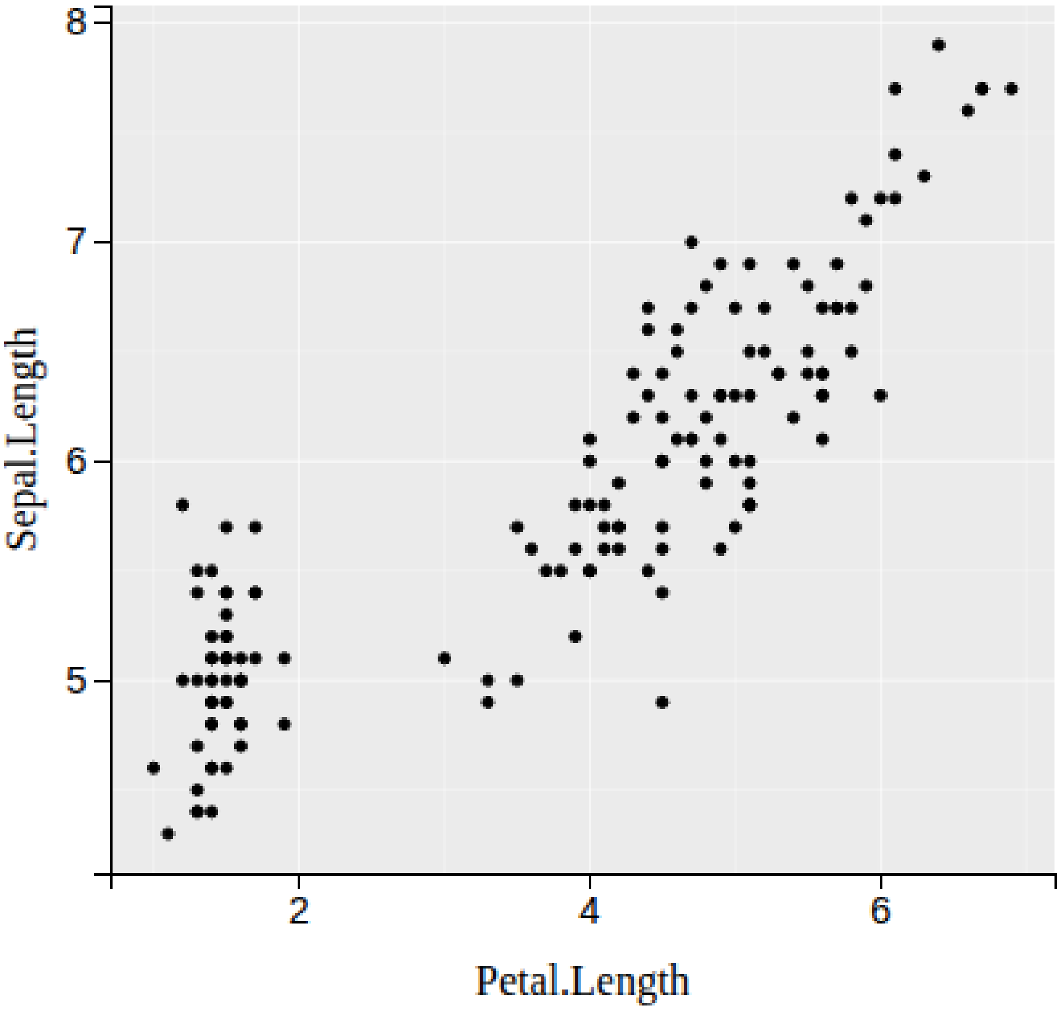
```
viz.title <- viz.scale.size
viz.title$title <- "Several size scales"
animint2dir(viz.title, "Ch06-title")
```

Note that `viz.scale.size` already has three ggplots, each with a `ggtitle`. Adding the global `title` option has the effect of defining a title for the web page.

6.5.3 Link R code with source option

The `source` option should be a character string: a link to the R source code which was used to create the animint.

```
animint(
  demo=ggplot()+
    geom_point(aes(
      Petal.Length, Sepal.Length),
      data=iris),
  source=
    "https://github.com/tdhock/animint-book/edit/master/Ch06-other.Rmd")
```



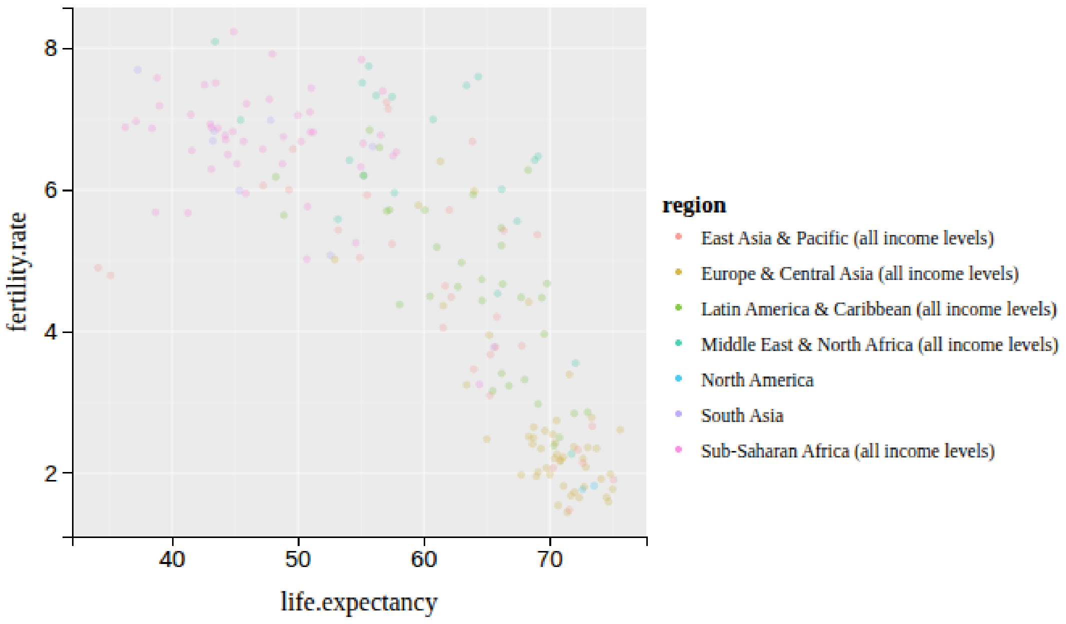
Note above how there is a source link at the bottom of the data viz.

Chapter 5 introduced the `animint2pages` function, which is used to publish an animint to GitHub Pages. It requires that the animint defines the `title` and `source` options, because that meta-data is required for organizing the animint in a gallery.

6.5.4 Link a video

The `video` option should be a character string: a link to a video which shows typical interactions with the animint. This mechanism can be used to help the users of your data visualization understand what is displayed, and what interactions they can use.

```
animint(
  video="https://vimeo.com/1050117030",
  scatter=ggplot()+
    geom_point(aes(
      x=life.expectancy, y=fertility.rate, color=region),
      clickSelects="country",
      alpha=0.7,
      data=WorldBank1975))
```



In the data visualization above, notice the "video" link which appears in the bottom right. Clicking that link leads to a video that was recorded to explain a more complex data visualization based on the World Bank data. The idea is that you can record a video for each of your animints, and then include a link to the video using this mechanism, so your users can more easily understand what is displayed, and what interactions are possible.

6.5.5 Show or hide selection menus with the selectize option

The selectize option should be a named list of logical values. Names should be selector variables, and values should indicate whether or not you would like to render a selection menu via selectize.js. By default, `animint2` will render a selection menu for every selection variable, with two exceptions:

- data-driven selection variables that are defined using named clickSelects/showSelected variables.
- selection variables that have a lot of values (they are slow to render).

These defaults should work well for the vast majority of animints. For those who are interested to see an example of how the `selectize` option works, please see the PredictedPeaks test in the `animint2` source code.

6.6 Chapter summary and exercises

This chapter explained several options for customizing animints at the observation, geom, plot, and global level.

Exercises:

- Create other versions of `viz.chunk.vars` with different values of `chunk_vars` for the `geom_point` and `geom_segment`. How does the choice of `chunk_vars` affect the appearance of the visualization? The disk space? The loading time?

Next, Chapter 7 explains the limitations of the current implementation of `animint2`.

7 Limitations

This chapter explains several known limitations of `animint2` for some interactive data visualization tasks. It also explains some workarounds that you can use in these situations. After reading this chapter, you will understand how to

- Use a normalized variable when different `showSelected` subsets have very different values of a variable you want to display.
- Compute statistics for each `showSelected` subset, rather than relying on the `stat_*` functions in `ggplot2`.
- Add data one at a time to a multiple selection variable set, rather than using a rectangular selection brush.
- Avoid using `vjust` and labels with multiple lines in `geom_text`.
- Order the plots on the page.
- Avoid using some unsupported `theme` options.
- Use facets with multiple variables per axis.
- Use the `shiny` web server package to interactively change the aesthetic mapping of an animint, or to perform computations based on selected values.

If you have an idea for improving `animint2` so that it overcomes one of these limitations, the `animint2` developers would be more than happy to review your Pull Request.

7.1 Use normalized variables to work with fixed scales

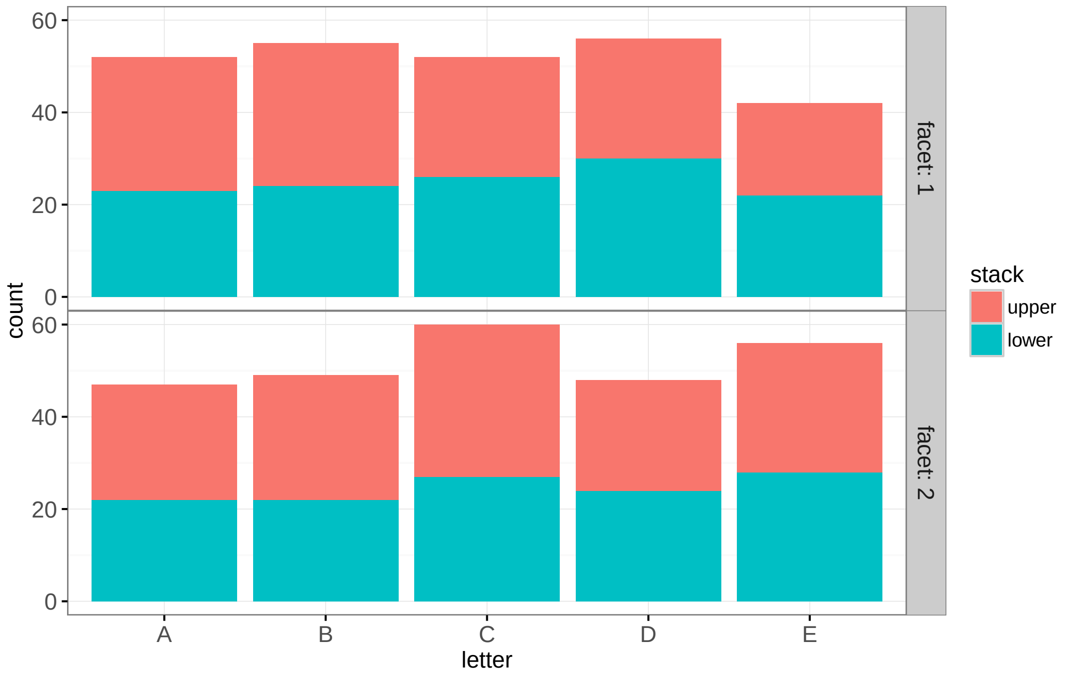
We implement axes and legends in the same way `ggplot2` does, by computing them once when the plot is first rendered. As a consequence, the axes and legends in each animint plot are not interactive. For most animated data visualizations, fixed axes make it easy to understand how the data changes along with the `time` variable.

There are some situations where it would be useful to have axes that interactively update. One example is when different showSelected subsets have very different values for variables that are shown with an axis or legend. In this case it would be useful to have interactive axes that update and change along with the data. We have experimental support for this, see the update axes test for details.

7.2 Compute statistics for each showSeleted subset

Animints do not support `ggplot2` statistics with `showSelected`. For example, consider the facetted ggplot below.

```r
set.seed(1)
library(data.table)
random.counts <- data.table(
  letter = c(replicate(4, LETTERS[1:5])),
  count = c(replicate(4, rbinom(5, 50, 0.5))),
  stack = rep(rep(factor(
    c("lower","upper"), c("upper","lower")
  ), each = 5), 2),
  facet = rep(1:2, each = 10))
library(animint2)
ggstat <- ggplot() +
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  geom_bar(aes(
    letter, count,
    key=paste(letter, stack),
    fill = stack),
    showSelected="facet",
    data = random.counts,
    stat = "identity",
    position="stack"
  )
ggstat+facet_grid(facet ~ ., labeller=label_both)
```

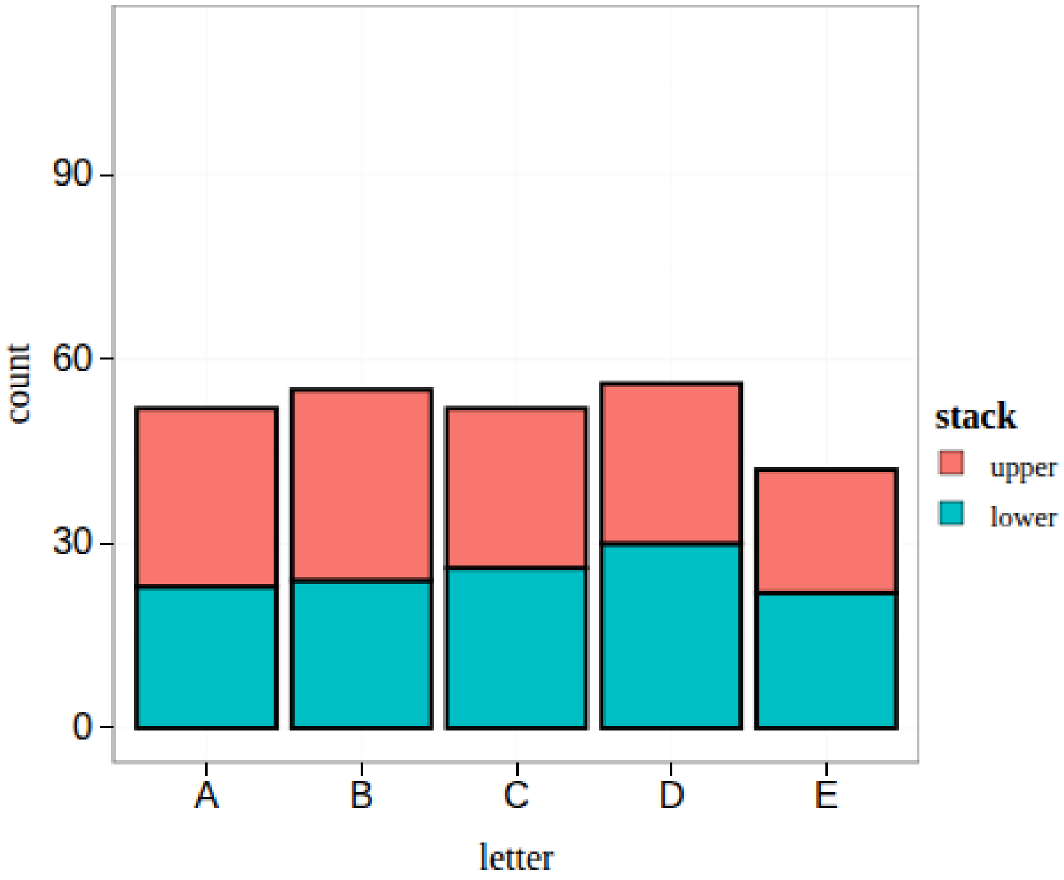


Using `showSelected` instead of facets does not result in what you may expect.

```
animint(
  plot = ggstat,
  duration = list(facet = 1000))
```

```
mapping: x = letter
y = count
key = paste(letter, stack)
fill = stack
showSelected1 = facet
showSelected2 = stack
geom_bar: width = NULL
na.rm = FALSE
stat_identity: na.rm = FALSE
position_stack
```

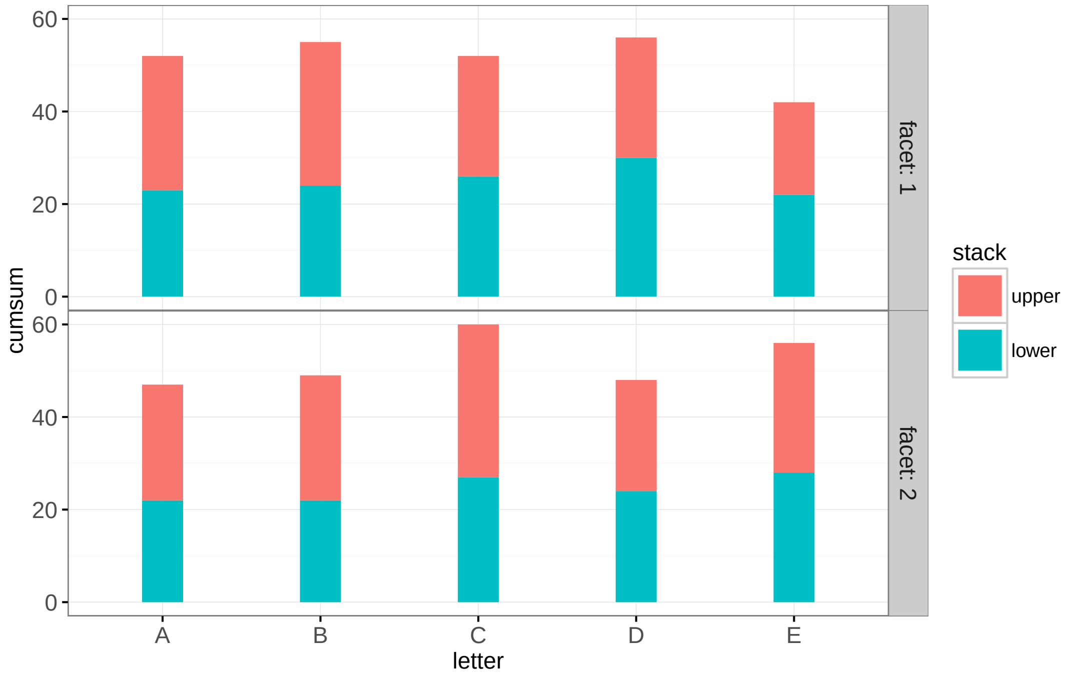
```
Warning in f(...): showSelected only works with position=identity, problem:
geom1_bar_plot
```



Try changing the selection from facet 1 to 2. The data viz above shows bars moving from top to bottom, indicating that `position=stack` has only been computed globally (not for each `showSelected` subset).

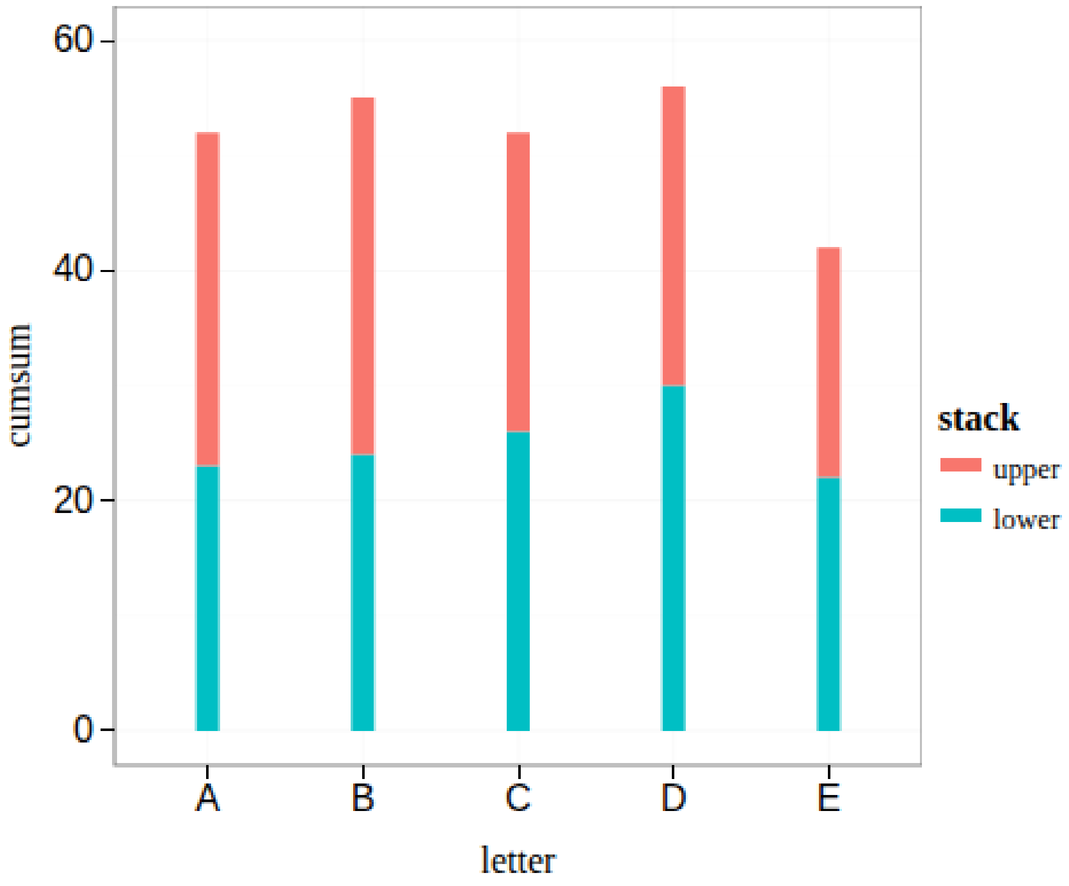
A workaround is to compute what you want to display, and use `position=identity`, which is demonstrated in the code below.

```
random.cumsums <- random.counts[, data.table(
  cumsum=cumsum(count),
  count,
  stack
), by=.(letter, facet)]
ggidentity <- ggplot() +
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  geom_segment(aes(
    x=letter, xend=letter,
    y=cumsum, yend=cumsum-count,
    key=paste(letter, stack),
    color=stack),
    showSelected="facet",
    data = random.cumsums,
    size=10,
    stat = "identity",
    position="identity")
ggidentity+facet_grid(facet ~ ., labeller=label_both)
```



Note how we used `geom_segment` instead of `geom_bar`, but their appearance is similar.

```
animint(
  plot = ggidentity,
  duration = list(facet = 1000))
```



Try changing the selection from facet 1 to 2. The data viz above shows bars adjusting their size, indicating that using `position=identity` is a valid work-around.

7.3 Add values to a multiple selection set one at a time

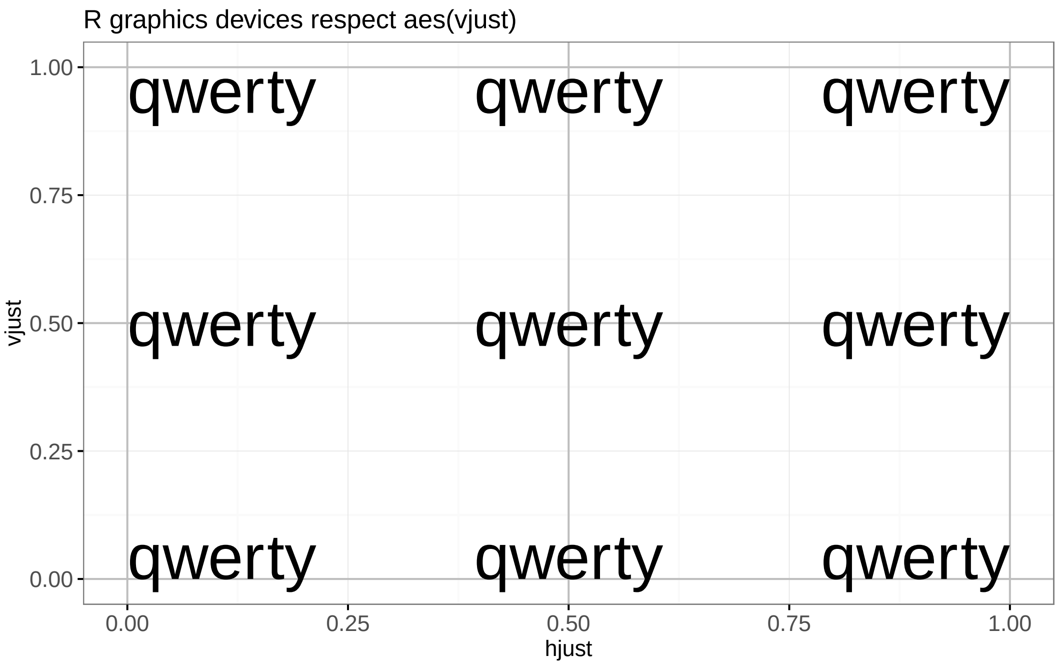
Animint does not support a rectangular brush or lasso for interactively defining a set of selected values. Instead, animint supports multiple selection by adding values one by one to the multiple selection set. Use the selector.types option to declare a multiple selection variable.

7.4 Adjust y instead of using vjust with `geom_text`

For horizontal text alignment, animint supports using `hjust` in R with the most common values: 0 for left alignment, 0.5 for middle alignment, and 1 for right alignment. Animint translates these three `hjust` values to the `text-anchor` property in the rendered data viz.

However, animint `geom_text` does not support vertical text alignment using `vjust` in R, because there is no property that can be used for vertical text alignment in SVG.

```
line.df <- data.frame(just=c(0, 0.5, 1))
text.df <- expand.grid(
  vjust=line.df$just,
  hjust=line.df$just)
gg.lines <- ggplot()+
  theme_bw()+
  geom_vline(aes(xintercept=just), data=line.df, color="grey")+
  geom_hline(aes(yintercept=just), data=line.df, color="grey")
gg.vjust <- gg.lines+
  geom_text(aes(
    hjust, vjust, label="qwerty", hjust=hjust, vjust=vjust),
    data=text.df)
gg.vjust+ggtitle("R graphics devices respect aes(vjust)")
```

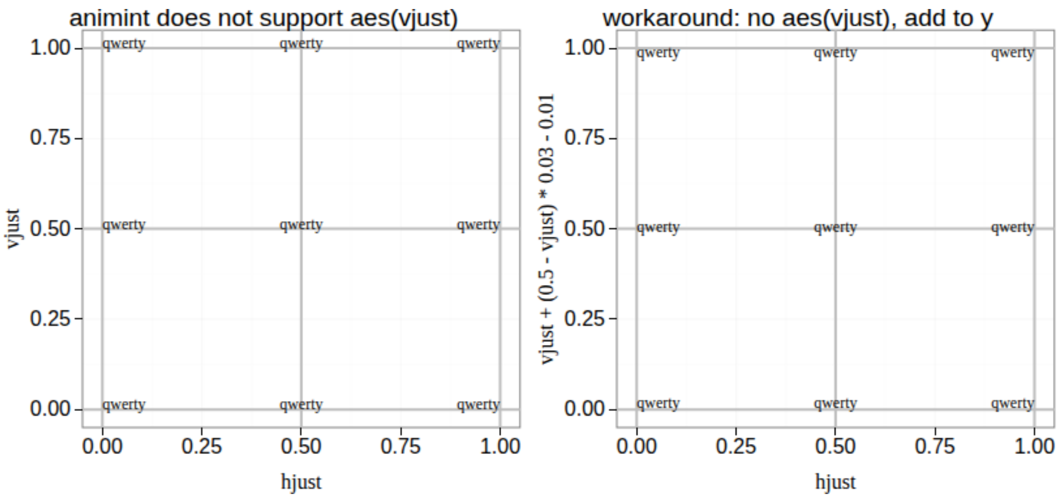


Note how both hjust and vjust are respected in the static ggplot above. In contrast, consider the animint below. The left plot should be the same as the ggplot above, but there are clear differences in terms of vertical placement of the text elements.

```
(viz.just <- animint(
  vjust=gg.vjust+
    ggtitle("animint does not support aes(vjust)"),
  workaround=gg.lines+
    ggtitle("workaround: no aes(vjust), add to y")+
    geom_text(aes(
      hjust, vjust + (0.5-vjust)*0.03 - 0.01,
      label="qwerty", hjust=hjust),
      data=text.df)))
```

```
[1] 0.5 1.0
```

```
Warning in vjustWarning(g.data$vjust): animint only supports vjust=0
```



The workaround in animint is shown in the right panel above. You can adjust the vertical position `y` values of the text elements for which you would have used `vjust`.

It is possible to implement `vjust` support for `geom_text` in animint, but we haven't yet had time to work on it. If you would like to implement it, we would be more than happy to accept a Pull Request. We already have an issue explaining how to implement it.

Another work-around is to use the new `geom_label_aligned`, which does support `vjust`. For an example, please read Chapter 8.

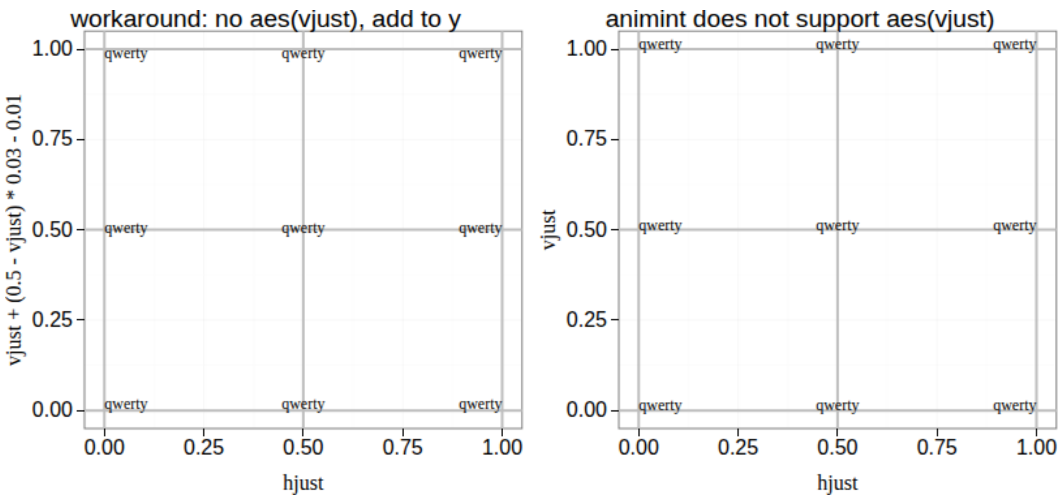
7.5 Order the plots on the page

Currently the only way to organize a multi-plot data viz is using the order of the ggplots in the animint `viz` list. The plots will appear on the web page in the same order as they occur in the animint `viz` list. For example, compare the data viz below with the data viz from the previous section.

```
animint(
  first=viz.just$workaround,
  second=viz.just$vjust)
```

```
[1] 0.5 1.0
```

```
Warning in vjustWarning(g.data$vjust): animint only supports vjust=0
```



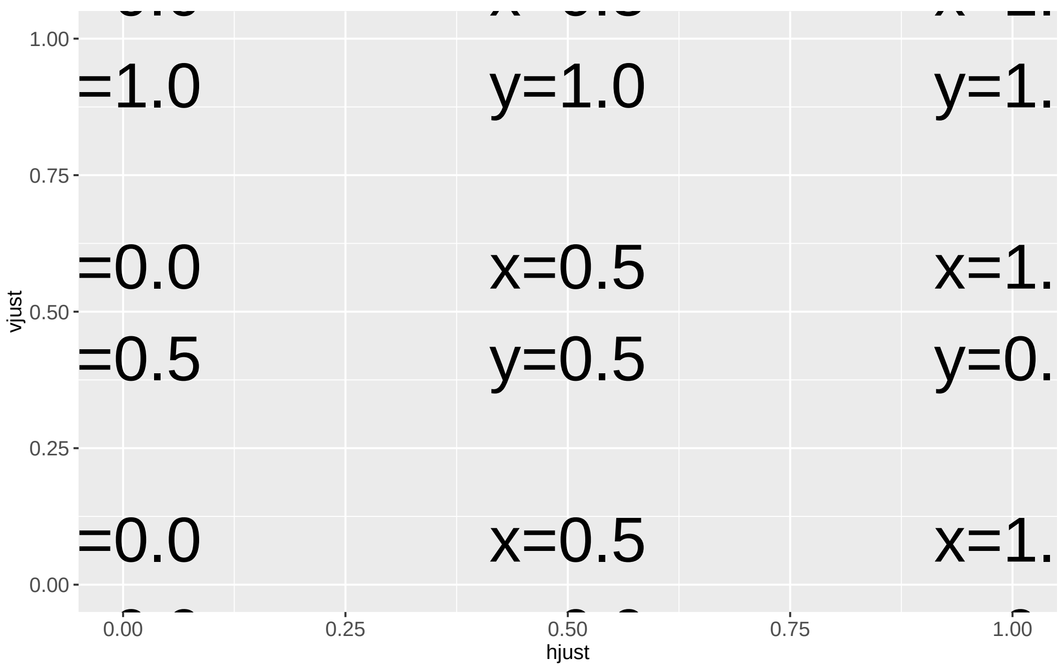
Note how the order of plots is reversed with respect to the data viz from the previous section. The main limitation of this method for plot layout is that only the order can be controlled. For example, depending on the width of the web page element in which the data viz above is rendered, the second plot will appear either below the first plot, or to the right of it.

If you have an idea for a better way to define the layout of plots in an animint, please tell us!

7.6 Avoid line breaks in text labels

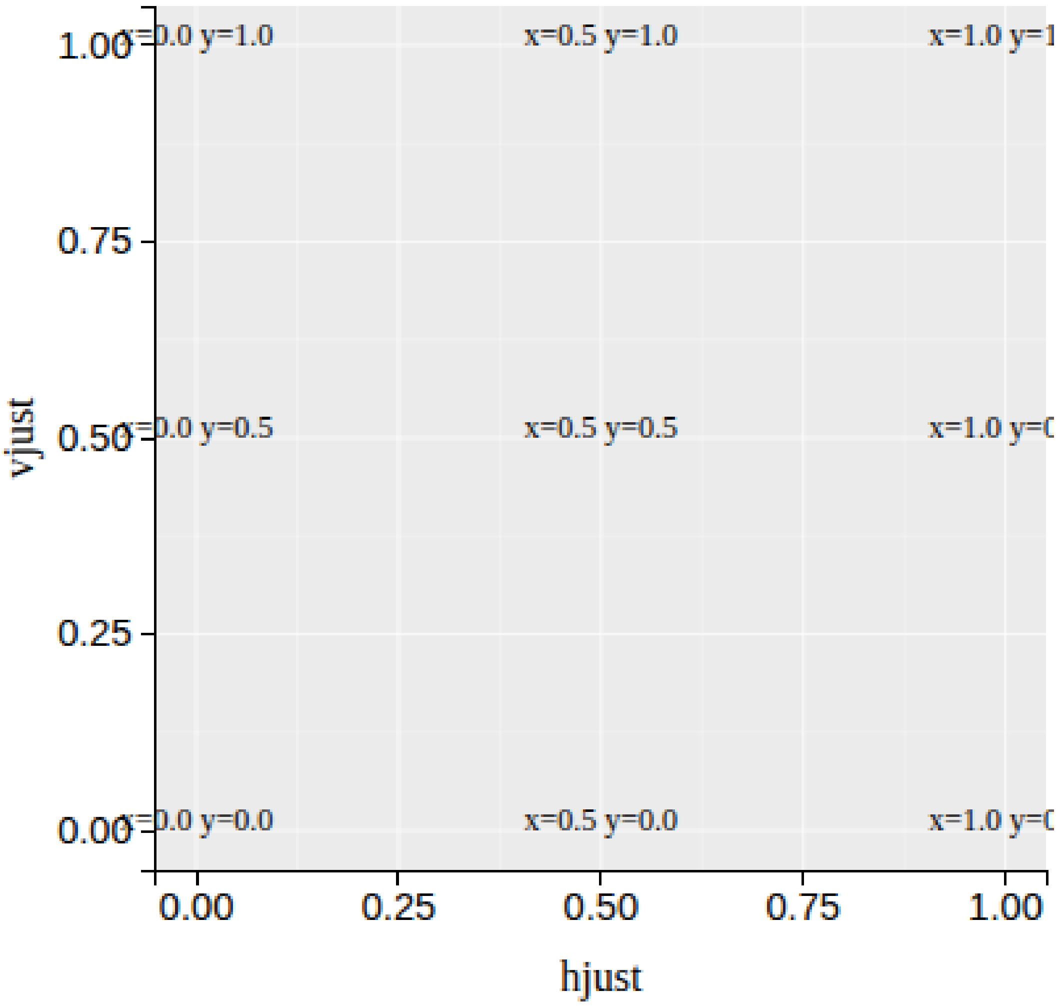
When rendering a ggplot using regular R graphics devices, using a line break or newline `\n` in a `geom_text` label results in multiple lines of text on the plot.

```r
gg.return <- ggplot()+
  geom_text(aes(
    hjust, vjust, label=sprintf("x=%.1f\ny=%.1f", hjust, vjust)),
    data=text.df)
gg.return
```



However, animint only supports drawing the first line of text.

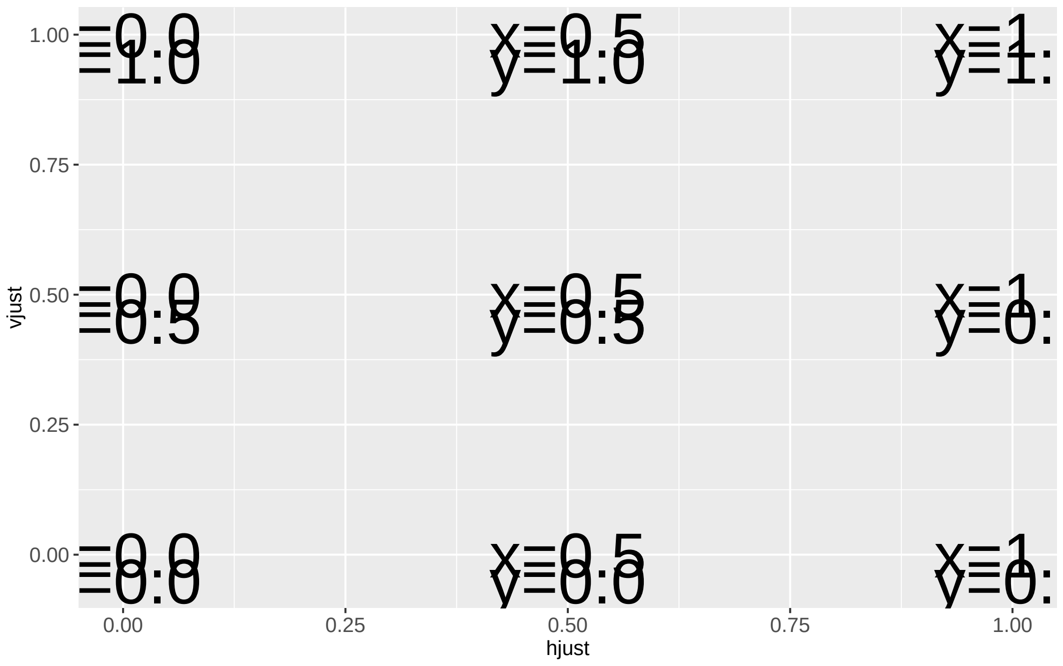
```
animint(gg.return)
```



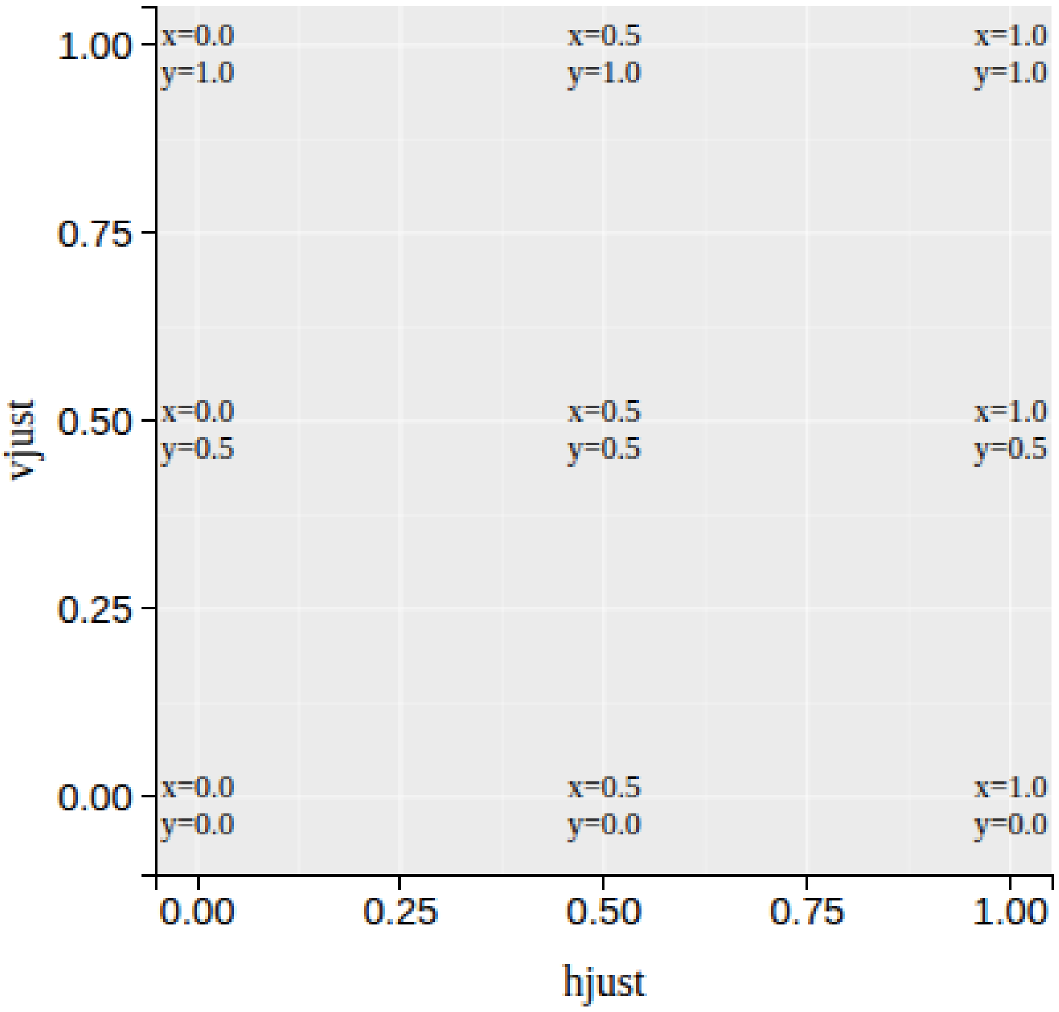
It would be nice to support multiple lines in `geom_text` labels, but we have not yet had time to implement that. However, we have an issue, and would be willing to accept a Pull Request which implements that.

Until then, the workaround is to use one `geom_text` layer for each line of text that you want to display:

```
gg.two.lines <- ggplot()+
  geom_text(aes(
    hjust, vjust, label=sprintf("x=%.1f", hjust)),
    data=text.df)+
  geom_text(aes(
    hjust, vjust-0.05, label=sprintf("y=%.1f", vjust)),
    data=text.df)
gg.two.lines
```



```
animint(gg.two.lines)
```



Also note in these examples that the text size is not consistent between static and interactive rendering, which is an issue

- in interactive plots, we want `size` to be the number of pixels, and
- in static plots, `size` interpretation is consistent with legacy `ggplot2` interpretation of size (could be changed for consistency with interactive rendering).

7.7 Avoid some ggplot theme options

One goal of animint is to support all of the theme options, but we have not yet had time to implement them all. If there is a theme option that you use and animint does not yet support, then please send us a Pull Request. The following list documents all the `theme` options that animint currently supports.

- `panel.margin` designates the distance between panels, and is used in the space saving facets idiom to eliminate the distance between panels.
- `panel.grid.major` is used to draw the the grid lines which are designated by the `breaks`

argument to the scale.
- panel.grid.minor is used to draw the grid lines between the major grid lines.
- panel.background is used for the <rect> in background of each panel.
- panel.border is used for the border <rect> of each panel (on top of the background <rect>).
- legend.position="none" works for hiding all of the legends, but none of the other legend positions are supported (the legends always appear on the right of the plot).

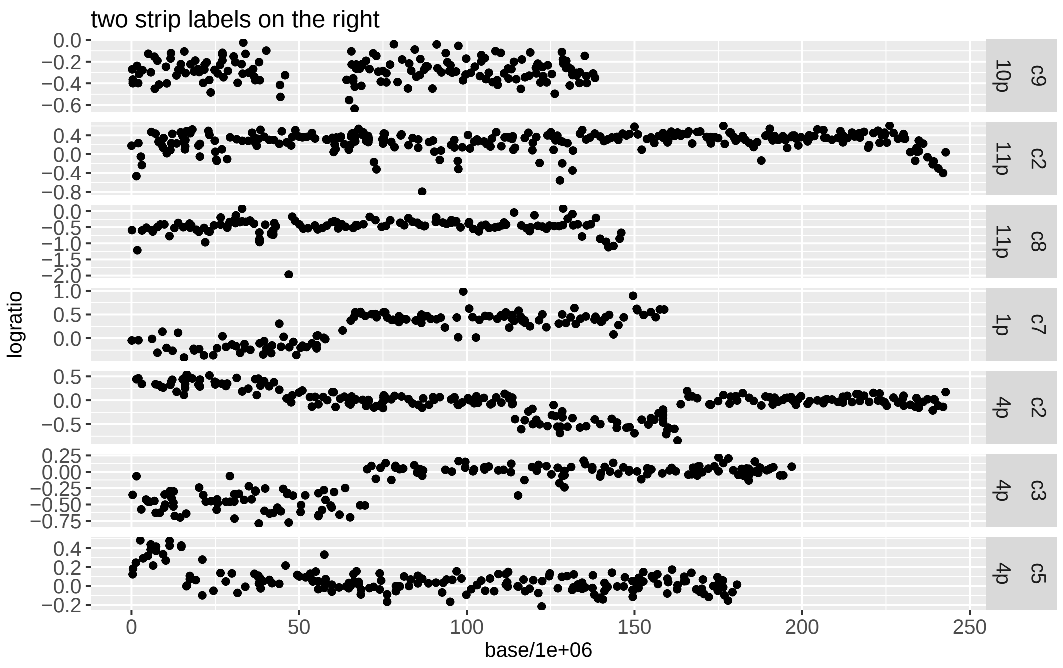
The following theme options can be set to element_blank to hide the axes.

- axis.title, axis.title.x, axis.title.y designate the axis title.
- axis.ticks, axis.ticks.x, axis.ticks.y designate teh axis ticks.
- axis.line, axis.line.x, axis.line.y designate the axis line.
- axis.text, axis.text.x, axis.text.y designate the axis tick label text, and support the angle and hjust arguments of element_text.

7.8 Facets with multiple variables per axis

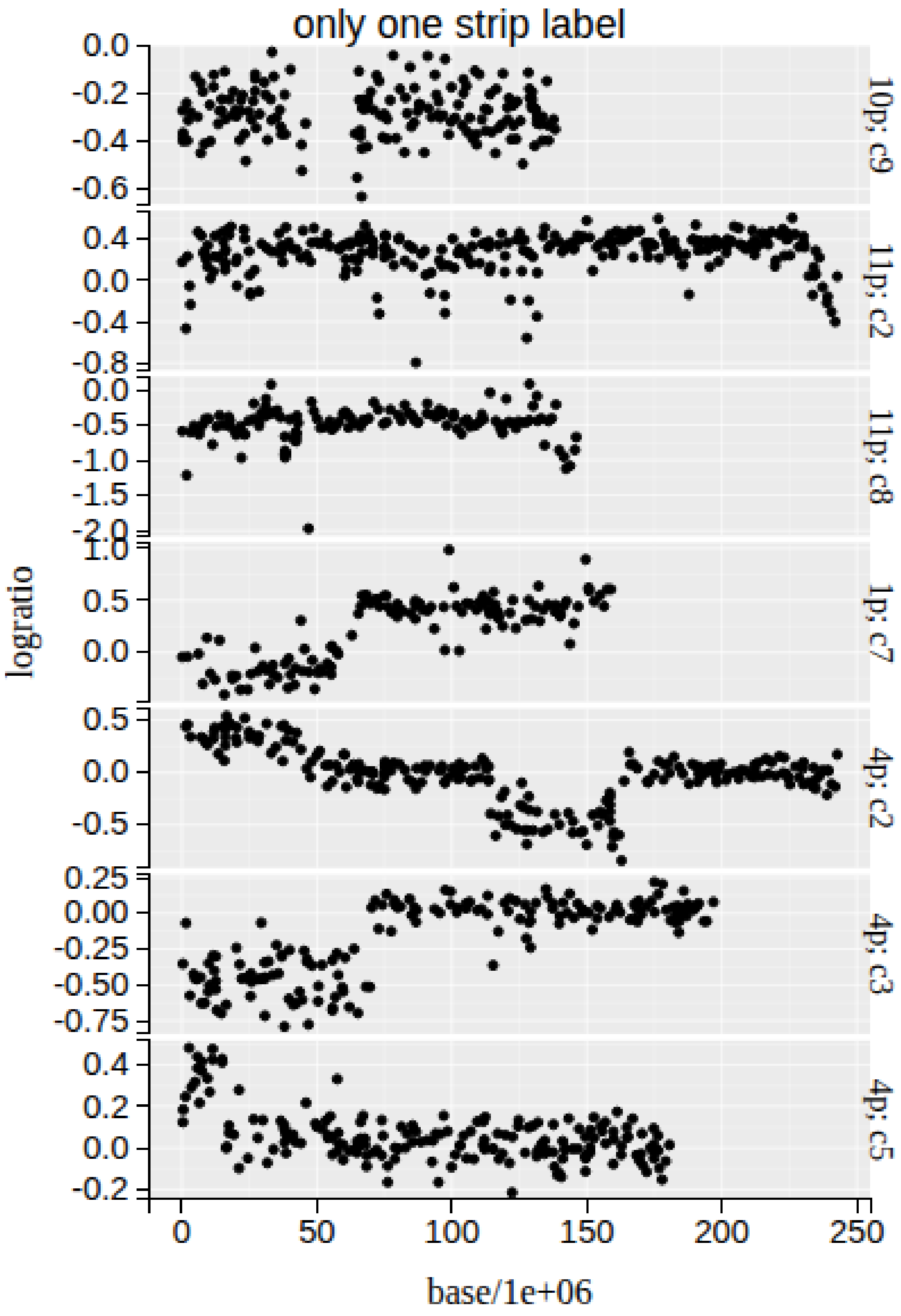
Animint supports facet_grid for creating multi-panel data visualizations, but only has limited support for multiple variables per axis. For example the ggplot below uses two variables to create vertical facets, which results in two strip labels when rendered with ggplot2.

```r
data(intreg)
signals.df <- transform(
  intreg$signals,
  person=sub("[.].*", "p", signal),
  chromosome=sub(".*[.]", "c", signal))
two.strips <- ggplot()+
  theme_animint(height=600)+
  facet_grid(person + chromosome ~ ., scales="free")+
  geom_point(aes(base/1e6, logratio), data=signals.df)
two.strips+ggtitle("two strip labels on the right")
```



In contrast, animint renders the same ggplot below using only one strip label.

```
animint(two.strips+ggtitle("only one strip label"))
```



It would be nice to support multiple strip labels per axis, but we have not yet had time to implement it. If you would like to implement this feature, we would be happy to accept your

Pull Request.

7.9 Interactive definition of aesthetic mappings using shiny

Normally, aesthetic mappings are defined once in R code, and can not be changed after rendering an animint. One way to overcome this limitation is by defining `shiny` inputs that are used as animint aesthetics, as in the following example.

```
shiny::runApp(system.file(
  "examples", "shiny-WorldBank", package="animint"))
```

7.10 Interactive computation

Another limitation is that animint can only display data that can be computed and stored in a data table before creating the visualization. This means that animint is not appropriate when there are more subsets of data to plot than you could ever compute. In that case, it would be better to use shiny.

7.11 Chapter summary and exercises

We discussed limitations of the current implementation of `animint2`, and explained several workarounds.

Exercises:

- Make a facetted ggplot with `stat_bin` that will not work with animint when the facet variable is instead used as a showSelected variable. Compute the stat yourself for each facet, and use `stat_identity` to make it work with animint.
- Make a ggplot which displays fine using `facet_grid(. ~ var, scales="free")` but does not display well in animint with `showSelected=var`. To fix the problem, compute a normalized version of `var` and use that for the `showSelected` variable.

Next, Chapter 8 explains how to create a multi-panel visualization of the World Bank data.

Part II

Advanced examples

8 World Bank

In this chapter we will explore several data visualizations of the World Bank data set.

Chapter outline:

- We begin by loading the World Bank data set and defining some helper functions for creating a multi-panel ggplot with several geoms.
- We then create a time series plot for life expectancy.
- We then add a scatterplot of life expectancy versus fertility rate as a second panel.
- We then add a third panel with a time series for fertility rate.

8.1 Load data and define helper functions

First we load the `WorldBank` data set, and consider only the subset which has both non-missing values for both `life.expectancy` and `fertility.rate`.

```
library(animint2)
data(WorldBank)
WorldBank$Region <- sub(
  " (all income levels)", "", WorldBank$region, fixed=TRUE)
library(data.table)
not.na <- data.table(WorldBank)[
  !(is.na(life.expectancy) | is.na(fertility.rate))
]
```

We will also be plotting the population variable using a size legend. Before plotting, we will make sure that none of the values are missing.

```
not.na[is.na(not.na$population)]
```

```
   iso2c country year fertility.rate life.expectancy population
1:    KW  Kuwait 1992          2.338        72.95266         NA
2:    KW  Kuwait 1993          2.341        73.07373         NA
3:    KW  Kuwait 1994          2.413        73.18724         NA
   GDP.per.capita.Current.USD 15.to.25.yr.female.literacy iso3c
1:                         NA                          NA   KWT
2:                         NA                          NA   KWT
3:                         NA                          NA   KWT
                                          region     capital longitude
1: Middle East & North Africa (all income levels) Kuwait City   47.9824
```

```
2: Middle East & North Africa (all income levels) Kuwait City   47.9824
3: Middle East & North Africa (all income levels) Kuwait City   47.9824
   latitude               income        lending                     Region
1:  29.3721 High income: nonOECD Not classified Middle East & North Africa
2:  29.3721 High income: nonOECD Not classified Middle East & North Africa
3:  29.3721 High income: nonOECD Not classified Middle East & North Africa
```

The table above shows that there are three rows with missing values for the population variable. They are for the country Kuwait during 1992-1994. The table below shows the data from the neighboring years, 1991-1995.

```
not.na[
  country == "Kuwait" & 1991 <= year & year <= 1995,
  .(country, year, population)]
```

```
   country year population
1:  Kuwait 1991    1999651
2:  Kuwait 1992         NA
3:  Kuwait 1993         NA
4:  Kuwait 1994         NA
5:  Kuwait 1995    1586123
```

The table above shows that the population of Kuwait decreased over the period 1991-1995, consistent with the Gulf War of that time period. We fill in those missing values below.

```
not.na[is.na(population), population := 1700000]
not.na[
  country == "Kuwait" & 1991 <= year & year <= 1995,
  .(country, year, population)]
```

```
   country year population
1:  Kuwait 1991    1999651
2:  Kuwait 1992    1700000
3:  Kuwait 1993    1700000
4:  Kuwait 1994    1700000
5:  Kuwait 1995    1586123
```

Next, we define the following helper function, which will be used to add columns to data sets in order to assign geoms to facets.

```
FACETS <- function(df, top, side)data.frame(
  df,
  top=factor(top, c("Fertility rate", "Years")),
  side=factor(side, c("Years", "Life expectancy")))
```

Note that the factor levels will specify the order of the facets in the ggplot. This is an example of the addColumn then facet idiom. Below, we define three more helper functions, one for each facet.

```
TS.LIFE <- function(df)FACETS(df, "Years", "Life expectancy")
SCATTER <- function(df)FACETS(df, "Fertility rate", "Life expectancy")
TS.FERT <- function(df)FACETS(df, "Fertility rate", "Years")
```

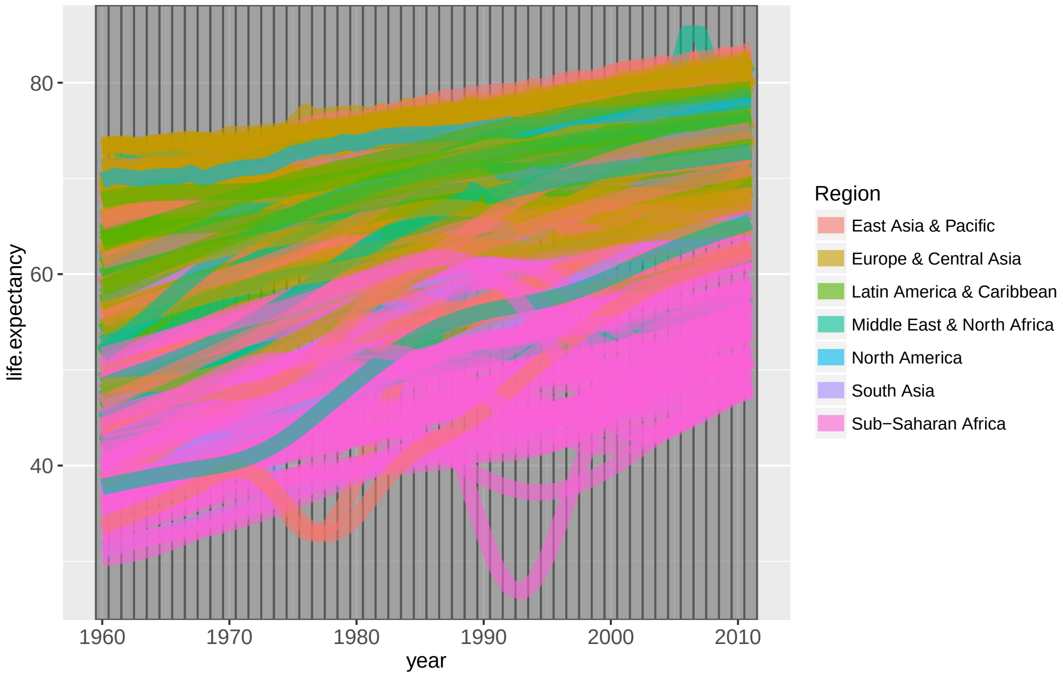
8.2 First time series plot

First we define a data set with one row for each year, which we will use for selecting years using a `geom_tallrect` in the background.

```
years <- unique(not.na[, .(year)])
```

We define the ggplot with a `geom_tallrect` in the background, and a `geom_line` for the time series.

```
line_alpha <- 3/5
line_size <- 4
ts.right <- ggplot()+
  geom_tallrect(aes(
    xmin=year-1/2, xmax=year+1/2),
    clickSelects="year",
    data=TS.LIFE(years), alpha=1/2)+
  geom_line(aes(
    year, life.expectancy, group=country, color=Region),
    clickSelects="country",
    data=TS.LIFE(not.na), size=line_size, alpha=line_alpha)
ts.right
```

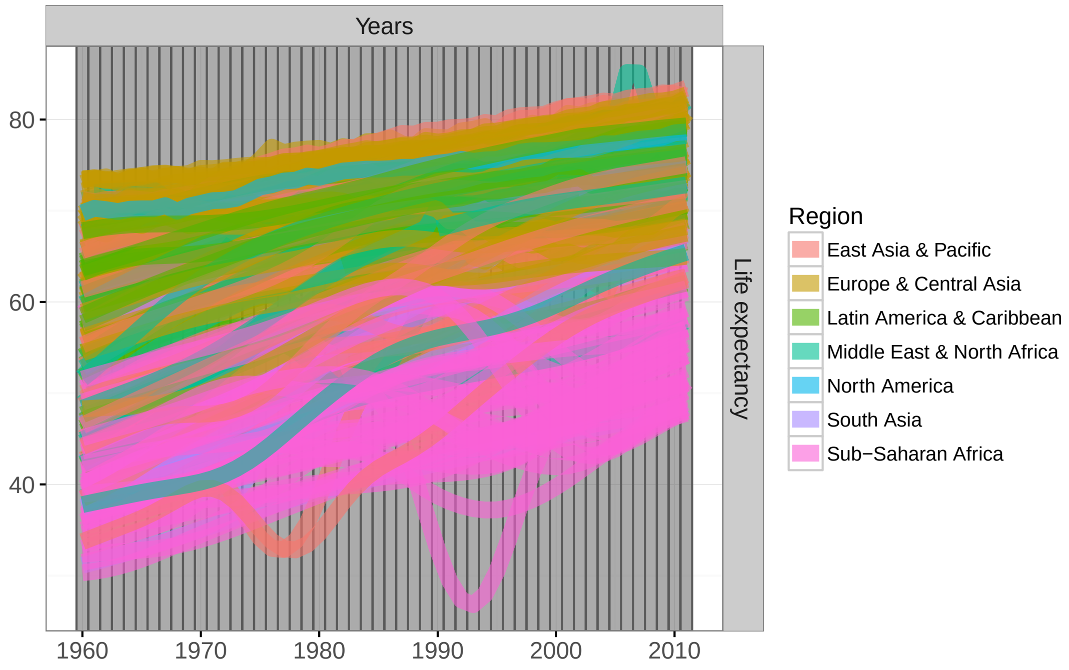


Note that we specified `clickSelects=year` so that clicking a tallrect will change the selected year, and `clickSelects=country` so that clicking a line will select or de-select a country. Also note that we used `TS.LIFE` to specify columns that we will use in the facet specification (next section).

8.3 Add a scatterplot facet

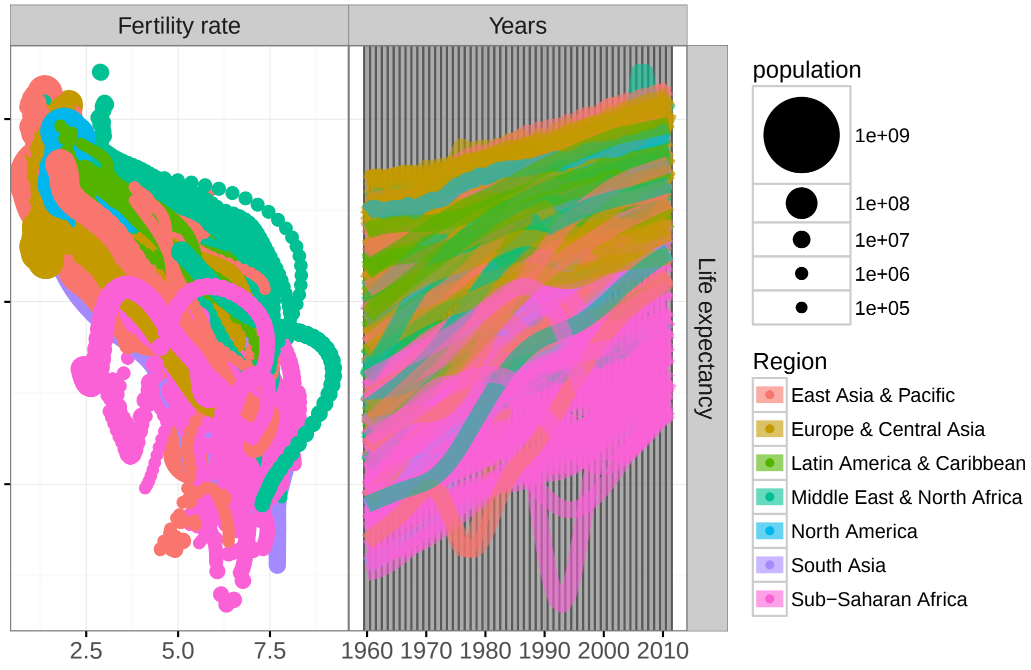
We begin by simply adding facets to the previous time series plot.

```
ts.facet <- ts.right+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(side ~ top, scales="free")+
  xlab("")+
  ylab("")
ts.facet
```



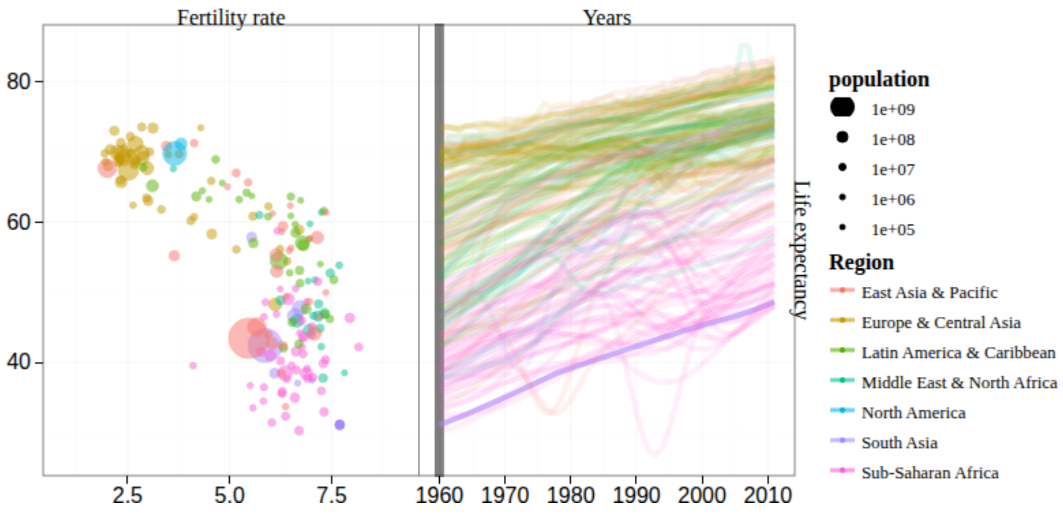
We set the `panel.margin` to 0, which is often a good idea to save space in a ggplot with facets. We use `scales="free"` and hide the axis labels, in an example of the addColumn then facet idiom. Instead, we use the facet label to show the variable encoded on each axis. Below, we add a scatterplot facet with a point for each year and country.

```
ts.scatter <- ts.facet+
  theme_animint(width=600)+
  geom_point(aes(
    fertility.rate, life.expectancy,
    color=Region, size=population,
    key=country), # key aesthetic for animated transitions!
    clickSelects="country",
    showSelected="year",
    data=SCATTER(not.na))+
  scale_size_animint(pixel.range=c(2, 20), breaks=10^(9:5))
ts.scatter
```



Note how we use `scale_size_animint` to specify the range of sizes in pixels, and the breaks in the legend. Also note that we use `SCATTER` to specify `top` and `side` columns which are used in the facet specification. We also render this ggplot interactively below.

```
animint(ts.scatter)
```



Note that single selection is used by default for both year and country.

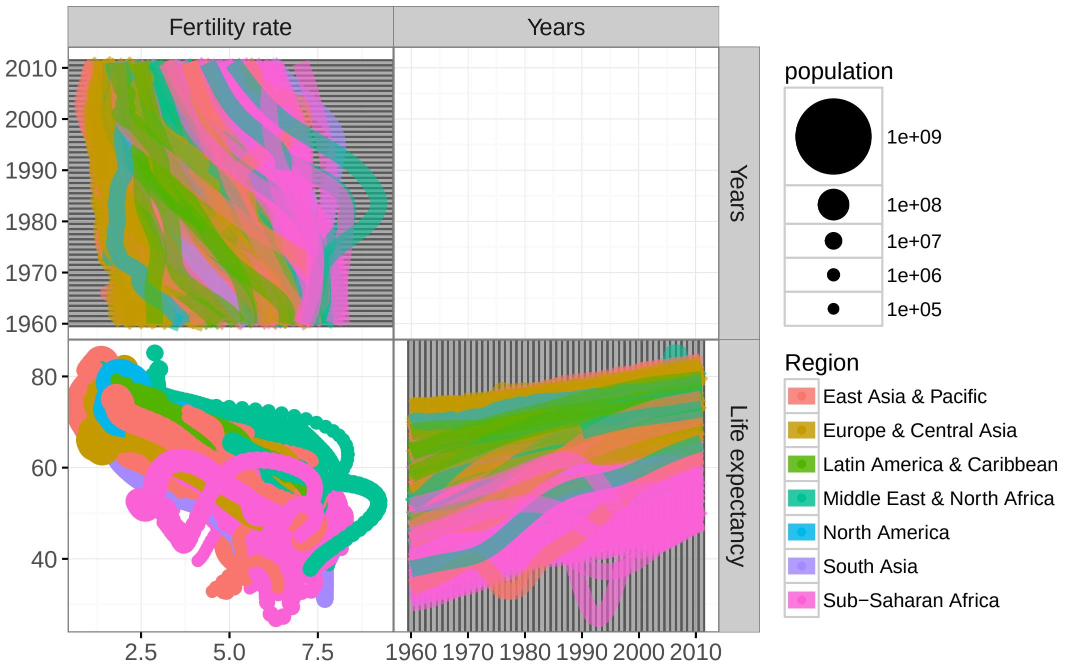
- The selected year is shown as a grey rectangle on the right.
- The selected country is shown with more opacity in the `geom_point` on the left, and in the `geom_line` on the right.

Exercise: how would you further emphasize the selected year and country? Hint: you can modify the `alpha_off` parameter from the default of `0.5` to a smaller value, like `0.2`. Try using `color_off`, which can not be used in combination with `aes(color)`, so try using `aes(fill)` instead in the `geom_point`.

8.4 Adding another time series facet

Below we add widerects for selecting years, and paths for showing fertility rate.

```
scatter.both <- ts.scatter+
  geom_widerect(aes(
    ymin=year-1/2, ymax=year+1/2),
    clickSelects="year",
    data=TS.FERT(years), alpha=1/2)+
  geom_path(aes(
    fertility.rate, year, group=country, color=Region),
    clickSelects="country",
    data=TS.FERT(not.na), size=line_size, alpha=line_alpha)
scatter.both
```



Note in the code above that `TS.FERT` was used to specify facet columns `top` and `side`. A final touch is to add text labels to the time series, using `geom_label_aligned`, which is new in `animint2` (it is not in `ggplot2`). It is a text label that adjusts its position to avoid overlaps with other labels with the same `y` value (in horizontal alignment). The code below first creates a data set with the extreme values of `year`, and then uses that with `alignment="horizontal"`.

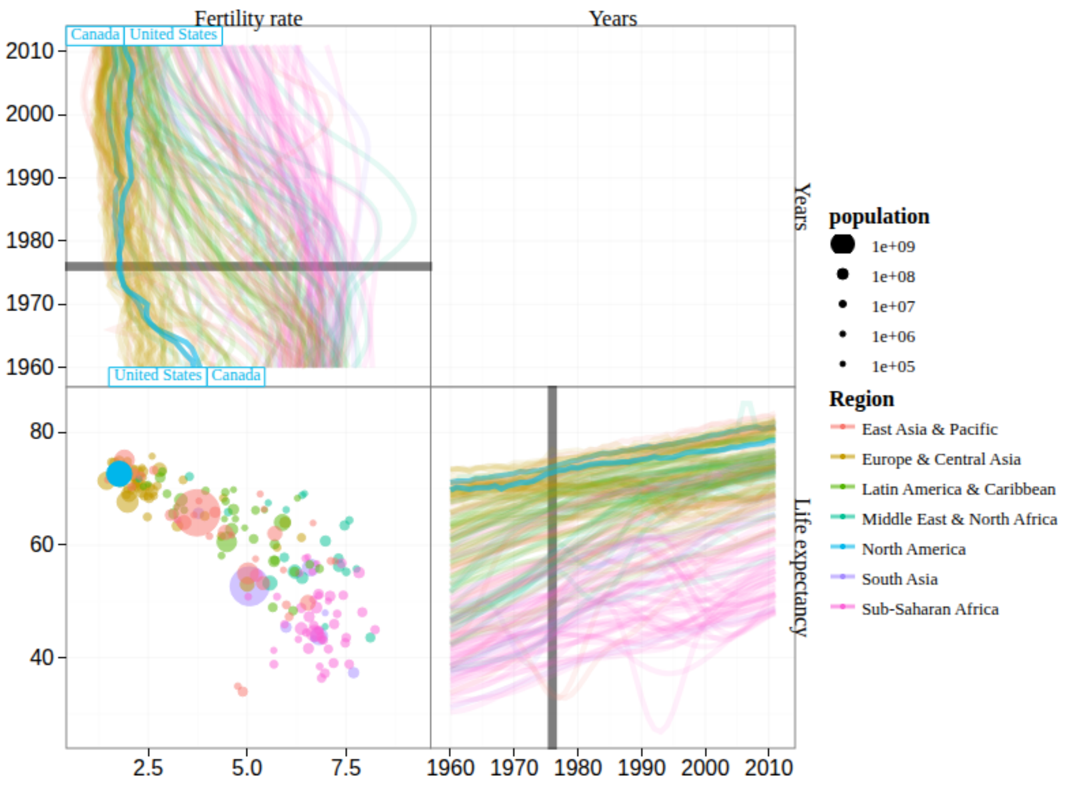
```
ext.years <- not.na[year %in% range(year)]
scatter.labels <- scatter.both+
  geom_label_aligned(aes(
    fertility.rate, year,
    vjust=ifelse(year==min(year), 1, 0),
    color=Region,
    label=country),
    data=TS.FERT(ext.years),
    showSelected="country",
    alignment="horizontal")
```

Note in the code above that we set `vjust`

- to 1 so that the top of the label is aligned with the min year at the bottom of the panel.
- to 0 so that the bottom of the label is aligned with the max year at the top of the panel.

We render an interactive version below.

```
animint(
  title="World Bank data (multiple selection, facets)",
  scatter=scatter.labels+
    theme_animint(width=600, height=600),
  duration=list(year=1000),
  time=list(variable="year", ms=3000),
  first=list(year=1975, country=c("United States", "Canada")),
  selector.types=list(country="multiple"))
```



The visualization above has three facets: two time series, and one scatter plot. The fertility rate time series shows two labels for each selected country, with a few special features:

- If the values of fertility rate for selected countries are too close, then the label positions are adjusted to avoid overlapping text.
- If there are selected countries near the left/right plot boundaries, then the labels are adjusted to avoid going outside of these boundaries.
- If there are too many countries selected to display all text labels in the available space (between left and right boundaries), then the text size is reduced until the text labels fit.
- These features are used in each group of labels with the same Y value, because `alignment="horizontal"` was specified.

Try selecting a few more neighboring countries to see how this works.

8.5 Chapter summary and exercises

We showed how to create a multi-layer, multi-panel (but single-plot) visualization of the World Bank data.

Exercises:

- Simplify the code by using `make_*()` instead of `geom_*()` for `tallrect` and `widerect`.
- The X axis for fertility rate shows default breaks 2.5, 5.0, 7.5. Change these to 2, 4, 6, 8. Hint: use `breaks` argument of `scale_x_continuous`.
- Since no smooth transition has been specified for `country`, the text labels appear and disappear instantaneously when the set of selected countries is modified. Try adding a smooth transition, by adding the global `duration` option, and by adding `aes(key)` to the `geom_label_aligned`. Hint: since there are two labels for each country, the key should depend on both `year` and `country`.
- Make it so that clicking a country label de-selects the corresponding country.
- Add text labels to the time series plot on the right, with names for each country, using `geom_label_aligned(alignment="vertical")`. Each label should appear only when the country is selected, and should disappear after clicking on the label.
- Add a text label to the scatterplot to indicate the selected year.
- Add text labels to the scatterplot, with names for each country. Each label should appear only when the country is selected, and should disappear after clicking on the label.
- Add points on each time series plot, with size proportional to population, as in the scatterplot. The points should appear only when the country is selected, and clicking the points should de-select that country.
- As in this gallery example, add world map in the Year/Year facet which is currently empty.

Next, Chapter 9 explains how to visualize the Montreal bike data set.

9 Montreal bikes

In this chapter we will explore several data visualizations of the Montreal bike data set.

Chapter outline:

- We begin with some static data visualizations.
- We create an interactive visualization of accident frequency over time.
- We create a interactive data viz with four plots, showing monthly accident trends, daily details, and a map of counter locations.

9.1 Static figures

We begin by loading the `montreal.bikes` data set, which is not available in the CRAN release of `animint2`, in order to save space on CRAN. Therefore to access this data set, you will need to install `animint2` from GitHub:

```
tryCatch({
  data(montreal.bikes, package="animint2")
}, warning=function(w){
  remotes::install_github("tdhock/animint2")
})
```

We begin by examining the accidents data table.

```
library(animint2)
data(montreal.bikes) #only present if installed from github
old.locale <- Sys.setlocale(locale="en_US.UTF-8")
for(col_name in c("nom","nom_comptage","Etat")){
  montreal.bikes$counter.locations[[col_name]] <- iconv(
    montreal.bikes$counter.locations[[col_name]], "latin1", "UTF-8")
}
library(data.table)
accidents.dt <- data.table(montreal.bikes$accidents)
accidents.dt[1]
```

```
     date.str time.str deaths people.severely.injured people.slightly.injured
1: 2012-01-02    18:35      0                       0                       1
   street.number            street cross.street location.int position.int
1:            NA ST JEAN BAPTISTE O   AV ROULEAU           32            6
              position                                    location
```

```
1: Voie de circulation En intersection (moins de 5 mètres)
```

Each accident has data about its date, time, location, and counts of death and slight/severe injury. Some of the values are in French (e.g. position Voie de circulation, location En intersection, etc).

We calculate the time period of the accidents below.

```
(accidents.dt[
, date.POSIXct := as.POSIXct(strptime(date.str, "%Y-%m-%d"))
][
, month.str := strftime(date.POSIXct, "%Y-%m")
][])
```

```
        date.str time.str deaths people.severely.injured
   1: 2012-01-02    18:35      0                       0
   2: 2012-01-05    21:50      0                       0
  ---
5594: 2014-12-27    12:35      0                       0
5595: 2014-12-30    11:55      0                       0
      people.slightly.injured street.number             street   cross.street
   1:                       1            NA ST JEAN BAPTISTE O     AV ROULEAU
   2:                       1            NA             FOSTER        JANELLE
  ---
5594:                       1            NA   CH DES PATRIOTES        1RE RUE
5595:                       1         14965     PIERREFONDS BD JACQUES BIZARD
      location.int position.int                         position
   1:           32            6              Voie de circulation
   2:           34            6              Voie de circulation
  ---
5594:           33            6              Voie de circulation
5595:           33            5 Voie cyclable / chaussée désignée
                                        location date.POSIXct month.str
   1:        En intersection (moins de 5 mètres)   2012-01-02   2012-01
   2:      Entre intersections (100 mètres et +)   2012-01-05   2012-01
  ---
5594: Près d'une intersection/carrefour giratoire   2014-12-27   2014-12
5595: Près d'une intersection/carrefour giratoire   2014-12-30   2014-12
```

```
range(accidents.dt$month.str)
```

```
[1] "2012-01" "2014-12"
```

Below we also compute the range of months for the bike counter data table.

```
(counts.dt <- data.table(montreal.bikes$counter.counts))
```

```
            location                date count
    1:         Berri 2009-01-01 05:00:00    29
    2:         Berri 2009-01-02 05:00:00    19
   ---
13382: Totem_Laurier 2013-09-17 04:00:00  3745
```

```
13383: Totem_Laurier 2013-09-18 04:00:00  3921
```

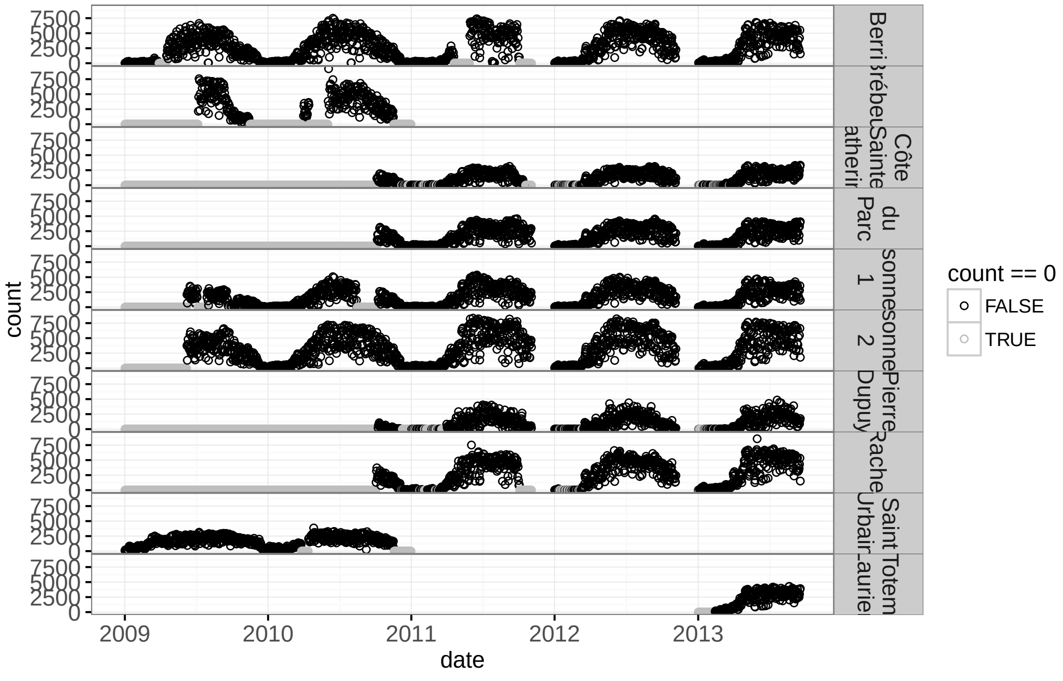
```
counts.dt[, month.str := strftime(date, "%Y-%m")]
range(counts.dt$month.str)
```

```
[1] "2009-01" "2013-09"
```

The bike counts are time series data which we visualize below.

```
counts.dt[, loc.lines := gsub("[- _]", "\n", location)]
ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(loc.lines ~ .)+
  geom_point(aes(
    date, count, color=count==0),
    shape=21,
    data=counts.dt)+
  scale_color_manual(values=c("TRUE"="grey", "FALSE"="black"))
```

```
Warning: Removed 407 rows containing missing values (geom_point).
```



Plotting with `geom_point` makes it easy to see the difference between zeros and missing values.

We will compute a summary of all accidents per month in this time period, so we first create a data table for each month below. (and make sure to set the locale to C for English month names)

```
uniq.month.vec <- unique(c(
  accidents.dt$month.str,
  counts.dt$month.str))
one.day <- 60 * 60 * 24
months <- data.table(month.str=uniq.month.vec)[
, month01.str := paste0(month.str, "-01")
][
, month01.POSIXct := as.POSIXct(strptime(month01.str, "%Y-%m-%d"))
][, let(
  next.POSIXct = month01.POSIXct + one.day * 31,
  month.str = strftime(month01.POSIXct, "%B %Y")
)][
, next01.str := paste0(strftime(next.POSIXct, "%Y-%m"), "-01")
][
, next01.POSIXct := as.POSIXct(strptime(next01.str, "%Y-%m-%d"))
]
month.levs <- months[order(month01.POSIXct), month.str]
(months[, month := factor(month.str, month.levs)][])
```

```
        month.str month01.str month01.POSIXct next.POSIXct next01.str
 1:  January 2012  2012-01-01      2012-01-01   2012-02-01 2012-02-01
 2: February 2012  2012-02-01      2012-02-01   2012-03-03 2012-03-01
---
71: November 2011  2011-11-01      2011-11-01   2011-12-02 2011-12-01
72: December 2011  2011-12-01      2011-12-01   2012-01-01 2012-01-01
    next01.POSIXct         month
 1:     2012-02-01  January 2012
 2:     2012-03-01 February 2012
---
71:     2011-12-01 November 2011
72:     2012-01-01 December 2011
```

Note that we created a `month` column which is a factor ordered by `month.levs`.

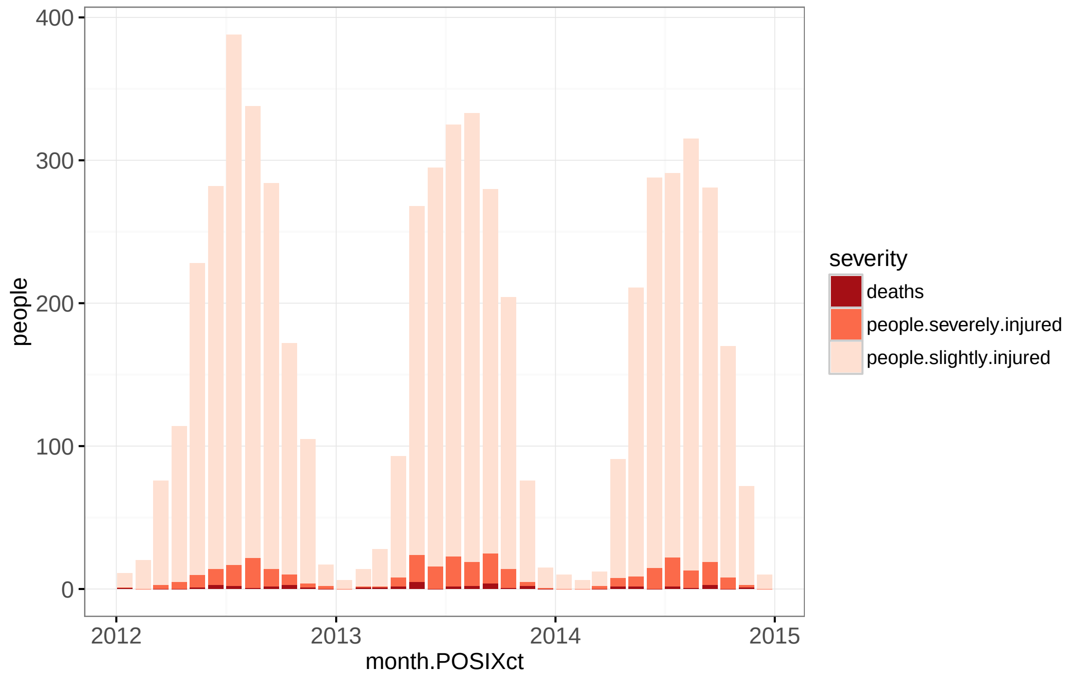
```
month_pos_ct <- function(mstr)as.POSIXct(
  strptime(paste0(mstr, "-15"), "%Y-%m-%d"))
accidents.dt[
, month.text := strftime(date.POSIXct, "%B %Y")
][
, month := factor(month.text, month.levs)
][
, month.POSIXct := month_pos_ct(month.str)
][]
```

```
        date.str time.str deaths people.severely.injured
   1: 2012-01-02    18:35      0                       0
   2: 2012-01-05    21:50      0                       0
  ---
5594: 2014-12-27    12:35      0                       0
5595: 2014-12-30    11:55      0                       0
      people.slightly.injured street.number             street    cross.street
   1:                       1            NA ST JEAN BAPTISTE O     AV ROULEAU
   2:                       1            NA              FOSTER       JANELLE
  ---
5594:                       1            NA   CH DES PATRIOTES        1RE RUE
5595:                       1         14965     PIERREFONDS BD JACQUES BIZARD
      location.int position.int                          position
   1:           32            6              Voie de circulation
   2:           34            6              Voie de circulation
  ---
5594:           33            6              Voie de circulation
5595:           33            5 Voie cyclable / chaussée désignée
                                       location date.POSIXct month.str
   1:         En intersection (moins de 5 mètres)   2012-01-02   2012-01
   2:       Entre intersections (100 mètres et +)   2012-01-05   2012-01
  ---
5594: Près d'une intersection/carrefour giratoire   2014-12-27   2014-12
5595: Près d'une intersection/carrefour giratoire   2014-12-30   2014-12
         month.text         month month.POSIXct
   1:  January 2012  January 2012    2012-01-15
   2:  January 2012  January 2012    2012-01-15
  ---
5594: December 2014 December 2014    2014-12-15
5595: December 2014 December 2014    2014-12-15
```

```r
stopifnot(!is.na(accidents.dt$month.POSIXct))
accidents.per.month <- accidents.dt[, list(
  total.accidents=.N,
  total.people=sum(
    deaths+people.severely.injured+people.slightly.injured),
  deaths=sum(deaths),
  people.severely.injured=sum(people.severely.injured),
  people.slightly.injured=sum(people.slightly.injured),
  next.POSIXct = month.POSIXct + one.day * 30,
  month01.str = paste0(strftime(month.POSIXct, "%Y-%m"), "-01")
), by=.(month, month.str, month.text, month.POSIXct)][, let(
  month01.POSIXct = as.POSIXct(strptime(month01.str, "%Y-%m-%d")),
  next01.str = paste0(strftime(next.POSIXct, "%Y-%m"), "-01")
)][
, next01.POSIXct := as.POSIXct(strptime(next01.str, "%Y-%m-%d"))
][]
```

We plot the accidents per month below.

```r
accidents.tall <- melt(
  accidents.per.month,
  measure.vars=c(
    "deaths", "people.severely.injured", "people.slightly.injured"),
  variable.name="severity",
  value.name="people")
severity.colors <- c(
  deaths="#A50F15",#dark red
  people.severely.injured="#FB6A4A",
  people.slightly.injured="#FEE0D2")#lite red
ggplot()+
  theme_bw()+
  geom_bar(aes(
    month.POSIXct, people, fill=severity),
    stat="identity",
    data=accidents.tall)+
  scale_fill_manual(values=severity.colors)
```



In each accident, there are counts of people who died, along with people who suffered severe and slight injuries. Below we classify the severity of each accident according to the worst outcome among the people affected.

```
  accidents.dt[
  , severity.str := fcase(
    0 < deaths, "deaths",
    0 < people.severely.injured, "people.severely.injured",
    default="people.slightly.injured")
  ][
  , severity := factor(severity.str, names(severity.colors))
  ][
  , table(severity)
  ]
```

```
severity
                 deaths people.severely.injured people.slightly.injured
```

```
                  44                    289                    5262
```

The output above shows that accidents with only slight injuries are most frequent, and accidents with at least one death are least frequent. Below we compute counts per month.

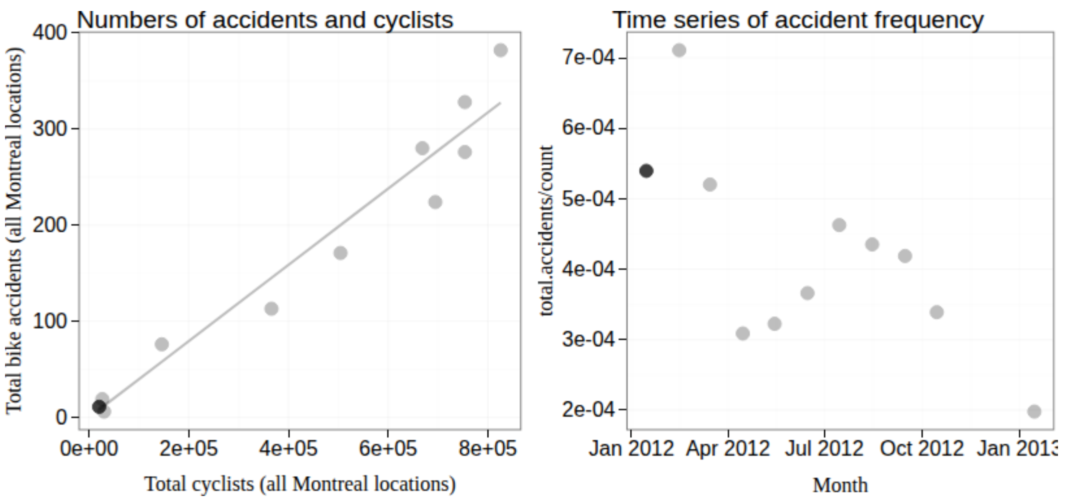
```
counts.per.month <- counts.dt[, let(
  month.POSIXct = month_pos_ct(month.str),
  month.text = strftime(date, "%B %Y"),
  day.of.the.month = as.integer(strftime(date, "%d"))
)][
, month := factor(month.text, month.levs)
][, list(
  days=.N,
  mean.per.day=mean(count),
  count=sum(count),
  month01.str = paste0(month.str, "-01")
), by=.(location, month, month.str, month.POSIXct)][
  0 < count
][
, month01.POSIXct := as.POSIXct(strptime(month01.str, "%Y-%m-%d"))
][
, next.POSIXct := month01.POSIXct + one.day * 31
][
, next01.str := paste0(strftime(next.POSIXct, "%Y-%m"), "-01")
][
, next01.POSIXct := as.POSIXct(strptime(next01.str, "%Y-%m-%d"))
][
, days.in.month := as.integer(round(difftime(next01.POSIXct,month01.POSIXct,units="days")))
][]
counts.per.month[days < days.in.month, {
  list(location, month, days, days.in.month)
}]
```

```
                  location          month days days.in.month
 1:                  Berri  November 2012    5            30
 2: Côte-Sainte-Catherine  November 2012    5            30
---
14:                 Rachel September 2013   18            30
15:          Totem_Laurier September 2013   18            30
```

As shown above, some months do not have observations for all days.

9.2 Interactive viz of accident frequency

```r
complete.months <- counts.per.month[days == days.in.month]
month.labels <- counts.per.month[, {
  .SD[which.max(count), ]
}, by=location]
day.labels <- counts.dt[, {
  .SD[which.max(count), ]
}, by=.(location, month)]
city.wide.cyclists <- counts.per.month[0 < count, list(
  locations=.N,
  count=sum(count),
  month01.str = paste0(month.str, "-01")
), by=.(month, month.str, month.POSIXct)][
, month01.POSIXct := as.POSIXct(strptime(month01.str, "%Y-%m-%d"))
][
, next.POSIXct := month01.POSIXct + one.day * 31
][
, next01.str := paste0(strftime(next.POSIXct, "%Y-%m"), "-01")
][
, next01.POSIXct := as.POSIXct(strptime(next01.str, "%Y-%m-%d"))
][]
month.str.vec <- strftime(seq(
  strptime("2012-01-15", "%Y-%m-%d"),
  strptime("2013-01-15", "%Y-%m-%d"),
  by="month"), "%Y-%m")
city.wide.complete <- complete.months[0 < count, list(
  locations=.N,
  count=sum(count),
  month01.str = paste0(month.str, "-01")
), by=.(month, month.str, month.POSIXct)]
setkey(city.wide.complete, month.str)
scatter.cyclists <- city.wide.complete[month.str.vec]
scatter.accidents <- accidents.per.month[
  scatter.cyclists, on=.(month.str)]
scatter.not.na <- scatter.accidents[!is.na(locations),]
scatter.max <- scatter.not.na[locations==max(locations)]
fit <- lm(total.accidents ~ count - 1, scatter.max)
scatter.max[, pred.accidents := predict(fit)]
animint(
  regression=ggplot()+
    theme_bw()+
    ggtitle("Numbers of accidents and cyclists")+
    geom_line(aes(
      count, pred.accidents),
      color="grey",
      data=scatter.max)+
      geom_point(aes(
        count, total.accidents),
        shape=1,
        clickSelects="month",
        size=5,
        alpha=0.75,
        data=scatter.max)+
      ylab("Total bike accidents (all Montreal locations)")+
      xlab("Total cyclists (all Montreal locations)"),
    timeSeries=ggplot()+
      theme_bw()+
      ggtitle("Time series of accident frequency")+
      xlab("Month")+
      geom_point(aes(
        month.POSIXct, total.accidents/count),
        clickSelects="month",
        size=5,
        alpha=0.75,
        data=scatter.max))
```

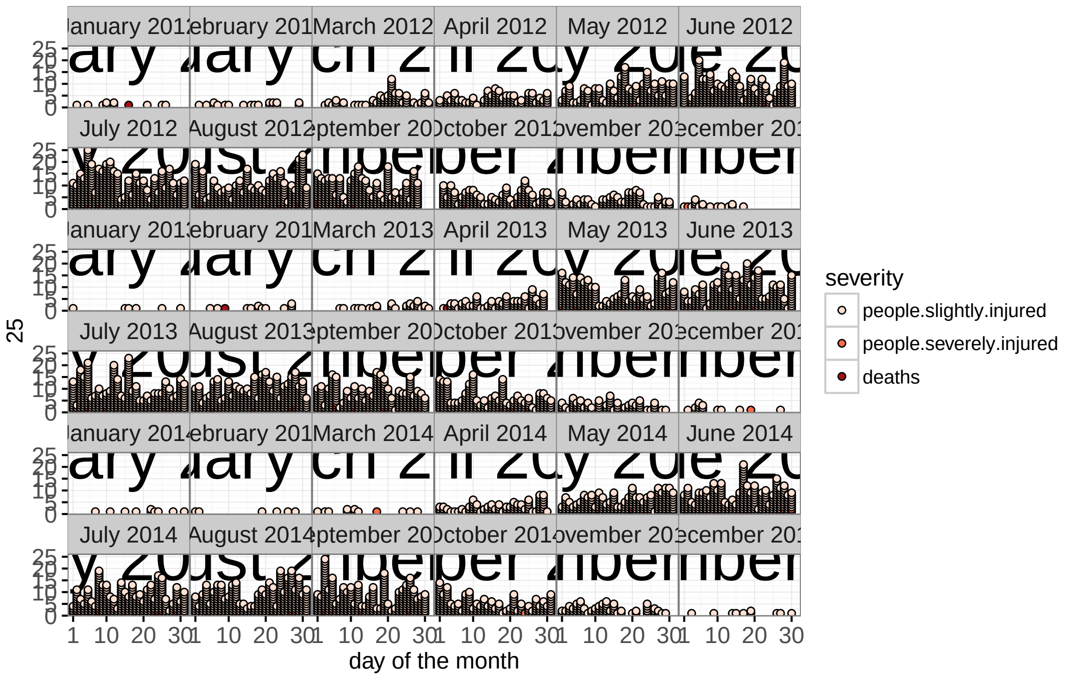


The data viz above shows two data visualizations of city-wide accident frequency over time. The plot on the left shows that the number of accidents grows with the number of cyclists. The plot on the right shows the frequency of accidents over time.

9.3 Interactive viz with map and details

The plot below is a dotplot of accidents for each month. Each dot represents one person who got in an accident.

```r
accidents.cumsum <- accidents.dt[
  order(date.POSIXct, month, severity)
][
, accident.i := seq_along(severity)
, by=.(date.POSIXct, month)
][
, day.of.the.month := as.integer(strftime(date.POSIXct, "%d"))
][]
ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "cm"))+
  facet_wrap("month")+
  geom_text(aes(15, 25, label=month), data=accidents.per.month)+
  scale_fill_manual(values=severity.colors, breaks=rev(names(severity.colors)))+
  scale_x_continuous("day of the month", breaks=c(1, 10, 20, 30))+
  geom_point(aes(
    day.of.the.month, accident.i, fill=severity),
    shape=21,
    data=accidents.cumsum)
```



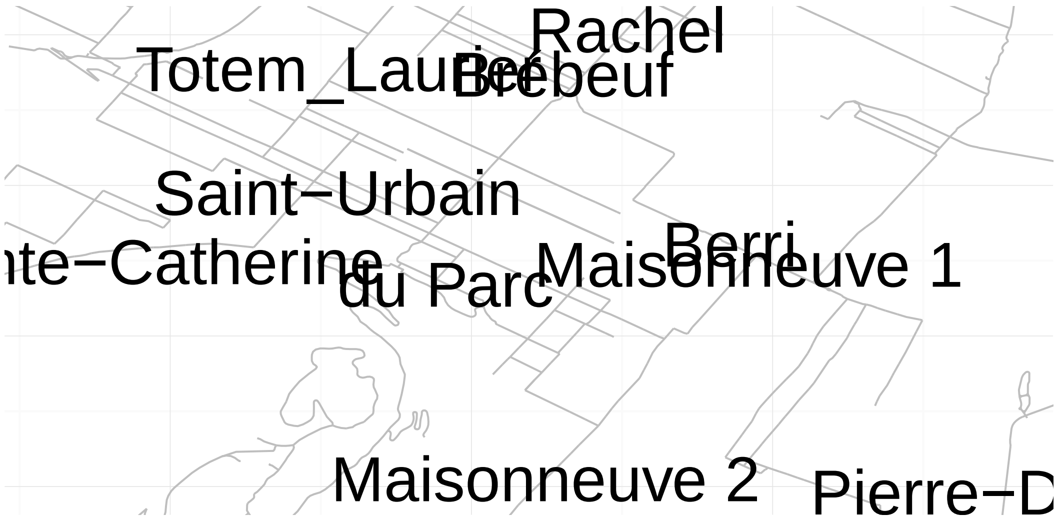
```r
counter.locations <- data.table(
  montreal.bikes$counter.locations
)[, let(
  lon = coord_X,
  lat = coord_Y
)][]
loc.name.code <- c(
    "Berri1"="Berri",
    "Brebeuf"="Brébeuf",
    CSC="Côte-Sainte-Catherine",
    "Maisonneuve_1"="Maisonneuve 1",
    "Maisonneuve_2"="Maisonneuve 2",
    "Parc"="du Parc",
    PierDup="Pierre-Dupuy",
    "Rachel/Papineau"="Rachel",
    "Saint-Urbain"="Saint-Urbain",
    "Totem_Laurier"="Totem_Laurier")
  counter.locations[, location := loc.name.code[nom_comptage] ]
  velo.counts <- table(counts.dt$location)
  (show.locations <- counter.locations[
    names(velo.counts), on=.(location)])
```

```
    id                  nom    nom_comptage          Etat     Type
 1:  3              Berri_1          Berri1      Existant compteur
 2:  2            Brebeuf_1         Brebeuf      Existant compteur
 3:  8 Cote-Ste-Catherine_1             CSC      Existant compteur
 4:  4        Maisonneuve_1   Maisonneuve_1 À réinstaller compteur
 5:  5        Maisonneuve_2   Maisonneuve_2      Existant compteur
 6: 22               Parc_1            Parc      Existant compteur
 7: 12       Pierre-Dupuy_1         PierDup      Existant compteur
 8:  6      Rachel/Papineau Rachel/Papineau      Existant compteur
 9:  1           St-Urbain_1    Saint-Urbain      Existant compteur
10: 37        Totem_Laurier   Totem_Laurier      Existant    totem
    Annee_implante   coord_X  coord_Y       lon      lat              location
 1:           2008 -73.56284 45.51613 -73.56284 45.51613                 Berri
 2:           2009 -73.57398 45.52741 -73.57398 45.52741               Brébeuf
 3:           2010 -73.60783 45.51496 -73.60783 45.51496 Côte-Sainte-
Catherine
 4:           2008 -73.56159 45.51479 -73.56159 45.51479         Maisonneuve 1
 5:           2008 -73.57508 45.50054 -73.57508 45.50054         Maisonneuve 2
 6:           2010 -73.58171 45.51346 -73.58171 45.51346               du Parc
 7:           2010 -73.54455 45.49966 -73.54455 45.49966          Pierre-
Dupuy
 8:           2007 -73.56965 45.53036 -73.56965 45.53036                Rachel
 9:           2014 -73.58888 45.51955 -73.58888 45.51955          Saint-
Urbain
10:           2013 -73.58883 45.52777 -73.58883 45.52777         Totem_Laurier
```

The counter locations above will be plotted below. Note that we use `showSelected=month` and `clickSelects=location`.

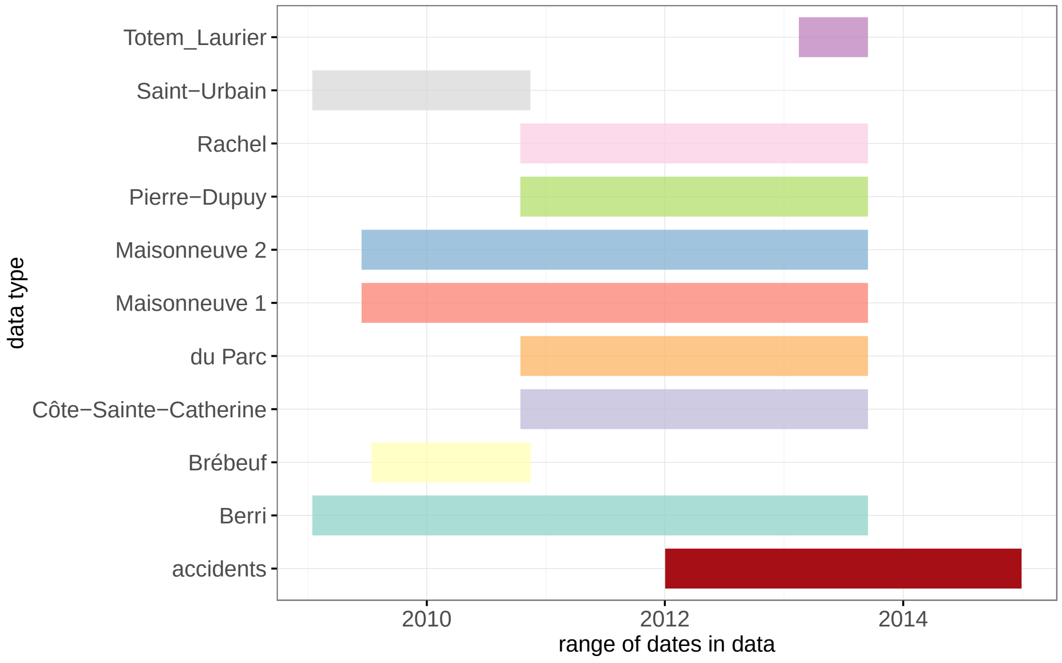
```r
  map.lim <- show.locations[, list(
    range.lat=range(lat),
    range.lon=range(lon)
  )]
  diff.vec <- sapply(map.lim, diff)
  diff.mat <- c(-1, 1) * matrix(diff.vec, 2, 2, byrow=TRUE)
```

```
scale.mat <- as.matrix(map.lim) + diff.mat
location.colors <-
  c("#8DD3C7", "#FFFFB3", "#BEBADA", "#FB8072", "#80B1D3", "#FDB462",
    "#B3DE69", "#FCCDE5", "#D9D9D9", "#BC80BD", "#CCEBC5", "#FFED6F")
names(location.colors) <- show.locations$location
counts.per.month.loc <- counts.per.month[
  show.locations, on=.(location)]
bike.paths <- data.table(montreal.bikes$path.locations)
some.paths <- bike.paths[
  scale.mat[1, "range.lat"] < lat &
    scale.mat[1, "range.lon"] < lon &
    lat < scale.mat[2, "range.lat"] &
    lon < scale.mat[2, "range.lon"]]
mtl.map <- ggplot()+
  theme_bw()+
  theme(
    panel.margin=grid::unit(0, "lines"),
    axis.line=element_blank(), axis.text=element_blank(),
    axis.ticks=element_blank(), axis.title=element_blank(),
    panel.background = element_blank(),
    panel.border = element_blank())+
  coord_equal(xlim=map.lim$range.lon, ylim=map.lim$range.lat)+
  scale_color_manual(values=location.colors)+
  scale_x_continuous(limits=scale.mat[, "range.lon"])+
  scale_y_continuous(limits=scale.mat[, "range.lat"])+
  geom_path(aes(
    lon, lat,
    tooltip=TYPE_VOIE,
    group=paste(feature.i, path.i)),
    color="grey",
    data=some.paths)+
  guides(color="none")+
  geom_text(aes(
    lon, lat,
    label=location),
    clickSelects="location",
    data=show.locations)
mtl.map
```



The plot below shows the time period that each counter was in operation. Note that we use `geom_tallrect` with `clickSelects` to select the month.

```r
location.ranges <- counts.per.month[0 < count, list(
  min=min(month.POSIXct),
  max=max(month.POSIXct)
), by=location]
accidents.range <- accidents.dt[, data.table(
  location="accidents",
  min=min(date.POSIXct),
  max=max(date.POSIXct))]
MonthSummary <- ggplot()+
  theme_bw()+
  theme_animint(width=450, height=250)+
  xlab("range of dates in data")+
  ylab("data type")+
  scale_color_manual(values=location.colors)+
  guides(color="none")+
  geom_segment(aes(
    min, location,
    xend=max, yend=location,
    color=location),
    clickSelects="location",
    data=location.ranges, alpha=3/4, size=10)+
  geom_segment(aes(
    min, location,
    xend=max, yend=location),
    color=severity.colors[["deaths"]],
    data=accidents.range,
    size=10)
MonthSummary
```



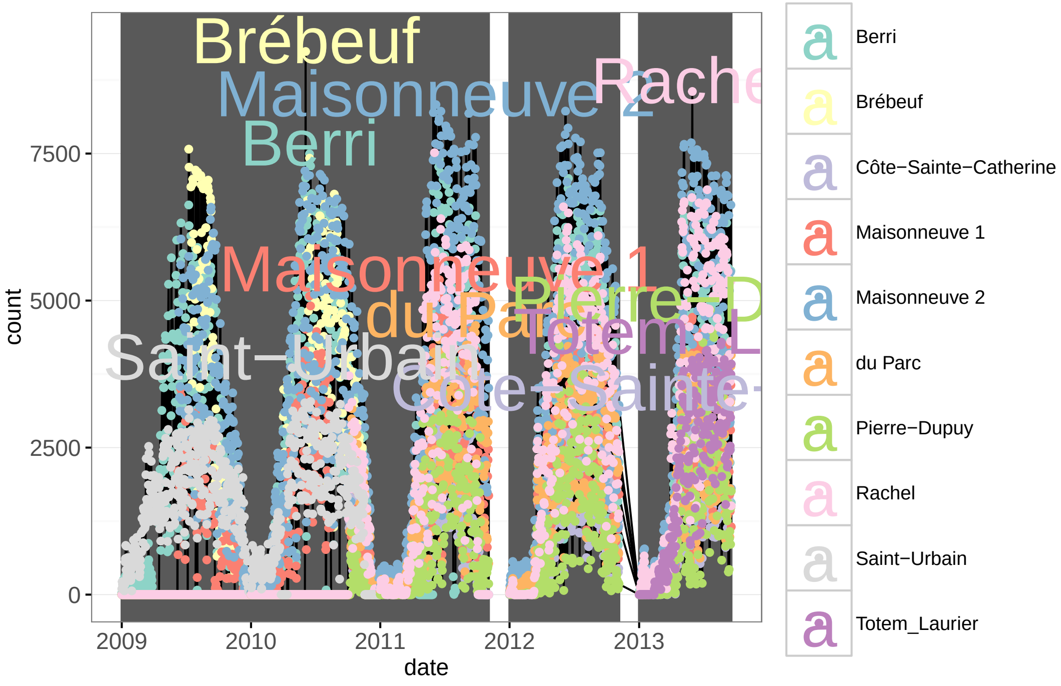
The plot below shows the bike counts at each location and day.

```
(dates <- counts.dt[, list(
  min.date = date-one.day/2,
  max.date = date+one.day/2,
  locations=sum(!is.na(count))
), by=list(date)][0 < locations])
```

```
                      date            min.date            max.date locations
   1: 2009-01-01 05:00:00 2008-12-31 17:00:00 2009-01-01 17:00:00         9
   2: 2009-01-02 05:00:00 2009-01-01 17:00:00 2009-01-02 17:00:00         9
  ---
1607: 2013-09-17 04:00:00 2013-09-16 16:00:00 2013-09-17 16:00:00         8
1608: 2013-09-18 04:00:00 2013-09-17 16:00:00 2013-09-18 16:00:00         8
```

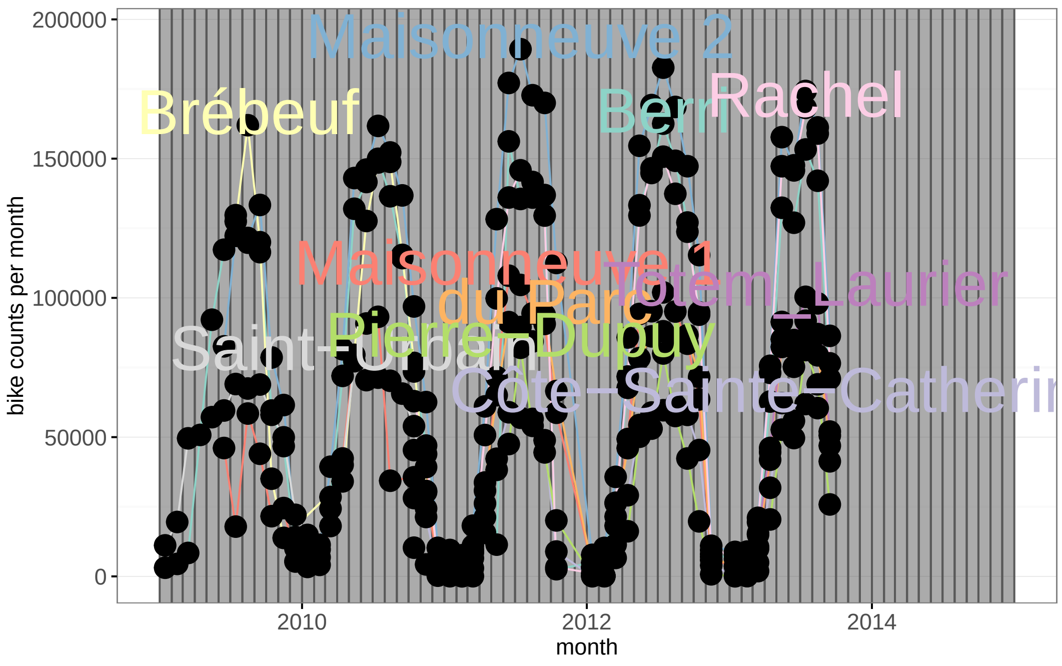
```
location.labels <- counts.dt[
, .SD[which.max(count)]
, by=list(location)]
TimeSeries <- ggplot()+
  theme_bw()+
  geom_tallrect(aes(
    xmin=date-one.day/2, xmax=date+one.day/2,
    clickSelects=date),
    data=dates, alpha=1/2)+
  geom_line(aes(
    date, count, group=location,
    showSelected=location,
    clickSelects=location),
    data=counts.dt)+
  scale_color_manual(values=location.colors)+
  geom_point(aes(
    date, count, color=location,
    showSelected=location,
    clickSelects=location),
    data=counts.dt)+
  geom_text(aes(
    date, count+200, color=location, label=location,
    showSelected=location,
    clickSelects=location),
    data=location.labels)
TimeSeries
```

```
Warning: Removed 407 rows containing missing values (geom_point).
```

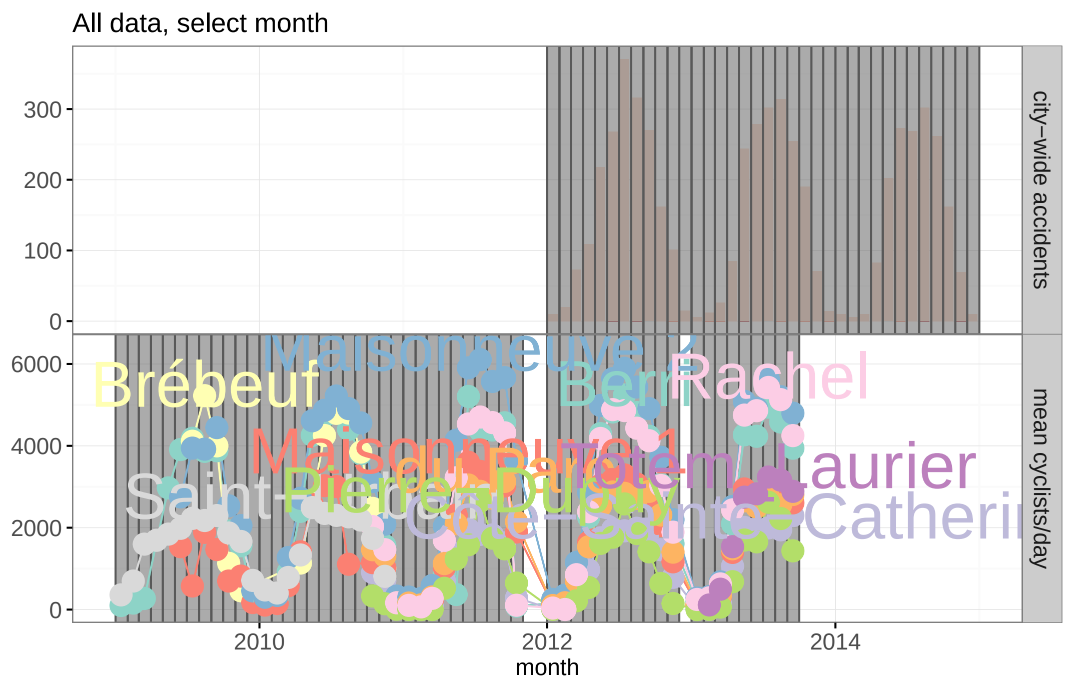


The plot below shows the same data but for each month.

```
MonthSeries <- ggplot()+
  guides(color="none", fill="none")+
  theme_bw()+
  geom_tallrect(aes(
    xmin=month01.POSIXct, xmax=next01.POSIXct),
    clickSelects="month",
    data=months,
    alpha=1/2)+
  geom_line(aes(
    month.POSIXct, count, group=location,
    color=location),
    showSelected="location",
    clickSelects="location",
    data=counts.per.month)+
  scale_color_manual(values=location.colors)+
  scale_fill_manual(values=location.colors)+
  xlab("month")+
  ylab("bike counts per month")+
  geom_point(aes(
    month.POSIXct, count, fill=location,
    tooltip=paste(
      count, "bikers counted at",
      location, "in", month)),
    showSelected="location",
    clickSelects="location",
    size=5,
    color="black",
    data=counts.per.month)+
  geom_text(aes(
    month.POSIXct, count+5000, color=location, label=location),
    showSelected="location",
    clickSelects="location",
    data=month.labels)
MonthSeries
```



```
counter.title <- "mean cyclists/day"
accidents.title <- "city-wide accidents"
person_people <- function(num, suffix)ifelse(
  num==0, "",
  sprintf(
    "%d %s %s",
    num,
    ifelse(num==1, "person", "people"),
    suffix))
deaths_severe_slight <- function(deaths, severe, slight)apply(cbind(
  ifelse(
    deaths==0, "",
    sprintf(
      "%d death%s",
      deaths,
      ifelse(deaths==1, "", "s"))),
  person_people(severe, "severely injured"),
  person_people(slight, "slightly injured")),
  1, function(x)paste(x[x!=""], collapse=", "))
MonthFacet <- ggplot()+
  ggtitle("All data, select month")+
  guides(color="none", fill="none")+
  theme_bw()+
  facet_grid(facet ~ ., scales="free")+
  theme(panel.margin=grid::unit(0, "lines"))+
  geom_tallrect(aes(
    xmin=month01.POSIXct, xmax=next01.POSIXct),
    clickSelects="month",
    data=data.table(
      city.wide.cyclists,
      facet=counter.title),
    alpha=1/2)+
  geom_line(aes(
    month.POSIXct, mean.per.day, group=location,
    color=location),
    showSelected="location",
    clickSelects="location",
    data=data.table(counts.per.month, facet=counter.title))+
  scale_color_manual(values=location.colors)+
  xlab("month")+
  ylab("")+
  geom_point(aes(
    month.POSIXct, mean.per.day, color=location,
    tooltip=paste(
      count, "cyclists counted at",
      location, "in",
      days, "days of", month,
      sprintf("(mean %d cyclists/day)", as.integer(mean.per.day)))),
    showSelected="location",
    clickSelects="location",
    size=5,
    fill="grey",
    data=data.table(counts.per.month, facet=counter.title))+
  geom_text(aes(
    month.POSIXct, mean.per.day+300, color=location, label=location),
    showSelected="location",
    clickSelects="location",
    data=data.table(month.labels, facet=counter.title))+
  scale_fill_manual(values=severity.colors)+
  geom_bar(aes(
    month.POSIXct, people,
    fill=severity),
    showSelected="severity",
    stat="identity",
    position="identity",
    color=NA,
    data=data.table(accidents.tall, facet=accidents.title))+
  geom_tallrect(aes(
    xmin=month01.POSIXct, xmax=next01.POSIXct,
    tooltip=paste(
      deaths_severe_slight(
        deaths,
        people.severely.injured,
        people.slightly.injured),
      "in", month)),
    clickSelects="month",
    alpha=0.5,
    data=data.table(accidents.per.month, facet=accidents.title))
MonthFacet
```



```r
(days.dt <- data.table(
  day.POSIXct=with(months, seq(
    min(month01.POSIXct),
    max(next01.POSIXct),
    by="day"))
)[
, day.of.the.week := strftime(day.POSIXct, "%a")
][])
```

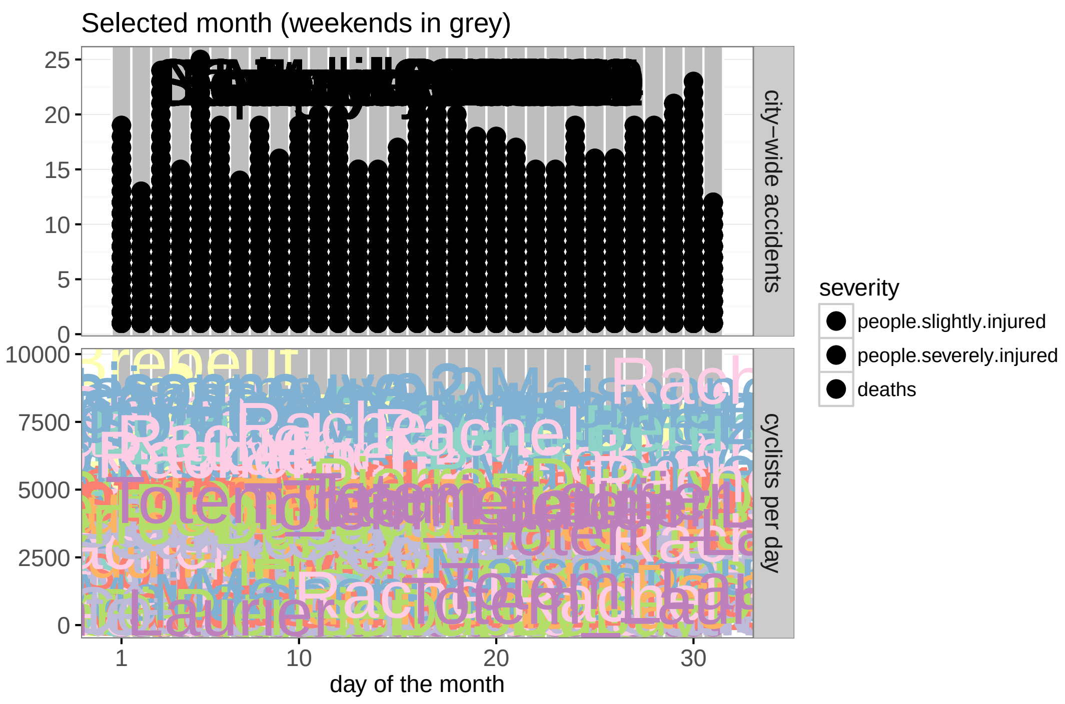
```
      day.POSIXct day.of.the.week
   1:  2009-01-01             Thu
   2:  2009-01-02             Fri
  ---
2191:  2014-12-31             Wed
2192:  2015-01-01             Thu
```

```r
## The following only works in locales with English days of the week.
(weekend.dt <- days.dt[
  day.of.the.week %in% c("Sat", "Sun")
][, let(
  month.text = strftime(day.POSIXct, "%B %Y"),
  day.of.the.month = as.integer(strftime(day.POSIXct, "%d"))
)][
, month := factor(month.text, month.levs)
][])
```

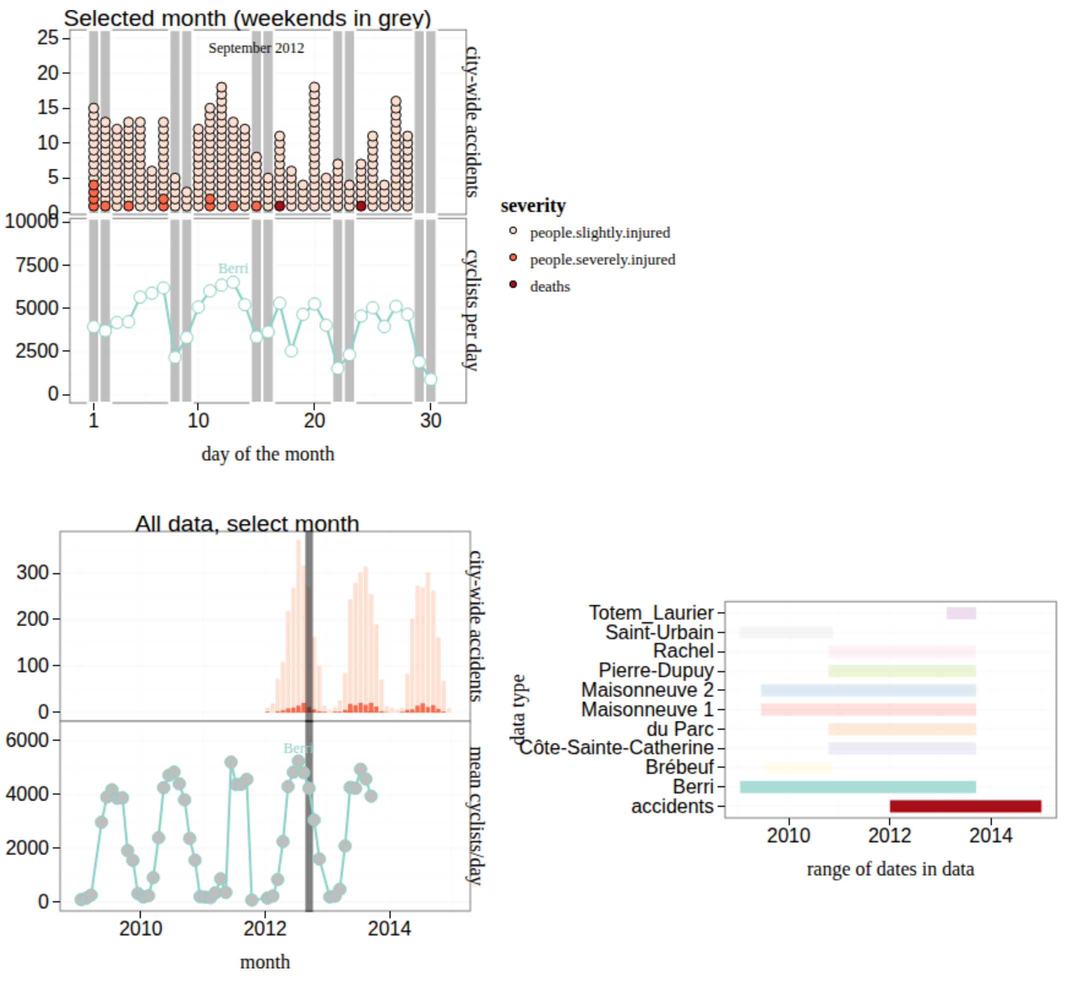
```
     day.POSIXct day.of.the.week    month.text day.of.the.month         month
  1:  2009-01-03             Sat  January 2009                3  January 2009
  2:  2009-01-04             Sun  January 2009                4  January 2009
 ---
625:  2014-12-27             Sat December 2014               27 December 2014
626:  2014-12-28             Sun December 2014               28 December 2014
```

```r
counter.title <- "cyclists per day"
DaysFacet <- ggplot()+
  ggtitle("Selected month (weekends in grey)")+
  theme_bw()+
  theme_animint(colspan=2, last_in_row=TRUE)+
  geom_tallrect(aes(
    xmin=day.of.the.month-0.5, xmax=day.of.the.month+0.5,
    key=paste(day.POSIXct)),
    showSelected="month",
    fill="grey",
    color="white",
    data=weekend.dt)+
  guides(color="none")+
  facet_grid(facet ~ ., scales="free")+
  geom_line(aes(
    day.of.the.month, count, group=location,
      key=location,
      color=location),
      showSelected=c("location", "month"),
      clickSelects="location",
      chunk_vars=c("month"),
      data=data.table(counts.dt, facet=counter.title))+
    scale_color_manual(values=location.colors)+
    ylab("")+
    geom_point(aes(
      day.of.the.month, count, color=location,
      key=paste(day.of.the.month, location),
      tooltip=paste(
        count, "cyclists counted at",
        location, "on",
        date)),
      showSelected=c("location", "month"),
      clickSelects="location",
      size=5,
      chunk_vars=c("month"),
      fill="white",
      data=data.table(counts.dt, facet=counter.title))+
    scale_fill_manual(
      values=severity.colors,
      breaks=rev(names(severity.colors)))+
    geom_text(aes(
      15, 23, label=month, key=1),
      showSelected="month",
      data=data.table(months, facet=accidents.title))+
    scale_x_continuous("day of the month", breaks=c(1, 10, 20, 30))+
    geom_text(aes(
      day.of.the.month, count+500, color=location, label=location,
      key=location),
      showSelected=c("location", "month"),
      clickSelects="location",
      data=data.table(day.labels, facet=counter.title))+
    geom_point(aes(
      day.of.the.month, accident.i,
      key=paste(date.str, accident.i),
      tooltip=paste(
        deaths_severe_slight(
          deaths,
          people.severely.injured,
          people.slightly.injured),
        "at",
        ifelse(is.na(street.number), "", street.number),
        street, "/", cross.street,
        date.str, time.str),
      fill=severity),
      showSelected="month",
    size=4,
    chunk_vars=c("month"),
    data=data.table(accidents.cumsum, facet=accidents.title))
DaysFacet
```

```
Warning: Removed 407 rows containing missing values (geom_point).
```



```
animint(
  DaysFacet,
  MonthFacet,
  MonthSummary,
  selector.types=list(severity="multiple"),
  duration=list(month=2000),
  first=list(
    location="Berri",
    month="September 2012"),
  time=list(variable="month", ms=5000))
```



9.4 Chapter summary and exercises

Exercises:

- Change location to a multiple selection variable.
- Add a plot for the map to the data viz.
- On the map, draw a circle for each location, with size that changes based on the `count` of the accidents in the currently selected `month`.
- On the `MonthSummary` plot, add a background rectangle that can be used to select the `month`.
- Remove the `MonthSummary` plot and add a similar visualization as a third panel in the `MonthFacet` plot.

Next, Chapter 10 explains how to visualize the K-Nearest-Neighbors machine learning model.

10

K-Nearest-Neighbors

In this chapter we will explore several data visualizations of the K-Nearest-Neighbors (KNN) classifier.

Chapter outline:

- We will start with the original static data visualization, re-designed as two ggplots rendered by animint. There is a plot of 10-fold cross-validation error, and a plot of the predictions of the 7-Nearest-Neighbors classifier.
- We propose a re-design that allows selecting the number of neighbors used for the model predictions.
- We propose a second re-design that allows selecting the number of folds used to compute the cross-validation error.

10.1 Original static figure

We start by reproducing a static version of Figure 13.4 from Elements of Statistical Learning by Hastie et al. That Figure consists of two plots:

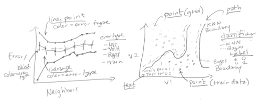


Figure 10.1: Static KNN viz

Left: mis-classification error curves, as a function of the number of neighbors.

- `geom_line` and `geom_point` for the error curves.
- `geom_linerange` for error bars of the validation error curve.
- `geom_hline` for the Bayes error.
- x = neighbors.
- y = percent error.
- color = error type.

Right: data and decision boundaries in the two-dimensional input feature space.

- `geom_point` for the data points.
- `geom_point` for the classification predictions on the grid in the background.
- `geom_path` for the decision boundaries.
- `geom_text` for the train/test/Bayes error rates.

10.1.1 Plot of mis-classification error curves

We begin by loading the data set.

```r
if(!requireNamespace("animint2data"))
  remotes::install_github("animint/animint2data")
```

```
Loading required namespace: animint2data
```

```r
data(ESL.mixture, package="animint2data")
str(ESL.mixture)
```

```
List of 8
 $ x       : num [1:200, 1:2] 2.5261 0.367 0.7682 0.6934 -0.0198 ...
 $ y       : num [1:200] 0 0 0 0 0 0 0 0 0 0 ...
 $ xnew    : 'matrix' num [1:6831, 1:2] -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2 -
1.9 -1.8 -1.7 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:6831] "1" "2" "3" "4" ...
  .. ..$ : chr [1:2] "x1" "x2"
 $ prob    : num [1:6831] 3.55e-05 3.05e-05 2.63e-05 2.27e-05 1.96e-05 ...
  ..- attr(*, ".Names")= chr [1:6831] "1" "2" "3" "4" ...
 $ marginal: num [1:6831] 6.65e-15 2.31e-14 7.62e-14 2.39e-13 7.15e-13 ...
  ..- attr(*, ".Names")= chr [1:6831] "1" "2" "3" "4" ...
 $ px1     : num [1:69] -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2 -1.9 -1.8 -1.7 ...
 $ px2     : num [1:99] -2 -1.95 -1.9 -1.85 -1.8 -1.75 -1.7 -1.65 -1.6 -
1.55 ...
 $ means   : num [1:20, 1:2] -0.2534 0.2667 2.0965 -0.0613 2.7035 ...
```

We will use the following components of this data set:

- `x`, the input matrix of the training data set (200 observations x 2 numeric features).
- `y`, the output vector of the training data set (200 class labels, either 0 or 1).
- `xnew`, the grid of points in the input space where we will show the classifier predictions (6831 grid points x 2 numeric features).
- `prob`, the true probability of class 1 at each of the grid points (6831 numeric values between 0 and 1).
- `px1`, the grid of points for the first input feature (69 numeric values between -2.6 and 4.2). These will be used to compute the Bayes decision boundary using the contourLines function.
- `px2`, the grid of points for the second input feature (99 numeric values between -2 and 2.9).
- `means`, the 20 centers of the normal distributions in the simulation model (20 centers x 2 input features).

First, we create a test set, following the example code from `help(ESL.mixture)`. Note that we use a `data.table` rather than a `data.frame` to store these big data, since `data.table` is often faster and more memory efficient for big data sets.

```r
library(MASS)
library(data.table)
set.seed(123)
```

```
centers <- c(
  sample(1:10, 5000, replace=TRUE),
  sample(11:20, 5000, replace=TRUE))
mix.test <- mvrnorm(10000, c(0,0), 0.2*diag(2))
test.points <- data.table(
  mix.test + ESL.mixture$means[centers,],
  label=factor(c(rep(0, 5000), rep(1, 5000))))
test.points
```

```
               V1        V2 label
    1:  2.0210959 1.3905124     0
    2:  2.7488414 1.0327241     0
   ---
 9999: -1.9089417 1.6135246     1
10000:  0.7678115 0.3154265     1
```

We then create a data table which includes all test points and grid points, which we will use in the test argument to the knn function.

```
pred.grid <- data.table(ESL.mixture$xnew, label=NA)
input.cols <- c("V1", "V2")
names(pred.grid)[1:2] <- input.cols
test.and.grid <- rbind(
  data.table(test.points, set="test"),
  data.table(pred.grid, set="grid"))
test.and.grid$fold <- NA
test.and.grid
```

```
             V1       V2 label  set fold
    1: 2.021096 1.390512     0 test   NA
    2: 2.748841 1.032724     0 test   NA
   ---
16830: 4.100000 2.900000  <NA> grid   NA
16831: 4.200000 2.900000  <NA> grid   NA
```

We randomly assign each observation of the training data set to one of ten folds.

```
n.folds <- 10
set.seed(2)
mixture <- with(ESL.mixture, data.table(x, label=factor(y)))
mixture$fold <- sample(rep(1:n.folds, l=nrow(mixture)))
mixture
```

```
               V1        V2 label fold
  1:  2.526092968 0.3210504     0    5
  2:  0.366954472 0.0314621     0    8
 ---
199:  0.008130556 2.2422639     1    4
200: -0.196246334 0.5514036     1    8
```

We define the following `OneFold` function, which divides the 200 observations into one train and one validation set. It then computes the predicted probability of the K-Nearest-Neighbors classifier for each of the data points in all sets (train, validation, test, and grid).

```
OneFold <- function(validation.fold){
  set <- ifelse(mixture$fold == validation.fold, "validation", "train")
  fold.data <- rbind(test.and.grid, data.table(mixture, set))
  fold.data$data.i <- 1:nrow(fold.data)
  only.train <- subset(fold.data, set == "train")
  data.by.neighbors <- list()
  for(neighbors in seq(1, 30, by=2)){
    if(interactive())cat(sprintf(
      "n.folds=%4d validation.fold=%d neighbors=%d\n",
      n.folds, validation.fold, neighbors))
    set.seed(1)
    pred.label <- class::knn( # random tie-breaking.
      only.train[, input.cols, with=FALSE],
      fold.data[, input.cols, with=FALSE],
      only.train$label,
      k=neighbors,
      prob=TRUE)
    prob.winning.class <- attr(pred.label, "prob")
    fold.data$probability <- ifelse(
      pred.label=="1", prob.winning.class, 1-prob.winning.class)
    fold.data[, pred.label := ifelse(0.5 < probability, "1", "0")]
    fold.data[, is.error := label != pred.label]
    fold.data[, prediction := ifelse(is.error, "error", "correct")]
    data.by.neighbors[[paste(neighbors)]] <-
      data.table(neighbors, fold.data)
  }#for(neighbors
  do.call(rbind, data.by.neighbors)
}#for(validation.fold
```

Below, we run the **`OneFold`** function in parallel using the **`future`** package. Note that validation folds 1:10 will be used to compute the validation set error. The validation fold 0 treats all 200 observations as a train set, and will be used for visualizing the learned decision boundaries of the K-Nearest-Neighbors classifier.

```
future::plan("multisession")
data.all.folds.list <- future.apply::future_lapply(
  0:n.folds, function(validation.fold){
    one.fold <- OneFold(validation.fold)
    data.table(validation.fold, one.fold)
  },
  future.seed = NULL)
data.all.folds <- do.call(rbind, data.all.folds.list)
```

The data table of predictions contains almost 3 million observations! When there are so many data, visualizing all of them at once is not practical or informative. Instead of visualizing them all at once, we will compute and plot summary statistics. In the code below we compute the mean and standard error of the mis-classification error for each model (over the 10 validation folds). This is an example of the summarize data table idiom which is generally useful for computing summary statistics for a single data table.

```
labeled.data <- data.all.folds[!is.na(label),]
error.stats <- labeled.data[, list(
  error.prop=mean(is.error)
  ), by=.(set, validation.fold, neighbors)]
validation.error <- error.stats[set=="validation", list(
  mean=mean(error.prop),
  sd=sd(error.prop)/sqrt(.N)
  ), by=.(set, neighbors)]
validation.error
```

```
           set neighbors  mean         sd
 1: validation         1 0.240 0.01943651
 2: validation         3 0.165 0.02362908
---
14: validation        27 0.195 0.02034426
15: validation        29 0.205 0.02291288
```

Below we construct data tables for the Bayes error (which we know is 0.21 for the mixture example data), and the train/test error.

```
Bayes.error <- data.table(
  set="Bayes",
  validation.fold=NA,
  neighbors=NA,
  error.prop=0.21)
Bayes.error
```

```
     set validation.fold neighbors error.prop
1: Bayes              NA        NA       0.21
```

```
other.error <- error.stats[validation.fold==0,]
head(other.error)
```

```
     set validation.fold neighbors error.prop
1:  test               0         1     0.2938
2: train               0         1     0.0000
3:  test               0         3     0.2400
4: train               0         3     0.1300
5:  test               0         5     0.2273
6: train               0         5     0.1300
```

Below we construct a color palette from `dput(RColorBrewer::brewer.pal(Inf, "Set1"))`, and linetype palettes.

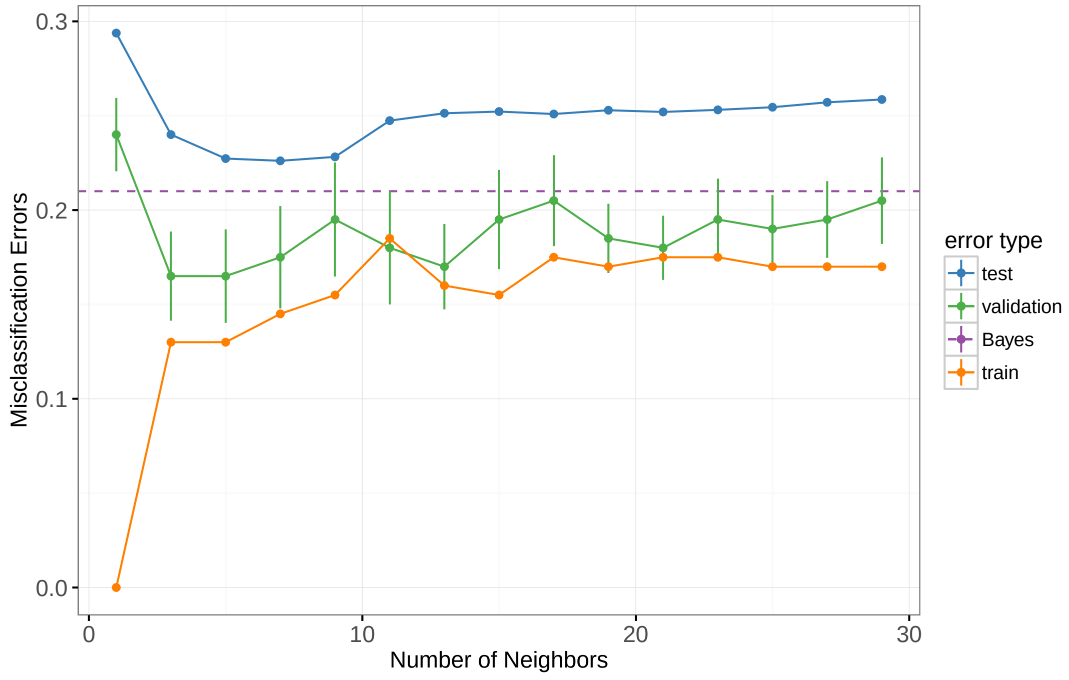
```
set.colors <- c(
  test="#377EB8", #blue
  validation="#4DAF4A",#green
```

```
    Bayes="#984EA3",#purple
    train="#FF7F00")#orange
  classifier.linetypes <- c(
    Bayes="dashed",
    KNN="solid")
  set.linetypes <- set.colors
  set.linetypes[] <- classifier.linetypes[["KNN"]]
  set.linetypes["Bayes"] <- classifier.linetypes[["Bayes"]]
  cbind(set.linetypes, set.colors)
```

```
           set.linetypes set.colors
test       "solid"       "#377EB8"
validation "solid"       "#4DAF4A"
Bayes      "dashed"      "#984EA3"
train      "solid"       "#FF7F00"
```

The code below reproduces the plot of the error curves from the original Figure.

```
  library(animint2)
  errorPlotStatic <- ggplot()+
    theme_bw()+
    theme_animint(width=300, rowspan=1)+
    geom_hline(aes(
      yintercept=error.prop, color=set, linetype=set),
      data=Bayes.error)+
    scale_color_manual(
      "error type", values=set.colors, breaks=names(set.colors))+
    scale_linetype_manual(
      "error type", values=set.linetypes, breaks=names(set.linetypes))+
    ylab("Misclassification Errors")+
    xlab("Number of Neighbors")+
    geom_linerange(aes(
      neighbors, ymin=mean-sd, ymax=mean+sd,
      color=set),
      data=validation.error)+
    geom_line(aes(
      neighbors, mean, linetype=set, color=set),
      data=validation.error)+
    geom_line(aes(
      neighbors, error.prop, group=set, linetype=set, color=set),
      data=other.error)+
    geom_point(aes(
      neighbors, mean, color=set),
      data=validation.error)+
    geom_point(aes(
      neighbors, error.prop, color=set),
      data=other.error)
  errorPlotStatic
```



10.1.2 Plot of decision boundaries in the input feature space

For the static data visualization of the feature space, we show only the model with 7 neighbors.

```
show.neighbors <- 7
show.data <- data.all.folds[
  validation.fold==0 & neighbors==show.neighbors]
show.points <- show.data[set=="train"]
```

Next, we compute the Train, Test, and Bayes mis-classification error rates which we will show in the bottom left of the feature space plot.

```
text.height <- 0.25
text.V1.prop <- 0
text.V2.bottom <- -2
text.V1.error <- -2.6
(error.text <- rbind(
  Bayes.error,
  other.error[neighbors==show.neighbors]
)[
, V2.top := text.V2.bottom + text.height * (1:.N)
][
, V2.bottom := V2.top - text.height
][])
```

```
     set validation.fold neighbors error.prop V2.top V2.bottom
1: Bayes              NA        NA     0.2100  -1.75     -2.00
```

```
2:  test               0         7     0.2261  -1.50     -1.75
3: train               0         7     0.1450  -1.25     -1.50
```

We define the following function which we will use to compute the decision boundaries.

```
getBoundaryDT <- function(prob.vec){
  stopifnot(length(prob.vec) == 6831)
  several.paths <- with(ESL.mixture, contourLines(
    px1, px2,
    matrix(prob.vec, length(px1), length(px2)),
    levels=0.5))
  contour.list <- list()
  for(path.i in seq_along(several.paths)){
    contour.list[[path.i]] <- with(several.paths[[path.i]], data.table(
      path.i, V1=x, V2=y))
  }
  do.call(rbind, contour.list)
}
```

We use this function to compute the decision boundaries for the learned 7-Nearest-Neighbors classifier, and for the optimal Bayes classifier.

```
boundary.grid <- show.data[set=="grid"][
, label := pred.label]
pred.boundary <- getBoundaryDT(
  boundary.grid$probability
)[
, classifier := "KNN"
][]
(Bayes.boundary <- getBoundaryDT(
  ESL.mixture$prob
)[
, classifier := "Bayes"
][])
```

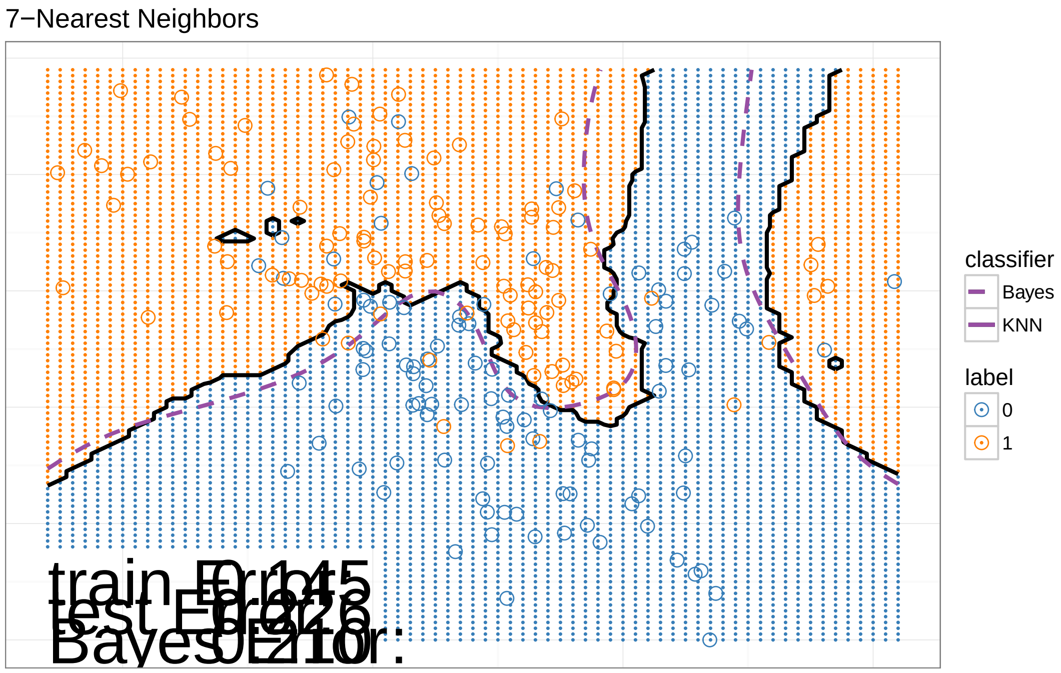
```
    path.i        V1        V2 classifier
  1:      1 -2.600000 -0.528615      Bayes
  2:      1 -2.557084 -0.500000      Bayes
 ---
249:      2  3.022480  2.850000      Bayes
250:      2  3.028586  2.900000      Bayes
```

Below, we consider only the grid points that do not overlap the text labels.

```
on.text <- function(V1, V2){
  V2 <= max(error.text$V2.top) & V1 <= text.V1.prop
}
show.grid <- boundary.grid[!on.text(V1, V2)]
```

The scatterplot below reproduces the 7-Nearest-Neighbors classifier of the original Figure.

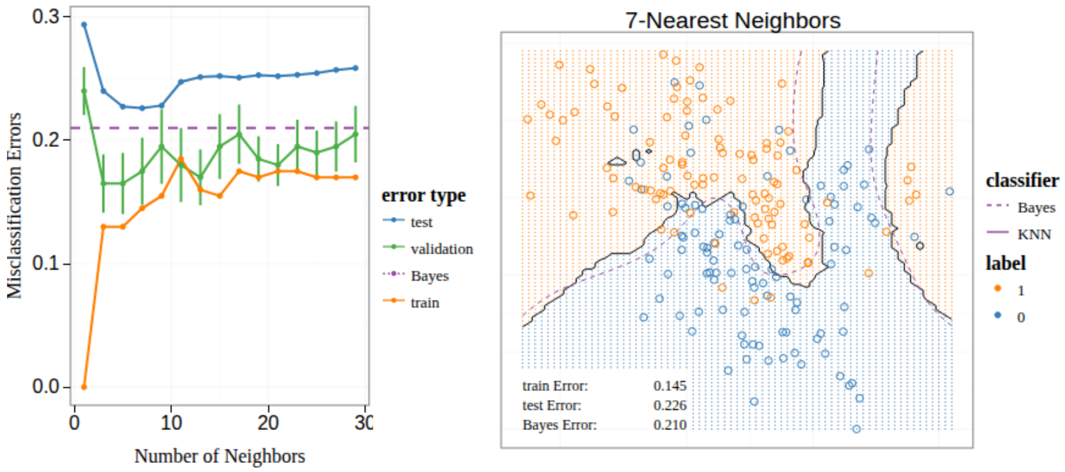
```r
label.colors <- c(
  "0"="#377EB8",
  "1"="#FF7F00")
scatterPlotStatic <- ggplot()+
  theme_bw()+
  theme(
    axis.text=element_blank(),
    axis.ticks=element_blank(),
    axis.title=element_blank())+
  ggtitle("7-Nearest Neighbors")+
  scale_color_manual(values=label.colors)+
  scale_linetype_manual(values=classifier.linetypes)+
  geom_point(aes(
    V1, V2, color=label),
    size=0.2,
    data=show.grid)+
  geom_path(aes(
    V1, V2, group=path.i, linetype=classifier),
    size=1,
    data=pred.boundary)+
  geom_path(aes(
    V1, V2, group=path.i, linetype=classifier),
    color=set.colors[["Bayes"]],
    size=1,
    data=Bayes.boundary)+
  geom_point(aes(
    V1, V2, color=label),
    fill=NA,
    size=3,
    shape=21,
    data=show.points)+
  geom_text(aes(
    text.V1.error, V2.bottom, label=paste(set, "Error:")),
    data=error.text,
    hjust=0)+
  geom_text(aes(
    text.V1.prop, V2.bottom, label=sprintf("%.3f", error.prop)),
    data=error.text,
    hjust=1)
scatterPlotStatic
```



10.1.3 Combined plots

Finally, we combine the two ggplots and render them as an animint.

```
animint(errorPlotStatic, scatterPlotStatic)
```



This data viz does have three interactive legends, but it is static in the sense that it displays only the model predictions for 7-Nearest Neighbors.

10.2 Select the number of neighbors using interactivity

In this section we propose an interactive re-design which allows the user to select K, the number of neighbors in the K-Nearest-Neighbors classifier.

Figure 10.2: Interactive KNN viz

10.2.1 Clickable error curves plot

We begin with a re-design of the error curves plot.

Note the following changes: * add a selector for the number of neighbors (`geom_tallrect`). * change the Bayes decision boundary from `geom_hline` with a legend entry, to a `geom_segment` with a text label. * add a linetype legend to distinguish error rates from the Bayes and KNN models. * change the error bars (`geom_linerange`) to error bands (`geom_ribbon`).

The only new data that we need to define are the endpoints of the segment that we will use to plot the Bayes decision boundary. Note that we also re-define the set "test" to emphasize the fact that the Bayes error is the best achievable error rate for test data.

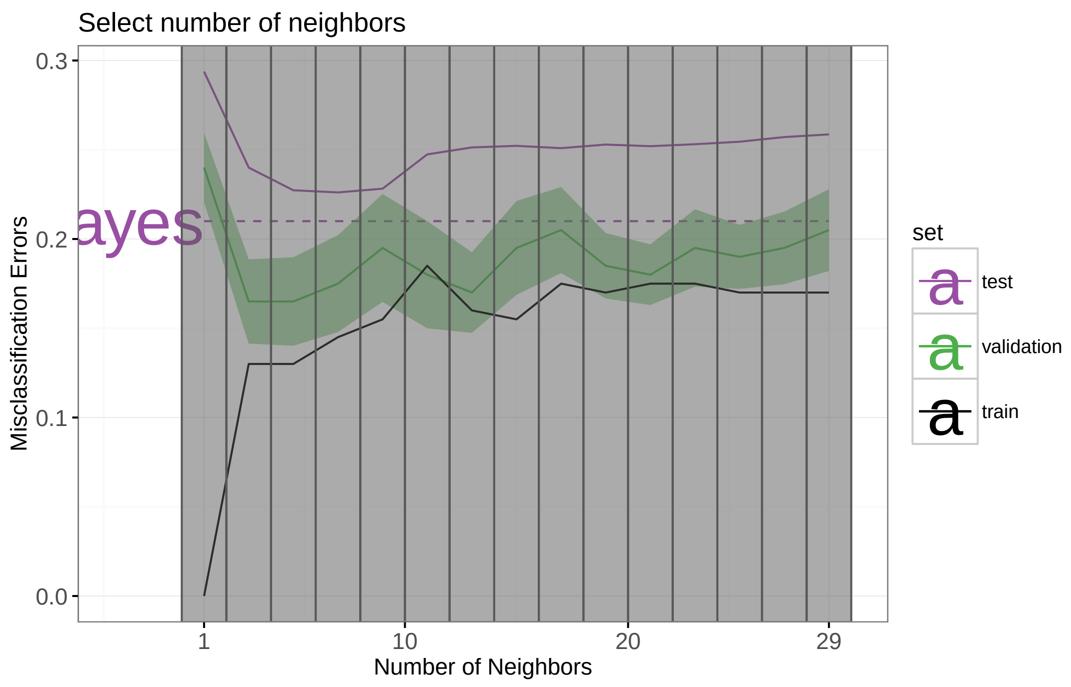
```
Bayes.segment <- data.table(
  Bayes.error,
  classifier="Bayes",
  min.neighbors=1,
  max.neighbors=29
)[, set := "test"]
```

We also add an error variable to the data tables that contain the prediction error of the K-Nearest-Neighbors models. This error variable will be used for the linetype legend.

```
validation.error$classifier <- "KNN"
other.error$classifier <- "KNN"
```

We re-define the plot of the error curves below. Note that * We use showSelected in `geom_text` and `geom_ribbon`, so that they will be hidden when the interactive legends are clicked. * We use clickSelects in `geom_tallrect`, to select the number of neighbors. Clickable geoms should be last (top layer) so that they are not obscured by non-clickable geoms (bottom layers).

```
set.colors <- c(
  test="#984EA3",#purple
  validation="#4DAF4A",#green
  Bayes="#984EA3",#purple
  train="black")
errorPlot <- ggplot()+
  ggtitle("Select number of neighbors")+
  theme_bw()+
  theme_animint(width=300)+
  geom_text(aes(
    min.neighbors, error.prop,
    color=set, label="Bayes"),
    showSelected="classifier",
    hjust=1,
    data=Bayes.segment)+
  geom_segment(aes(
    min.neighbors, error.prop,
    xend=max.neighbors, yend=error.prop,
    color=set,
    linetype=classifier),
    showSelected="classifier",
    data=Bayes.segment)+
  scale_color_manual(values=set.colors, breaks=names(set.colors))+
  scale_fill_manual(values=set.colors)+
  guides(fill="none", linetype="none")+
  scale_linetype_manual(values=classifier.linetypes)+
  ylab("Misclassification Errors")+
  scale_x_continuous(
    "Number of Neighbors",
    limits=c(-3, 30),
    breaks=c(1, 10, 20, 29))+
  geom_ribbon(aes(
    neighbors, ymin=mean-sd, ymax=mean+sd,
    fill=set),
    showSelected=c("classifier", "set"),
    alpha=0.5,
    color="transparent",
    data=validation.error)+
  geom_line(aes(
    neighbors, mean, color=set,
    linetype=classifier),
    showSelected="classifier",
    data=validation.error)+
  geom_line(aes(
    neighbors, error.prop, group=set, color=set,
    linetype=classifier),
    showSelected="classifier",
    data=other.error)+
  geom_tallrect(aes(
    xmin=neighbors-1, xmax=neighbors+1),
    clickSelects="neighbors",
    alpha=0.5,
    data=validation.error)
errorPlot
```



10.2.2 Feature space plot that shows the selected number of neighbors

Next, we focus on a re-design of the feature space plot. In the previous section we considered only the subset of data from the model with 7 neighbors. Our re-design includes the following changes: * We use neighbors as a showSelected variable. * We add a legend to show which training data points are mis-classified. * We use equal spaced coordinates so that visual distance (pixels) is the same as the Euclidean distance in the feature space.

```
show.data <- data.all.folds[validation.fold==0]
show.points <- show.data[set=="train"]
```

Below, we compute the predicted decision boundaries separately for each K-Nearest-Neighbors model.

```
boundary.grid <- show.data[set=="grid"][
, label := pred.label]
show.grid <- boundary.grid[!on.text(V1, V2)]
(pred.boundary <- boundary.grid[
, getBoundaryDT(probability), by=neighbors
][, classifier := "KNN"][])
```

```
     neighbors path.i        V1        V2 classifier
   1:         1      1 -2.60000 -1.025000        KNN
   2:         1      1 -2.55000 -1.000000        KNN
  ---
4491:        29      2  2.80099  1.900000        KNN
4492:        29      2  2.80000  1.897619        KNN
```

Instead of showing the number of neighbors in the plot title, below we create a `geom_text` element that will be updated based on the number of selected neighbors.

```
show.text <- show.grid[, .(
  V1=mean(range(V1)),
  V2=3.05
), by=neighbors]
```

Below we compute the position of the text in the bottom left, which we will use to display the error rate of the selected model.

```
other.error[, V2.bottom := rep(
  text.V2.bottom + text.height * 1:2, l=.N)]
```

Below we re-define the Bayes error data without a neighbors column, so that it appears in each showSelected subset.

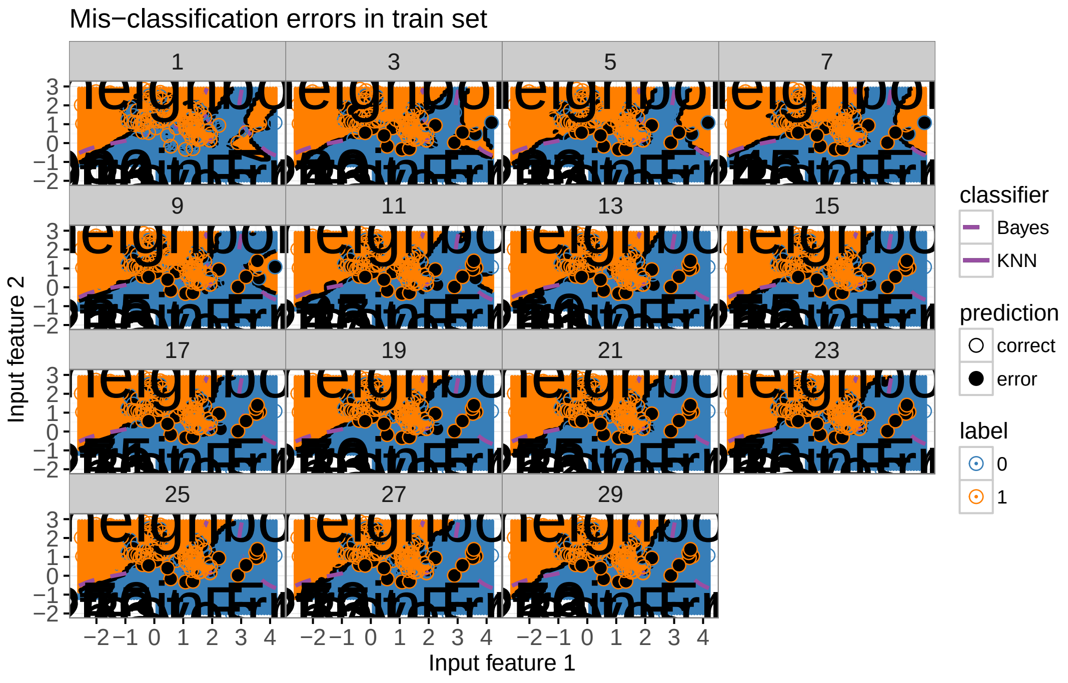
```
Bayes.error <- data.table(
  set="Bayes",
  error.prop=0.21)
```

Finally, we re-define the ggplot, using neighbors as a showSelected variable in the point, path, and text geoms.

```
scatterPlot <- ggplot()+
  ggtitle("Mis-classification errors in train set")+
  theme_bw()+
  theme_animint(width=450, colspan=1)+
  scale_x_continuous(
    "Input feature 1",
    breaks=seq(-2, 4))+
  ylab("Input feature 2")+
  scale_color_manual(values=label.colors)+
  scale_linetype_manual(values=classifier.linetypes)+
  geom_point(aes(
    V1, V2, color=label),
    showSelected="neighbors",
    size=0.2,
    data=show.grid)+
  geom_path(aes(
    V1, V2, group=path.i, linetype=classifier),
    showSelected="neighbors",
    size=1,
    data=pred.boundary)+
  geom_path(aes(
    V1, V2, group=path.i, linetype=classifier),
    color=set.colors[["test"]],
    size=1,
    data=Bayes.boundary)+
  geom_point(aes(
      V1, V2, color=label,
      fill=prediction),
      showSelected="neighbors",
      size=3,
      shape=21,
      data=show.points)+
    scale_fill_manual(values=c(error="black", correct="transparent"))+
    geom_text(aes(
      text.V1.error, text.V2.bottom, label=paste(set, "Error:")),
      data=Bayes.error,
      hjust=0)+
    geom_text(aes(
      text.V1.prop, text.V2.bottom, label=sprintf("%.3f", error.prop)),
      data=Bayes.error,
      hjust=1)+
    geom_text(aes(
      text.V1.error, V2.bottom, label=paste(set, "Error:")),
      showSelected="neighbors",
      data=other.error,
      hjust=0)+
    geom_text(aes(
      text.V1.prop, V2.bottom, label=sprintf("%.3f", error.prop)),
      showSelected="neighbors",
      data=other.error,
      hjust=1)+
    geom_text(aes(
      V1, V2,
      label=paste0(
        neighbors,
        " nearest neighbor",
        ifelse(neighbors==1, "", "s"),
        " classifier")),
      showSelected="neighbors",
      data=show.text)
```

Before compiling the interactive data viz, we print a static ggplot with a facet for each value of neighbors.

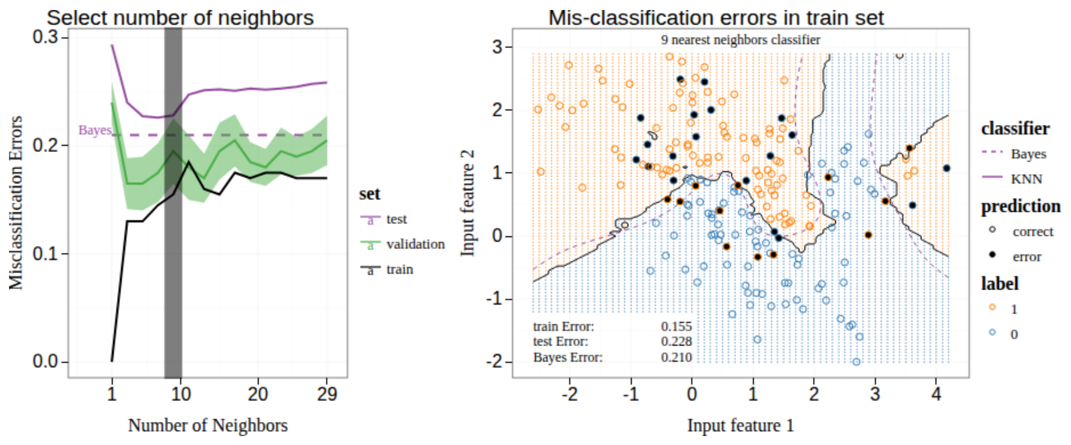
```
scatterPlot+
  facet_wrap("neighbors")+
  theme(panel.margin=grid::unit(0, "lines"))
```



10.2.3 Combined interactive data viz

Finally, we combine the two plots in a single data viz with neighbors as a selector variable.

```
animint(
  errorPlot,
  scatterPlot,
  first=list(neighbors=7),
  time=list(variable="neighbors", ms=3000))
```



Note that neighbors is used as a time variable, so animation shows the predictions of the different models.

10.3 Select the number of cross-validation folds using interactivity

In this section we discuss a second re-design which allows the user to select the number of folds used to compute the validation error curve.

The for loop below computes the validation error curve for several different values of `n.folds`.

```
error.by.folds <- list()
error.by.folds[["10"]] <- data.table(n.folds=10, validation.error)
for(n.folds in c(3, 5, 15)){
  set.seed(2)
  mixture <- with(ESL.mixture, data.table(x, label=factor(y)))
  mixture$fold <- sample(rep(1:n.folds, l=nrow(mixture)))
  only.validation.list <- future.apply::future_lapply(
    1:n.folds, function(validation.fold){
      one.fold <- OneFold(validation.fold)
      data.table(validation.fold, one.fold[set=="validation"])
    }, future.seed=NULL)
  only.validation <- do.call(rbind, only.validation.list)
  only.validation.error <- only.validation[, list(
    error.prop=mean(is.error)
  ), by=.(set, validation.fold, neighbors)]
  only.validation.stats <- only.validation.error[, list(
    mean=mean(error.prop),
    sd=sd(error.prop)/sqrt(.N)
  ), by=.(set, neighbors)]
  error.by.folds[[paste(n.folds)]] <-
    data.table(n.folds, only.validation.stats, classifier="KNN")
}
validation.error.several <- do.call(rbind, error.by.folds)
```

The code below computes the minimum of the error curve for each value of `n.folds`.

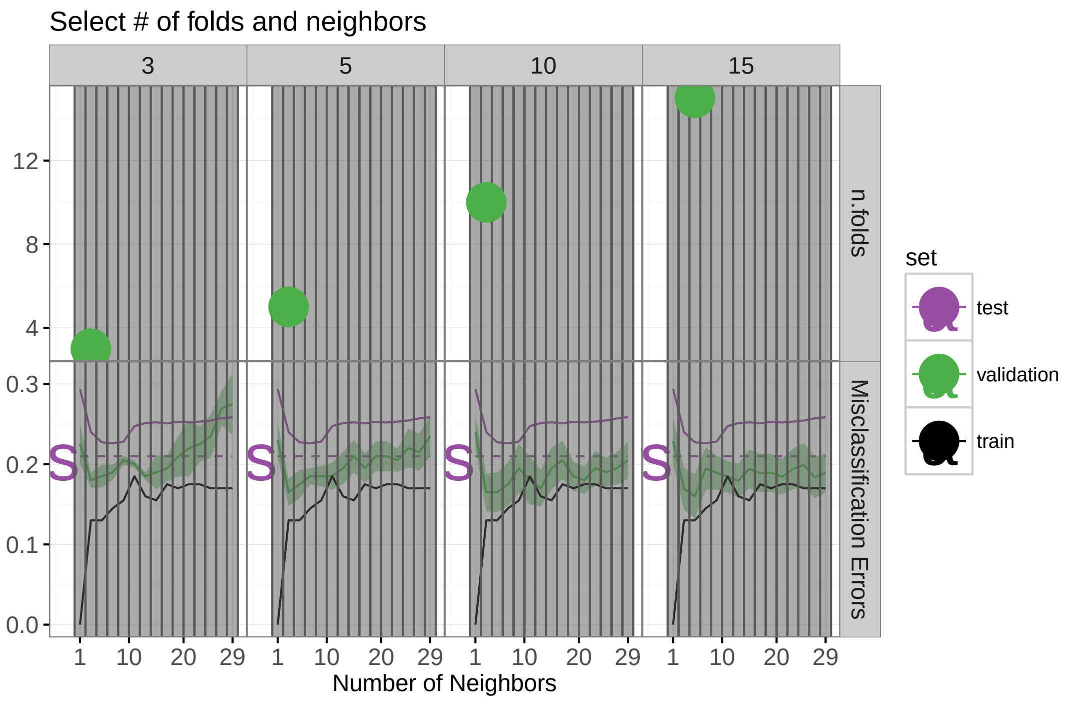
```
min.validation <- validation.error.several[
, .SD[which.min(mean)]
, by=n.folds]
```

The code below creates a new error curve plot with two facets.

```
facets <- function(df, facet)data.frame(df, facet=factor(
  facet, c("n.folds", "Misclassification Errors")))
errorPlotNew <- ggplot()+
  ggtitle("Select # of folds and neighbors")+
  theme_bw()+
  theme_animint(width=325)+
  theme(panel.margin=grid::unit(0, "cm"))+
  facet_grid(facet ~ ., scales="free")+
  geom_text(aes(
      min.neighbors, error.prop,
      color=set, label="Bayes"),
      showSelected="classifier",
      hjust=1,
      data=facets(Bayes.segment, "Misclassification Errors"))+
    geom_segment(aes(
      min.neighbors, error.prop,
      xend=max.neighbors, yend=error.prop,
      color=set,
      linetype=classifier),
      showSelected="classifier",
      data=facets(Bayes.segment, "Misclassification Errors"))+
    scale_color_manual(values=set.colors, breaks=names(set.colors))+
    scale_fill_manual(values=set.colors, breaks=names(set.colors))+
    guides(fill="none", linetype="none")+
    scale_linetype_manual(values=classifier.linetypes)+
    ylab("")+
    scale_x_continuous(
      "Number of Neighbors",
      limits=c(-3, 30),
      breaks=c(1, 10, 20, 29))+
    geom_ribbon(aes(
      neighbors, ymin=mean-sd, ymax=mean+sd,
      fill=set),
      showSelected=c("classifier", "set", "n.folds"),
      alpha=0.5,
      color="transparent",
      data=facets(validation.error.several, "Misclassification Errors"))+
    geom_line(aes(
      neighbors, mean, color=set,
      linetype=classifier),
      showSelected=c("classifier", "n.folds"),
      data=facets(validation.error.several, "Misclassification Errors"))+
    geom_line(aes(
      neighbors, error.prop, group=set, color=set,
      linetype=classifier),
      showSelected="classifier",
      data=facets(other.error, "Misclassification Errors"))+
    geom_tallrect(aes(
      xmin=neighbors-1, xmax=neighbors+1),
      clickSelects="neighbors",
      alpha=0.5,
      data=validation.error)+
    geom_point(aes(
      neighbors, n.folds, color=set),
      clickSelects="n.folds",
      size=9,
      data=facets(min.validation, "n.folds"))
```

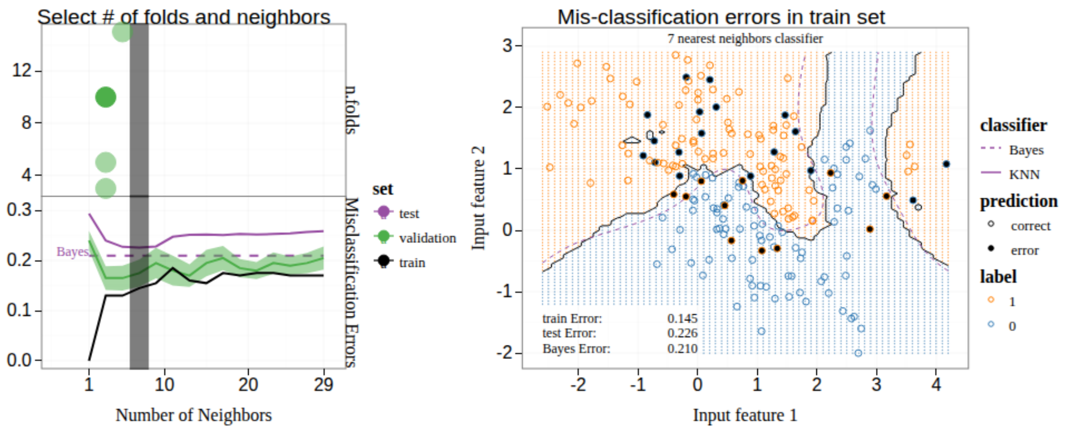
The code below previews the new error curve plot, adding an additional facet for the showSelected variable.

```
errorPlotNew+facet_grid(facet ~ n.folds, scales="free")
```



The code below creates an interactive data viz using the new error curve plot.

```
animint(
  errorPlotNew,
  scatterPlot,
  first=list(neighbors=7, n.folds=10))
```



10.4 Chapter summary and exercises

We showed how to add two interactive features to a data visualization of predictions of the K-Nearest-Neighbors model. We started with a static data visualization which only showed predictions of the 7-Nearest-Neighbors model. Then, we created an interactive re-design which allowed selecting K, the number of neighbors. We did another re-design which added a facet for selecting the number of cross-validation folds.

Exercises:

- Make it so that text error rates in the bottom left of the second plot are hidden after clicking the legend entries for Bayes, train, test. Hint: you can either use one `geom_text` with `showSelected=c(selectorNameColumn="selectorValueColumn")` (as explained in Chapter 14) or two `geom_text`, each with a different showSelected parameter.
- The probability column of the show.grid data table is the predicted probability of class 1. How would you re-design the visualization to show the predicted probability rather than the predicted class at each grid point? The main challenge is that probability is a numeric variable, but ggplot scales must be either continuous or discrete (not both). You could use a continuous fill scale, but then you would have to use a different scale to show the prediction variable.
- Add a new plot that shows the relative sizes of the train, validation, and test sets. Make sure that the plotted size of the validation and train sets change based on the selected value of `n.folds`.
- So far, the feature space plots only showed model predictions and errors for the entire train data set (validation.fold==0). Create a re-design which includes a new plot or facet for selecting validation.fold, and a facetted feature space plot (one facet for train set, one facet for validation set).

Next, Chapter 11 explains how to visualize the Lasso, a machine learning model.

11 Lasso

This goal of this chapter is to create an interactive data visualization that explains the Lasso, a machine learning model for regularized linear regression.

Chapter outline:

- We begin with several static data visualizations of the lasso path.
- We then create an interactive version with a facet and plot showing subtrain/validation error and residuals.
- Finally we re-design the interactive data visualization with simplified legends and moving tallrects.

11.1 Static plots of the coefficient regularization path

We begin by loading the prostate cancer data set.

```
if(!requireNamespace("animint2data"))
  remotes::install_github("animint/animint2data")
```

```
Loading required namespace: animint2data
```

```
data(prostate, package="animint2data")
library(data.table)
print(prostate, topn=1, trunc.cols = TRUE)
```

```
        lcavol  lweight age       lbph svi       lcp gleason pgg45       lpsa
 1: -0.5798185 2.769459  50 -1.3862944   0 -1.386294       6     0 -
0.4307829
---
97:  3.4719665 3.974998  68  0.4382549   1  2.904165       7    20  5.5829322
1 variable not shown: [train]
```

The output above shows the first and last row of the data table. The `train` column indicates a pre-defined split in the data. In this visualization, we will study the regularization path of the Lasso, for which we need a hold-out set to learn the optimal degree of regularization. We will use the split set name `subtrain` for the data used to compute linear model coefficients, and `validation` for the data used for selecting the regularization parameter (by minimizing prediction error on this set).

```
prostate[
, set := ifelse(train, "subtrain", "validation")
][, table(set)]
```

```
set
  subtrain validation
        67         30
```

We construct subtrain inputs `x` and outputs `y` using the code below.

```
input.cols <- c(
  "lcavol", "lweight", "age", "lbph",
  "svi", "lcp", "gleason", "pgg45")
prostate.inputs <- prostate[, ..input.cols]
is.subtrain <- prostate$set == "subtrain"
x <- as.matrix(prostate.inputs[is.subtrain])
head(x)
```

```
         lcavol  lweight age      lbph svi       lcp gleason pgg45
[1,] -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0
[2,] -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0
[3,] -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20
[4,] -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0
[5,]  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0
[6,] -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0
```

```
y <- prostate[is.subtrain, lpsa]
head(y)
```

```
[1] -0.4307829 -0.1625189 -0.1625189 -0.1625189  0.3715636  0.7654678
```

Below we fit the full path of lasso solutions using the `lars` package.

```
library(lars)
```

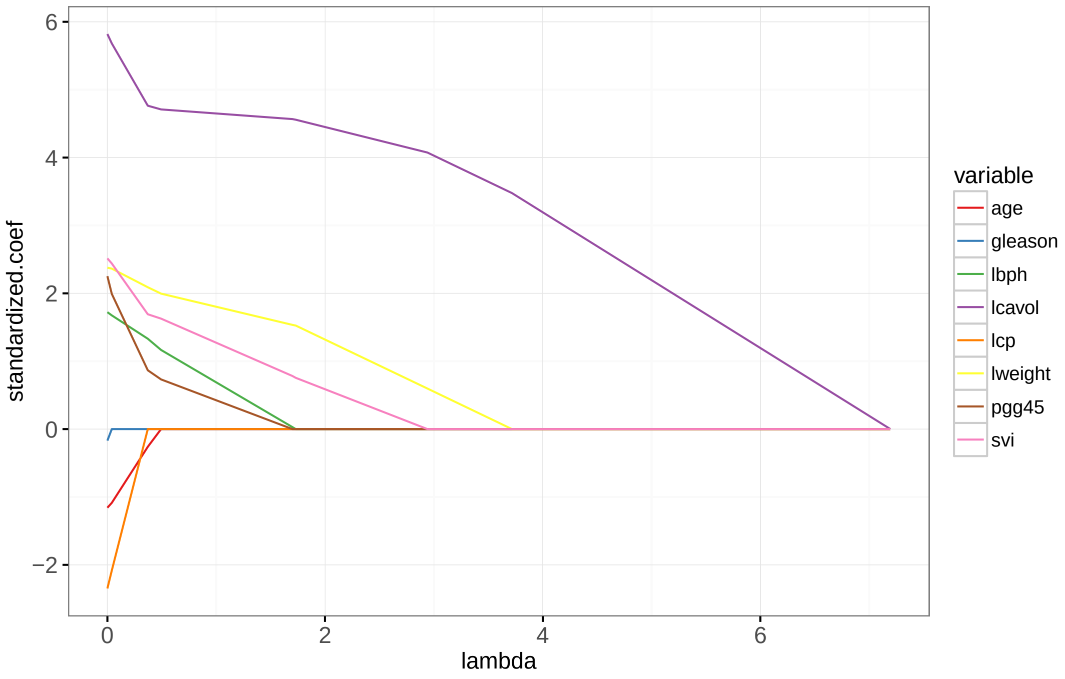
```
Loaded lars 1.3
```

```
fit <- lars(x,y,type="lasso")
fit$lambda
```

```
[1] 7.1939462 3.7172742 2.9403866 1.7305064 1.7002813 0.4933166 0.3711651
[8] 0.0403451
```

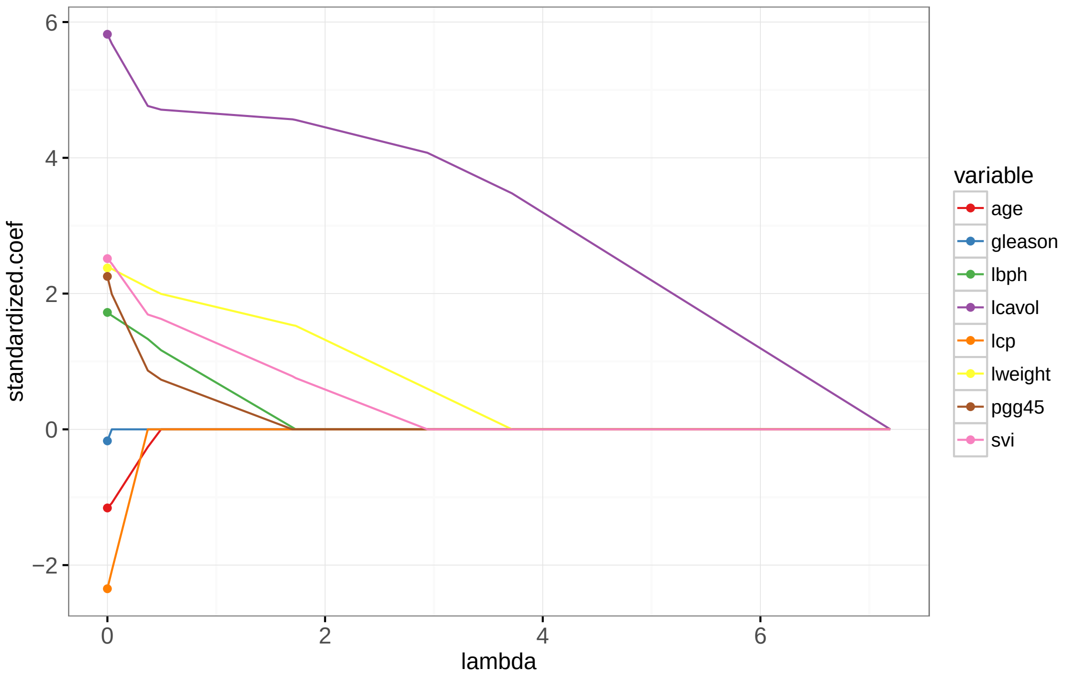
The path of `lambda` values are not evenly spaced.

```
pred.nox <- predict(fit, type="coef")
beta <- scale(pred.nox$coefficients, FALSE, 1/fit$normx)
arclength <- rowSums(abs(beta))
path.list <- list()
for(variable in colnames(beta)){
  standardized.coef <- beta[, variable]
  path.list[[variable]] <- data.table::data.table(
    step=seq_along(standardized.coef),
    lambda=c(fit$lambda, 0),
    variable,
    standardized.coef,
    fraction=pred.nox$fraction,
    arclength)
}
path <- do.call(rbind, path.list)
variable.colors <- c(
  "#E41A1C", "#377EB8", "#4DAF4A", "#984EA3", "#FF7F00", "#FFFF33",
  "#A65628", "#F781BF", "#999999")
library(animint2)
gg.lambda <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  scale_color_manual(values=variable.colors)+
  geom_line(aes(
    lambda, standardized.coef, color=variable, group=variable),
    data=path)
gg.lambda
```



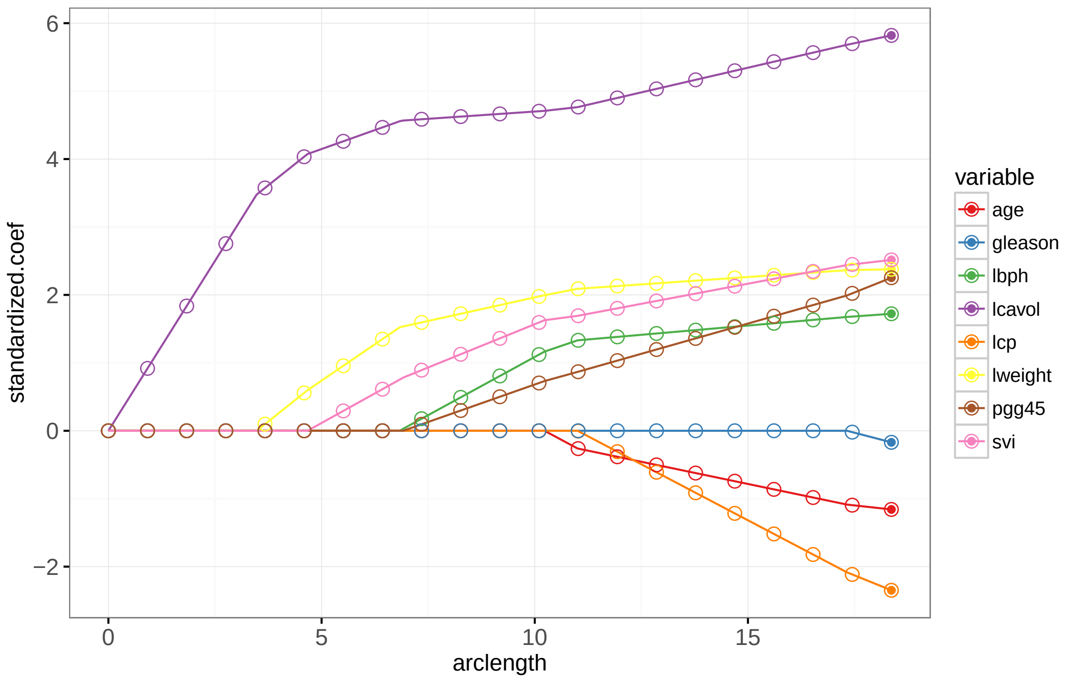
The plot above shows the entire lasso path, the optimal weights in the L1-regularized least squares regression problem, for every regularization parameter lambda. The path begins at the least squares solution, lambda=0 on the left. It ends at the completely regularized intercept-only model on the right. To see the equivalence with the ordinary least squares solution, we add dots in the plot below.

```
x.scaled <- with(fit, scale(x, meanx, normx))
lfit <- lm.fit(x.scaled, y)
lpoints <- data.table::data.table(
  variable=colnames(x),
  standardized.coef=lfit$coefficients,
  arclength=sum(abs(lfit$coefficients)))
gg.lambda+
  geom_point(aes(
    0, standardized.coef, color=variable),
    data=lpoints)
```



In the next plot below, we show the path as a function of L1 norm (arclength), with some more points on an evenly spaced grid that we will use later for animation.

```
fraction <- sort(unique(c(
  seq(0, 1, l=21))))
pred.fraction <- predict(
  fit, prostate.inputs,
  type="coef", mode="fraction", s=fraction)
coef.grid.list <- list()
coef.grid.mat <- scale(pred.fraction$coefficients, FALSE, 1/fit$normx)
for(fraction.i in seq_along(fraction)){
  standardized.coef <- coef.grid.mat[fraction.i,]
  coef.grid.list[[fraction.i]] <- data.table::data.table(
    fraction=fraction[[fraction.i]],
    variable=colnames(x),
    standardized.coef,
    arclength=sum(abs(standardized.coef)))
}
coef.grid <- do.call(rbind, coef.grid.list)
ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  scale_color_manual(values=variable.colors)+
  geom_line(aes(
    arclength, standardized.coef, color=variable, group=variable),
    data=path)+
  geom_point(aes(
    arclength, standardized.coef, color=variable),
    data=lpoints)+
  geom_point(aes(
    arclength, standardized.coef, color=variable),
    shape=21,
    fill=NA,
    size=3,
    data=coef.grid)
```



The plot above shows that the weights at the grid points are consistent with the lines that represent the entire path of solutions. The LARS algorithm quickly provides Lasso solutions for as many grid points as you like. More precisely, since the LARS only computes the change-points in the piecewise linear path, its time complexity only depends on the number of change-points (not the number of grid points).

11.2 Interactive visualization of the regularization path

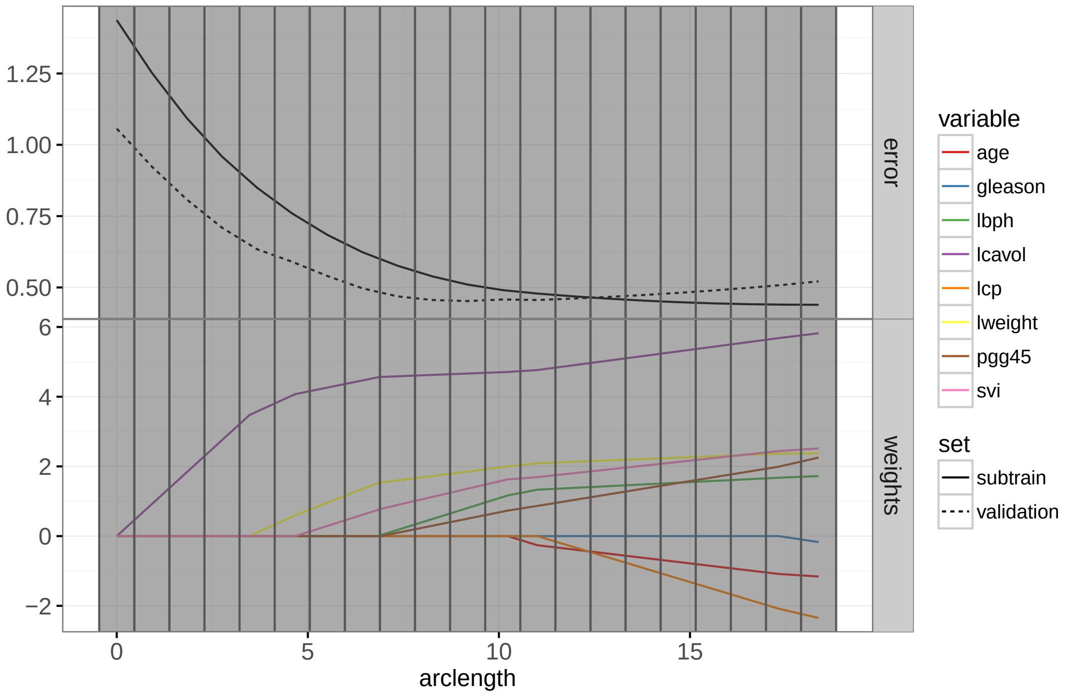
In this section, we combine the lasso weight path with the subtrain/validation error plot. First, we compute a data table with one row per model size and set.

```
pred.list <- predict(
  fit, prostate.inputs,
  mode="fraction", s=fraction)
residual.mat <- pred.list$fit - prostate$lpsa
squares.mat <- residual.mat * residual.mat
mean.error <- prostate[, data.table(
  fraction,
  mse=colMeans(squares.mat[.I, ]),
  arclength=rowSums(abs(coef.grid.mat))
), by=set]
print(mean.error, topn=2)
```

```
           set fraction       mse  arclength
 1:   subtrain     0.00 1.4370365  0.0000000
 2:   subtrain     0.05 1.2524384  0.9182159
---
41: validation     0.95 0.5090004 17.4461020
42: validation     1.00 0.5212740 18.3643178
```

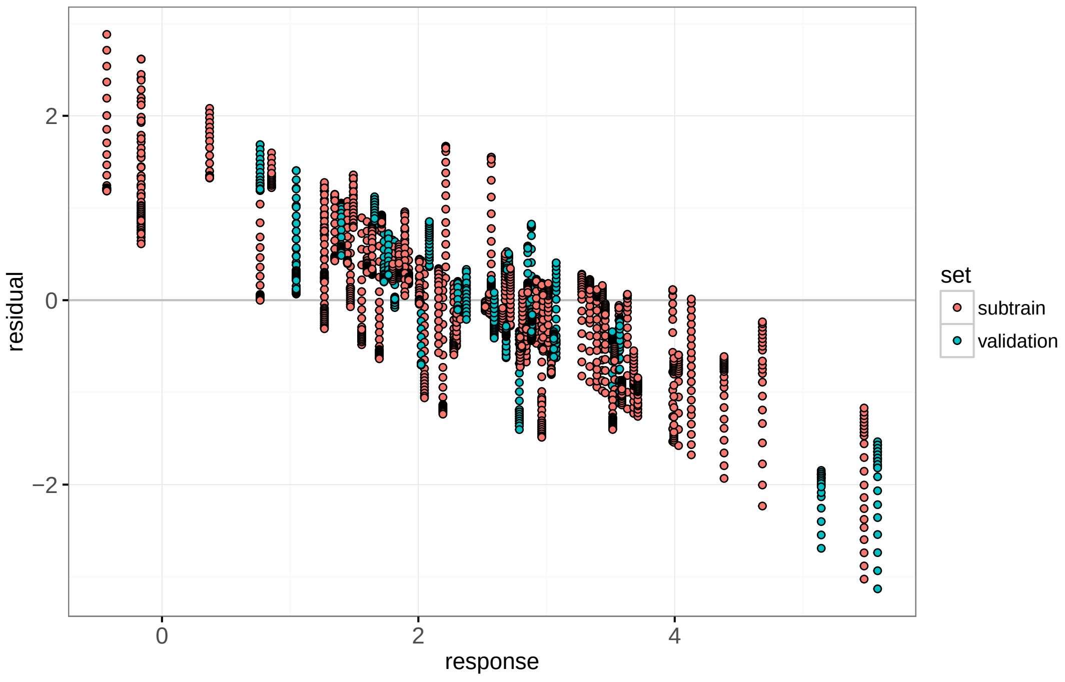
Note in the code above that we used the data table special symbol `.I`, which is set to the indices corresponding to the current value of `by=set`, used to compute the `mse` for each set. The table in the output above is used to plot the error curves below.

```
rect.width <- diff(mean.error$arclength[1:2])/2
addY <- function(dt, y){
  data.table::data.table(dt, y.var=factor(y, c("error", "weights")))
}
tallrect.dt <- coef.grid[variable==variable[1],]
gg.path <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  theme_animint(width=300, rowspan=1)+
  facet_grid(y.var ~ ., scales="free")+
  ylab("")+
  scale_color_manual(values=variable.colors)+
  geom_line(aes(
    arclength, standardized.coef, color=variable, group=variable),
    data=addY(path, "weights"))+
  geom_line(aes(
    arclength, mse, linetype=set, group=set),
    data=addY(mean.error, "error"))+
  geom_tallrect(aes(
    xmin=arclength-rect.width,
    xmax=arclength+rect.width),
    clickSelects="arclength",
    alpha=0.5,
    data=tallrect.dt)
gg.path
```



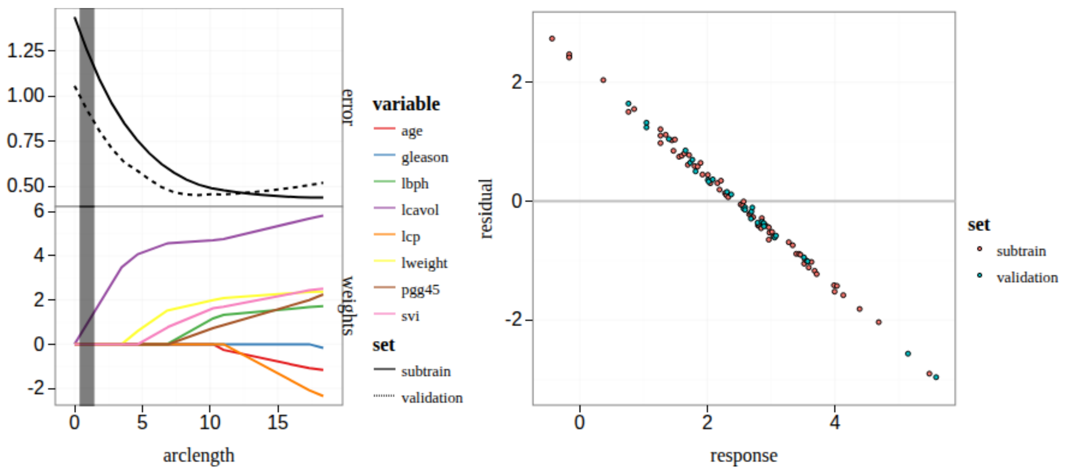
Finally, we add a plot of residuals versus actual values.

```
lasso.res.list <- list()
for(fraction.i in seq_along(fraction)){
  lasso.res.list[[fraction.i]] <- data.table::data.table(
    observation.i=1:nrow(prostate),
    fraction=fraction[[fraction.i]],
    residual=residual.mat[, fraction.i],
    response=prostate$lpsa,
    arclength=sum(abs(coef.grid.mat[fraction.i,])),
    set=prostate$set)
}
lasso.res <- do.call(rbind, lasso.res.list)
hline.dt <- data.table::data.table(residual=0)
gg.res <- ggplot()+
  theme_bw()+
  geom_hline(aes(
    yintercept=residual),
    data=hline.dt,
    color="grey")+
  geom_point(aes(
    response, residual, fill=set,
    key=observation.i),
    showSelected="arclength",
    shape=21,
    data=lasso.res)
gg.res
```



Below, we combine the ggplots above in a single animint below. Clicking the first plot changes the regularization parameter, and the residuals that are shown in the second plot.

```
animint(
  gg.path,
  gg.res,
  duration=list(arclength=2000),
  time=list(variable="arclength", ms=2000))
```

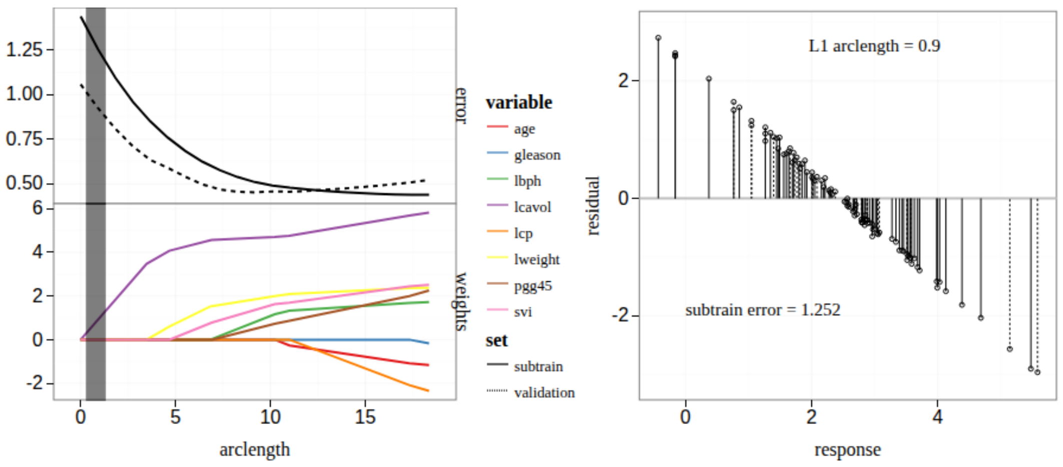


11.3 Re-design with moving tallrects

The re-design below has two changes. First, you may have noticed that there are two different set legends in the previous animint (linetype=set in the first path plot, and color=set in the second residual plot). It would be easier for the reader to decode if the set variable had just one mapping. So in the re-design below we replace the `geom_point` in the second plot with a `geom_segment` with `linetype=set`.

Second, we have replaced the single tallrect in the first plot with two tallrects. The first tallrect has `showSelected=arclength` and is used to show the selected arclength using a grey rectangle. Since we specify a `duration` for the `arclength` variable, and the same `key=1` value, we will observe a smooth transition of the selected grey tallrect. The second tallrect has `clickSelects=arclength` and so clicking it has the effect of changing the selected value of `arclength`. We specify a another data set with more rows, and use the named clickSelects/showSelected variables to indicate that `arclength` should also be used as a `showSelected` variable.

```
tallrect.show.list <- list()
for(a in tallrect.dt$arclength){
  is.selected <- tallrect.dt$arclength == a
  not.selected <- tallrect.dt[!is.selected]
  tallrect.show.list[[paste(a)]] <- data.table::data.table(
    not.selected, show.val=a, show.var="arclength")
}
tallrect.show <- do.call(rbind, tallrect.show.list)
animint(
  path=ggplot()+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "lines"))+
    facet_grid(y.var ~ ., scales="free")+
    ylab("")+
    scale_color_manual(values=variable.colors)+
    geom_line(aes(
      arclength, standardized.coef, color=variable, group=variable),
      data=addY(path, "weights"))+
    geom_line(aes(
      arclength, mse, linetype=set, group=set),
      data=addY(mean.error, "error"))+
    geom_tallrect(aes(
      xmin=arclength-rect.width,
      xmax=arclength+rect.width,
      key=1),
      showSelected="arclength",
      alpha=0.5,
      data=tallrect.dt)+
    geom_tallrect(aes(
      xmin=arclength-rect.width,
      xmax=arclength+rect.width,
      key=paste(arclength, show.val)),
      clickSelects="arclength",
      showSelected=c("show.var"="show.val"),
      alpha=0.5,
      data=tallrect.show),
  res=ggplot()+
    theme_bw()+
    geom_hline(aes(
      yintercept=residual),
      data=hline.dt,
      color="grey")+
    guides(linetype="none")+
    geom_point(aes(
      response, residual,
      key=observation.i),
      showSelected=c("set", "arclength"),
      shape=21,
      fill=NA,
      color="black",
      data=lasso.res)+
    geom_text(aes(
      3, 2.5, label=sprintf("L1 arclength = %.1f", arclength),
      key=1),
      size=15,
      showSelected="arclength",
      data=tallrect.dt)+
    geom_text(aes(
      0, ifelse(set=="subtrain", -2, -2.5),
      label=sprintf("%s error = %.3f", set, mse),
      key=1),
      size=15,
      showSelected=c("set", "arclength"),
      hjust=0,
        data=mean.error[set=="subtrain"])+
      geom_segment(aes(
        response, residual,
        xend=response, yend=0,
        linetype=set,
        key=observation.i),
        showSelected=c("set", "arclength"),
        size=1,
        data=lasso.res),
    duration=list(arclength=2000),
    time=list(variable="arclength", ms=2000))
```



11.4 Chapter summary and exercises

We created a visualization of the Lasso machine learning model, which simulataneously shows the regularization path and error curves. Interactivity was used to show details for different values of the regularization parameter.

Exercises:

- Re-make this data viz, including the same visual effect for the tallrects, using only one `geom_tallrect`. Hint: create another data set with `expand.grid(arclength.click=arclength, arclength.show=arclength)`, as in the definition of the `make_tallrect_or_widerect` function.
- Add another scatterplot that shows predicted values versus response, with a `geom_abline` in the background to indicate perfect prediction.
- How would the error curves look if other train/validation splits were chosen? Perform 4-fold cross-validation and add a plot that can be used to select test fold.

Next, Chapter 12 explains how to visualize the Support Vector Machine.

12

Support Vector Machines

This goal of this chapter is to create an interactive data visualization that explains the Support Vector Machine, a machine learning model for binary classification.

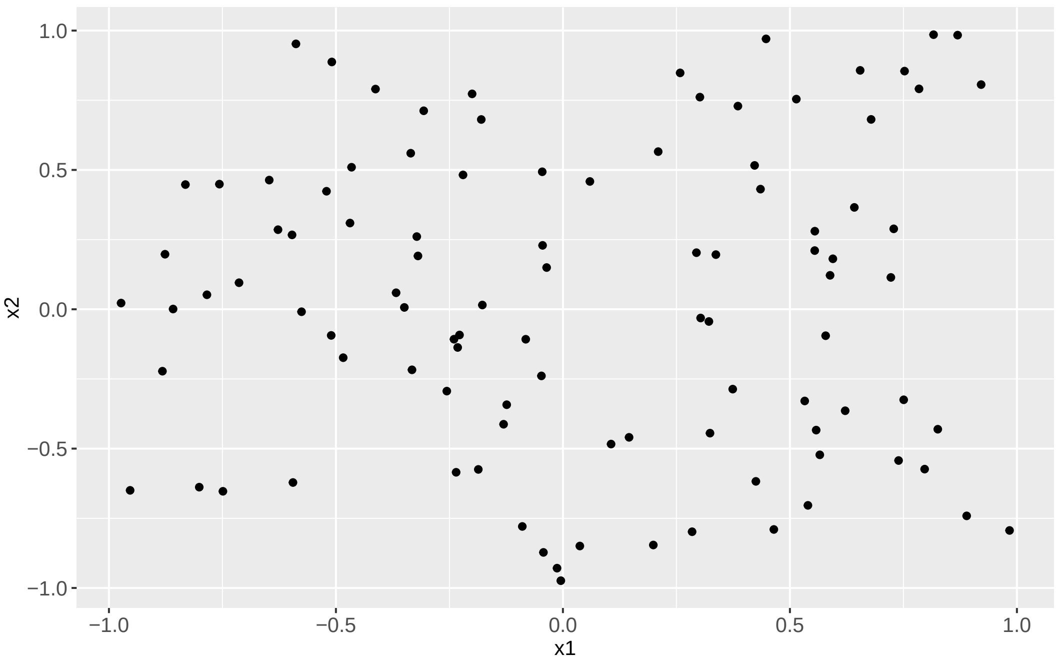
Chapter summary:

- We begin by simulating some data for binary classification in two dimensions, and making some static plots.
- In the second section, we make an interactive data visualization to show how the linear Support Vector Machine decision boundary changes as a function of the cost hyper-parameter.
- In the last section, we make an interactive data visualization to show how the decision boundary of the polynomial kernel Support Vector Machine changes as a function of the two hyper-parameters (cost and degree).

12.1 Generate and plot some data

We begin by generating two input features, `x1` and `x2`.

```r
library(data.table)
N <- 50
set.seed(1)
getInput <- function(){
  c(#rnorm(N, sd=0.3),
    runif(N, -1, 1),
    runif(N, -1, 1)
    )
}
data.dt <- data.table(
  x1=getInput(),
  x2=getInput())
library(animint2)
ggplot()+
  geom_point(aes(
    x1, x2),
    data=data.dt)
```

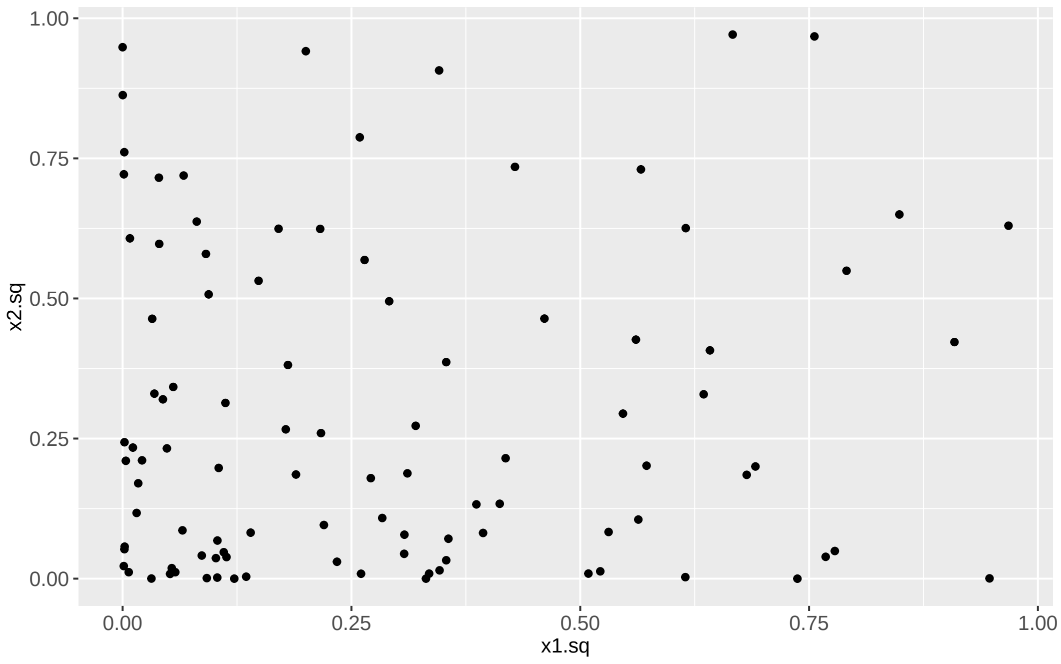


The plot below shows the same data, after computing two additional input features (the squares of the original two inputs).

```
data.dt[, let(
  x1.sq = x1^2,
  x2.sq = x2^2)][]
```

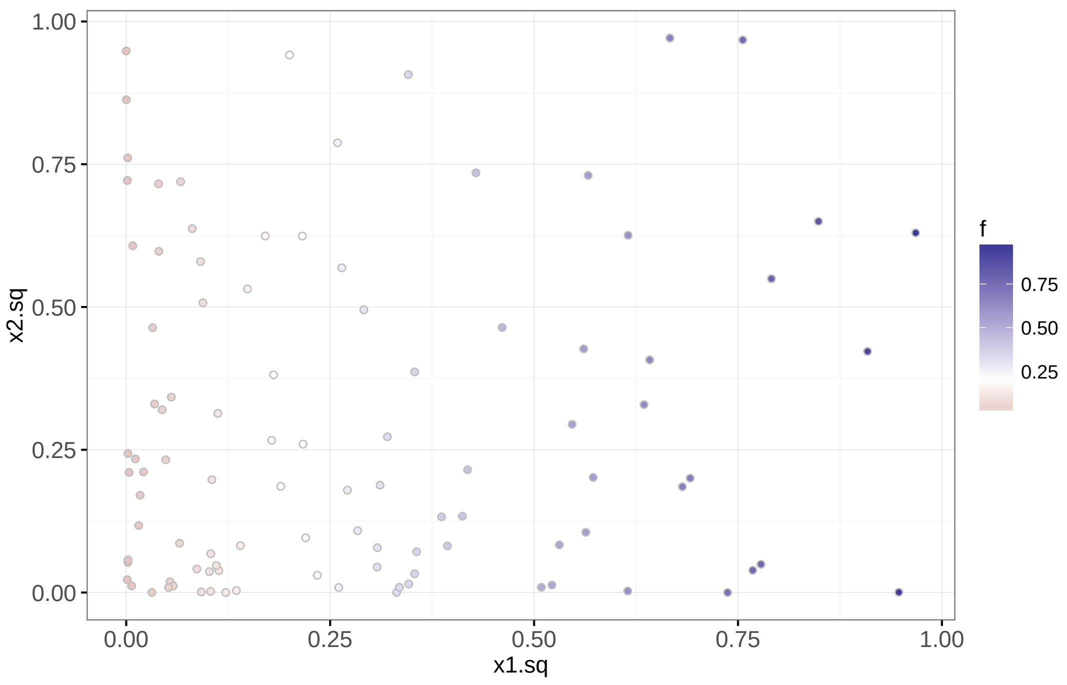
```
             x1         x2      x1.sq      x2.sq
  1: -0.4689827  0.3094479 0.21994475 0.09575798
  2: -0.2557522 -0.2936055 0.06540919 0.08620416
 ---
 99:  0.6217405 -0.3640726 0.38656123 0.13254888
100:  0.2098666  0.5657027 0.04404398 0.32001952
```

```
ggplot()+
  geom_point(aes(
    x1.sq, x2.sq),
    data=data.dt)
```



In our simulation, we assume that the output score `f` is a linear function of `x1.sq`, and ignores `x2.sq`. The plot below visualizes the output scores using the point fill aesthetic.

```
data.dt[, f := x1.sq]
true.decision.boundary <- 0.2
ggplot()+
  theme_bw()+
  scale_fill_gradient2(midpoint=true.decision.boundary)+
  geom_point(aes(
    x1.sq, x2.sq, fill=f),
    shape=21,
    color="grey",
    data=data.dt)
```

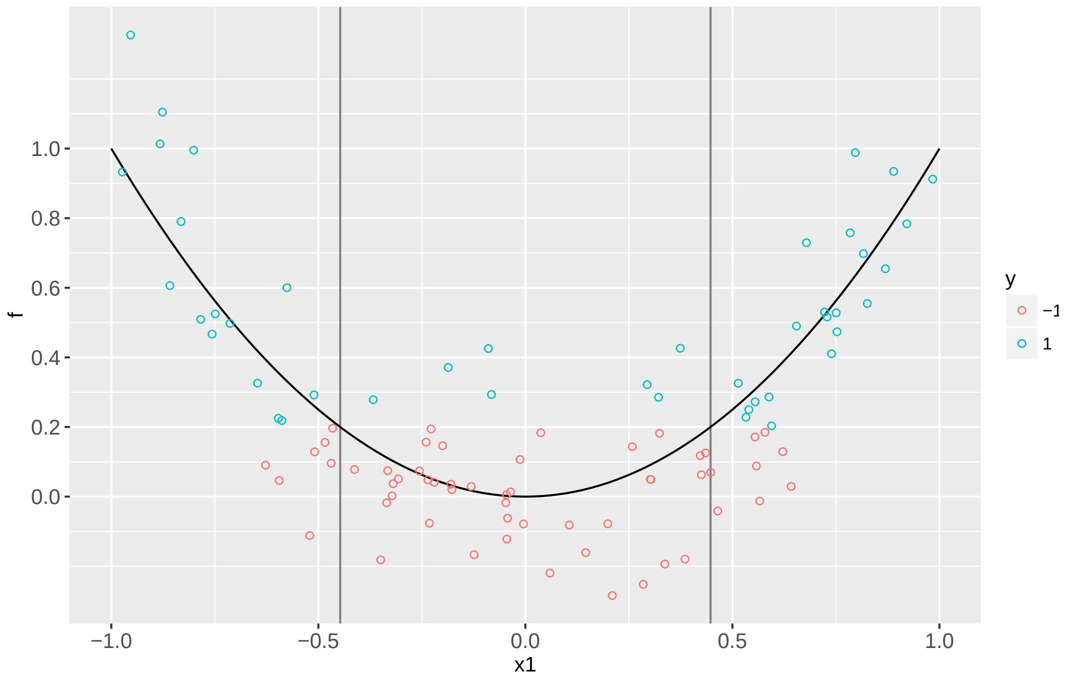


In particular, we assume that the label `y` is negative (-1) if `x1.sq + noise < threshold`, and positive (1) otherwise. The plot below visualizes the scores and labels, as a function of the input feature `x1`. It also shows the true score function in black. Of course, we would not be able to make this visualization with real data (only the labels are known in real data, not the scores).

```
data.dt[
, f.noise := f+rnorm(N, 0, 0.2)
][
, y.num := ifelse(f.noise<true.decision.boundary, -1, 1)
][
, y := factor(y.num)
]
table(data.dt$y)
```

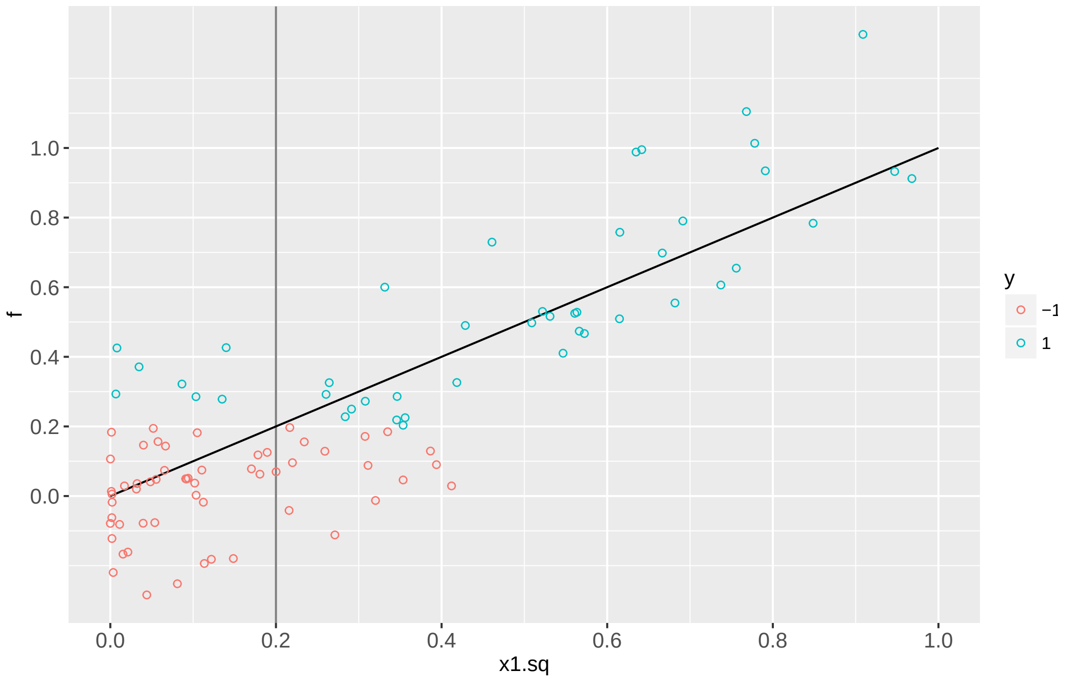
```
-1  1
56 44
```

```
scores <- data.table(x1=seq(-1, 1, l=101))[
, x1.sq := x1^2
][
, f := x1.sq ]
x1.boundaries <- data.table(
  boundary=c(1, -1)*sqrt(true.decision.boundary))
ggplot()+
  scale_y_continuous(breaks=seq(0, 1, by=0.2))+
  geom_vline(aes(
    xintercept=boundary),
    color="grey50",
    data=x1.boundaries)+
  geom_line(aes(
    x1, f),
    data=scores)+
  geom_point(aes(
    x1, f.noise, color=y),
    shape=21,
    fill=NA,
    data=data.dt)
```



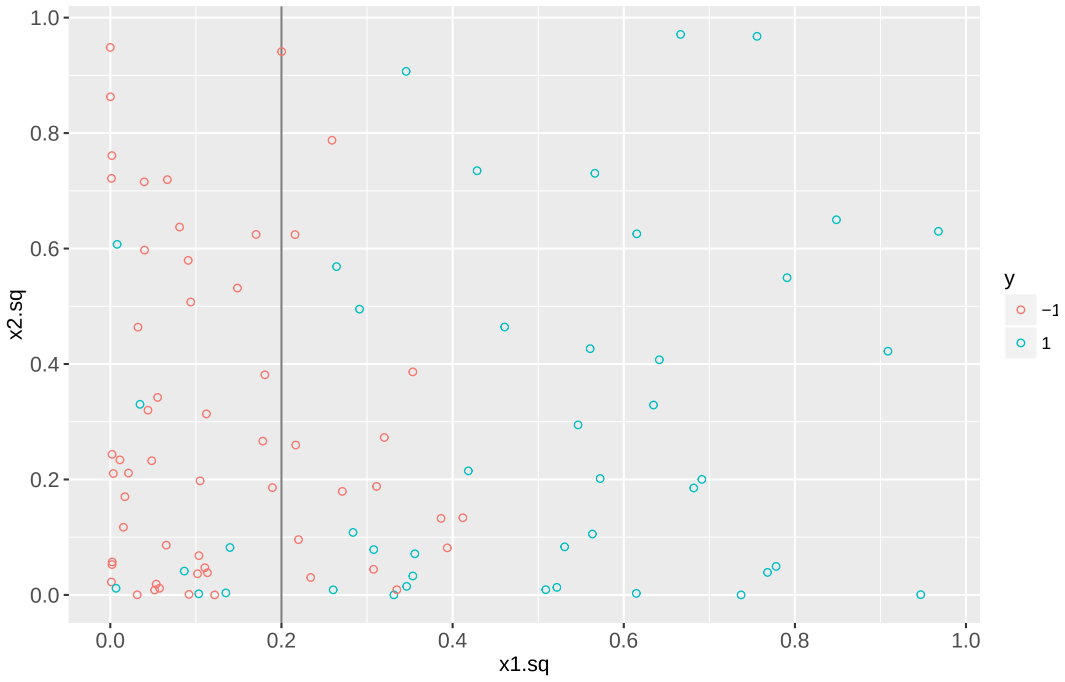
The plot below shows the scores and labels, as a function of the squared feature `x1.sq`. It is clear that the score function that we want to learn is linear in `x1.sq`.

```
x1sq.boundary <- data.table(boundary=true.decision.boundary)
ggplot()+
  scale_y_continuous(breaks=seq(0, 1, by=0.2))+
  scale_x_continuous(breaks=seq(0, 1, by=0.2))+
  geom_vline(aes(
    xintercept=boundary),
    color="grey50",
    data=x1sq.boundary)+
  geom_line(aes(x1.sq, f), data=scores)+
  geom_point(aes(
    x1.sq, f.noise, color=y),
    shape=21,
    fill=NA,
    data=data.dt)
```



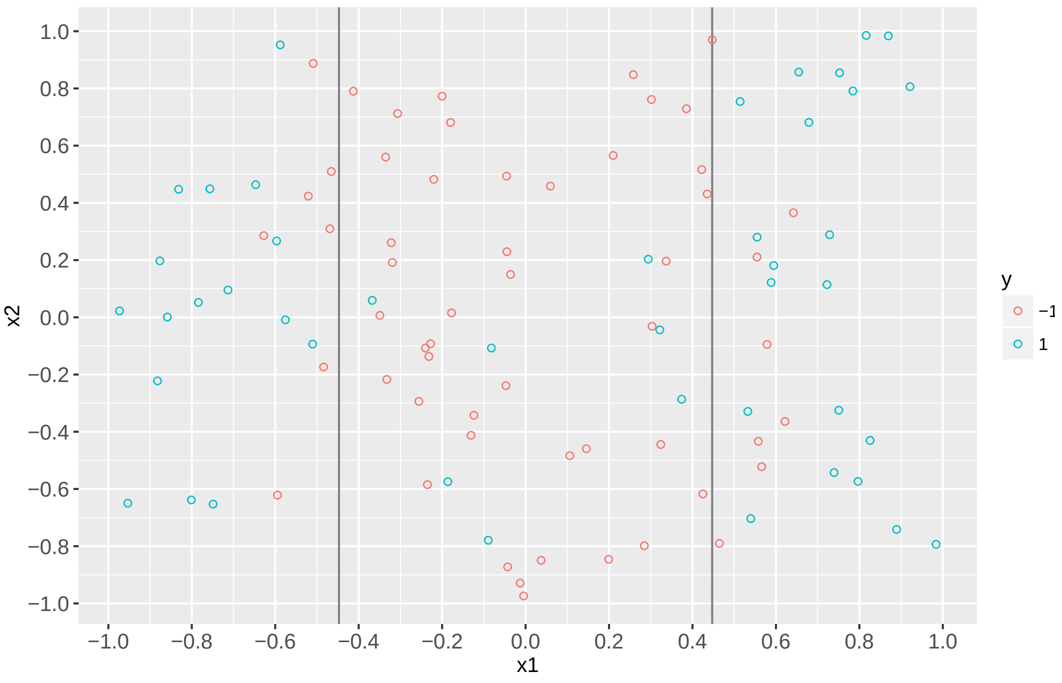
Next, we visualize the labels in the two-dimensional squared feature space. It is clear that the decision boundary is linear in this space.

```
ggplot()+
  scale_y_continuous(breaks=seq(0, 1, by=0.2))+
  scale_x_continuous(breaks=seq(0, 1, by=0.2))+
  geom_vline(aes(
    xintercept=boundary),
    color="grey50",
    data=x1sq.boundary)+
  geom_point(aes(
    x1.sq, x2.sq, color=y),
    shape=21,
    fill=NA,
    data=data.dt)
```



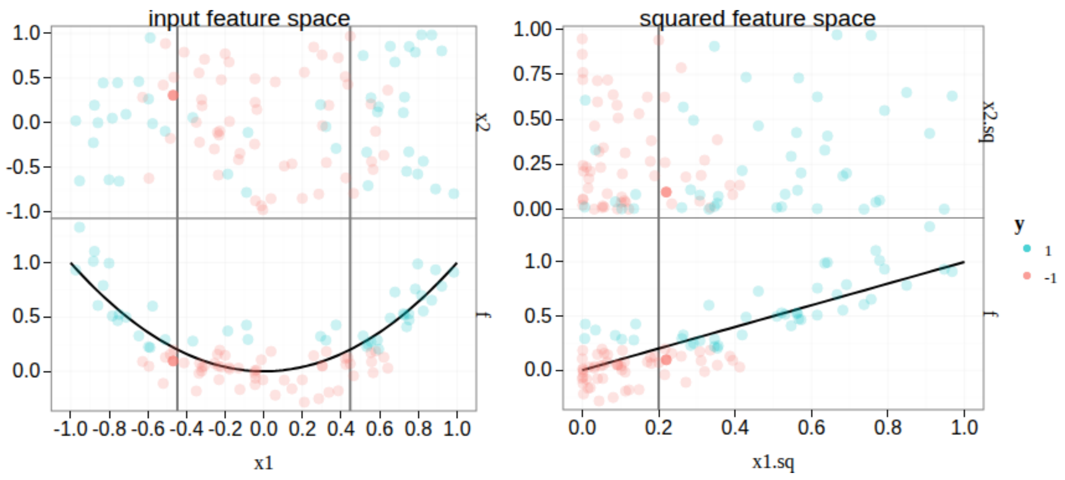
The plot below shows the input feature space (`x1` and `x2`). It is clear that the decision boundary is non-linear in `x1`.

```
ggplot()+
  scale_y_continuous(breaks=seq(-1, 1, by=0.2))+
  scale_x_continuous(breaks=seq(-1, 1, by=0.2))+
  geom_vline(aes(
    xintercept=boundary),
    color="grey50",
    data=x1.boundaries)+
  geom_point(aes(
    x1, x2, color=y),
    shape=21,
    fill=NA,
    data=data.dt)
```



The animint below uses `clickSelects` to show which points in the input and squared space correspond. We just need to create an `data.i` variable that has a unique ID for each data point.

```
data.dt[, data.i := 1:.N]
YVAR <- function(dt, y.var){
  dt$y.var <- factor(y.var, c("x2", "x2.sq", "f"))
  dt
}
animint(
  input=ggplot()+
    ggtitle("input feature space")+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "lines"))+
    facet_grid(y.var ~ ., scales="free")+
    scale_x_continuous(breaks=seq(-1, 1, by=0.2))+
    ylab("")+
    guides(color="none")+
    geom_vline(aes(
      xintercept=boundary),
      color="grey50",
      data=x1.boundaries)+
    geom_point(aes(
      x1, x2, color=y),
      clickSelects="data.i",
      size=4,
      alpha=0.7,
      data=YVAR(data.dt, "x2"))+
    geom_line(aes(
      x1, f),
      data=YVAR(scores, "f"))+
    geom_point(aes(
      x1, f.noise, color=y),
      clickSelects="data.i",
      size=4,
      alpha=0.7,
      data=YVAR(data.dt, "f")),
  square=ggplot()+
    ggtitle("squared feature space")+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "lines"))+
    facet_grid(y.var ~ ., scales="free")+
    ylab("")+
    scale_x_continuous(breaks=seq(0, 1, by=0.2))+
    geom_vline(aes(
      xintercept=boundary),
      color="grey50",
      data=x1sq.boundary)+
    geom_point(aes(
      x1.sq, x2.sq, color=y),
      clickSelects="data.i",
      size=4,
      alpha=0.7,
      data=YVAR(data.dt, "x2.sq"))+
    geom_line(aes(
      x1.sq, f),
      data=YVAR(scores, "f"))+
    geom_point(aes(
      x1.sq, f.noise, color=y),
      clickSelects="data.i",
      size=4,
      alpha=0.7,
      data=YVAR(data.dt, "f")))
```



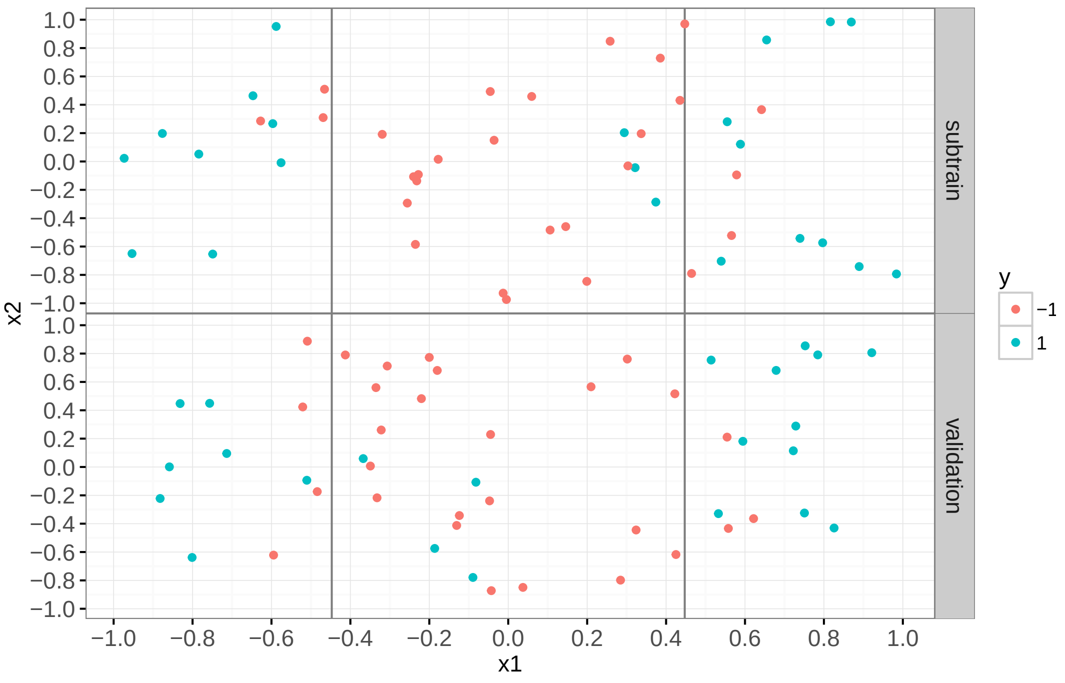
Note how we used two multi-panel plots with the addColumn then facet idiom, rather than creating four separate plots. This emphasizes the fact that some plots/facets have a common `x1` or `x1.sq` axis. Note that we also hid the color legend in the first plot, since it is sufficient to just have one color legend.

12.2 Linear SVM

```
train.i <- 1:N
data.dt[
, set := "validation"
][
  train.i, set := "subtrain"
]
table(data.dt$set)
```

```
subtrain validation
      50         50
```

```
subtrain.dt <- data.dt[set=="subtrain",]
ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(set ~ .)+
  geom_vline(aes(
    xintercept=boundary),
    color="grey50",
    data=x1.boundaries)+
  scale_y_continuous(breaks=seq(-1, 1, by=0.2))+
  scale_x_continuous(breaks=seq(-1, 1, by=0.2))+
  geom_point(aes(
    x1, x2, color=y),
    data=data.dt)
```



We begin by fitting a linear SVM to the train data in the squared feature space, and visualizing the true labels `y` along with the predicted labels `pred.y`.

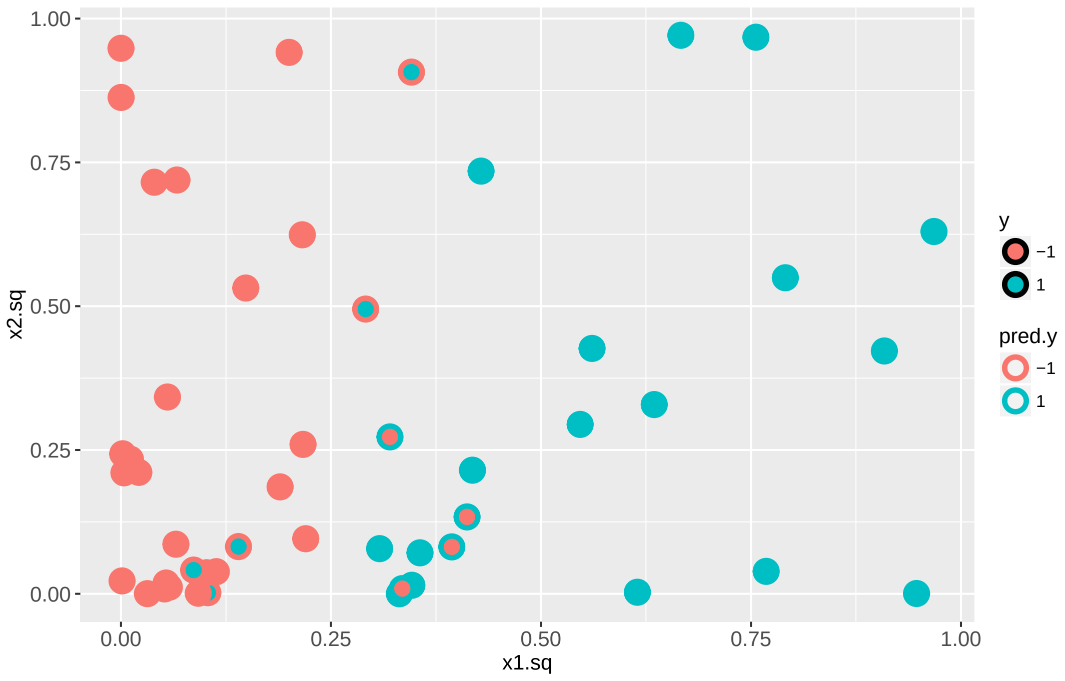
```
library(kernlab)
```

```
Attaching package: 'kernlab'

The following object is masked from 'package:animint2':

    alpha
```

```
squared.mat <- subtrain.dt[, cbind(x1.sq, x2.sq)]
y.vec <- subtrain.dt$y
fit <- ksvm(squared.mat, y.vec, kernel="vanilladot")
```

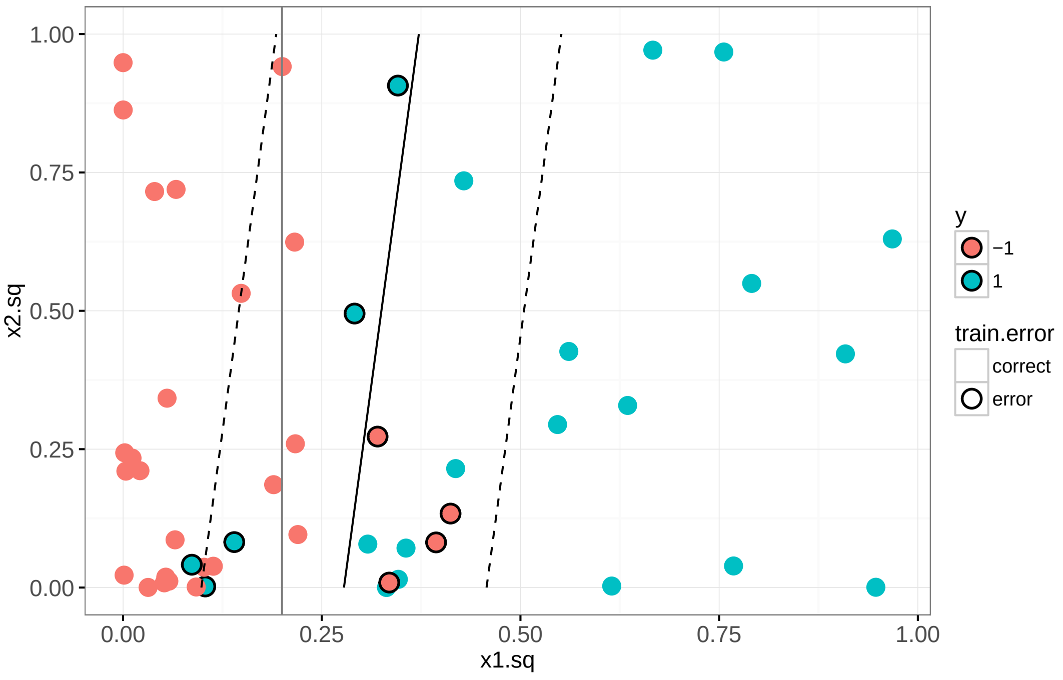
```
 Setting default kernel parameters
```

```
subtrain.dt$pred.y <- predict(fit)
ggplot()+
  geom_point(aes(
    x1.sq, x2.sq, color=pred.y, fill=y),
    shape=21,
    size=4,
    stroke=2,
    data=subtrain.dt)
```



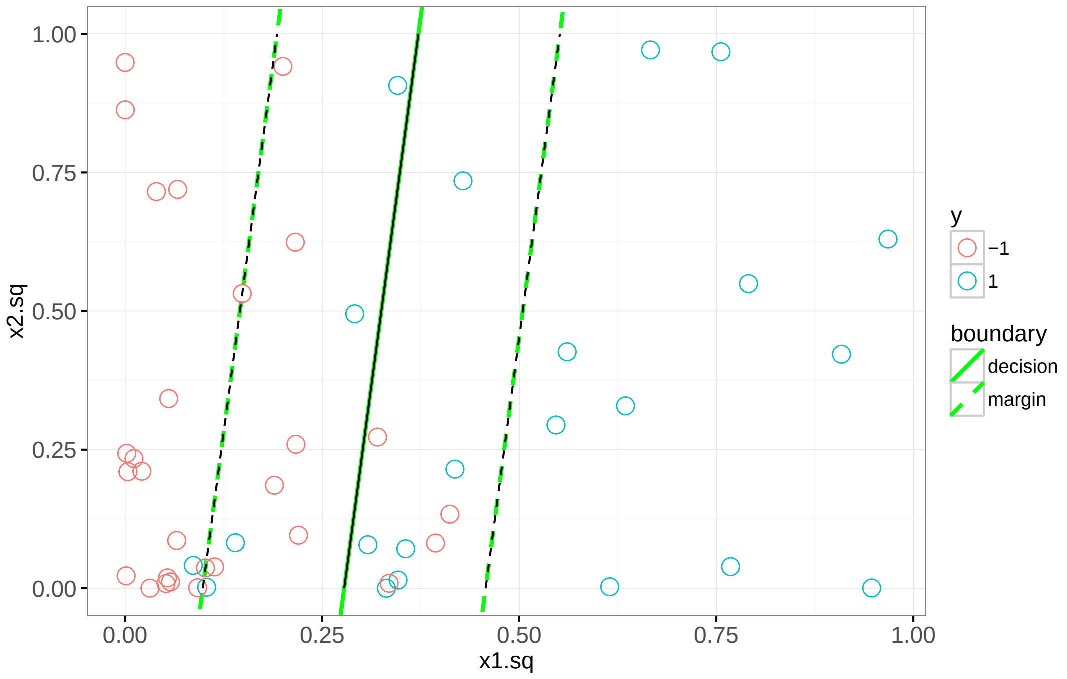
It is clear from the plot above that there are several mis-classified train data points. In the plot below we visualize the decision boundary and margin.

```
predF <- function(fit, X){
  fit.sc <- scaling(fit)$x.scale
  if(is.null(fit.sc)){
    fit.sc <- list(
      "scaled:center"=c(0,0),
      "scaled:scale"=c(1,1))
  }
  mu <- fit.sc[["scaled:center"]]
  sigma <- fit.sc[["scaled:scale"]]
  X.sc <- scale(X, mu, sigma)
  kernelMult(
    kernelf(fit),
    X.sc,
    xmatrix(fit)[[1]],
    coef(fit)[[1]])-b(fit)
}
xsq.vec <- seq(0, 1, l=41)
grid.sq.dt <- data.table(expand.grid(
  x1.sq=xsq.vec,
  x2.sq=xsq.vec
))[
, pred.f := predF(fit, cbind(x1.sq, x2.sq))]
subtrain.dt[, train.error := ifelse(y==pred.y, "correct", "error")]
ggplot()+
  theme_bw()+
  scale_color_manual(values=c(error="black", correct=NA))+
  geom_point(aes(
    x1.sq, x2.sq, fill=y, color=train.error),
    shape=21,
    stroke=1,
    size=4,
    data=subtrain.dt)+
  geom_vline(aes(
    xintercept=boundary), color="grey50",
    data=x1sq.boundary)+
  geom_contour(aes(
    x1.sq, x2.sq, z=pred.f),
    breaks=0,
    color="black",
    data=grid.sq.dt)+
  geom_contour(aes(
    x1.sq, x2.sq, z=pred.f),
    breaks=c(-1, 1),
    color="black",
    linetype="dashed",
    data=grid.sq.dt)
```



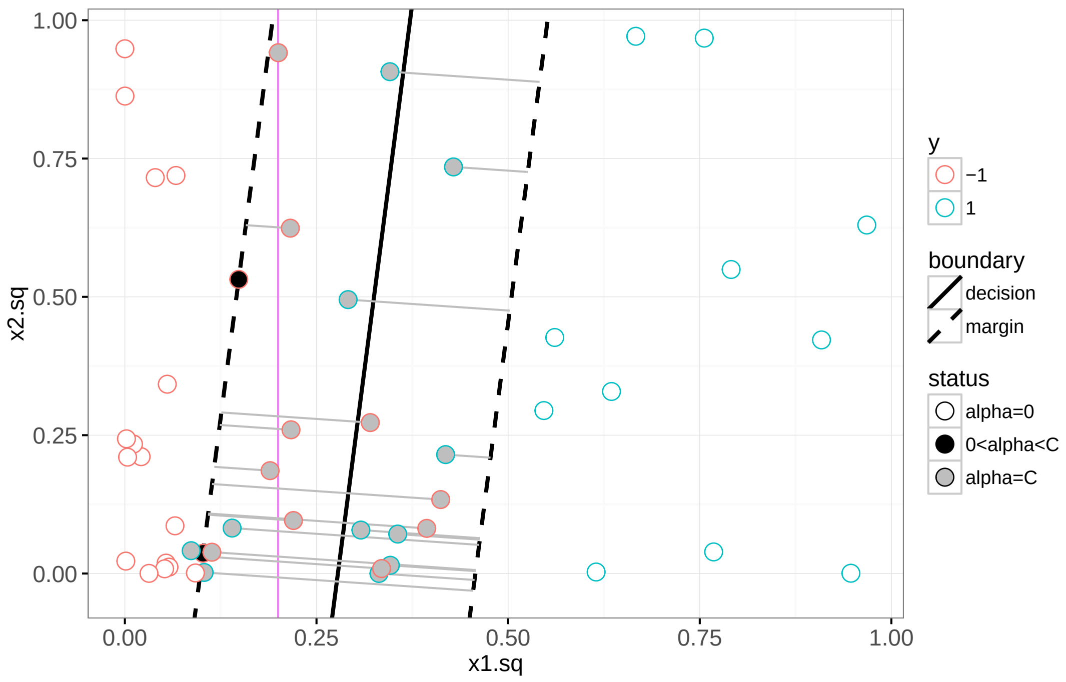
The plot above shows the true decision boundary using a grey `vline`. It also uses `geom_contour` to display the decision boundary (solid black line, predicted score 0) and the margin (dashed black line, predicted score -1 and 1). Since the decision boundary and margin are linear in this space, we can also use `geom_abline` to display them. To do that we need to do some math, and work out the equations for the slope and intercepts of those lines (as a function of the learned bias `b(fit)` and `weight.vec`, as well as the scale parameters mu and sigma).

```
## The equation of the margin lines is x2 = m2 + s2/w2[c+b+w1*m1/s1]
## -s2*w1/(w2*s1)*x1 for c=1 and -1. x is input feature, m is mean, s
## is scale, w is learned weight.
fit.sc <- scaling(fit)$x.scale
if(is.null(fit.sc)){
  fit.sc <- list(
    "scaled:center"=c(0,0),
    "scaled:scale"=c(1,1))
}
mu <- fit.sc[["scaled:center"]]
sigma <- fit.sc[["scaled:scale"]]
weight.vec <- colSums(xmatrix(fit)[[1]]*coef(fit)[[1]])
predF.linear <- function(fit, X){
  X.sc <- scale(X, mu, sigma)
  X.sc %*% weight.vec - b(fit)
}
abline.dt <- data.table(
  y=factor(c(-1,0,1)),
  boundary=c("margin", "decision", "margin"),
  intercept=mu[2]+sigma[2]/weight.vec[2]*(
    c(-1, 0, 1)+b(fit)+weight.vec[1]*mu[1]/sigma[1]),
  slope=-weight.vec[1]*sigma[2]/(weight.vec[2]*sigma[1]))
ggplot()+
  theme_bw()+
  scale_linetype_manual(values=c(margin="dashed", decision="solid"))+
  geom_abline(aes(
    slope=slope, intercept=intercept, linetype=boundary),
    color="green",
    size=1,
    data=abline.dt)+
  geom_point(aes(
    x1.sq, x2.sq, color=y),
    shape=21,
    fill=NA,
    size=4,
    data=subtrain.dt)+
  geom_contour(aes(
    x1.sq, x2.sq, z=pred.f),
    breaks=0,
    color="black",
    data=grid.sq.dt)+
  geom_contour(aes(
    x1.sq, x2.sq, z=pred.f),
    breaks=c(-1, 1),
    color="black",
    linetype="dashed",
    data=grid.sq.dt)
```



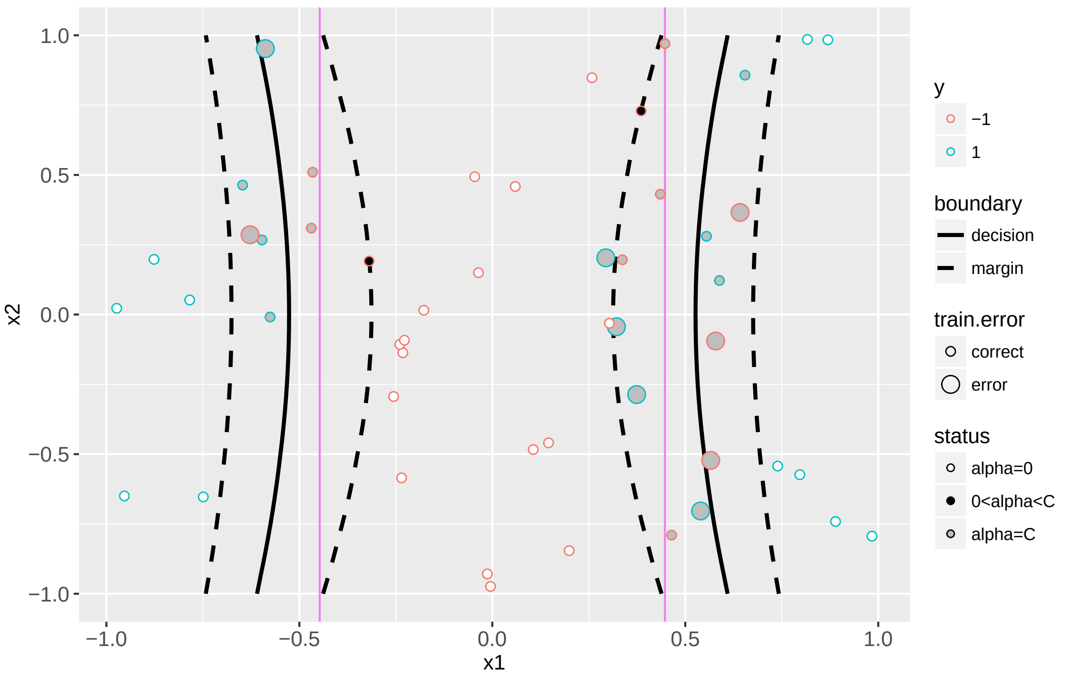
The plot above confirms that our computation of the slope and intercepts (green lines) agrees with the contours (black lines). In the plot below, we show the learned `alpha` coefficients, and add a `geom_segment` to visualize the slack.

```
subtrain.dt[, alpha := 0]
train.row.vec <- as.integer(rownames(xmatrix(fit)[[1]]))
subtrain.dt[train.row.vec, alpha := kernlab::alpha(fit)[[1]] ]
subtrain.dt[, status := ifelse(
  alpha==0, "alpha=0",
  ifelse(alpha==1, "alpha=C", "0<alpha<C"))]
slack.slope <- weight.vec[2]*sigma[1]/(weight.vec[1]*sigma[2])
slack.dt <- subtrain.dt[alpha==1,]
slack.join <- abline.dt[slack.dt, on=list(y)]
slack.join[, x1.sq.margin := (
  x2.sq-slack.slope*x1.sq-intercept)/(slope-slack.slope)]
slack.join[, x2.sq.margin := slope*x1.sq.margin + intercept]
sv.colors <- c(
  "alpha=0"="white",
  "0<alpha<C"="black",
  "alpha=C"="grey")
ggplot()+
  theme_bw()+
  scale_linetype_manual(values=c(margin="dashed", decision="solid"))+
  geom_vline(aes(
    xintercept=boundary), color="violet",
    data=x1sq.boundary)+
  geom_abline(aes(
    slope=slope, intercept=intercept, linetype=boundary),
    size=1,
    data=abline.dt)+
  geom_segment(aes(
    x1.sq, x2.sq,
    xend=x1.sq.margin, yend=x2.sq.margin),
    color="grey",
    data=slack.join)+
  scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
  geom_point(aes(
    x1.sq, x2.sq, color=y, fill=status),
    shape=21,
    size=4,
    data=subtrain.dt)
```



The plot above shows the slack in grey segments, and the decision and margin lines in black. The Bayes decision boundary is shown in the background as a vertical violet line. The support vectors are the points with non-zero alpha coefficients. Black filled support vectors are on the margin, and grey support vectors are on the wrong side of the margin (and have non-zero slack). The plot below shows the model that was learned in the original feature space,

```
n.grid <- 41
x.vec <- seq(-1, 1, l=n.grid)
grid.dt <- data.table(expand.grid(
  x1=x.vec,
  x2=x.vec))
getBoundaryDF <- function(score.vec, level.vec=c(-1, 0, 1)){
  stopifnot(length(score.vec) == n.grid * n.grid)
  several.paths <- contourLines(
    x.vec, x.vec,
    matrix(score.vec, n.grid, n.grid),
    levels=level.vec)
  contour.list <- list()
  for(path.i in seq_along(several.paths)){
    contour.list[[path.i]] <- with(several.paths[[path.i]], data.table(
      path.i,
      level.num=as.numeric(level),
      level.fac=factor(level, level.vec),
      boundary=ifelse(level==0, "decision", "margin"),
      x1=x, x2=y))
  }
  do.call(rbind, contour.list)
}
grid.dt[, pred.f := predF(fit, cbind(x1^2, x2^2))]
boundaries <- grid.dt[, getBoundaryDF(pred.f)]
ggplot()+
  scale_linetype_manual(values=c(margin="dashed", decision="solid"))+
  geom_vline(aes(
    xintercept=boundary),
    color="violet",
    data=x1.boundaries)+
  geom_path(aes(
    x1, x2, group=path.i, linetype=boundary),
    size=1,
    data=boundaries)+
  scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
  scale_size_manual(values=c(correct=2, error=4))+
  geom_point(aes(
    x1, x2, color=y,
    size=train.error,
    fill=status),
    shape=21,
    data=subtrain.dt)
```



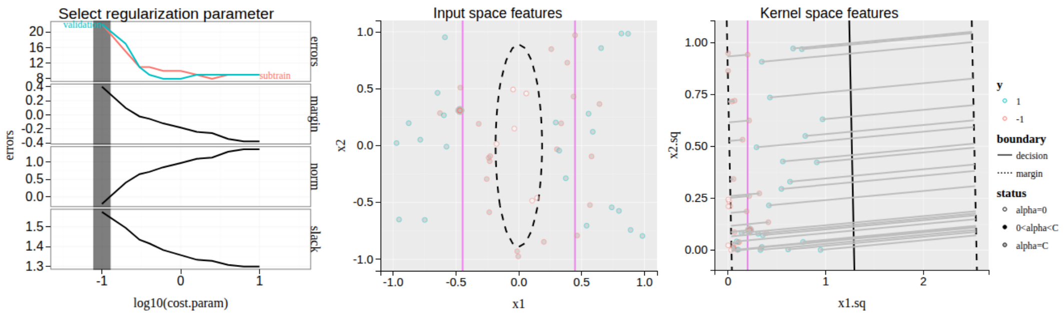
The goal below will be to make an animint that shows how the decision boundary, margin, and slack change as a function of the cost parameter.

```
modelInfo.list <- list()
predictions.list <- list()
slackSegs.list <- list()
modelLines.list <- list()
inputBoundaries.list <- list()
setErrors.list <- list()
cost.by <- 0.2
for(cost.param in round(10^seq(-1, 1, by=cost.by),1)){
  fit <- ksvm(
    squared.mat, y.vec, kernel="vanilladot", scaled=FALSE, C=cost.param)
  fit.sc <- scaling(fit)$x.scale
  if(is.null(fit.sc)){
    fit.sc <- list(
      "scaled:center"=c(0,0),
      "scaled:scale"=c(1,1))
  }
  mu <- fit.sc[["scaled:center"]]
  sigma <- fit.sc[["scaled:scale"]]
  weight.vec <- colSums(xmatrix(fit)[[1]]*coef(fit)[[1]])
  grid.sq.dt[, pred.f := predF(fit, cbind(x1.sq, x2.sq))]
  data.dt[, pred.y := predict(fit, cbind(x1.sq, x2.sq))]
  one.error <- data.dt[, list(errors=sum(y!=pred.y)), by=set]
  setErrors.list[[paste(cost.param)]] <- data.table(
    cost.param, one.error)
  subtrain.dt[, pred.f := predF(fit, cbind(x1^2, x2^2))]
```

```
    grid.dt[, pred.f := predF(fit, cbind(x1^2, x2^2))]
    boundaries <- getBoundaryDF(grid.dt$pred.f)
    inputBoundaries.list[[paste(cost.param)]] <- data.table(
      cost.param, boundaries)
    subtrain.dt$alpha <- 0
    train.row.vec <- as.integer(rownames(xmatrix(fit)[[1]]))
    subtrain.dt[train.row.vec, alpha := kernlab::alpha(fit)[[1]] ]
    subtrain.dt[, status := ifelse(
      alpha==0, "alpha=0",
      ifelse(alpha==cost.param, "alpha=C", "0<alpha<C"))]
    ## The equation of the margin lines is x2 = m2 + s2/w2[c+b+w1*m1/s1]
    ## -s2*w1/(w2*s1)*x1 for c=1 and -1. x is input feature, m is mean, s
    ## is scale, w is learned weight.
    slack.slope <- weight.vec[2]*sigma[1]/(weight.vec[1]*sigma[2])
    abline.dt <- data.table(
      y=factor(c(-1,0,1)),
      boundary=c("margin", "decision", "margin"),
      intercept=mu[2]+sigma[2]/weight.vec[2]*(
        c(-1, 0, 1)+b(fit)+weight.vec[1]*mu[1]/sigma[1]),
      slope=-weight.vec[1]*sigma[2]/(weight.vec[2]*sigma[1]))
    slack.dt <- subtrain.dt[alpha==cost.param]
    slack.join <- abline.dt[slack.dt, on=list(y)]
    slack.join[, x1.sq.margin := (
      x2.sq-slack.slope*x1.sq-intercept)/(slope-slack.slope)]
    slack.join[, x2.sq.margin := slope*x1.sq.margin + intercept]
    norm.weights <- as.numeric(weight.vec %*% weight.vec)
    modelInfo.list[[paste(cost.param)]] <- data.table(
      cost.param,
      slack=slack.join[, sum(1-pred.f*y.num)],
      norm=norm.weights,
      margin=2/sqrt(norm.weights))
    predictions.list[[paste(cost.param)]] <- data.table(
      cost.param, subtrain.dt)
    slackSegs.list[[paste(cost.param)]] <- data.table(
      cost.param, slack.join)
    modelLines.list[[paste(cost.param)]] <- data.table(
      cost.param, abline.dt)
  }
```

```
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
```

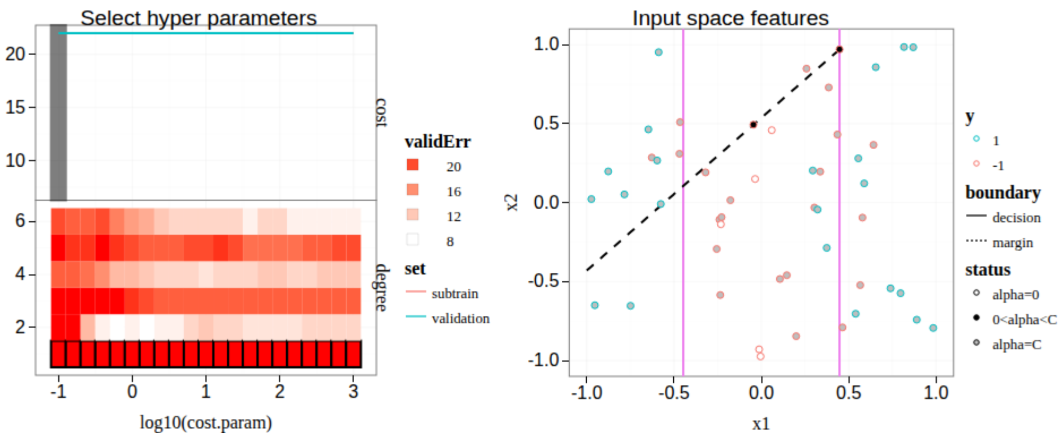
```
inputBoundaries <- do.call(rbind, inputBoundaries.list)
predictions <- do.call(rbind, predictions.list)
slackSegs <- do.call(rbind, slackSegs.list)
modelLines <- do.call(rbind, modelLines.list)
modelInfo <- do.call(rbind, modelInfo.list)
setErrors <- do.call(rbind, setErrors.list)
modelInfo.tall <- melt(modelInfo, id.vars="cost.param")
grid.sq.dt$boundary <- "true"
setErrors$variable <- "errors"
inputBoundaries[, boundary := ifelse(level.num==0, "decision", "margin")]
slackSegs$boundary <- "margin"
set.label.select <- data.table(
  cost.param=range(setErrors$cost.param),
  set=c("validation", "subtrain"),
  hjust=c(1, 0))
set.labels <- setErrors[set.label.select, on=list(cost.param, set)]
viz.linear.svm <- animint(
  selectModel=ggplot()+
    ggtitle("Select regularization parameter")+
    scale_x_continuous(limits=c(-1.5, 1.5))+
    geom_tallrect(aes(
      xmin=log10(cost.param)-cost.by/2,
      xmax=log10(cost.param)+cost.by/2),
      clickSelects="cost.param",
      alpha=0.5,
      data=modelInfo)+
    theme_bw()+
    facet_grid(variable ~ ., scales="free")+
    geom_line(aes(
      log10(cost.param), errors,
      group=set, color=set),
      data=setErrors)+
    geom_text(aes(
      log10(cost.param), errors-1, label=set,
      hjust=hjust,
      color=set),
      data=set.labels)+
    guides(color="none")+
    geom_line(aes(
      log10(cost.param), log10(value)),
      data=modelInfo.tall),
  inputSpace=ggplot()+
    ggtitle("Input space features")+
    scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
    geom_vline(aes(
      xintercept=boundary),
      color="violet",
      data=x1.boundaries)+
    guides(color="none", fill="none", linetype="none")+
    scale_linetype_manual(values=c(
      "-1"="dashed",
      "0"="solid",
      "1"="dashed"))+
    geom_path(aes(
      x1, x2,
      group=path.i,
      linetype=level.fac),
      showSelected=c("boundary", "cost.param"),
      color="black",
      data=inputBoundaries)+
    geom_point(aes(
      x1, x2, fill=status),
      showSelected=c("status", "y", "data.i", "cost.param"),
      size=5,
      color="grey",
      data=predictions)+
    geom_point(aes(
      x1, x2, color=y, fill=status),
      showSelected=c("cost.param", "status", "y"),
      clickSelects="data.i",
      size=3,
      data=predictions),
  kernelSpace=ggplot()+
    ggtitle("Kernel space features")+
    geom_vline(aes(
      xintercept=boundary), color="violet",
      data=x1sq.boundary)+
    ##coord_cartesian(xlim=c(0, 1), ylim=c(0, 1))+
    geom_abline(aes(
      slope=slope, intercept=intercept, linetype=boundary),
      showSelected="cost.param",
      color="black",
      data=modelLines)+
    scale_linetype_manual(values=c(
      decision="solid",
      margin="dashed",
      true="solid"))+
    geom_point(aes(
      x1.sq, x2.sq, fill=status),
      showSelected=c("data.i", "cost.param"),
      size=5,
      color="grey",
      data=predictions)+
    geom_point(aes(
      x1.sq, x2.sq, color=y, fill=status),
      clickSelects="data.i",
      showSelected="cost.param",
      size=3,
      data=predictions)+
    scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
    geom_segment(aes(
      x1.sq, x2.sq,
      xend=x1.sq.margin, yend=x2.sq.margin),
      showSelected=c("cost.param", "boundary"),
      color="grey",
      data=slackSegs))
viz.linear.svm
```



12.3 Non-linear polynomial kernel SVM

In the previous section we fit a linear kernel in the squared feature space, which resulted in learning a function which is non-linear in terms of the original feature space. In this section we directly fit a non-linear polynomial kernel in the original space.

```
predictions.list <- list()
inputBoundaries.list <- list()
setErrors.list <- list()
cost.by <- 0.2
orig.mat <- subtrain.dt[, cbind(x1, x2)]
for(cost.param in 10^seq(-1, 3, by=cost.by)){
  for(degree.num in seq(1, 6, by=1)){
    k <- polydot(degree.num, offset=0)
    fit <- ksvm(
      orig.mat, y.vec, kernel=k, scaled=FALSE, C=cost.param)
    grid.dt[, pred.f := predF(fit, cbind(x1, x2))]
    grid.dt[, pred.y := predict(fit, cbind(x1, x2))]
    grid.dt[, stopifnot(sign(pred.f) == pred.y)]
    data.dt[, pred.y := predict(fit, cbind(x1, x2))]
    one.error <- data.dt[, list(errors=sum(y != pred.y)), by=set]
    setErrors.list[[paste(cost.param, degree.num)]] <- data.table(
      cost.param, degree.num, one.error)
    boundaries <- getBoundaryDF(grid.dt$pred.f)
    if(is.data.frame(boundaries) && nrow(boundaries)){
      cost.deg <- paste(cost.param, degree.num)
      inputBoundaries.list[[cost.deg]] <- data.table(
        cost.param, degree.num, boundaries)
    }
    subtrain.dt[, alpha := 0]
    train.row.vec <- as.integer(rownames(xmatrix(fit)[[1]]))
    subtrain.dt[train.row.vec, alpha := kernlab::alpha(fit)[[1]] ]
    subtrain.dt[, status := ifelse(
      alpha==0, "alpha=0",
      ifelse(alpha==cost.param, "alpha=C", "0<alpha<C"))]
    predictions.list[[paste(cost.param, degree.num)]] <- data.table(
      cost.param, degree.num, subtrain.dt)
  }
}
inputBoundaries <- do.call(rbind, inputBoundaries.list)
predictions <- do.call(rbind, predictions.list)
setErrors <- do.call(rbind, setErrors.list)
validationErrors <- setErrors[set=="validation"]
validationErrors$select <- "degree"
setErrors$select <- "cost"
animint(
  selectModel=ggplot()+
    ggtitle("Select hyper parameters")+
    geom_tallrect(aes(
      xmin=log10(cost.param)-cost.by/2,
      xmax=log10(cost.param)+cost.by/2),
      clickSelects="cost.param",
      alpha=0.5,
      data=setErrors[degree.num==1 & set=="subtrain",])+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "lines"))+
    theme_animint(width=350, rowspan=1)+
    facet_grid(select ~ ., scales="free")+
    ylab("")+
    geom_line(aes(
      log10(cost.param), errors,
      key=set,
      group=set,
      color=set),
      showSelected="degree.num",
      data=setErrors)+
    scale_fill_gradient("validErr", low="white", high="red")+
    geom_tile(aes(
      log10(cost.param), degree.num, fill=errors),
      clickSelects="degree.num",
      data=validationErrors),
  inputSpace=ggplot()+
    theme_bw()+
    ggtitle("Input space features")+
    scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
    geom_vline(aes(
      xintercept=boundary),
      color="violet",
      data=x1.boundaries)+
    scale_linetype_manual(values=c(
      margin="dashed",
      decision="solid"))+
    geom_path(aes(
      x1, x2,
      group=path.i,
      linetype=boundary),
      showSelected=c("degree.num", "cost.param"),
      color="black",
      data=inputBoundaries)+
    geom_point(aes(
      x1, x2, color=y, fill=status),
      showSelected=c("cost.param", "degree.num"),
      size=3,
      data=predictions))
```



12.4 Chapter summary and exercises

We used ggplots to visualize the Support Vector Machine model for binary classification. We used animint and interactivity to show how the SVM decision boundary changes as a function of the model hyper-parameters.

Exercises:

- Use HTML table layout for `viz.linear.svm`, so that the two feature space plots appear beside each other, and the "Select regularization parameter" plot appears above or below.
- Use `rbfdot` as the kernel function. Compute subtrain and validation error, then add a new panel to the "select hyper parameters" plot.
- Default scales use the same two colors for the **y** and **set** legends, which could be confusing.

  Change the colors in one of the two legends so that they are different.
- Use `color` and `color_off` parameters to change the appearance of the `geom_tile` when selected or not, as explained in Chapter 6, section Specifying how selection state is displayed.

Next, Chapter 13 explains how to visualize the Poisson regression model.

13

Poisson regression

This goal of this chapter is to create an interactive data visualization that explains Poisson regression, a machine learning model for predicting an integer-valued output from inputs that are real-valued vectors. This is a "linear regression" model since it learns a linear function from the inputs to the output. Like least squares regression, Poisson regression can be formulated as a maximum likelihood problem. However, it differs from least squares linear regression since it uses a Poisson distribution to model the output labels, instead of a Gaussian distribution. This modeling choice is appropriate when output labels are non-negative integers.

Chapter outline:

- We begin by creating a plot that shows the probability mass function for a Poisson distribution mean parameter that can be interactively selected.
- We then add a second panel that shows the cumulative distribution function.
- We then add a second plot which shows the Poisson loss, with a second selector for label value.

13.1 Plot the probability mass function and select the Poisson mean parameter

The goal of this section is to create a data visualization that shows the probability mass function for a selected Poisson mean parameter.

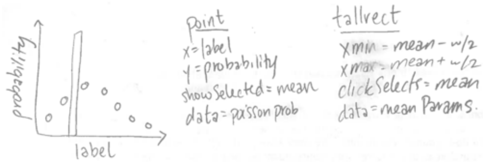


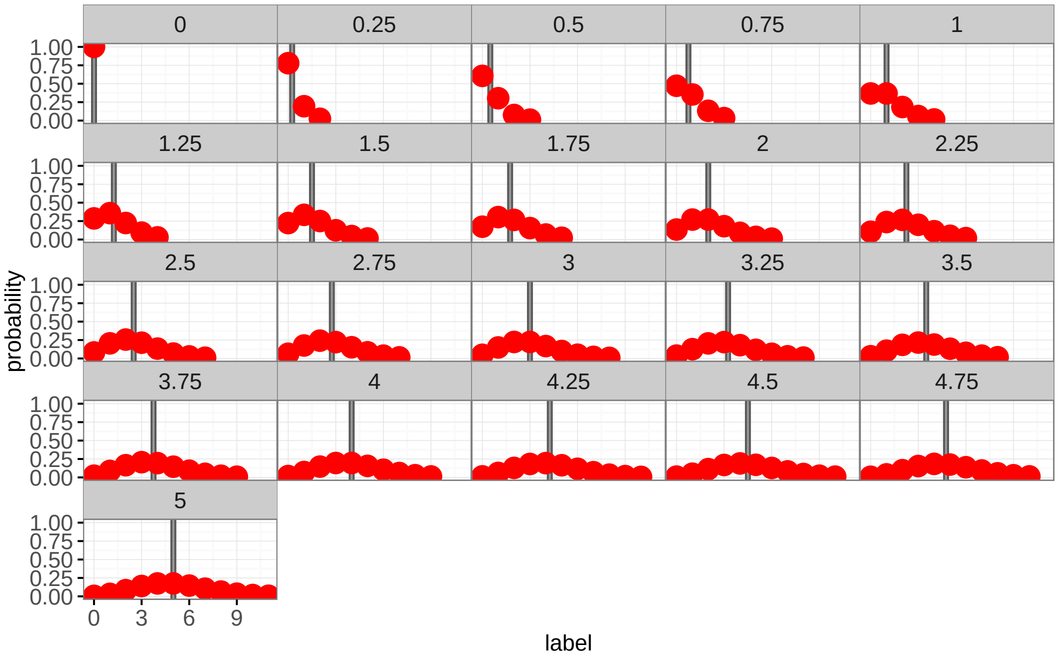
Figure 13.1: Poisson regression viz prob

```r
library(data.table)
poisson.mean.diff <- 0.25
poisson.mean.vec <- seq(0, 5, by=poisson.mean.diff)
quantile.max <- 0.99
poisson.prob.list <- list()
for(poisson.mean in poisson.mean.vec){
  label.max <- qpois(quantile.max, poisson.mean)
  label <- 0:label.max
  probability <- dpois(label, poisson.mean)
  poisson.prob.list[[paste(poisson.mean)]] <- data.table(
    poisson.mean,
    label,
    probability,
    cum.prob=cumsum(probability))
}
poisson.prob <- do.call(rbind, poisson.prob.list)
poisson.prob
```

```
     poisson.mean label probability  cum.prob
  1:         0.00     0 1.000000000 1.0000000
  2:         0.25     0 0.778800783 0.7788008
 ---
155:         5.00    10 0.018132789 0.9863047
156:         5.00    11 0.008242177 0.9945469
```

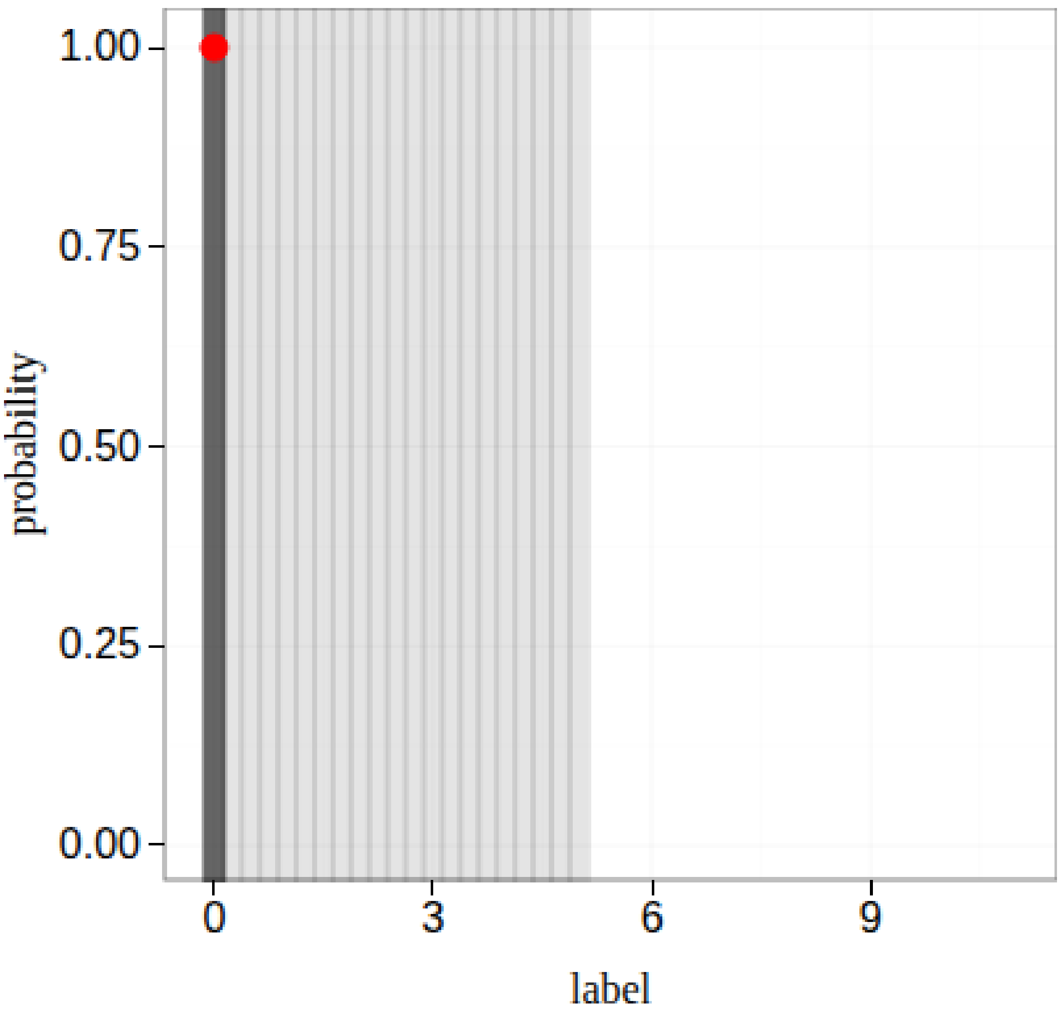
The static data viz below shows one facet for each Poisson distribution.

```
  mean.tallrects <- data.table(
    poisson.mean=poisson.mean.vec,
    min=poisson.mean.vec - poisson.mean.diff/2,
    max=poisson.mean.vec + poisson.mean.diff/2)
  library(animint2)
  prob.mass <- ggplot()+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "cm"))+
    geom_tallrect(aes(
      xmin=min, xmax=max),
      clickSelects="poisson.mean",
      alpha=0.6,
      data=mean.tallrects)+
    geom_point(aes(
      label, probability,
      tooltip=sprintf("prob(label = %d) = %f", label, probability)),
      color="red",
      showSelected="poisson.mean",
      size=5,
      data=poisson.prob)
  prob.mass+
    facet_wrap("poisson.mean")
```



Note that we used `alpha=0.6` with `geom_tallrect`, which means that the tallrect for the selected mean has 0.6 opacity, and the other tallrects have 0.1 opacity. Note also that we use `color="red"` and `size=5` with `geom_point` so that it is easier to see the points on a grey background, and to hover the cursor over the points to see the tooltip. We next create an interactive version with animint.

```
animint(prob.mass)
```



You can click the viz above to change the mean of the Poisson distribution. You can also hover the cursor over a data point to see its probability. Note that for integer values of the Poisson mean, there are two labels that are the most probable (the mode of the Poisson distribution). For example the Poisson distribution with a mean of 3 attains its maximum probability of about 0.224 at label values of 2 and 3.

13.2 Add a panel for the cumulative distribution function

To add a panel for the cumulative distribution function, we will re-make the ggplot based on the sketch below.

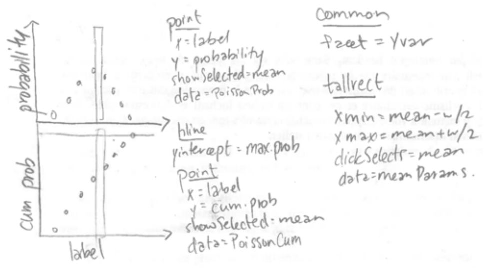
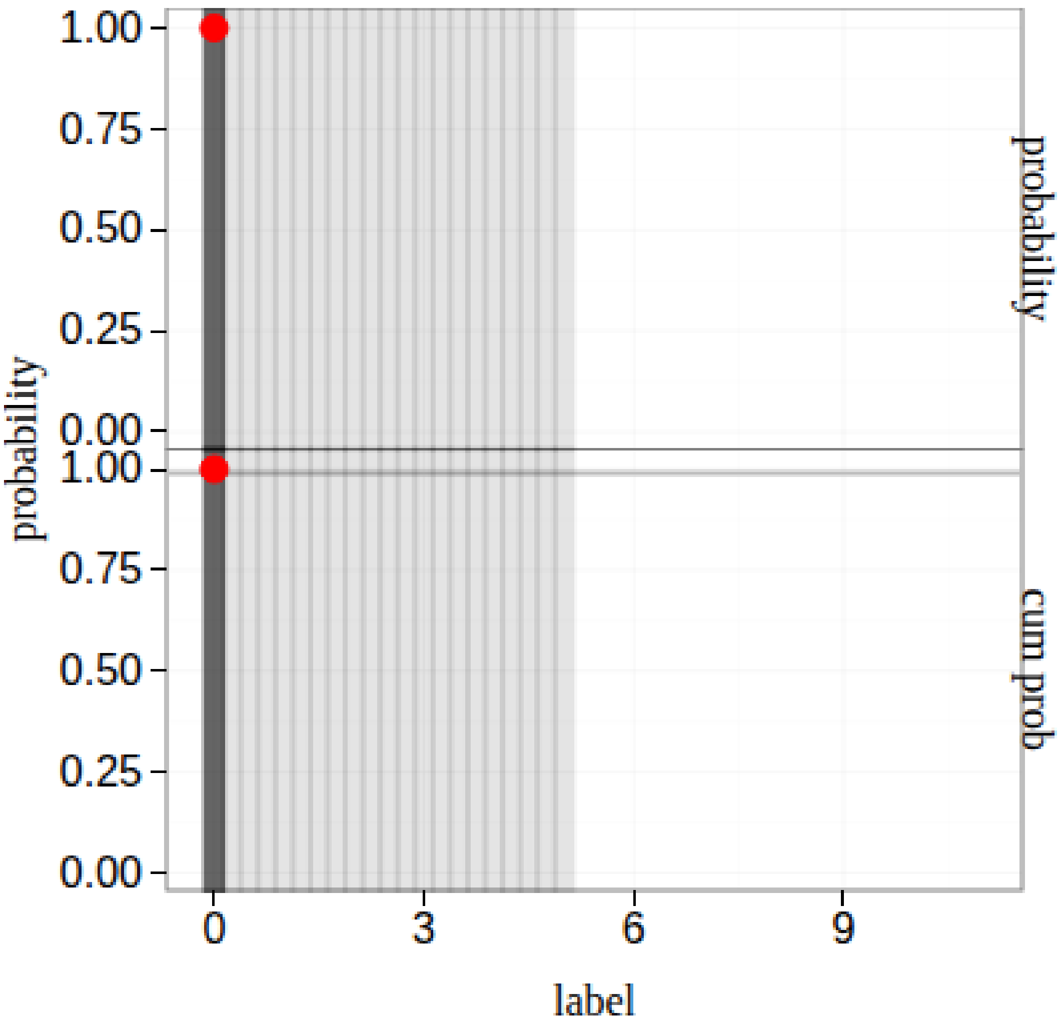


Figure 13.2: Poisson regression viz cum prob

When we specify the data sets, we will use the addColumn then facet idiom to add a `panel` variable.

```r
addPanel <- function(dt, panel){
  data.table(dt, panel=factor(panel, c("probability", "cum prob")))
}
quantile.max.dt <- data.table(quantile.max)
animint(
  prob=ggplot()+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "cm"))+
    facet_grid(panel ~ ., scales="free")+
    geom_hline(aes(
      yintercept=quantile.max),
      color="grey",
      data=addPanel(quantile.max.dt, "cum prob"))+
    geom_tallrect(aes(
      xmin=min, xmax=max),
      clickSelects="poisson.mean",
      alpha=0.6,
      data=mean.tallrects)+
    geom_point(aes(
      label, probability,
      tooltip=sprintf(
        "prob(label = %d) = %f", label, probability)),
      showSelected="poisson.mean",
      color="red",
      size=5,
      data=addPanel(poisson.prob, "probability"))+
    geom_point(aes(
      label, cum.prob,
      tooltip=sprintf(
        "prob(label <= %d) = %f", label, cum.prob)),
      showSelected="poisson.mean",
      color="red",
      size=5,
      data=addPanel(poisson.prob, "cum prob")))
```



Note how we used `addPanel` to add a `panel` variable to all the data sets for each geom except `geom_tallrect`. Using `panel` as a facet variable has the effect of drawing each geom in only one panel, except the `geom_tallrect` which is drawn in each panel.

Note that we also used a `geom_hline` to show 0.99, the cumulative distribution function threshold that was used to determine the set of points to plot for each Poisson distribution. This is an example of "show your arbitrary choices," one of the general principles of designing good interactive data visualizations.

13.3 Add a plot of the Poisson loss and a selector for label value

Next we will compute the Poisson loss for several values of the output label.

```
PoissonLoss <- function(label, seg.mean){
  stopifnot(is.numeric(label))
```

```
    stopifnot(is.numeric(seg.mean))
    if(any(seg.mean < 0)){
      stop("PoissonLoss undefined for negative segment mean")
    }
    if(length(seg.mean)==1)seg.mean <- rep(seg.mean, length(label))
    if(length(label)==1)label <- rep(label, length(seg.mean))
    stopifnot(length(seg.mean) == length(label))
    not.integer <- round(label) != label
    is.negative <- label < 0
    loss <- ifelse(
      not.integer | is.negative, Inf,
      ifelse(seg.mean == 0, ifelse(label == 0, 0, Inf),
             seg.mean - label * log(seg.mean)
             ## This term makes all the minima zero.
             -ifelse(label == 0, 0, label - label*log(label))))
    loss
  }
```

Below we compute the loss for several label values, using the list of data tables idiom.

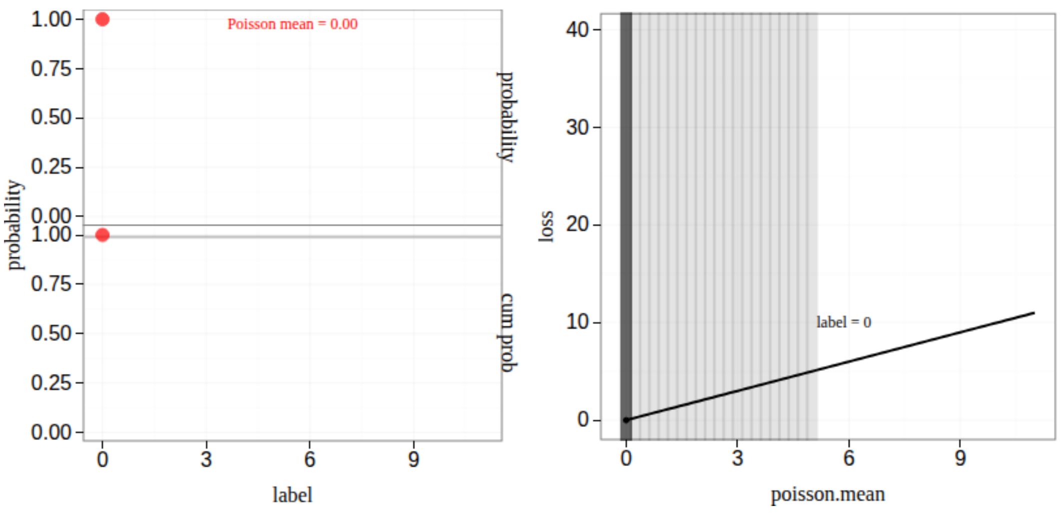
```
label.vec <- unique(poisson.prob$label)
label.range <- range(label.vec)
mean.vec <- seq(label.range[1], label.range[2], l=100)
loss.min.list <- list()
loss.fun.list <- list()
for(label in label.vec){
  loss <- PoissonLoss(label, mean.vec)
  loss.fun.list[[paste(label)]] <- data.table(
    label, poisson.mean=mean.vec, loss)
  loss.min.list[[paste(label)]] <- data.table(
    label, loss=0)
}
loss.fun <- do.call(rbind, loss.fun.list)
loss.min <- do.call(rbind, loss.min.list)
```

We also make a data table to display text labels for the selected mean and label values.

```
mean.text <- data.table(
  label=max(poisson.prob$label)/2,
  probability=0.95,
  poisson.mean=poisson.mean.vec)
loss.max <- 10
label.text <- data.table(
  poisson.mean=max(mean.tallrects$max),
  loss=loss.max*0.95,
  label=label.vec)
```

Next we make a data viz with an additional panel.

```
(viz.loss <- animint(
  prob=ggplot()+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "cm"))+
    facet_grid(panel ~ ., scales="free")+
    geom_text(aes(
      label, probability, label=sprintf(
      "Poisson mean = %.2f", poisson.mean)),
      color="red",
      showSelected="poisson.mean",
      data=addPanel(mean.text, "probability"))+
    geom_hline(aes(
      yintercept=quantile.max),
      color="grey",
      data=addPanel(quantile.max.dt, "cum prob"))+
    geom_point(aes(
      label, probability,
      tooltip=sprintf(
        "prob(label = %d) = %f", label, probability)),
      showSelected="poisson.mean",
      clickSelects="label",
      color="red",
      size=5,
      alpha=0.7,
      data=addPanel(poisson.prob, "probability"))+
    geom_point(aes(
      label, cum.prob,
      tooltip=sprintf(
        "prob(label <= %d) = %f", label, cum.prob)),
      color="red",
      showSelected="poisson.mean",
      clickSelects="label",
      size=5,
      alpha=0.7,
      data=addPanel(poisson.prob, "cum prob")),
  loss=ggplot()+
    theme_bw()+
    geom_text(aes(
      poisson.mean, loss,
      label=sprintf("label = %d", label)),
      showSelected="label",
      hjust=0,
      data=label.text)+
    geom_line(aes(
      poisson.mean, loss),
      showSelected="label",
      data=loss.fun)+
    geom_point(aes(
      label, loss),
        showSelected="label",
        data=loss.min)+
      geom_tallrect(aes(
        xmin=min, xmax=max),
        clickSelects="poisson.mean",
        alpha=0.6,
        data=mean.tallrects)))
```

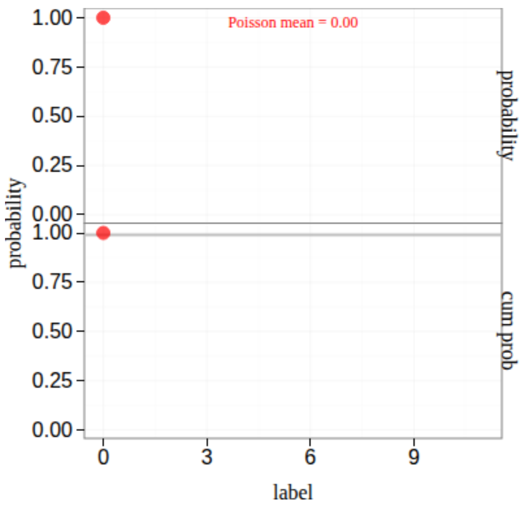


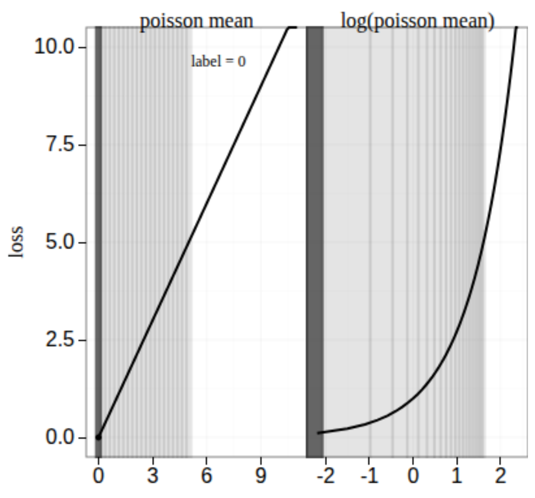
The data viz above shows the probability on the left and the Poisson loss on the right.

```
viz.log.loss <- viz.loss
addX <- function(dt, x.var)data.table(dt, x.var=factor(
  x.var, c("poisson mean", "log(poisson mean)")))
finite.loss <- loss.fun[is.finite(loss)]
finite.loss[, log.poisson.mean := log(poisson.mean)]
finite.log.loss <- finite.loss[is.finite(log.poisson.mean)]
mean.tallrects[, log.min := ifelse(min < 0, -Inf, log(min))]
```

```
Warning in log(min): NaNs produced
```

```
viz.log.loss$loss <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(. ~ x.var, scales="free")+
  xlab("")+
  coord_cartesian(ylim=c(0, loss.max))+
  geom_text(aes(
    poisson.mean, loss, label=sprintf(
                          "label = %d", label)),
    showSelected="label",
    hjust=0,
    data=addX(label.text, "poisson mean"))+
  geom_line(aes(
      poisson.mean, loss),
      showSelected="label",
      data=addX(finite.loss, "poisson mean"))+
    geom_point(aes(
      label, loss),
      showSelected="label",
      data=addX(loss.min, "poisson mean"))+
    geom_tallrect(aes(
      xmin=min, xmax=max),
      clickSelects="poisson.mean",
      alpha=0.6,
      data=addX(mean.tallrects, "poisson mean"))+
    geom_line(aes(
      log.poisson.mean, loss),
      showSelected="label",
      data=addX(finite.log.loss, "log(poisson mean)"))+
    geom_point(aes(
      log(label), loss),
      showSelected="label",
      data=addX(loss.min[0<label,], "log(poisson mean)"))+
    geom_tallrect(aes(
      xmin=log.min, xmax=log(max)),
      clickSelects="poisson.mean",
      alpha=0.6,
      data=addX(mean.tallrects, "log(poisson mean)"))
  viz.log.loss
```





13.4 Chapter summary and exercises

We explained how to visualize the Poisson distribution and loss, which are used for the Poisson regression model.

Exercises:

- The code above used `addPanel` and `addX` helper functions with several geoms to create multi-panel plots, which results in repetition. To avoid that repetition, create a new data viz which uses a single geom with a larger data set. For example, the red points in the two panels of the first plot could be defined using one `geom_point` with a larger data set (Hint: use `data.table::melt` with `measure.vars=c("cum.prob", "probability")`.
- Create a similar sequence of data visualizations for the Binomial regression model.

Next, Chapter 14 explains how to use the named clickSelects/showSelected to visualize the PeakSegJoint machine learning model with data-driven selector variables.

14 *Named clickSelects/showSelected*

This chapter explains how to use named clickSelects/showSelected variables for creating data-driven selector names. This feature makes it easier to write animint code, and makes it faster to compile.

Chapter outline:

- We begin by downloading the `PSJ` data set and computing the data to plot.
- We show one method of defining an animint with many selectors, using for loops. This method is technically correct, but computationally inefficient.
- We then explain the preferred method for defining an animint with many selectors, using named clickSelects/showSelected. This method is more computationally efficient, and easier to code.

14.1 Download data set

The example data come from the PeakSegJoint package. The code below downloads the data set.

```
if(!requireNamespace("animint2data"))
  remotes::install_github("animint/animint2data")
```

```
Loading required namespace: animint2data
```

```
data(PSJ, package="animint2data")
```

14.2 Compute data to plot

The section below computes some common data that we will use in two data visualizations below.

```
res.error <- PSJ$error.total.chunk
ann.colors <- c(
  noPeaks="#f6f4bf",
  peakStart="#ffafaf",
  peakEnd="#ff4c4c",
  peaks="#a445ee")
## prob.regions are the black segments that show which regions are
## mapped to which segmentation problems.
library(data.table)
all.regions <- data.table(do.call(rbind, PSJ$regions.by.problem))
prob.regions.names <- c(
  "bases.per.problem", "problem.i", "problem.name",
  "chromStart", "chromEnd")
prob.regions <- unique(data.frame(all.regions)[, prob.regions.names])
prob.regions$sample.id <- "problems"
all.modelSelection <- data.table(do.call(
  rbind, PSJ$modelSelection.by.problem))
modelSelection.errors <- all.modelSelection[!is.na(errors)]
penalty.range <- all.modelSelection[, c(
  min(max.log.lambda), max(min.log.lambda))]
penalty.mid <- mean(penalty.range)
coverage.counts <- table(PSJ$coverage$sample.id)
facet.rows <- length(coverage.counts)+1
dvec <- diff(log(res.error$bases.per.problem))
dval <- exp(mean(dvec))
dval2 <- (dval-1)/2 + 1
res.error$min.bases.per.problem <- res.error$bases.per.problem/dval2
res.error$max.bases.per.problem <- res.error$bases.per.problem*dval2
modelSelection.labels <- unique(all.modelSelection[, data.table(
  problem.name=problem.name,
  bases.per.problem=bases.per.problem,
  problemStart=problemStart,
  problemEnd=problemEnd,
  min.log.lambda=penalty.mid,
  peaks=max(peaks)+0.5)])
```

14.3 Define data viz using for loops

The R code below constructs a data viz using for loops.

```
library(animint2)
print(timing.for.construct <- system.time({
  viz.for <- list(
    coverage=ggplot()+
      geom_segment(aes(
        chromStart/1e3, problem.i,
        xend=chromEnd/1e3, yend=problem.i),
        showSelected="bases.per.problem",
        clickSelects="problem.name",
        data=prob.regions)+
      ggtitle("select problem")+
      geom_text(aes(
        chromStart/1e3, problem.i,
        label=sprintf("%d problems mean size %.1f kb",
                      problems, mean.bases/1e3)),
        showSelected="bases.per.problem",
        data=PSJ$problem.labels,
        hjust=0)+
      geom_segment(aes(
        problemStart/1e3, problem.i,
        xend=problemEnd/1e3, yend=problem.i),
        showSelected="bases.per.problem",
        clickSelects="problem.name",
        size=5,
        data=PSJ$problems)+
      scale_y_continuous(
        "aligned read coverage",
        breaks=function(limits){
          floor(limits[2])
        })+
      scale_linetype_manual(
        "error type",
        limits=c(
          "correct",
          "false negative",
          "false positive"),
        values=c(
          correct=0,
          "false negative"=3,
          "false positive"=1))+
      scale_x_continuous(paste(
        "position on chr11",
        "(kilo bases = kb)"))+
      coord_cartesian(xlim=c(118167.406, 118238.833))+
      geom_tallrect(aes(
        xmin=chromStart/1e3, xmax=chromEnd/1e3,
        fill=annotation),
        alpha=0.5,
        color="grey",
        data=PSJ$filled.regions)+
      scale_fill_manual(values=ann.colors)+
      theme_bw()+
      theme_animint(width=1500, height=facet.rows*100)+
      theme(panel.margin=grid::unit(0, "cm"))+
      facet_grid(sample.id ~ ., labeller=function(df){
        df$sample.id <- sub("McGill0", "", sub(" ", "\n", df$sample.id))
        df
      }, scales="free")+
      geom_line(aes(
        base/1e3, count),
        data=PSJ$coverage,
        color="grey50"),
    resError=ggplot()+
      ggtitle("select problem size")+
      ylab("minimum percent incorrect regions")+
      geom_tallrect(aes(
        xmin=min.bases.per.problem,
        xmax=max.bases.per.problem),
        clickSelects="bases.per.problem",
        alpha=0.5,
        data=res.error)+
      scale_x_log10()+
      geom_line(aes(
        bases.per.problem, errors/regions*100,
        color=chunks, size=chunks),
        data=data.frame(res.error, chunks="this"))+
      geom_line(aes(
        bases.per.problem, errors/regions*100,
        color=chunks, size=chunks),
        data=data.frame(PSJ$error.total.all, chunks="all")),
    modelSelection=ggplot()+
      geom_segment(aes(
        min.log.lambda, peaks,
        xend=max.log.lambda, yend=peaks),
        showSelected=c("bases.per.problem", "problem.name"),
        data=data.frame(all.modelSelection, what="peaks"),
        size=5)+
      geom_text(aes(
        min.log.lambda, peaks,
        label=sprintf(
          "%.1f kb in problem %s",
          (problemEnd-problemStart)/1e3, problem.name)),
        showSelected=c("problem.name", "bases.per.problem"),
        data=data.frame(modelSelection.labels, what="peaks"))+
      geom_segment(aes(
        min.log.lambda, as.integer(errors),
        xend=max.log.lambda, yend=as.integer(errors)),
        showSelected=c("bases.per.problem", "problem.name"),
        data=data.frame(modelSelection.errors, what="errors"),
        size=5)+
      ggtitle("select number of samples with 1 peak")+
      ylab("")+
      facet_grid(what ~ ., scales="free"),
    title="Animint compiler with for loops",
    first=PSJ$first)
  ## For every problem there is a selector (called problem.dot) for the
  ## number of peaks in that problem. So in this for loop we add a few
  ## layers with aes_string(clickSelects=problem.dot) or
```

```
    ## aes_string(showSelected=problem.dot) to the coverage and
    ## modelSelection plots.
    for(problem.dot in names(PSJ$modelSelection.by.problem)){
      regions.dt <- PSJ$regions.by.problem[[problem.dot]]
      regions.dt[[problem.dot]] <- regions.dt$peaks
      if(!is.null(regions.dt)){
        viz.for$coverage <- viz.for$coverage+
          geom_tallrect(aes(
            xmin=chromStart/1e3,
            xmax=chromEnd/1e3,
            linetype=status),
            showSelected=c(problem.dot, "bases.per.problem"),
            data=data.frame(regions.dt),
            fill=NA,
            color="black")
      }
      if(problem.dot %in% names(PSJ$peaks.by.problem)){
        peaks <- PSJ$peaks.by.problem[[problem.dot]]
        peaks[[problem.dot]] <- peaks$peaks
        prob.peaks.names <- c(
          "bases.per.problem", "problem.i", "problem.name",
          "chromStart", "chromEnd", problem.dot)
        prob.peaks <- unique(data.frame(peaks)[, prob.peaks.names])
        prob.peaks$sample.id <- "problems"
        viz.for$coverage <- viz.for$coverage +
          geom_segment(aes(
            chromStart/1e3, 0,
            xend=chromEnd/1e3, yend=0),
            clickSelects="problem.name",
            showSelected=c(problem.dot, "bases.per.problem"),
            data=peaks, size=7, color="deepskyblue")+
          geom_segment(aes(
            chromStart/1e3, problem.i,
            xend=chromEnd/1e3, yend=problem.i),
            clickSelects="problem.name",
            showSelected=c(problem.dot, "bases.per.problem"),
            data=prob.peaks, size=7, color="deepskyblue")
      }
      modelSelection.dt <- PSJ$modelSelection.by.problem[[problem.dot]]
      modelSelection.dt[[problem.dot]] <- modelSelection.dt$peaks
      viz.for$modelSelection <- viz.for$modelSelection+
        geom_tallrect(aes(
          xmin=min.log.lambda,
          xmax=max.log.lambda),
          clickSelects=problem.dot,
          showSelected=c("problem.name", "bases.per.problem"),
          data=modelSelection.dt, alpha=0.5)
    }
  }))
```

```
   user  system elapsed 
  3.091   0.000   3.091 
```

Note the timing of the code above. It takes a long time just to evaluate the R code that defines this data viz, since it has so many geoms. Next, we compile the data visualization.

```r
print(timing.for.compile <- system.time({
  animint2dir(viz.for, "Ch14-for")
}))
```

```
Warning: Using size for a discrete variable is not advised.
Warning: Using size for a discrete variable is not advised.
```

```
Warning in checkSingleShowSelectedValue(meta$selectors): showSelected variables
with only 1 level: chr11.118184422.118184700peaks,
chr11.118192951.118193582peaks, chr11.118203893.118204314peaks
```

```
   user  system elapsed 
241.081   0.270 242.610 
```

Note that the compilation also takes a long time, since there are so many geoms. The data viz can be viewed on Ch14-for/index.html. In the next section we will create the same data viz, but more efficiently.

14.4 Define data viz using named clickSelects/showSelected

In this section we use named clickSelects/showSelected to create a more efficient version of the previous data visualization. In general, any data visualization defined using for loops in R code can be made more efficient by instead using this method.

```r
sample.peaks <- data.table(do.call(rbind, PSJ$peaks.by.problem))
prob.peaks.names <- c(
  "bases.per.problem", "problem.i", "problem.name", "peaks",
  "chromStart", "chromEnd")
problem.peaks <- unique(sample.peaks[, ..prob.peaks.names])
problem.peaks$sample.id <- "problems"
peakvar <- function(position){
  paste0(gsub("[-:]", ".", position), "peaks")
}
all.regions[, selector := peakvar(problem.name)]
sample.peaks[, selector := peakvar(problem.name)]
problem.peaks[, selector := peakvar(problem.name)]
all.modelSelection[, selector := peakvar(problem.name)]
print(timing.named.construct <- system.time({
  viz.named <- list(
    coverage=ggplot()+
      ggtitle("select problem")+
      geom_segment(aes(
        chromStart/1e3, problem.i,
        xend=chromEnd/1e3, yend=problem.i),
        showSelected="bases.per.problem",
        clickSelects="problem.name",
        data=prob.regions)+
      geom_text(aes(
        chromStart/1e3, problem.i,
        label=sprintf(
          "%d problems mean size %.1f kb",
          problems, mean.bases/1e3)),
        showSelected="bases.per.problem",
        data=PSJ$problem.labels,
        hjust=0)+
      geom_segment(aes(
        problemStart/1e3, problem.i,
        xend=problemEnd/1e3, yend=problem.i),
        showSelected="bases.per.problem",
        clickSelects="problem.name",
        size=5,
        data=PSJ$problems)+
      scale_y_continuous(
        "aligned read coverage",
        breaks=function(limits){
          floor(limits[2])
        })+
      scale_linetype_manual(
        "error type",
        limits=c(
          "correct",
          "false negative",
          "false positive"),
        values=c(
          correct=0,
          "false negative"=3,
          "false positive"=1))+
      scale_x_continuous(paste(
        "position on chr11",
        "(kilo bases = kb)"))+
      coord_cartesian(xlim=c(118167.406, 118238.833))+
      geom_tallrect(aes(
        xmin=chromStart/1e3, xmax=chromEnd/1e3,
        fill=annotation),
        alpha=0.5,
        color="grey",
        data=PSJ$filled.regions)+
      scale_fill_manual(values=ann.colors)+
      theme_bw()+
      theme_animint(width=1500, height=facet.rows*100)+
      theme(panel.margin=grid::unit(0, "cm"))+
      facet_grid(sample.id ~ ., labeller=function(df){
```

```
        df$sample.id <- sub("McGill0", "", sub(" ", "\n", df$sample.id))
        df
      }, scales="free")+
      geom_line(aes(
        base/1e3, count),
        data=PSJ$coverage,
        color="grey50")+
      geom_tallrect(aes(
        xmin=chromStart/1e3,
        xmax=chromEnd/1e3,
        linetype=status),
        showSelected=c("selector"="peaks", "bases.per.problem"),
        data=all.regions,
        fill=NA,
        color="black")+
      geom_segment(aes(
        chromStart/1e3, 0,
        xend=chromEnd/1e3, yend=0),
        clickSelects="problem.name",
        showSelected=c("selector"="peaks", "bases.per.problem"),
        data=sample.peaks, size=7, color="deepskyblue")+
      geom_segment(aes(
        chromStart/1e3, problem.i,
        xend=chromEnd/1e3, yend=problem.i),
        clickSelects="problem.name",
        showSelected=c("selector"="peaks", "bases.per.problem"),
        data=problem.peaks, size=7, color="deepskyblue"),
    resError=ggplot()+
      ggtitle("select problem size")+
      ylab("minimum percent incorrect regions")+
      geom_tallrect(aes(
        xmin=min.bases.per.problem,
        xmax=max.bases.per.problem),
        clickSelects="bases.per.problem",
        alpha=0.5,
        data=res.error)+
      scale_x_log10()+
      geom_line(aes(
        bases.per.problem, errors/regions*100,
        color=chunks, size=chunks),
        data=data.frame(res.error, chunks="this"))+
      geom_line(aes(
        bases.per.problem, errors/regions*100,
        color=chunks, size=chunks),
        data=data.frame(PSJ$error.total.all, chunks="all")),
    modelSelection=ggplot()+
      geom_segment(aes(
        min.log.lambda, peaks,
        xend=max.log.lambda, yend=peaks),
        showSelected=c("problem.name", "bases.per.problem"),
        data=data.frame(all.modelSelection, what="peaks"),
        size=5)+
      geom_text(aes(
        min.log.lambda, peaks,
        label=sprintf(
          "%.1f kb in problem %s",
          (problemEnd-problemStart)/1e3, problem.name)),
        showSelected=c("problem.name", "bases.per.problem"),
        data=data.frame(modelSelection.labels, what="peaks"))+
      geom_segment(aes(
        min.log.lambda, as.integer(errors),
        xend=max.log.lambda, yend=as.integer(errors)),
        showSelected=c("problem.name", "bases.per.problem"),
        data=data.frame(modelSelection.errors, what="errors"),
        size=5)+
      ggtitle("select number of samples with 1 peak")+
      ylab("")+
      geom_tallrect(aes(
        xmin=min.log.lambda,
        xmax=max.log.lambda),
        clickSelects=c("selector"="peaks"),
        showSelected=c("problem.name", "bases.per.problem"),
        data=all.modelSelection, alpha=0.5)+
      facet_grid(what ~ ., scales="free"),
    title="Animint compiler with named clickSelects/showSelected",
    first=PSJ$first)
  ### For every problem there is a selector (called problem.name) for
  ### the number of peaks in that problem. The animint2dir compiler
  ### creates a selection variable for every unique value of
  ### clickSelects/showSelected names (and it uses corresponding values
  ### to set/update the selected value/geoms).
  }))
```

```
   user  system elapsed 
  0.024   0.000   0.024 
```

It is clear that it takes much less time to evaluate the R code above which uses the named clickSelects/showSelected. We compile it below.

```
  print(timing.named.compile <- system.time({
    animint2dir(viz.named, "Ch14-named")
  }))
```

```
Warning: Using size for a discrete variable is not advised.
Warning: Using size for a discrete variable is not advised.
```

```
   user  system elapsed 
  1.752   0.055   7.632 
```

The animint produced above can be viewed on Ch14-named/index.html. Note that it should appear to be the same as the other data viz above. The timings above show that named clickSelects/showSelected are much faster than for loops, in both the definition and compilation steps.

14.5 Disk usage comparison

In this section we compute the disk usage of both methods.

```
viz.dirs.vec <- c("Ch14-for", "Ch14-named")
viz.dirs.text <- paste(viz.dirs.vec, collapse=" ")
(cmd <- paste("du -ks", viz.dirs.text))
```

```
[1] "du -ks Ch14-for Ch14-named"
```

```
kb.dt <- fread(cmd=cmd)
setnames(kb.dt, c("kilobytes", "path"))
kb.dt
```

```
   kilobytes       path
1:      4508   Ch14-for
2:      1768 Ch14-named
```

The table above shows that the data viz defined using for loops takes about twice as much disk space as the data viz that used named clickSelects/showSelected.

14.6 Chapter summary and exercises

The table below summarizes the disk usage and timings presented in this chapter. It is clear that named clickSelects/showSelected are more efficient in both respects, and should be used instead of for loops.

```
data.frame(
  kilobytes=kb.dt$kilobytes,
  construct.seconds=c(
    timing.for.construct[["elapsed"]],
    timing.named.construct[["elapsed"]]),
  compile.seconds=c(
    timing.for.compile[["elapsed"]],
    timing.named.compile[["elapsed"]]),
  row.names=c("for", "named"))
```

```
      kilobytes construct.seconds compile.seconds
for        4508             3.091         242.610
named      1768             0.024           7.632
```

Exercises:

- Use named clickSelects/showSelected to create a visualization of some data from your domain of expertise.

Next, Chapter 15 explains how to visualize root-finding algorithms.

15

Newton's root-finding method

Roots of a function `f(x)` are values `x` such that `f(x)=0`. Some functions `f` have an explicit expression for their roots. For example:

- the linear function `f(x)=b*x+c=0` has a single root `x=-c/b`, if b is not zero.
- the quadratic function `f(x)=a*x^2+b*x+c=0` has two roots `x=(-b±sqrt(b^2-4*a*c))/(2*a)`, if the discriminant is positive `b^2-4*a*c>0`.
- the sin function `f(x)=sin(a*x)=0` has an infinite number of roots: `pi*z/a` for all integers `z`.

However, there are some functions which have no explicit expression for their roots. For example, the roots of the Poisson loss `f(x)=a*x+b*log(x)+c` have no explicit expression in terms of common mathematical functions. (actually it has a solution in terms of the Lambert W function but that function is not commonly available) This goal of this chapter is to create an interactive data visualization that explains Newton's method for finding the roots of such functions.

Chapter outline:

- We begin by implementing the Newton method for the Poisson loss, to find the root which is larger than the minimum. We create several static and one interactive data visualization.
- We then suggest an exercise for finding the root which is smaller than the minimum.

15.1 Larger root in mean space

We begin by defining coefficients of a Poisson Loss function with two roots.

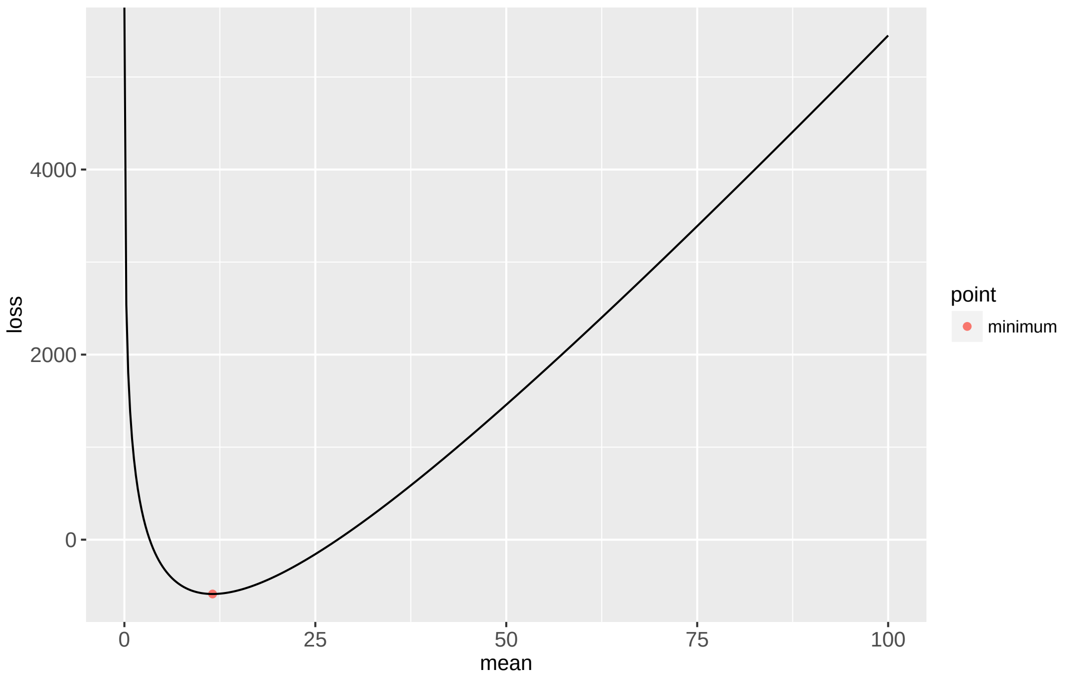
```
Linear <- 95
Log <- -1097
Constant <- 1000
loss.fun <- function(Mean){
  Linear*Mean + Log*log(Mean) + Constant
}
(mean.at.optimum <- -Log/Linear)
```

```
[1] 11.54737
```

```
(loss.at.optimum <- loss.fun(mean.at.optimum))
```

```
[1] -586.764
```

```
library(data.table)
loss.dt <- data.table(mean=seq(0, 100, l=400))
loss.dt[, loss := loss.fun(mean)]
opt.dt <- data.table(
  mean=mean.at.optimum,
  loss=loss.at.optimum,
  point="minimum")
library(animint2)
gg.loss <- ggplot()+
  geom_point(aes(mean, loss, color=point), data=opt.dt)+
  geom_line(aes(mean, loss), data=loss.dt)
print(gg.loss)
```

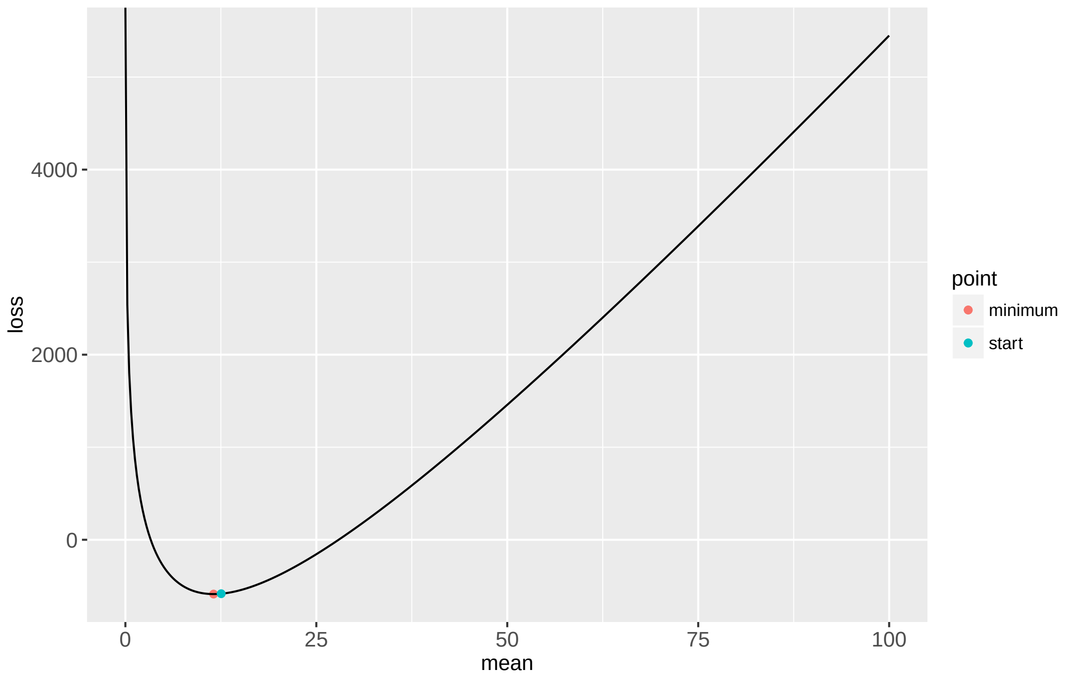


Our goal is to find the two roots of this function. Newton's root finding method starts from an arbitrary candidate root, and then repeatedly uses linear approximations to find more accurate candidate roots. To compute the linear approximation, we need the derivative:

```
loss.deriv <- function(Mean){
  Linear + Log/Mean
}
```

We begin the root finding at a point larger than the minimum,

```
possible.root <- mean.at.optimum+1
gg.loss+
  geom_point(aes(
    mean, loss, color=point),
    data=data.table(
      point="start",
      mean=possible.root,
      loss=loss.fun(possible.root)))
```



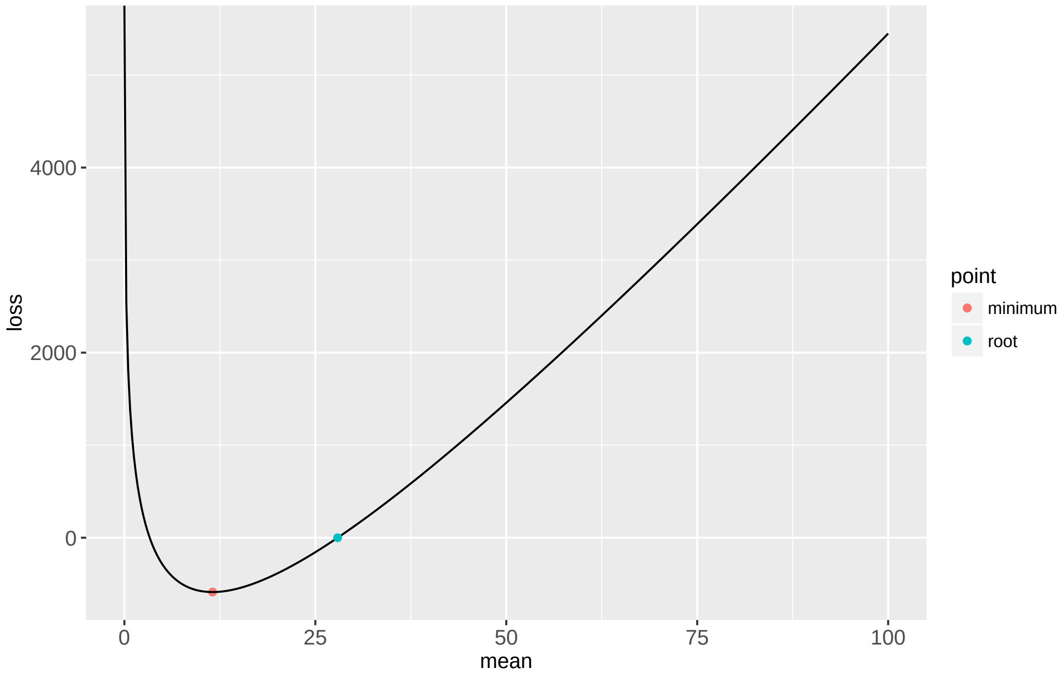
We then use the following implementation of Newton's method to find a root,

```
iteration <- 1
solution.list <- list()
thresh.dt <- data.table(thresh=1e-6)
while(thresh.dt$thresh < abs({
  fun.value <- loss.fun(possible.root)
})){
  cat(sprintf("mean=%e loss=%e\n", possible.root, fun.value))
  deriv.value <- loss.deriv(possible.root)
  new.root <- possible.root - fun.value/deriv.value
  solution.list[[iteration]] <- data.table(
    iteration, possible.root, fun.value, deriv.value, new.root)
  iteration <- iteration+1
  possible.root <- new.root
}
```

```
mean=1.254737e+01 loss=-5.828735e+02
mean=8.953188e+01 loss=4.574958e+03
mean=3.424363e+01 loss=3.768946e+02
mean=2.825783e+01 loss=1.901051e+01
mean=2.791944e+01 loss=7.929111e-02
```

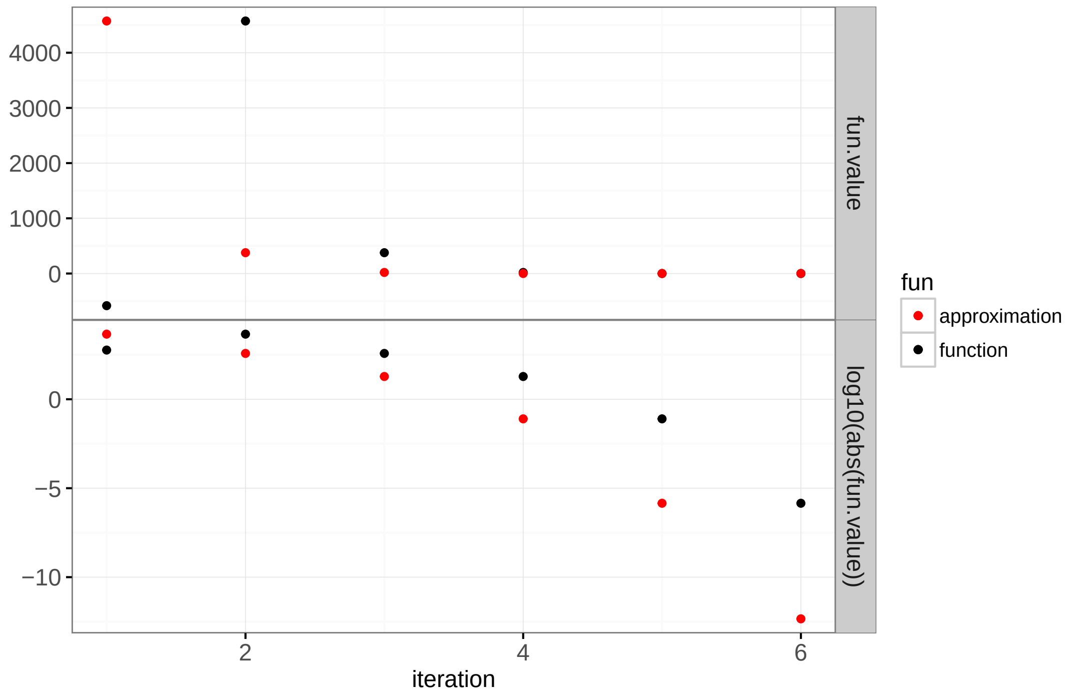
```
mean=2.791802e+01 loss=1.425562e-06
```

```
root.dt <- data.table(point="root", possible.root, fun.value)
gg.loss+
  geom_point(aes(possible.root, fun.value, color=point),
             data=root.dt)
```



The plot above shows the root that was found. The stopping criterion was an absolute cost value less than `1e-6` so we know that this root is at least that accurate. The following plot shows the accuracy of the root as a function of the number of iterations.

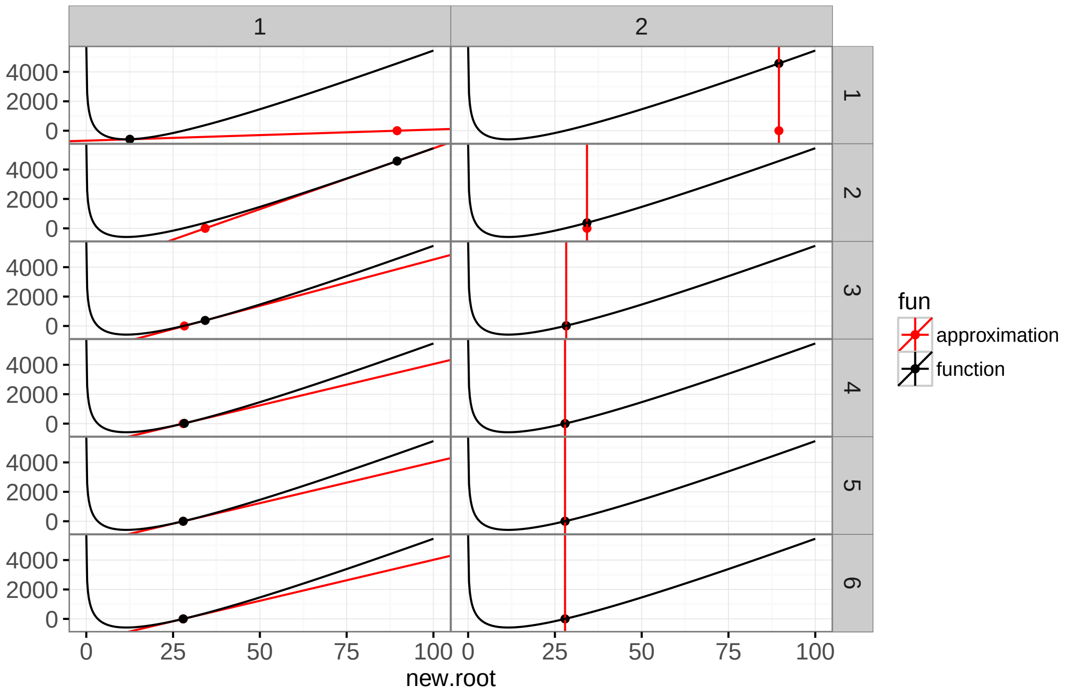
```
solution <- do.call(rbind, solution.list)
solution$new.value <- c(solution$fun.value[-1], fun.value)
gg.it <- ggplot()+
  geom_point(aes(
    iteration, fun.value, color=fun),
    data=data.table(solution, y="fun.value", fun="function"))+
  geom_point(aes(
    iteration, log10(abs(fun.value)), color=fun),
    data=data.table(
      solution, y="log10(abs(fun.value))", fun="function"))+
  scale_color_manual(values=c("function"="black", approximation="red"))+
  geom_point(aes(
    iteration, new.value, color=fun),
    data=data.table(solution, y="fun.value", fun="approximation"))+
  geom_point(aes(
    iteration, log10(abs(new.value)), color=fun),
    data=data.table(
      solution, y="log10(abs(fun.value))", fun="approximation"))+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(y ~ ., scales="free")+
  ylab("")
print(gg.it)
```



The plot above shows a horizontal line for the stopping criterion threshold, on the log scale. It is clear that the red dot in the last iteration is much below that threshold.

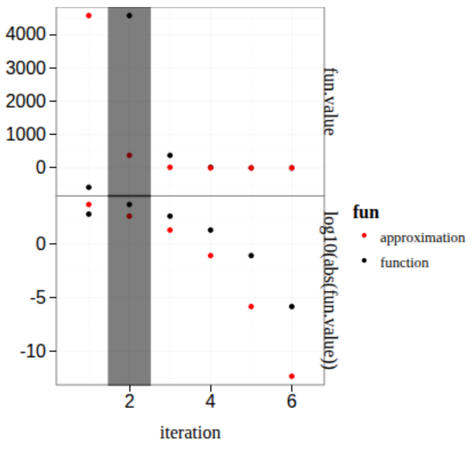
The plot below shows each step of the algorithm. The left panels show the linear approximation at the candidate root, along with the root of the linear approximation. The right panels show the root of the linear approximation, along with the corresponding function value (the new candidate root).

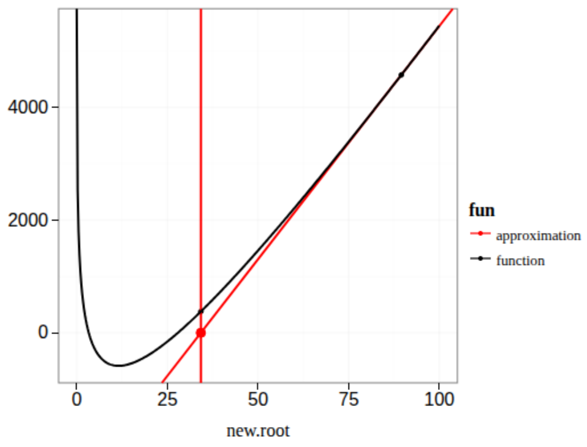
```
## y - fun.value = deriv.value * (x - possible.root)
## y = deriv.value*x + fun.value-possible.root*deriv.value
ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(iteration ~ step)+
  scale_color_manual(values=c("function"="black", approximation="red"))+
  geom_abline(aes(
    slope=deriv.value, intercept=fun.value-possible.root*deriv.value,
    color=fun),
    data=data.table(solution, fun="approximation", step=1))+
  geom_point(aes(
    new.root, 0, color=fun),
    data=data.table(solution, fun="approximation"))+
  geom_point(aes(
    new.root, new.value, color=fun),
    data=data.table(solution, fun="function", step=2))+
  geom_vline(aes(
    xintercept=new.root, color=fun),
    data=data.table(solution, fun="approximation", step=2))+
  geom_point(aes(
    possible.root, fun.value, color=fun),
    data=data.table(solution, fun="function", step=1))+
  geom_line(aes(
    mean, loss, color=fun),
    data=data.table(loss.dt, fun="function"))+
  ylab("")
```



It is clear that the algorithm quickly converges to the root. The following is an animated interactive version of the same data viz.

```
animint(
  time=list(variable="iteration", ms=2000),
  iterations=gg.it+
    theme_animint(width=300, colspan=1)+
    geom_tallrect(aes(
      xmin=iteration-0.5,
      xmax=iteration+0.5),
      clickSelects="iteration",
      alpha=0.5,
      data=solution),
  loss=ggplot()+
    theme_bw()+
    scale_color_manual(values=c(
      "function"="black",
      approximation="red"))+
    geom_abline(aes(
      slope=deriv.value, intercept=fun.value-possible.root*deriv.value,
      color=fun),
      showSelected="iteration",
      data=data.table(solution, fun="approximation"))+
    geom_point(aes(
      new.root, 0, color=fun),
      showSelected="iteration",
      size=4,
      data=data.table(solution, fun="approximation"))+
    geom_point(aes(
      new.root, new.value, color=fun),
      showSelected="iteration",
      data=data.table(solution, fun="function"))+
    geom_vline(aes(
      xintercept=new.root, color=fun),
      showSelected="iteration",
      data=data.table(solution, fun="approximation"))+
    geom_point(aes(
      possible.root, fun.value, color=fun),
      showSelected="iteration",
      data=data.table(solution, fun="function"))+
    geom_line(aes(
      mean, loss, color=fun),
      data=data.table(loss.dt, fun="function"))+
    ylab(""))
```





15.2 Comparison with Lambert W solution

The code below uses the Lambert W function to compute a root, and compares its solution to the one we computed using Newton's method.

```
inside <- Linear*exp(-Constant/Log)/Log
root.vec <- Log/Linear*c(
  LambertW::W(inside, 0),
  LambertW::W(inside, -1))
```

```
Registered S3 methods overwritten by 'ggplot2':
  method                      from
  drawDetails.zeroGrob        animint2
  grobHeight.absoluteGrob     animint2
  grobHeight.zeroGrob         animint2
  grobWidth.absoluteGrob      animint2
  grobWidth.zeroGrob          animint2
  grobX.absoluteGrob          animint2
  grobY.absoluteGrob          animint2
  heightDetails.titleGrob     animint2
  heightDetails.zeroGrob      animint2
  makeContext.dotstackGrob    animint2
  print.ggplot2_bins          animint2
  print.rel                   animint2
  widthDetails.titleGrob      animint2
  widthDetails.zeroGrob       animint2
```

```
loss.fun(c(
  Newton=root.dt$possible.root,
  Lambert=root.vec[2]))
```

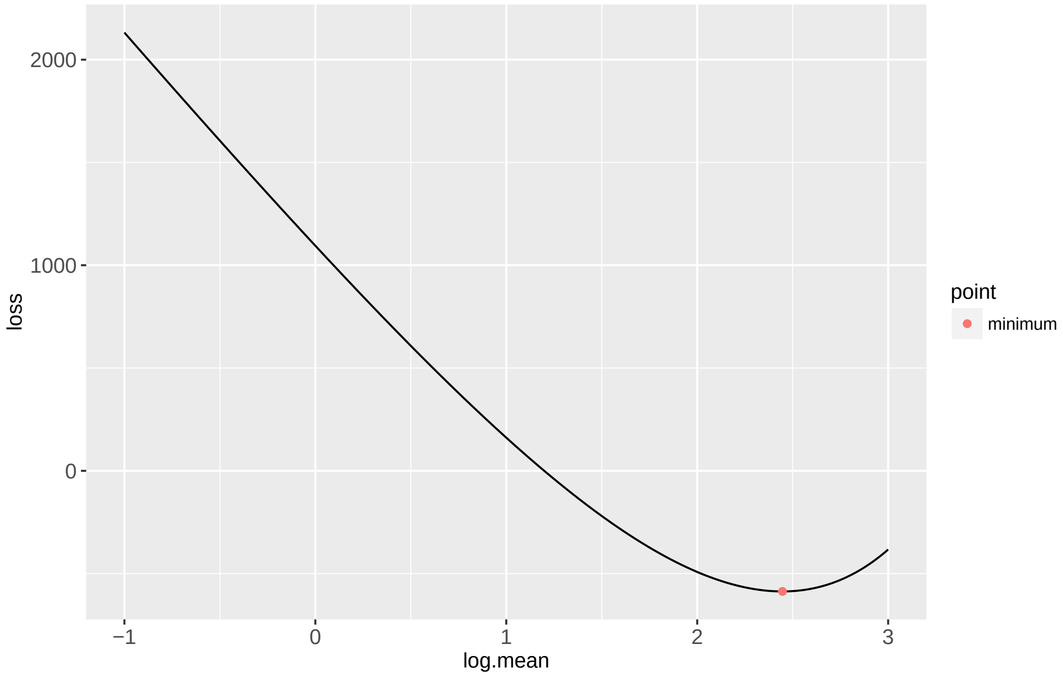
```
       Newton       Lambert
-4.547474e-13  0.000000e+00
```

For these data, the Lambert W function yields a root which is slightly more accurate than our implementation of Newton's method.

15.3 Root-finding in the log space

The previous section showed an algorithm for finding the root which is larger than the minimum. In this section we explore an algorithm for finding the other root (smaller than the minimum). Note that the Poisson loss is highly non-linear as mean goes to zero, so the linear approximation in the Newton root finding will not work very well. Instead, we equivalently perform root finding in the log space:

```
log.loss.fun <- function(log.mean){
  Linear*exp(log.mean) + Log*log.mean + Constant
}
log.loss.dt <- data.table(
  log.mean=seq(-1, 3, l=400)
)[
, loss := log.loss.fun(log.mean)]
ggplot()+
  geom_line(aes(
    log.mean, loss),
    data=log.loss.dt)+
  geom_point(aes(
    log(mean), loss, color=point),
    data=opt.dt)
```



Exercise: derive and implement the Newton method for this function, in order to find the root that is smaller than the minimum. Create an animint similar to the previous section.

15.4 Chapter summary and exercises

In this chapter we explored several visualizations of the Newton method for finding roots of smooth functions.

Exercises:

- Add a title to each plot.

- Add a size legend to the first plot (black points larger than red), so that we can see when red and black have the same value.
- Add a `geom_hline` to emphasize the loss=0 value in the second plot.
- Add a `geom_hline` to emphasize the stopping threshold in the first plot.
- Turn off one of the two legends, to save space.
- How to specify smooth transitions between iterations?
- Instead of using iteration as the animation/time variable, create a new one in order to show two distinct states/steps for each iteration, i.e. the `step` variable in the facetted plot above.
- What happens to the rate of convergence when you try to find the larger root in the log space, or the smaller root in the original space? Theoretically it should not converge as fast, since the functions are more nonlinear for those roots. Make a data visualization that allows you to select the starting value, and shows how many iterations it takes to converge to within the threshold.
- Create another plot that allows you to select the threshold. Plot the number of iterations as a function of threshold.
- Derive the loss function for Binomial regression, and visualize the corresponding Newton root finding method.
- Refactor `gg.it` code to use only one `geom_point`, instead of the four geoms in the current code. Hint: use `rbind()` to create a single table with all of the data.

Next, Chapter 16 explains how to visualize change-point detection models.

16

Supervised change-point detection

In this chapter we will explore several data visualizations of supervised changepoint detection models.

Chapter outline:

- We begin by making several static visualizations of the `intreg` data set.
- We then create an interactive visualization in which one plot can be click to select the number of changepoints/segments, and the other plot shows the corresponding model.
- We end by showing a static visualization of the max margin linear regression model, and suggesting exercises about creating an interactive version.

16.1 Static figures

We begin by loading the `intreg` data set.

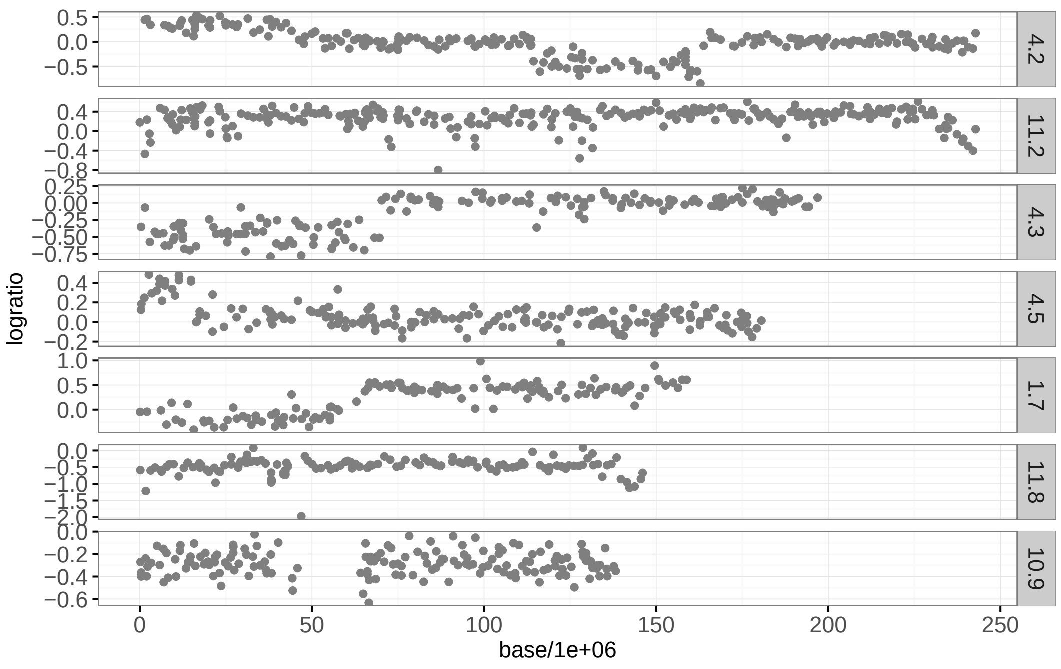
```r
library(animint2)
data(intreg)
library(data.table)
lapply(intreg, function(df)data.table(df)[1:2])
```

```
$model
         line      min.L    max.L min.feature max.feature
1: regression -1.4001088 2.139100   -2.476391   -1.584656
2:      limit -0.1493744 3.389835   -2.476391   -1.584656

$annotations
   signal first.base last.base   annotation  logratio
1:   11.2   55103411 161558770 0breakpoints 0.9847716
2:    4.2  140080934 201712984  1breakpoint 0.9847716

$intervals
   signal   feature       min.L    max.L
1:    4.2 -2.152421 -1.36503599 1.136433
2:   11.2 -1.797948  0.04183316      Inf

$selection
   signal     min.L     max.L segments cost
1:    4.2      -Inf -3.566634       20    2
2:    4.2 -3.566634 -3.301154       19    2
```

```
$segments
   signal segments first.base last.base        mean
1:    4.2        1    1472476 242801018 -0.02092153
2:    4.2        2    1472476  45164626  0.35123108

$breaks
   signal      base segments
1:    4.2  45164626        2
2:    4.2 114042112        3

$signals
   signal    base  logratio
1:    4.2 1472476 0.4404207
2:    4.2 2063049 0.4594316
```

As shown above, it is a named list of 7 related data.frames. We begin our exploration of these data by plotting the signals in separate facets.

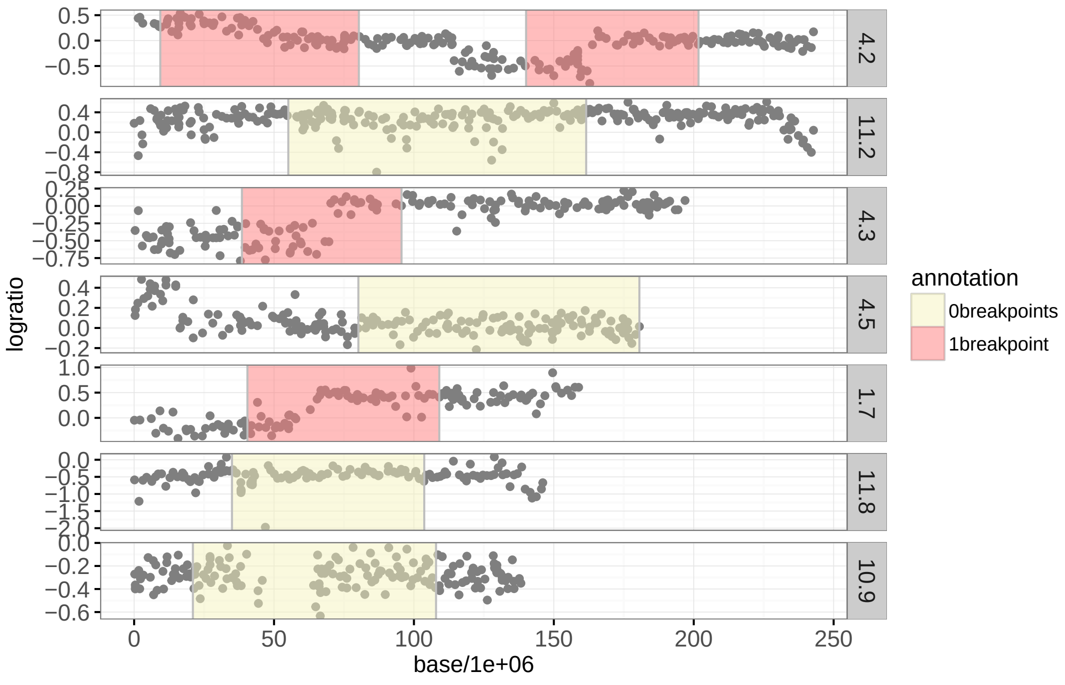
```
data.color <- "grey50"
gg.signals <- ggplot()+
  theme_bw()+
  facet_grid(signal ~ ., scales="free")+
  geom_point(aes(
    base/1e6, logratio,
    showSelected="signal"),
    color=data.color,
    data=intreg$signals)
gg.signals
```



Each data point plotted above shows an approximate measurement of DNA copy number (logratio), as a function of base position on a chromosome. Such data come from high-throughput assays which are important for diagnosing certain types of cancer such as neuroblastoma.

An important part of the diagnosis is detecting "breakpoints" or abrupt changes, within a given chromosome (panel). It is clear from the plot above that there are several breakpoint in these data. In particular signal 4.2 appears to have three breakpoints, signal 4.3 appears to have one, etc. In fact these data come from medical doctors at the Institute Curie (Paris, France) who have visually annotated regions with and without breakpoints. These data are available as `intreg$annotations` and are plotted below.

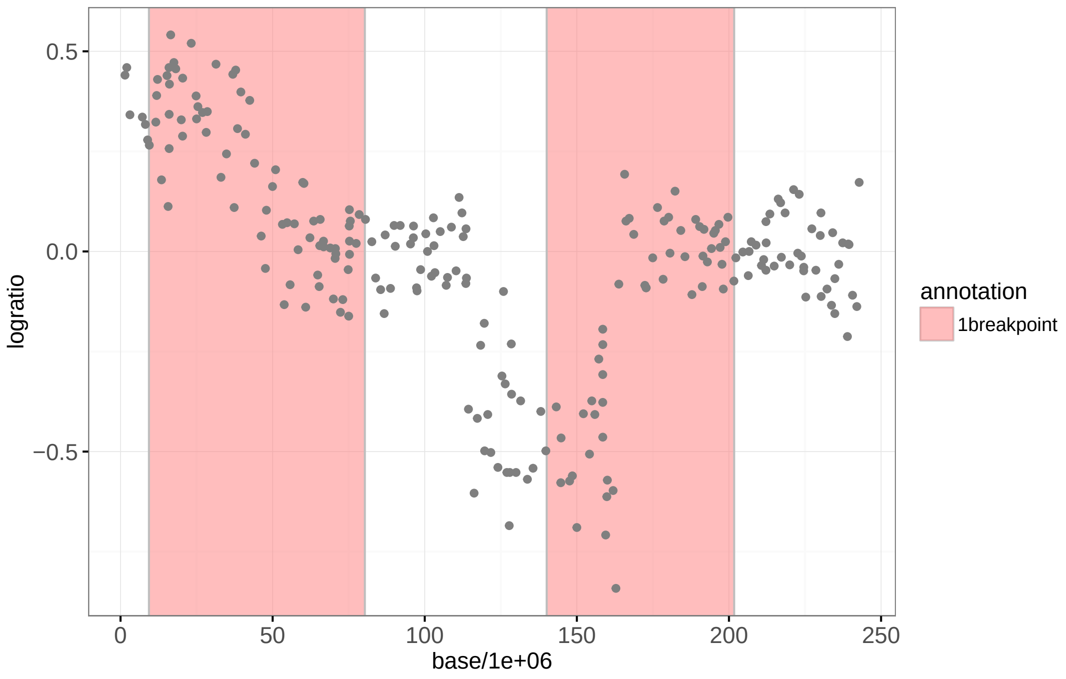
```
breakpoint.colors <- c("1breakpoint"="#ff7d7d", "0breakpoints"='#f6f4bf')
gg.ann <- gg.signals+
  scale_fill_manual(values=breakpoint.colors)+
  geom_tallrect(aes(
    xmin=first.base/1e6, xmax=last.base/1e6,
    fill=annotation),
    color="grey",
    alpha=0.5,
    data=intreg$annotations)
gg.ann
```



The plot above shows yellow regions where the doctors have determined that there are no significant breakpoints, and red regions where there is one breakpoint. The goal in analyzing these data is to learn from the limited labeled data (colored regions) and provide consistent breakpoint predictions throughout (even in un-labeled regions).

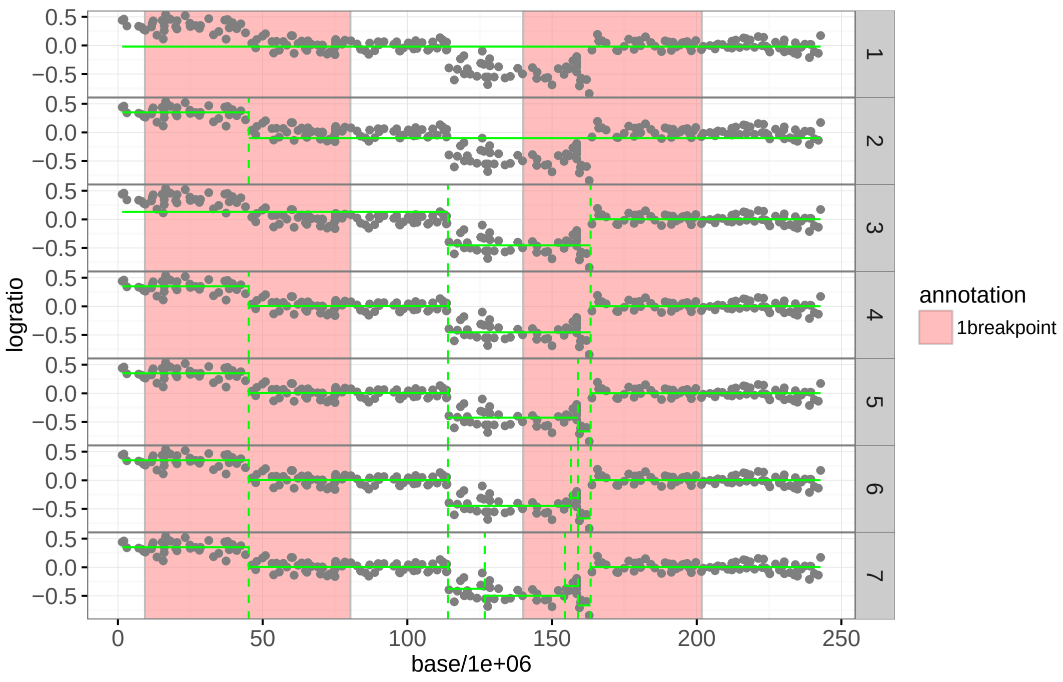
In order to detect these breakpoints we have fit some maximum likelihood segmentation models, using the efficient algorithm implemented in `jointseg::Fpsn`. The segment means are available in `intreg$segments` and the predicted breakpoints are available in `intreg$breaks`. For each signal there is a sequence of models from 1 to 20 segments. First let's zoom in on one signal:

```r
sig.name <- "4.2"
show.segs <- 7
sig.labels <- subset(intreg$annotations, signal==sig.name)
gg.one <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  geom_tallrect(aes(
    xmin=first.base/1e6, xmax=last.base/1e6,
    fill=annotation),
    color="grey",
    alpha=0.5,
    data=sig.labels)+
  geom_point(aes(
    base/1e6, logratio),
    color=data.color,
    data=subset(intreg$signals, signal==sig.name))+
  scale_fill_manual(values=breakpoint.colors)
gg.one
```



We plot some of these models for one of the signals below:

```r
sig.segs <- data.table(
  intreg$segments)[signal == sig.name & segments <= show.segs]
sig.breaks <- data.table(
    intreg$breaks)[signal == sig.name & segments <= show.segs]
model.color <- "green"
gg.models <- gg.one+
  facet_grid(segments ~ .)+
  geom_segment(aes(
    first.base/1e6, mean,
    xend=last.base/1e6, yend=mean),
    color=model.color,
    data=sig.segs)+
  geom_vline(aes(
    xintercept=base/1e6),
    color=model.color,
    linetype="dashed",
    data=sig.breaks)
gg.models
```



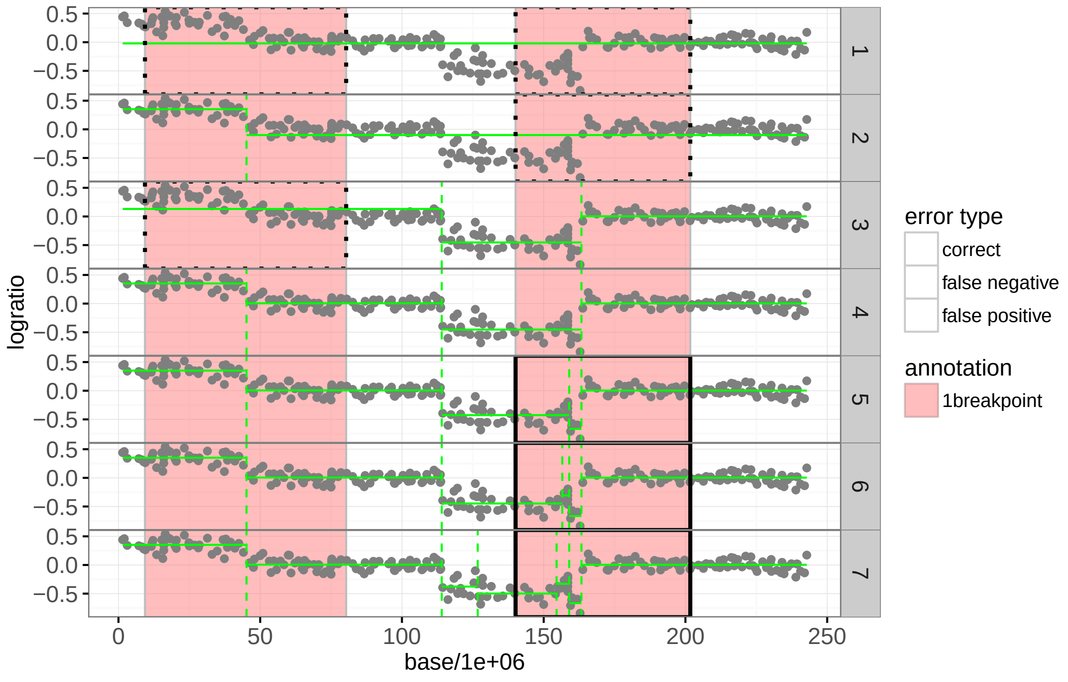
The plot above shows the maximum likelihood segmentation models in green (from one to six segments). Below we use the `penaltyLearning::labelError` function to compute the label error, which quantifies which models agree with which labels.

```
sig.models <- data.table(segments=1:show.segs, signal=sig.name)
sig.errors <- penaltyLearning::labelError(
  sig.models, sig.labels, sig.breaks,
  change.var="base",
  label.vars=c("first.base", "last.base"),
  model.vars="segments",
  problem.vars="signal")
```

```
Registered S3 methods overwritten by 'ggplot2':
  method                     from
  drawDetails.zeroGrob       animint2
  grobHeight.absoluteGrob    animint2
  grobHeight.zeroGrob        animint2
  grobWidth.absoluteGrob     animint2
  grobWidth.zeroGrob         animint2
  grobX.absoluteGrob         animint2
  grobY.absoluteGrob         animint2
  heightDetails.titleGrob    animint2
  heightDetails.zeroGrob     animint2
  makeContext.dotstackGrob   animint2
  print.ggplot2_bins         animint2
  print.rel                  animint2
  widthDetails.titleGrob     animint2
  widthDetails.zeroGrob      animint2
```

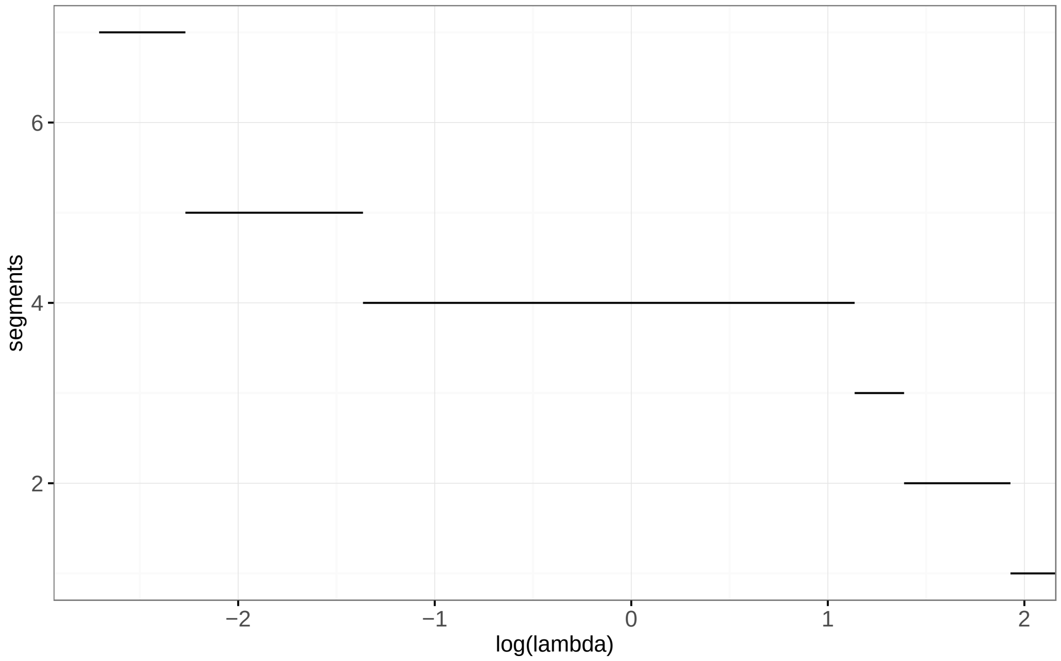
The `sig.errors$label.errors` data.table contains one row for every (model,label) combination. The `status` column can be used to show the label error: `false negative` for too few changes, `false positive` for too many changes, or `correct` for the right number of changes.

```r
gg.models+
  geom_tallrect(aes(
    xmin=first.base/1e6, xmax=last.base/1e6,
    linetype=status),
    data=sig.errors$label.errors,
    color="black",
    size=1,
    fill=NA)+
  scale_linetype_manual(
    "error type",
    values=c(
      correct=0,
      "false negative"=3,
      "false positive"=1))
```



Looking at the label error plot above, it is clear that the model with four segments should be selected, because it achieves zero label errors. There are a number of criteria that can be used to select which one of these models is best. One way to do that is by selecting the model with s segments is $S^*(\lambda) = L_s + \lambda * s$, where L_s is the total loss of the model with s segments, and λ is a non-negative penalty. In the plot below we show the model selection function $S^*(\lambda)$ for this data set:

```
sig.selection <- data.table(
  intreg$selection)[signal == sig.name & segments <= show.segs]
gg.selection <- ggplot()+
  theme_bw()+
  geom_segment(aes(
    min.L, segments,
    xend=max.L, yend=segments),
    data=sig.selection)+
  xlab("log(lambda)")
gg.selection
```

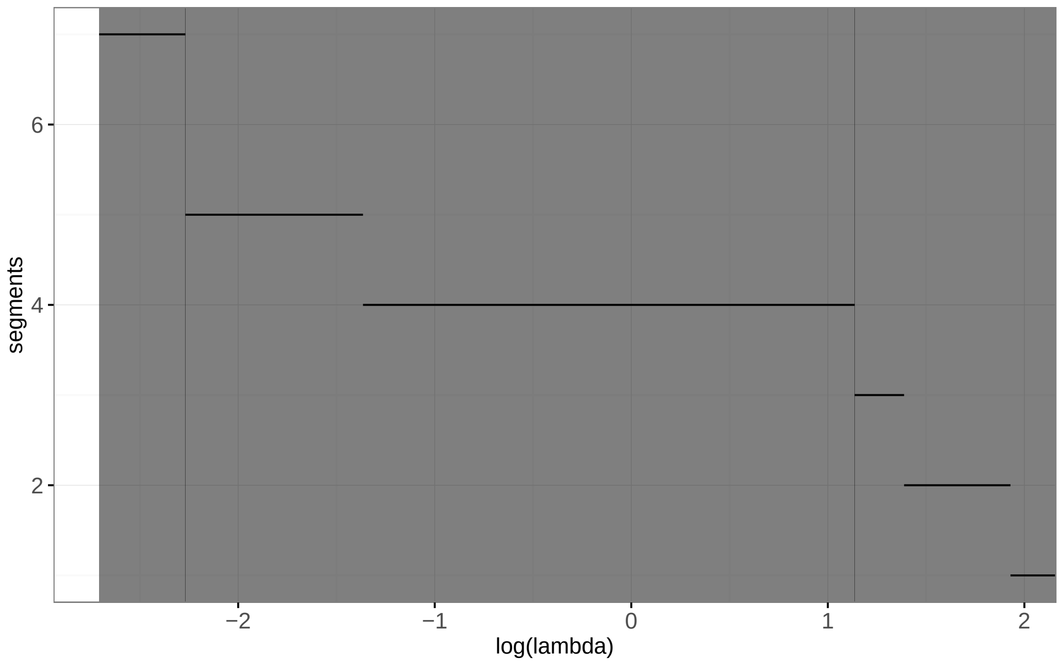


It is clear from the plot above that the model selection function is decreasing. In the next section we make an interactive version of these two plots where we can actually click on the model selection plot in order to select the model.

16.2 Interactive figures for one signal

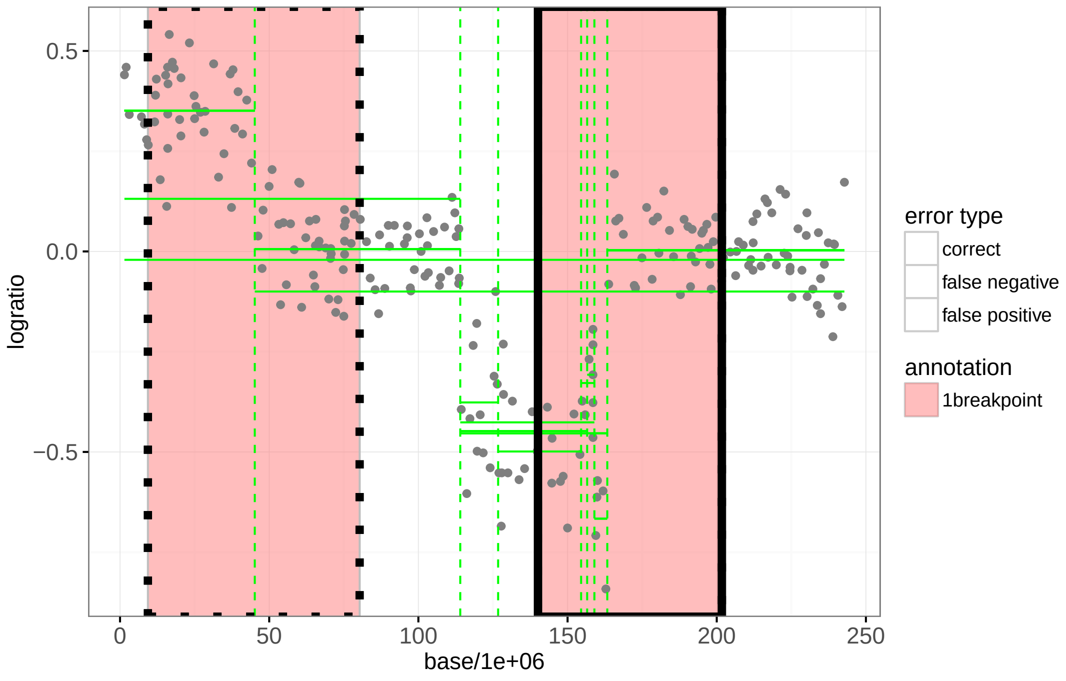
We will create an interactive figure for one signal by adding a `geom_tallrect` with `clickSelects=segments` to the plot above:

```
interactive.selection <- gg.selection+
  geom_tallrect(aes(
    xmin=min.L, xmax=max.L),
    clickSelects="segments",
    data=sig.selection,
    color=NA,
    fill="black",
    alpha=0.5)
interactive.selection
```



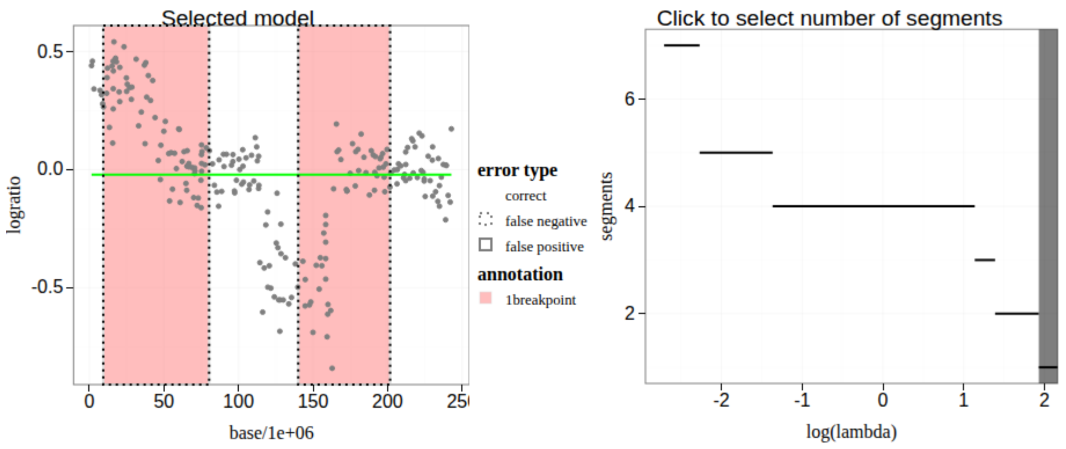
We will combine that with the non-facetted version of the data/models plot below, in which we have added `showSelected=segments` to the model geoms:

```
interactive.models <- gg.one+
  geom_segment(aes(
    first.base/1e6, mean,
    xend=last.base/1e6, yend=mean),
    showSelected="segments",
    color=model.color,
    data=sig.segs)+
  geom_vline(aes(
    xintercept=base/1e6),
    showSelected="segments",
    color=model.color,
    linetype="dashed",
    data=sig.breaks)+
  geom_tallrect(aes(
    xmin=first.base/1e6, xmax=last.base/1e6,
    linetype=status),
    showSelected="segments",
    data=sig.errors$label.errors,
    size=2,
    color="black",
    fill=NA)+
  scale_linetype_manual(
    "error type",
    values=c(
      correct=0,
      "false negative"=3,
      "false positive"=1))
interactive.models
```



Of course the plot above is not very informative because it is not interactive. Below we combine the two interactive ggplots in a single linked animint:

```
animint(
  models=interactive.models+
    ggtitle("Selected model"),
  selection=interactive.selection+
    ggtitle("Click to select number of segments"))
```

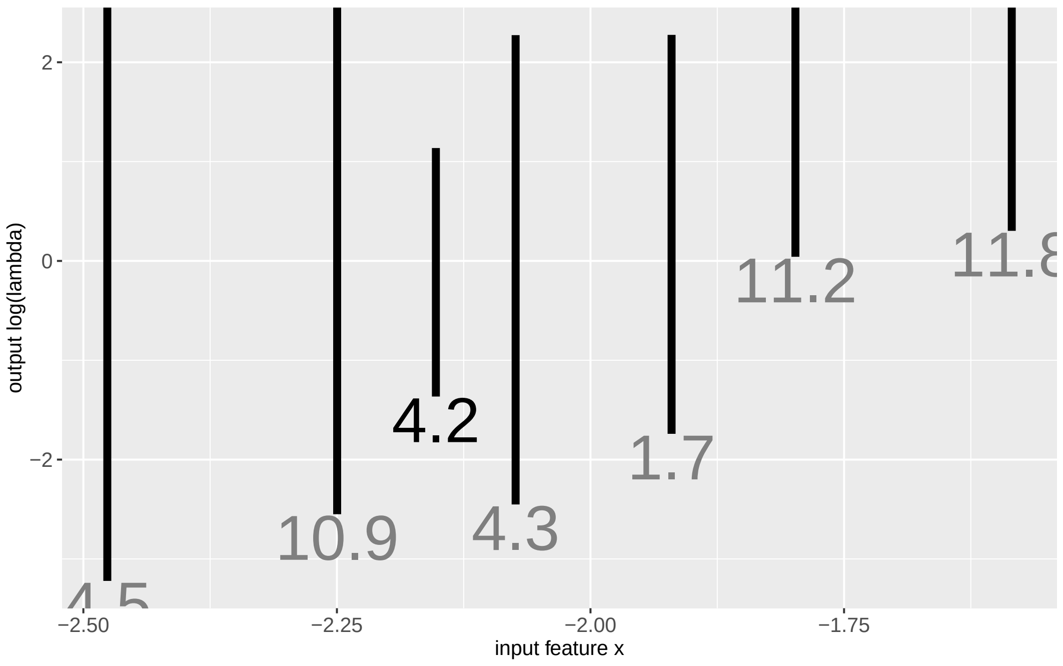


Note that in the data viz above the model with 6 segments is not selectable for any value of lambda, so there is no way to click on the plot to select that model. However it is possible to select the model using the segments selection menu (click “Show selection menus” at the bottom of the data viz).

16.3 Static max margin regression plot

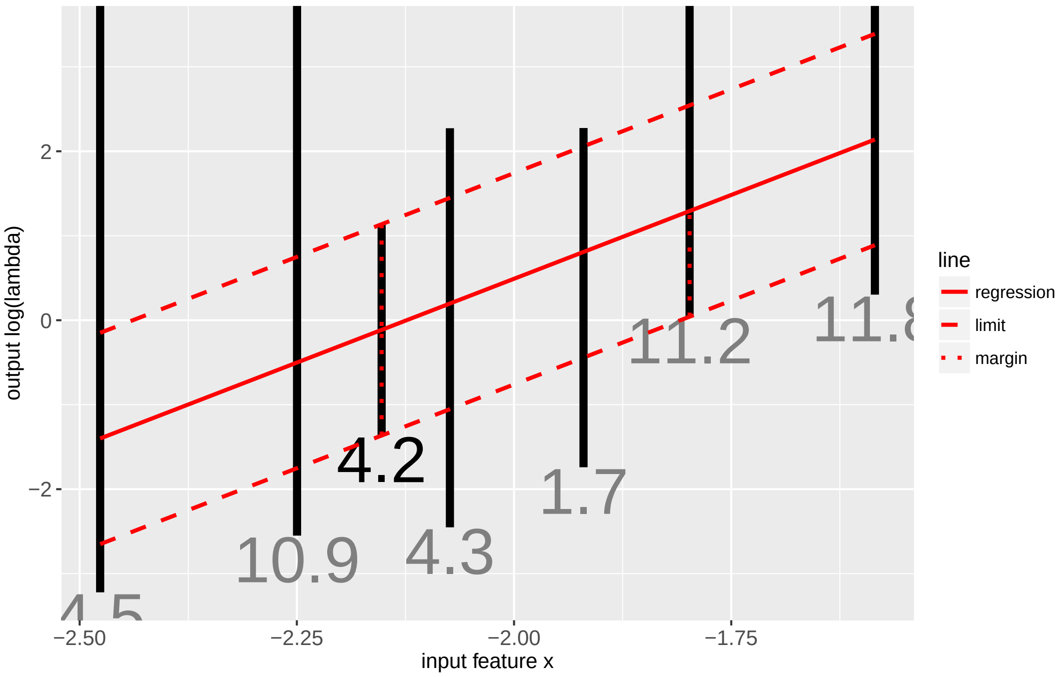
Another part of this data set is `intreg$intervals` which has one row for every signal. The columns `min.L` and `max.L` indicate the min/max values of the target interval, which is the largest range of log(penalty) values with minimum label errors. Below we plot this interval as a function of a feature of the data (log number of data points):

```r
gg.intervals <- ggplot()+
  geom_segment(aes(
    feature, min.L,
    xend=feature, yend=max.L),
    size=2,
    data=intreg$intervals)+
  geom_text(aes(
    feature, min.L, label=signal,
    color=ifelse(signal==sig.name, "black", "grey50")),
    vjust=1,
    data=intreg$intervals)+
  scale_color_identity()+
  ylab("output log(lambda)")+
  xlab("input feature x")
gg.intervals
```



The target intervals in the plot above denote the region of log(lambda) space that will select a model with minimum label errors. There is one interval for each signal; we made an animint in the previous section for the signal indicated in black text. Machine learning algorithms can be used to find a penalty function that intersects each of the intervals, and maximizes the margin (the distance between the regression function and the nearest interval limit). Data for the linear max margin regression function are in `intreg$model` which is shown in the plot below:

```r
gg.mm <- gg.intervals+
  geom_segment(aes(
    min.feature, min.L,
    xend=max.feature, yend=max.L,
    linetype=line),
    color="red",
    size=1,
    data=intreg$model)+
  scale_linetype_manual(
    values=c(
      regression="solid",
      margin="dotted",
      limit="dashed"))
gg.mm
```



The plot above shows the linear max margin regression function f(x) as the solid red line. It is clear that it intersects each of the black target intervals, and maximizes the margin (red vertical dotted lines). For more information on the subject of supervised changepoint detection, please see my useR 2017 tutorial.

Now that you know how to visualize each of the seven parts of the `intreg` data set, the rest of the chapter is devoted to exercises.

16.4 Chapter summary and exercises

Exercises:

- Add a `geom_text` which shows the currently selected signal name at the top of the plot, in `interactive.models` in the first animint above.
- Make an animint with two plots that shows the data set that corresponds to each interval on the max margin regression plot. One plot should show an interactive version of the max margin regression plot where you can click on an interval to select a signal. The other plot should show the data set for the currently selected signal.
- In the animint you created in the previous exercise, add a third plot with the model selection function for the currently selected signal.
- Re-design the previous animint so that instead of using a third plot, add a facet to the max margin regression regression plot such that the log(lambda) axes are aligned. Add another facet that shows the number of incorrect labels (`intreg$selection$cost`) for each log(lambda) value.
- Add geoms for selecting the number of segments. Clicking the model selection plot should select the number of segments, which should update the displayed model and label errors on the plot of the data for the currently selected signal. Furthermore add a visual indication of the selected model to the max margin regression plot. The result should look something like this.
- Make another data viz by starting with the facetted `gg.signals` plot in the beginning of this chapter. Add a plot that can be used to select the number of segments for each signal. For each signal in the facetted plot of the data, show the currently selected model for that signal (there should be a separate selection variable for each signal – you can use named clickSelects/showSelected as explained in Chapter 14). The result should look something like this.

Next, Chapter 17 explains how to visualize the K-means clustering algorithm.

17

K-means clustering

In this chapter we will explore several data visualizations of K-means clustering, which is an unsupervised learning algorithm.

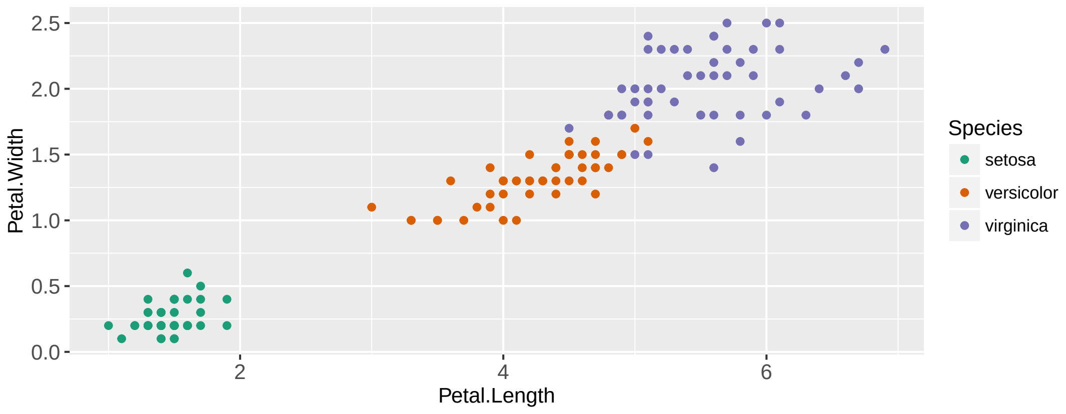
Chapter outline:

- We begin by visualizing two features of the iris data.
- We choose three random data points to use as cluster centers.
- We show how all distances between data points and cluster centers can be computed and visualized.
- We end by showing a visualization of how the k-means model parameters change with each iteration.

17.1 Visualize iris data with labels

We begin with a typical visualization of the iris data, including a color legend to indicate the Species.

```r
library(animint2)
color.code <- c(
  setosa="#1B9E77",
  versicolor="#D95F02",
  virginica="#7570B3",
  "1"="#E7298A",
  "2"="#66A61E",
  "3"="#E6AB02",
  "4"="#A6761D")
ggplot()+
  scale_color_manual(values=color.code)+
  geom_point(aes(
    Petal.Length, Petal.Width, color=Species),
    data=iris)+
  coord_equal()
```



We will illustrate the K-means clustering algorithm using these two dimensions,

```r
data.mat <- as.matrix(iris[,c("Petal.Width","Petal.Length")])
head(data.mat)
```

```
     Petal.Width Petal.Length
[1,]         0.2          1.4
[2,]         0.2          1.4
[3,]         0.2          1.3
[4,]         0.2          1.5
[5,]         0.2          1.4
[6,]         0.4          1.7
```

```r
str(data.mat)
```

```
 num [1:150, 1:2] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:2] "Petal.Width" "Petal.Length"
```

To run K-means, the number of clusters hyper-parameter (K) must be fixed in advance. Then K random data points are selected as the initial cluster centers,

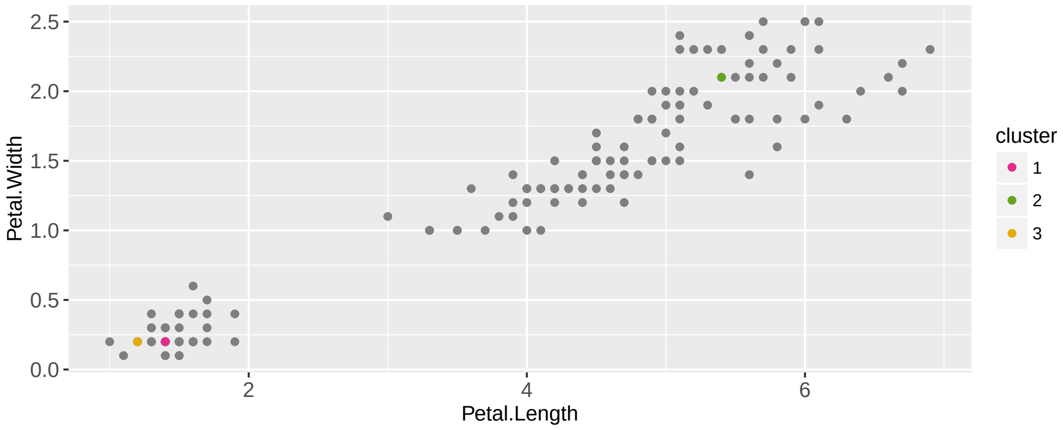
```r
K <- 3
library(data.table)
data.dt <- data.table(data.mat)
set.seed(3)
centers.dt <- data.dt[sample(1:.N, K)]
(centers.mat <- as.matrix(centers.dt))
```

```
     Petal.Width Petal.Length
[1,]         0.2          1.4
[2,]         2.1          5.4
[3,]         0.2          1.2
```

```r
centers.dt[, cluster := factor(1:K)]
centers.dt
```

```
   Petal.Width Petal.Length cluster
1:         0.2          1.4       1
2:         2.1          5.4       2
3:         0.2          1.2       3
```

```
gg.centers <- ggplot()+
  scale_color_manual(values=color.code)+
  geom_point(aes(
    Petal.Length, Petal.Width),
    color="grey50",
    data=data.dt)+
  geom_point(aes(
    Petal.Length, Petal.Width, color=cluster),
    data=centers.dt)+
  coord_equal()
gg.centers
```

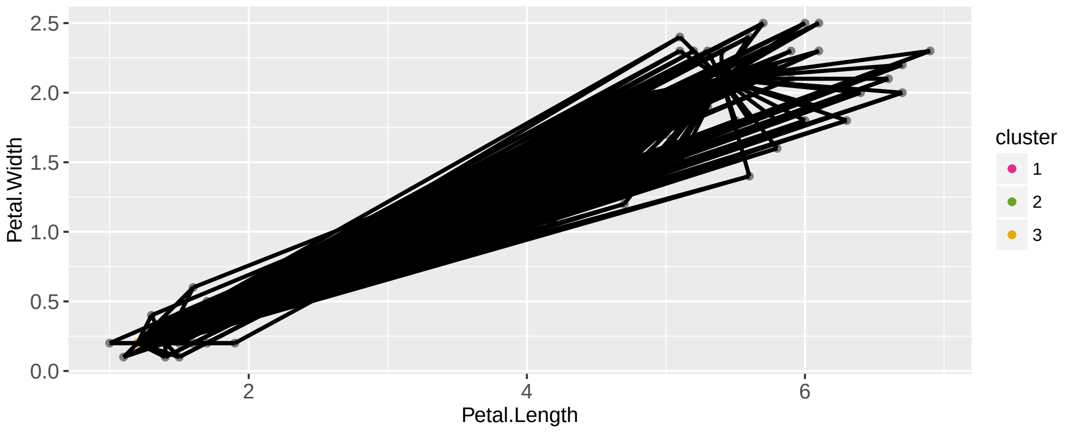


Above we displayed the two data sets (cluster centers and data) using two instances of `geom_point`. Below we compute the distance between each data point and each cluster center,

```
pairs.dt <- data.table(expand.grid(
  centers.i=1:nrow(centers.mat),
  data.i=1:nrow(data.mat)))
```

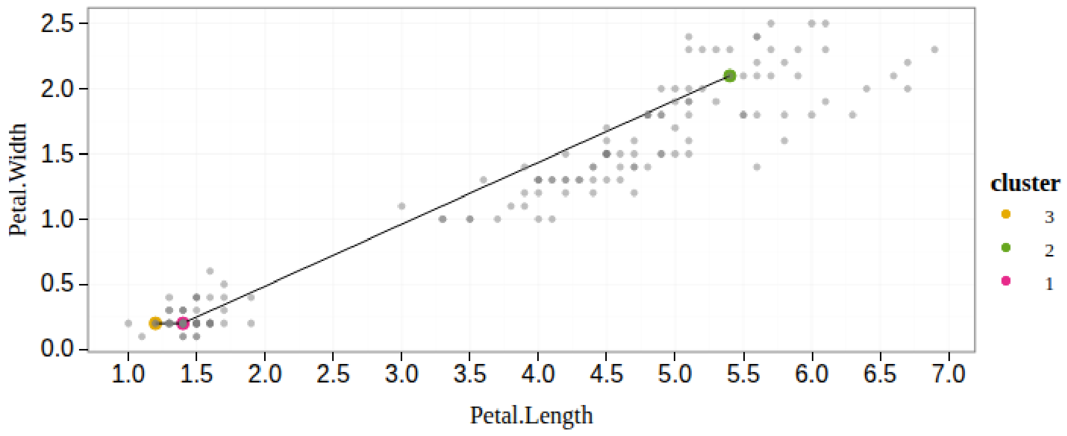
These can be visualized via a `geom_point`,

```
seg.dt <- pairs.dt[, data.table(
  data.i,
  data=data.mat[data.i,],
  center=centers.mat[centers.i,])]
gg.centers+
  geom_segment(aes(
    data.Petal.Length, data.Petal.Width,
    xend=center.Petal.Length, yend=center.Petal.Width),
    size=1,
    data=seg.dt)
```



There are 450 segments overplotted above, so interactivity would be useful to emphasize the segments connected to a particular data point. To do that we create a `data.i` selection variable,

```
animint(
  ggplot()+
    theme_bw()+
    theme_animint(height=300, width=640)+
    scale_color_manual(values=color.code)+
    scale_x_continuous(breaks=seq(1,7,by=0.5))+
    scale_y_continuous(breaks=seq(0, 2.5, by=0.5))+
    geom_point(aes(
      Petal.Length, Petal.Width, color=cluster),
      size=4,
      data=centers.dt)+
    geom_segment(aes(
      data.Petal.Length, data.Petal.Width,
      xend=center.Petal.Length, yend=center.Petal.Width),
      size=1,
      showSelected="data.i",
      data=seg.dt)+
    geom_point(aes(
      Petal.Length, Petal.Width),
      clickSelects="data.i",
      size=2,
      color="grey50",
      data=data.table(data.mat, data.i=1:nrow(data.mat))))
```



In the data viz above you can click on a data point to show the distances from that data point to each cluster center.

Exercises for this section:

- edit the x/y scales so that the same ticks are shown.
- change the color of each segment so that it matches the corresponding cluster.
- add a tooltip that shows the distance value.
- make the segment width depend on its optimality (segment connected to closest cluster center should be emphasized with greater width).

17.2 Visualizing iterations of algorithm

Next we compute the closest cluster center for each data point,

```
pairs.dt[, error := rowSums(
(data.mat[data.i,]-centers.mat[centers.i,])^2)]
(closest.dt <- pairs.dt[, .SD[which.min(error)], by=data.i])
```

```
     data.i centers.i error
  1:      1         1  0.00
  2:      2         1  0.00
 ---
149:    149         2  0.04
150:    150         2  0.18
```

```
(closest.data <- closest.dt[, .(
  data.dt[data.i],
  cluster=factor(centers.i)
)])
```

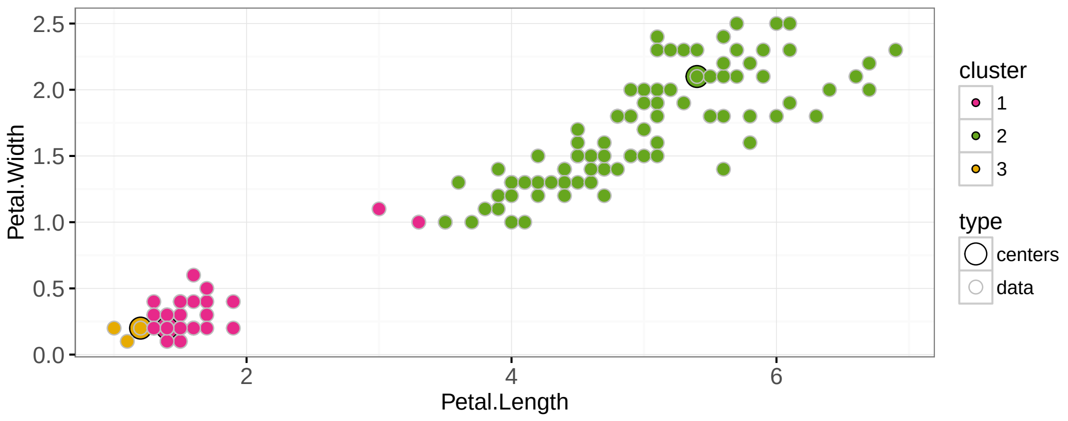
```
     Petal.Width Petal.Length cluster
  1:         0.2          1.4       1
  2:         0.2          1.4       1
 ---
```

```
149:          2.3          5.4       2
150:          1.8          5.1       2
```

```
(both.dt <- rbind(
  data.table(type="centers", centers.dt),
  data.table(type="data", closest.data)))
```

```
        type Petal.Width Petal.Length cluster
  1: centers         0.2          1.4       1
  2: centers         2.1          5.4       2
 ---
152:    data         2.3          5.4       2
153:    data         1.8          5.1       2
```

```
ggplot()+
  scale_fill_manual(values=color.code)+
  scale_color_manual(values=c(centers="black", data="grey"))+
  scale_size_manual(values=c(centers=5, data=3))+
  geom_point(aes(
    Petal.Length, Petal.Width, fill=cluster, size=type, color=type),
    shape=21,
    data=both.dt)+
  coord_equal()+
  theme_bw()
```

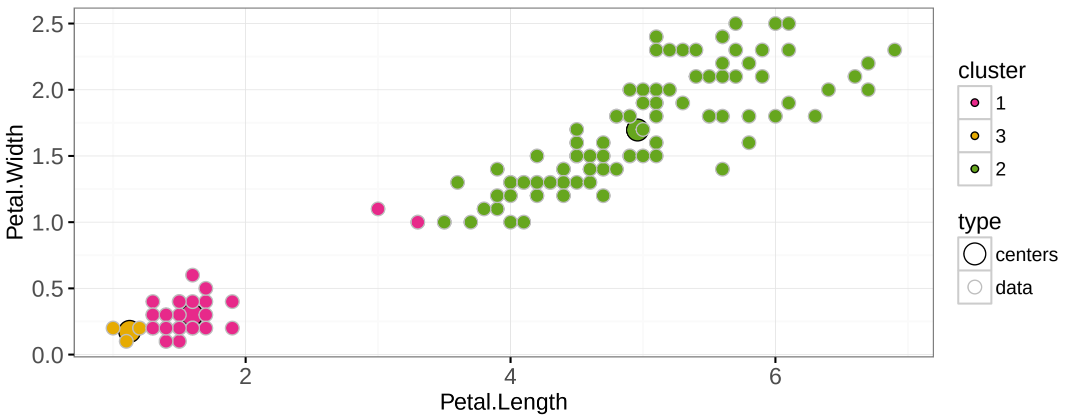


Then we update the cluster centers,

```
new.centers <- closest.dt[, data.table(
  t(colMeans(data.dt[data.i]))
), by=.(cluster=centers.i)]
(new.both <- rbind(
  data.table(type="centers", new.centers),
  data.table(type="data", closest.data)))
```

```
        type cluster Petal.Width Petal.Length
  1: centers       1       0.300     1.595918
  2: centers       3       0.175     1.125000
 ---
152:     data       2       2.300     5.400000
153:     data       2       1.800     5.100000
```

```
ggplot()+
  scale_fill_manual(values=color.code)+
  scale_color_manual(values=c(centers="black", data="grey"))+
  scale_size_manual(values=c(centers=5, data=3))+
  geom_point(aes(
    Petal.Length, Petal.Width, fill=cluster, size=type, color=type),
    shape=21,
    data=new.both)+
  coord_equal()+
  theme_bw()
```



So the visualizations above show the steps of k-means: (1) updating cluster assignment based on closest center, then (2) updating center based on data assigned to that cluster. To visualize several iterations of the above two steps, we can use a for loop,

```
set.seed(3)
centers.dt <- data.dt[sample(1:.N, K)]
(centers.mat <- as.matrix(centers.dt))
```

```
     Petal.Width Petal.Length
[1,]         0.2          1.4
[2,]         2.1          5.4
[3,]         0.2          1.2
```

```
data.and.centers.list <- list()
iteration.error.list <- list()
for(iteration in 1:20){
  pairs.dt[, error := {
    rowSums((data.mat[data.i,]-centers.mat[centers.i,])^2)
  }]
  closest.dt <- pairs.dt[, .SD[which.min(error)], by=data.i]
```

```
    iteration.error.list[[iteration]] <- data.table(
      iteration, error=sum(closest.dt[["error"]]))
    iteration.both <- rbind(
      data.table(type="centers", centers.dt, cluster=1:K),
      closest.dt[, data.table(
        type="data", data.dt[data.i], cluster=factor(centers.i))])
    data.and.centers.list[[iteration]] <- data.table(
      iteration, iteration.both)
    new.centers <- closest.dt[, data.table(
      t(colMeans(data.dt[data.i]))
    ), keyby=.(cluster=centers.i)]
    centers.dt <- new.centers[, names(centers.dt), with=FALSE]
    centers.mat <- as.matrix(centers.dt)
  }
  (data.and.centers <- do.call(rbind, data.and.centers.list))
```

```
      iteration    type Petal.Width Petal.Length cluster
   1:         1 centers         0.2          1.4       1
   2:         1 centers         2.1          5.4       2
  ---
3059:        20    data         2.3          5.4       2
3060:        20    data         1.8          5.1       2
```

```
  (iteration.error <- do.call(rbind, iteration.error.list))
```

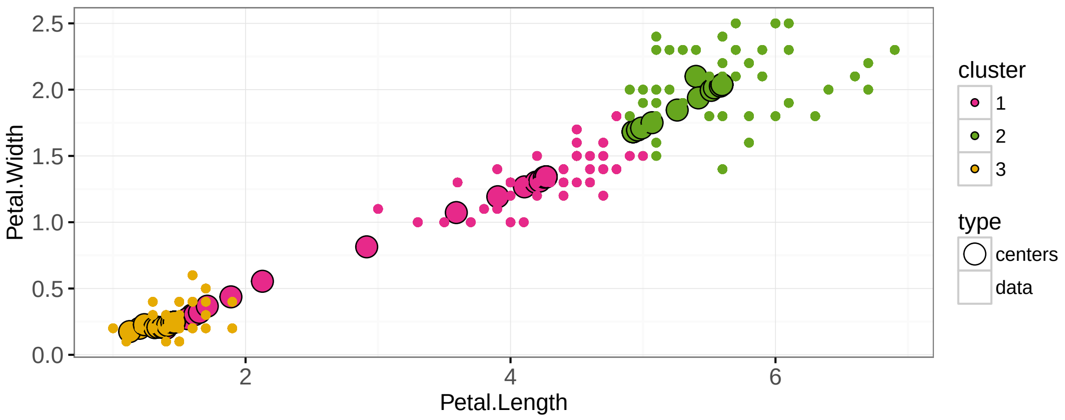
```
    iteration     error
 1:         1 123.63000
 2:         2  85.82705
---
19:        19  31.37136
20:        20  31.37136
```

First we create an overview plot with an error curve that will be used to select the model size,

```
  gg.err <- ggplot()+
    theme_bw()+
    geom_point(aes(
      iteration, error),
      data=iteration.error)+
    make_tallrect(iteration.error, "iteration", alpha=0.3)
```

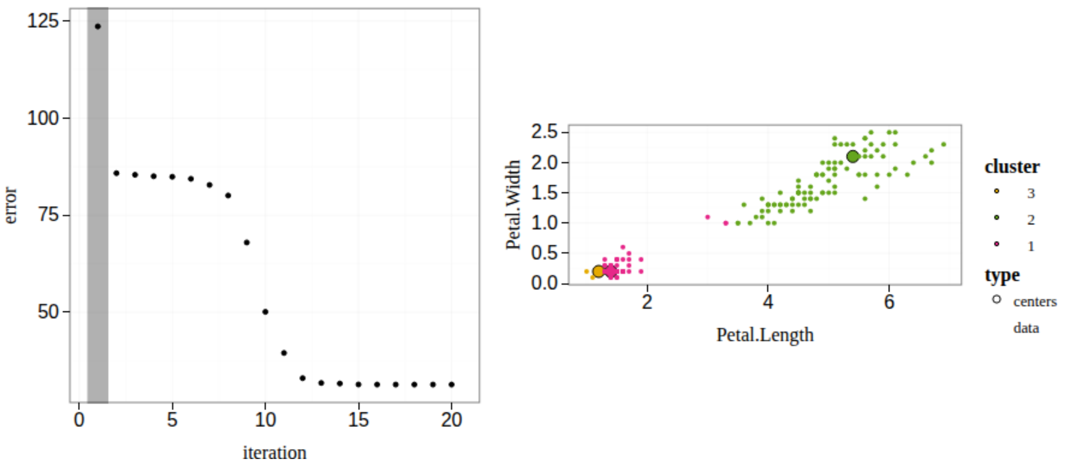
We also make a plot which will show the current iteration,

```
  gg.iteration <- ggplot()+
    scale_fill_manual(values=color.code)+
    scale_color_manual(values=c(centers="black", data=NA))+
    scale_size_manual(values=c(centers=5, data=2))+
    geom_point(aes(
      Petal.Length, Petal.Width, fill=cluster, size=type, color=type),
      shape=21,
      showSelected="iteration",
      data=data.and.centers)+
    coord_equal()+
    theme_bw()
  gg.iteration
```



Combining the two plots results in an interactive data viz,

```
  animint(gg.err, gg.iteration)
```



17.3 Chapter summary and exercises

Exercises:

- Make centers always show up in front (on top) of the data.
- Add smooth transitions.

- Add animation on iteration variable.
- Current code has fixed max number of iterations, so it is possible for the last few iterations to make no progress. For example in the viz above, iteration=16 was the last one that resulted in a decrease in error (iterations 17-20 resulted in no decrease). Modify the code so that it stops iterating if there is no decrease in error.
- Current viz has only one animation frame (showSelected subset) per iteration (the mean shown is before it is updated). Add another animation frame that shows the mean after the update.
- Add interactive segments that show the distance from each data point to each cluster center (as in first animint on this page).
- Add the features described in the exercises in the previous section on this page.
- Compute results for several different random seeds, then display error rates for each seed on the error overview plot, and allow the user to select any of those results.
- Compute results for several different numbers of clusters (K). Compute the Adjusted Rand Index using `pdfCluster::adj.rand.index(species, cluster)` for each different K and seed. Add an overview plot that shows the ARI value of each model, and allows selecting the number of clusters.
- Make a similar visualization using another data set such as `data("penguins", package="palmerpenguins")`.

Next, Chapter 18 explains how to visualize the gradient descent learning algorithm for neural network learning.

18

Neural networks

In this chapter we will explore several data visualizations of the gradient descent learning algorithm for Neural networks.

Chapter outline:

- We begin by simulating and visualizing some 2d data for binary classification.
- We then show how a classification function in 2d can be visualized by computing predictions on a grid, and then using `geom_tile` or `geom_path` with contour lines.
- We compute linear model predictions, and gradient descent updates, using a simple automatic differentiation (auto-grad) system.
- We end by implementing gradient descent for a neural network, and using an interactive data visualization to show how the predictions get more accurate with iterations of the learning algorithm.

18.1 Visualize simulated data

In this section, we simulate a simple data set with a non-linear pattern for binary classification.

```r
sim.col <- 2
sim.row <- 100
set.seed(1)
features.hidden <- matrix(runif(sim.row*sim.col), sim.row, sim.col)
head(features.hidden)
```

```
          [,1]      [,2]
[1,] 0.2655087 0.6547239
[2,] 0.3721239 0.3531973
[3,] 0.5728534 0.2702601
[4,] 0.9082078 0.9926841
[5,] 0.2016819 0.6334933
[6,] 0.8983897 0.2132081
```

In the simulation, the data table above has the “hidden” features which are used to create the labels, but are not available for learning. The latent/true function used for classification is the following,

```r
bayes <- function(DT)DT[, (V1>0.2 & V2<0.8)]
library(data.table)
```

```r
hidden.dt <- data.table(features.hidden)
label.vec <- ifelse(bayes(hidden.dt), 1, -1)
table(label.vec)
```

```
label.vec
-1  1
31 69
```

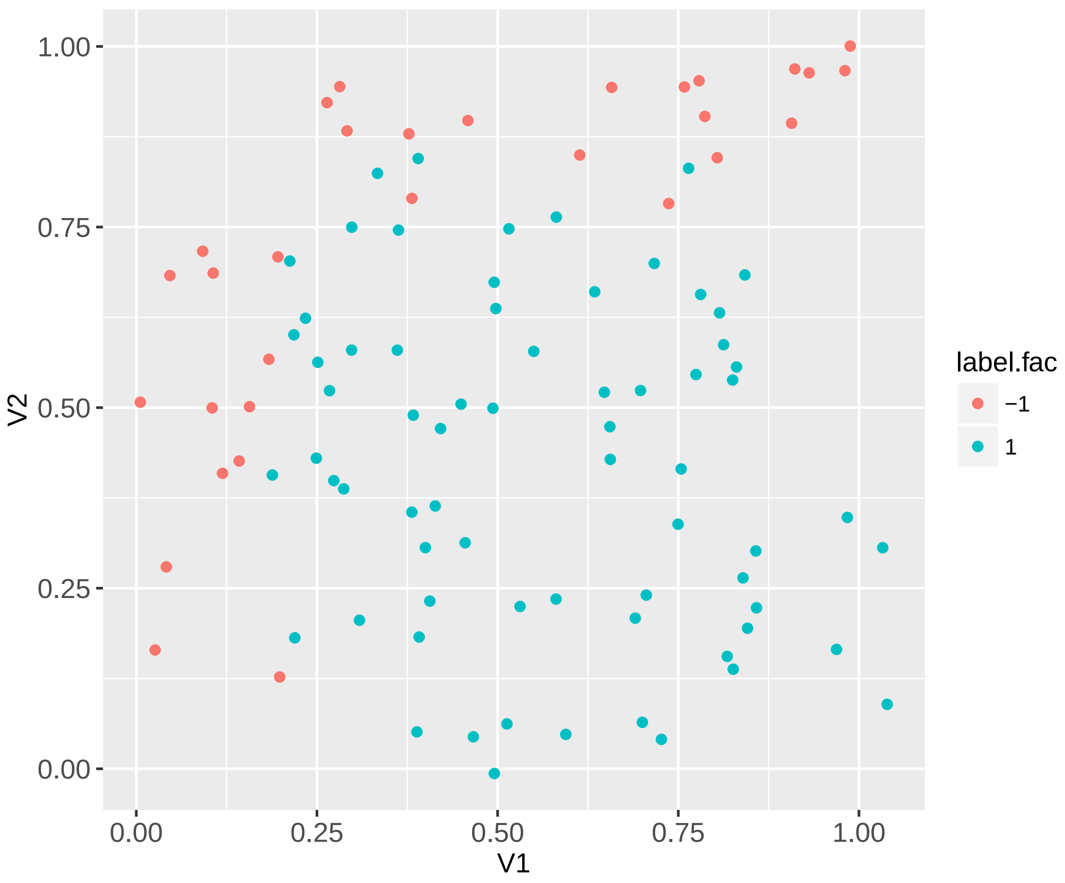
The binary labels above are created from the hidden features, but for learning we only have access to the noisy features below,

```r
set.seed(1)
features.noisy <- features.hidden+rnorm(sim.row*sim.col, sd=0.05)
head(features.noisy)
```

```
          [,1]      [,2]
[1,] 0.2341860 0.6237056
[2,] 0.3813061 0.3553031
[3,] 0.5310719 0.2247141
[4,] 0.9879718 1.0005855
[5,] 0.2181573 0.6007640
[6,] 0.8573663 0.3015725
```

To plot the data and visualize the pattern, we use the code below,

```r
library(animint2)
label.fac <- factor(label.vec)
sim.dt <- data.table(features.noisy, label.fac)
ggplot()+
  geom_point(aes(
    V1, V2, color=label.fac),
    data=sim.dt)+
  coord_equal()
```



The plot above shows each row in the data set as a point, with the two features on the two axes, and the two labels in two different colors. The lower right part of the feature space tends to have positive labels, and the left and top areas have negative labels. This is the pattern that the neural network will learn. To properly train a neural network, we need to split the data into two sets:

- subtrain: used to compute gradients, which are used to update weight parameters, and predicted values. With enough iterations/epochs of the gradient descent learning algorithm, and a powerful enough neural network model (large enough number of hidden units/layers), it should be possible to get perfect prediction on the subtrain set.
- validation: used to avoid overfitting. By computing the prediction error on the validation set, and choosing the number of gradient descent iterations/epochs which minimizes the validation error, we can ensure the learned model has good generalization properties (provides good predictions on not only the subtrain set, but also new data points like in the validation set).

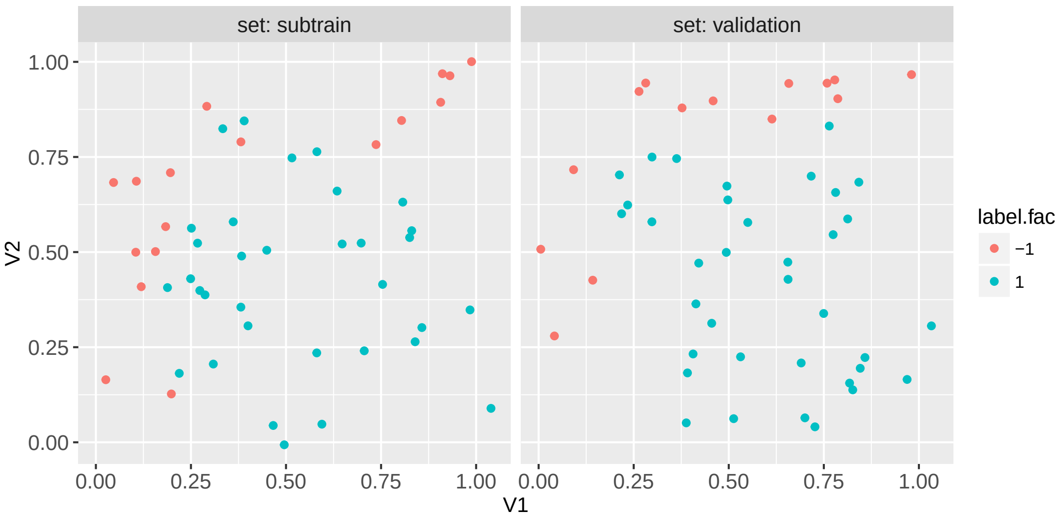
```
is.set.list <- list(
  validation=rep(c(TRUE,FALSE), l=nrow(features.noisy)))
is.set.list$subtrain <- !is.set.list$validation
set.vec <- ifelse(is.set.list$validation, "validation", "subtrain")
table(set.vec)
```

```
set.vec
```

```
  subtrain validation
        50         50
```

The code above is used to randomly assign half of the data into each of the subtrain and validation sets. Below, we plot the two sets in separate facets,

```
sim.dt[, set := set.vec]
ggplot()+
  facet_grid(. ~ set, labeller=label_both)+
  geom_point(aes(
    V1, V2, color=label.fac),
    data=sim.dt)+
  coord_equal()
```



18.2 Visualize Bayes optimal classification function

To visualize the optimal/Bayes decision boundary, we need to evaluate the function on a 2d grid of points that spans the feature space. To create such a grid, we first create a list which contains the two 1d grids for each feature,

```
(grid.list <- lapply(sim.dt[, .(V1, V2)], function(V){
  seq(min(V), max(V), l=30)
}))
```

```
$V1
 [1] 0.005600558 0.041238397 0.076876237 0.112514077 0.148151916 0.183789756
 [7] 0.219427595 0.255065435 0.290703274 0.326341114 0.361978954 0.397616793
[13] 0.433254633 0.468892472 0.504530312 0.540168151 0.575805991 0.611443831
[19] 0.647081670 0.682719510 0.718357349 0.753995189 0.789633028 0.825270868
[25] 0.860908708 0.896546547 0.932184387 0.967822226 1.003460066 1.039097905
```

```
$V2
 [1] -0.006562821  0.028166431  0.062895684  0.097624936  0.132354189
 [6]  0.167083441  0.201812694  0.236541946  0.271271199  0.306000451
[11]  0.340729703  0.375458956  0.410188208  0.444917461  0.479646713
[16]  0.514375966  0.549105218  0.583834471  0.618563723  0.653292975
[21]  0.688022228  0.722751480  0.757480733  0.792209985  0.826939238
[26]  0.861668490  0.896397742  0.931126995  0.965856247  1.000585500
```

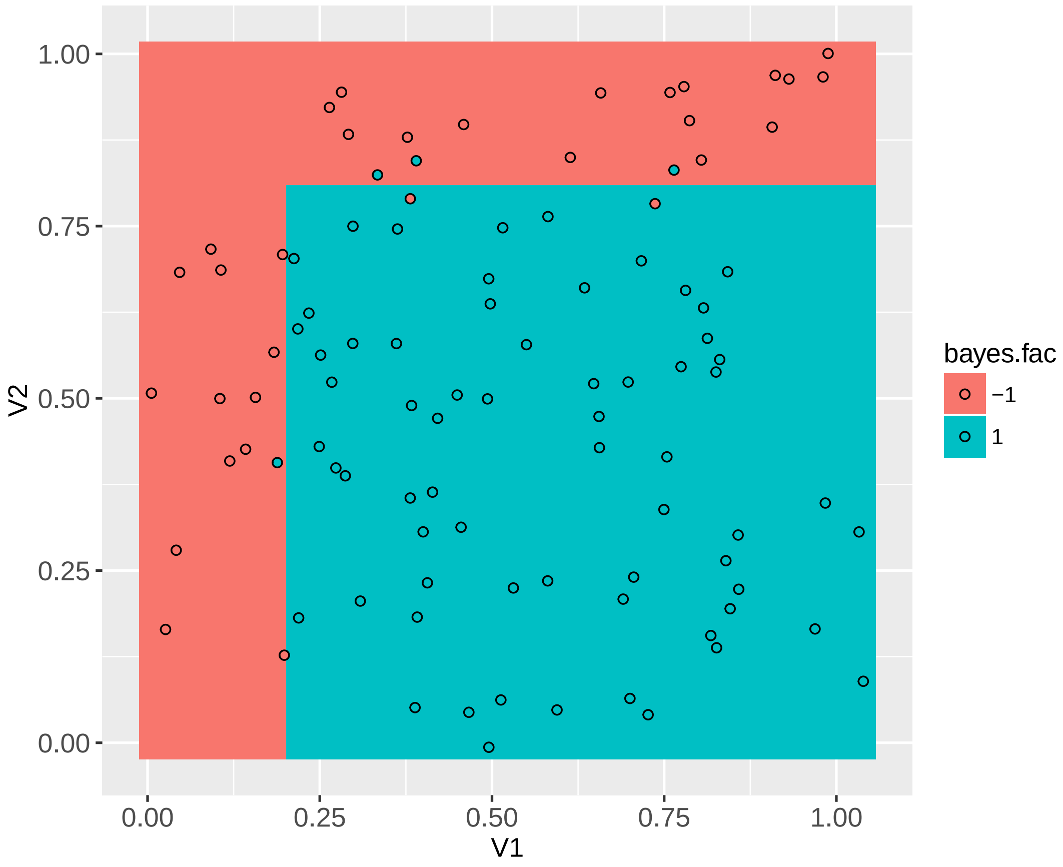
Then we use `CJ` (cross-join) to create a data table representing the 2d grid, for which we evaluate the best/Bayes classification function,

```
(grid.dt <- do.call(
  CJ, grid.list
)[
, bayes.num := ifelse(bayes(.SD), 1, -1)
][
, bayes.fac := factor(bayes.num)
][])
```

```
              V1           V2 bayes.num bayes.fac
  1: 0.005600558 -0.006562821        -1        -1
  2: 0.005600558  0.028166431        -1        -1
 ---
899: 1.039097905  0.965856247        -1        -1
900: 1.039097905  1.000585500        -1        -1
```

The best classifier is visualized below in the feature space,

```
ggplot()+
  geom_tile(aes(
    V1, V2, fill=bayes.fac),
    color=NA,
    data=grid.dt)+
  geom_point(aes(
    V1, V2, fill=label.fac),
    color="black",
    shape=21,
    data=sim.dt)+
  coord_equal()
```



The plot above shows that even using the best possible function, there are still some prediction errors (points on a background of different color). Another way to visualize that best classification function is via the decision boundary, which can be computed using the code below,

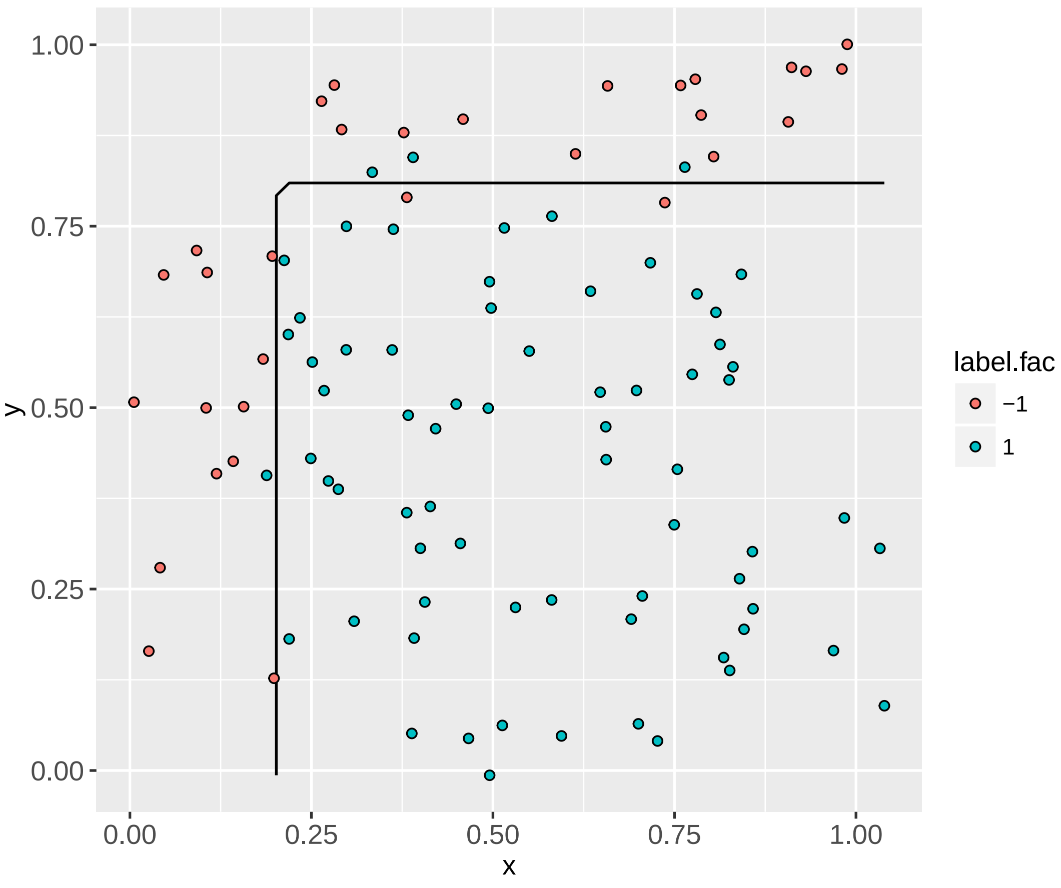
```
get_boundary <- function(score){
  contour.list <- contourLines(
    grid.list$V1, grid.list$V2,
    matrix(
      score,
      length(grid.list$V1),
      length(grid.list$V2),
      byrow=TRUE),
    levels=0)
  if(length(contour.list)){
    data.table(contour.i=seq_along(contour.list))[, {
      with(contour.list[[contour.i]], data.table(level, x, y))
    }, by=contour.i]
  }
}
(bayes.contour.dt <- get_boundary(grid.dt$bayes.num))
```

```
    contour.i level         x            y
```

```
 1:          1     0 0.2016087 -0.006562821
 2:          1     0 0.2016087  0.028166431
---
47:          1     0 1.0034601  0.809574611
48:          1     0 1.0390979  0.809574611
```

The best decision boundary is visualized in the feature space below,

```
ggplot()+
  geom_path(aes(
    x, y, group=contour.i),
    data=bayes.contour.dt)+
  geom_point(aes(
    V1, V2, fill=label.fac),
    color="black",
    shape=21,
    data=sim.dt)+
  coord_equal()
```



18.3 Forward and back propagation in linear model

To implement the gradient descent algorithm for learning neural network model parameters, we will use a simple auto-grad system. Auto-grad is the idea that the neural network model structure should be defined once, and that structure should be used to derive both the forward (prediction) and backward (gradient) propagation computations. Below we use a simple auto-grad system where each node in the computation graph is represented by an R environment (a mutable data structure, necessary so that the gradients are back-propagated to all the model parameters). The function below is a constructor for the most basic building block of the auto-grad system, a node in the computation graph:

```r
new_node <- function(value, gradient=NULL, ...){
  node <- new.env()
  node$value <- value
  node$parent.list <- list(...)
  node$backward <- function(){
    grad.list <- gradient(node)
    for(parent.name in names(grad.list)){
      parent.node <- node$parent.list[[parent.name]]
      parent.node$grad <- grad.list[[parent.name]]
      parent.node$backward()
    }
  }
  node
}
```

The code in the function above starts by creating a new environment, then populates it with three objects:

- `value` is a matrix computed by forward propagation at this node in the computation graph.
- `parent.list` is a list of parent nodes, each of which is used to compute `value`.
- `backward` is a function which should be called by the user on the final/loss node in the computation graph. It calls `gradient`, which should compute the gradient of the loss with respect to the parent nodes, which are stored in the `grad` attribute in each corresponding parent node, before recursively calling `backward` on each parent node.

The simplest kind of node is an initial node, defined by the code below,

```r
initial_node <- function(mat){
  new_node(mat, gradient=function(...)list())
}
```

The code above says that an initial node simply stores the input matrix `mat` as the value, and has a `gradient` method that does nothing (because initial nodes in the computation graph have no parents for which gradients could be computed). The code below defines `mm`, a node in the computation graph which represents a matrix multiplication,

```
mm <- function(feature.node, weight.node)new_node(
  cbind(1, feature.node$value) %*% weight.node$value,
  features=feature.node,
  weights=weight.node,
  gradient=function(node)list(
    features=node$grad %*% t(weight.node$value),
    weights=t(cbind(1, feature.node$value)) %*% node$grad))
```

The `mm` definition above assumes that there is a weight node with the same number of rows as the number of columns (plus one for intercept) in the feature node. The forward/value and gradient computations use matrix multiplication. For instance, we can use `mm` as follows to define a simple linear model,

```
feature.node <- initial_node(features.noisy[is.set.list$subtrain,])
weight.node <- initial_node(rep(0, ncol(features.noisy)+1))
linear.pred.node <- mm(feature.node, weight.node)
str(linear.pred.node$value)
```

```
 num [1:50, 1] 0 0 0 0 0 0 0 0 0 0 ...
```

It can be seen in the code above that the `mm` function returns a node representing predicted values, one for each row in the feature matrix. To use the gradient features we need a loss function, which in the case of binary classification is the logistic (cross-entropy) loss,

```
log_loss <- function(pred.node, label.node)new_node(
  mean(log(1+exp(-label.node$value*pred.node$value))),
  pred=pred.node,
  label=label.node,
  gradient=function(...)list(
    pred=-label.node$value/(
      1+exp(label.node$value*pred.node$value)
    )/length(label.node$value)))
```

The code above defines the logistic loss and gradient, assuming the label is either -1 or 1, and the prediction is a real number (not necessarily between 0 and 1, maybe negative). The code below creates nodes for the labels and loss,

```
label.node <- initial_node(label.vec[is.set.list$subtrain])
loss.node <- log_loss(linear.pred.node, label.node)
loss.node$value
```

```
[1] 0.6931472
```

Now that we have computed the loss, we can compute the gradient of the loss with respect to the weights, which is used to perform the updates during learning. Remember that we should now call `backward` (on the subtrain loss), which should eventually store the gradient as `weight.node$grad`. Below we first verify that it has not yet been computed, then we compute it:

```
weight.node$grad
```

```
NULL
```

```
loss.node$backward()
weight.node$grad
```

```
            [,1]
[1,] -0.16000000
[2,] -0.10511619
[3,] -0.02447615
```

Note that since `loss.node` contains recursive back-references to its parent nodes (including predictions and weights), the `backward` call above is able to conveniently compute and store `weight.node$grad`, the gradient of the loss with respect to the weight parameters. The gradient is the direction of steepest ascent, meaning the direction weights could be modified to maximize the loss. Because we want to minimize the loss, the learning algorithm performs updates in the negative gradient direction, of steepest descent.

```
(descent.direction <- -weight.node$grad)
```

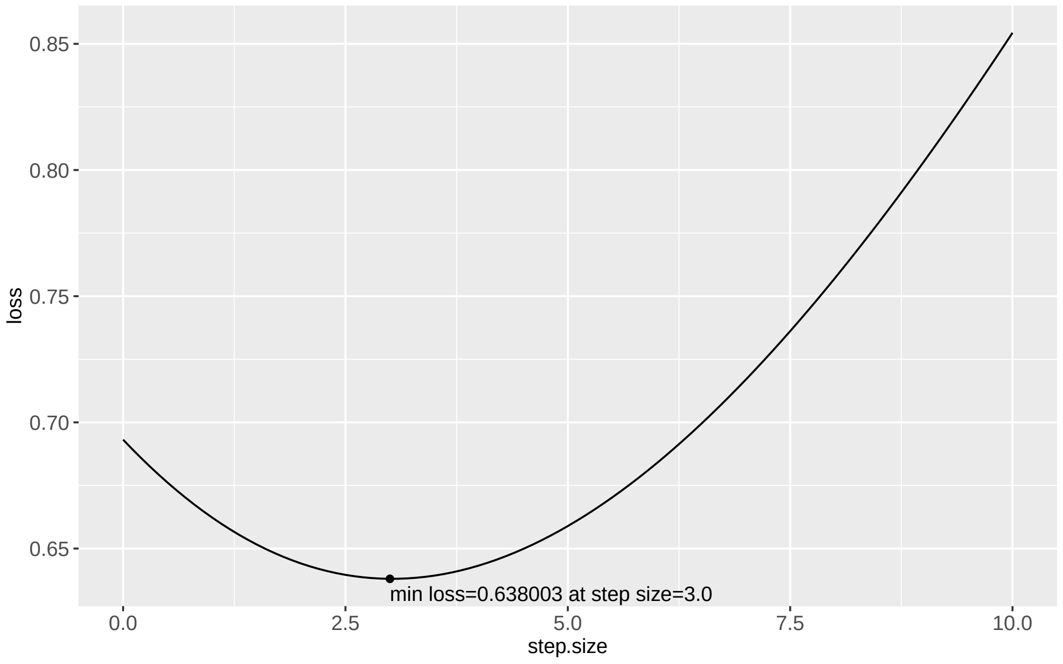
```
           [,1]
[1,] 0.16000000
[2,] 0.10511619
[3,] 0.02447615
```

In gradient descent for this linear model, we update the weight vector in this direction. Each update to the weight vector is referred to as an iteration or step. A small step in this direction is guaranteed to decrease the loss, but too small of a step will not make much progress toward minimizing the loss. It is unknown how far in this direction is best, so we typically need to search over a grid of step sizes (aka learning rates). Or we can perform a line search, which means making the following plot of loss as a function of step size, then choosing the step size with minimal loss.

```
(line.search.dt <- data.table(step.size=seq(0, 10, l=101))[, .(
  loss=log_loss(mm(
    feature.node,
    initial_node(weight.node$value+step.size*descent.direction)
  ), label.node)$value
), by=step.size])
```

```
     step.size      loss
  1:       0.0 0.6931472
  2:       0.1 0.6894865
 ---
100:       9.9 0.8490474
101:      10.0 0.8543739
```

```
line.search.min <- line.search.dt[which.min(loss)]
ggplot()+
  geom_line(aes(
    step.size, loss),
    data=line.search.dt)+
  geom_point(aes(
    step.size, loss),
    data=line.search.min)+
  geom_text(aes(
    step.size, loss, label=sprintf(
      "min loss=%f at step size=%.1f",
      loss, step.size)),
    data=line.search.min,
    size=4,
    hjust=0,
    vjust=1.5)
```



The plot above shows that the min loss occurs at a step size of about 3, which means that the line search would choose that step size for this gradient descent parameter update/iteration, before re-computing the gradient in the next iteration.

18.4 Neural network learning

Defining a linear model in the previous section was relatively simple, because there is only one weight matrix parameter (actually, a weight vector, with the same number of elements as the number of columns/features, plus one for an intercept). In contrast, a neural network has more than one weight matrix parameter to learn. We initialize these weights as nodes in the code below,

```
new_weight_node_list <- function(units.per.layer, intercept=TRUE){
  weight.node.list <- list()
  for(layer.i in seq(1, length(units.per.layer)-1)){
    input.units <- units.per.layer[[layer.i]]+intercept
    output.units <- units.per.layer[[layer.i+1]]
    weight.mat <- matrix(
      rnorm(input.units*output.units), input.units, output.units)
    weight.node.list[[layer.i]] <- initial_node(weight.mat)
  }
  weight.node.list
}
(units.per.layer <- c(ncol(features.noisy), 40, 1))
```

```
[1]  2 40  1
```

```
(weight.node.list <- new_weight_node_list(units.per.layer))
```

```
[[1]]
<environment: 0x562964fdb680>

[[2]]
<environment: 0x562964fde480>
```

```
lapply(weight.node.list, function(node)dim(node$value))
```

```
[[1]]
[1]  3 40

[[2]]
[1] 41  1
```

The output above shows that there is a single layer with 40 hidden units in the neural network, meaning there are two weight matrices to learn. Each of these weight matrices is used to predict the units in a given layer, from the units in the previous layer. In order to learn a prediction function which is a non-linear function of the features, each layer except for the last must have a non-linear activation function, applied element-wise to the units after matrix multiplication. For example, a common and efficient non-linear activation function is the ReLU (Rectified Linear Units), which is implemented below,

```
relu <- function(before.node)new_node(
  ifelse(before.node$value < 0, 0, before.node$value),
  before=before.node,
  gradient=function(node)list(
    before=ifelse(before.node$value < 0, 0, node$grad)))
hidden.before.act <- mm(feature.node, weight.node.list[[1]])
str(hidden.before.act$value)
```

```
 num [1:50, 1:40] 1.62 3.67 2.34 2.42 1.57 ...
```

```
hidden.before.act$value[1:5,1:5]
```

```
         [,1]        [,2]      [,3]      [,4]       [,5]
[1,] 1.617099 -0.31485515 0.8687167 0.5969050 -0.5717334
[2,] 3.665478 -0.08953216 1.1884750 0.7686507 -0.7655569
[3,] 2.335856 -1.53696561 1.1271410 0.4960481 -1.2771119
[4,] 2.418533 -0.61750628 1.0377415 0.6157081 -0.8390015
[5,] 1.571396  1.26836784 0.6830969 0.7897418  0.2105804
```

```
hidden.after.act <- relu(hidden.before.act)
hidden.after.act$value[1:5,1:5]
```

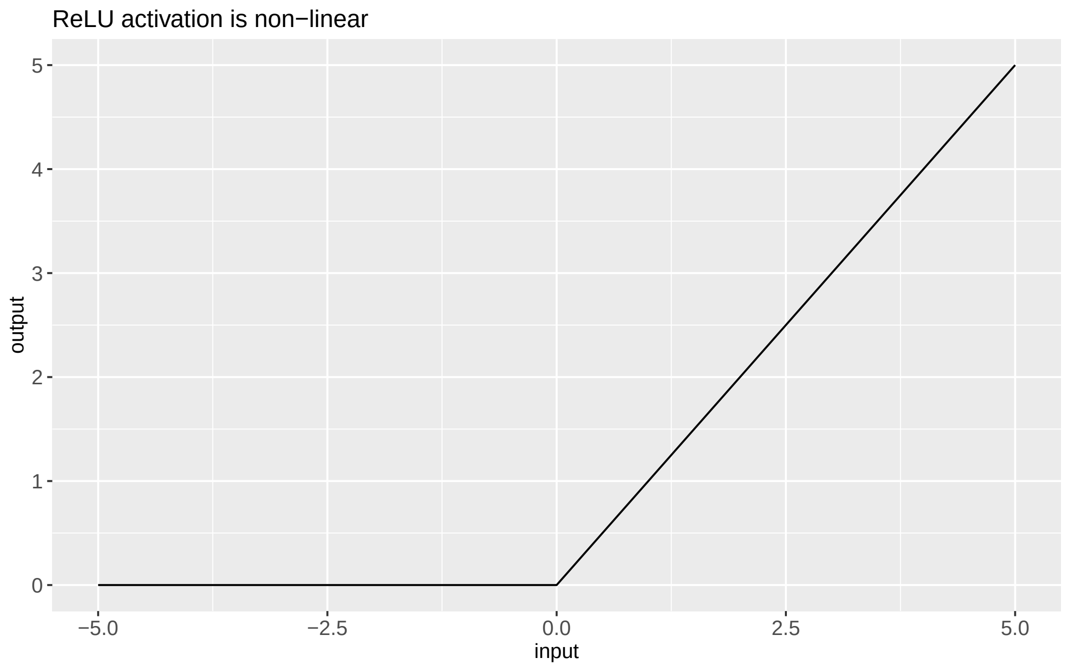
```
         [,1]     [,2]      [,3]      [,4]      [,5]
[1,] 1.617099 0.000000 0.8687167 0.5969050 0.0000000
[2,] 3.665478 0.000000 1.1884750 0.7686507 0.0000000
[3,] 2.335856 0.000000 1.1271410 0.4960481 0.0000000
[4,] 2.418533 0.000000 1.0377415 0.6157081 0.0000000
[5,] 1.571396 1.268368 0.6830969 0.7897418 0.2105804
```

Note in the output above how the ReLU activation sets negative values to zero, and keeps positive values the same:

```
(relu.dt <- data.table(
  input=seq(-5, 5, l=101)
)[, output := relu(initial_node(input))$value][])
```

```
     input output
  1:  -5.0    0.0
  2:  -4.9    0.0
 ---
100:   4.9    4.9
101:   5.0    5.0
```

```
ggplot()+
  ggtitle("ReLU activation is non-linear")+
  geom_line(aes(
    input, output),
    data=relu.dt)
```



Finally, the last node that we need to implement our neural network is a node for predictions, computed via the for loop over weight nodes in the function below,

```
pred_node <- function(set.features){
  feature.node <- initial_node(set.features)
  for(layer.i in seq_along(weight.node.list)){
    weight.node <- weight.node.list[[layer.i]]
    before.node <- mm(feature.node, weight.node)
    feature.node <- if(layer.i < length(weight.node.list)){
      relu(before.node)
    }else{
      before.node
    }
  }
  feature.node
}
nn.pred.node <- pred_node(features.noisy[is.set.list$subtrain,])
str(nn.pred.node$value)
```

```
num [1:50, 1] 3.85 6.42 2.14 3.54 8.18 ...
```

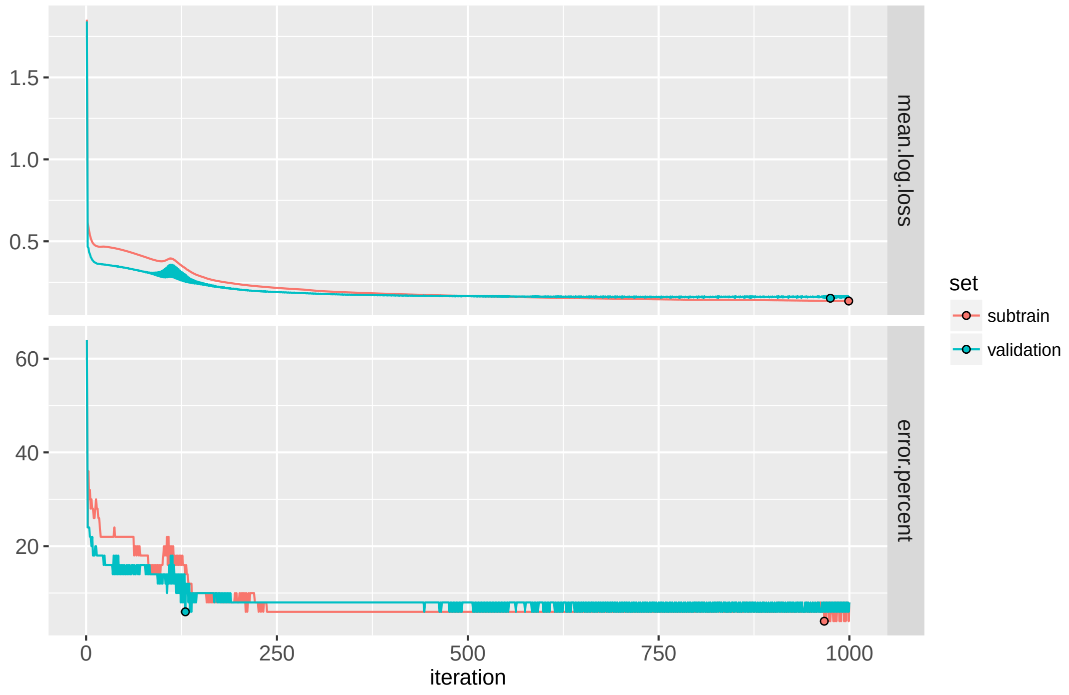
The `pred_node` function is also useful for computing predictions on the grid of features, which will be useful later for visualizing the learned function,

```
grid.mat <- grid.dt[, cbind(V1,V2)]
nn.grid.node <- pred_node(grid.mat)
str(nn.grid.node$value)
```

```
num [1:900, 1] 1.74 2.09 2.43 2.78 3.12 ...
```

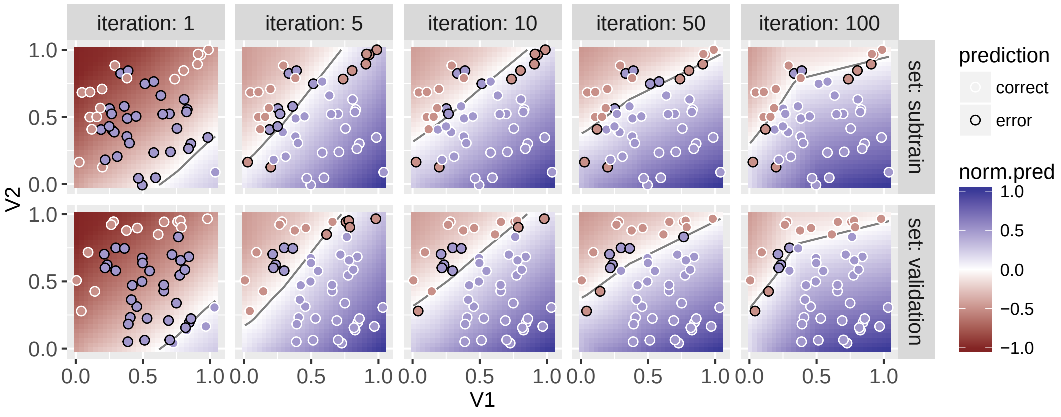
The code below combines all of the pieces above into a gradient descent learning algorithm. The hyper-parameters are the constant step size, and the maximum number of iterations.

```
step.size <- 0.5
max.iterations <- 1000
units.per.layer <- c(ncol(features.noisy), 40, 1)
loss.dt.list <- list()
err.dt.list <- list()
pred.dt.list <- list()
set.seed(10)
weight.node.list <- new_weight_node_list(units.per.layer)
for(iteration in 1:max.iterations){
  loss.node.list <- list()
  for(set in names(is.set.list)){
    is.set <- is.set.list[[set]]
    set.label.node <- initial_node(label.vec[is.set])
    set.features <- features.noisy[is.set,]
    set.pred.node <- pred_node(set.features)
    set.loss.node <- log_loss(set.pred.node, set.label.node)
    loss.node.list[[set]] <- set.loss.node
    set.pred.num <- ifelse(set.pred.node$value<0, -1, 1)
    is.error <- set.pred.num != set.label.node$value
    err.dt.list[[paste(iteration, set)]] <- data.table(
      iteration, set,
      set.features,
      label=set.label.node$value,
      pred.num=as.numeric(set.pred.num))
    loss.dt.list[[paste(iteration, set)]] <- data.table(
      iteration, set,
      mean.log.loss=set.loss.node$value,
      error.percent=100*mean(is.error))
  }
  grid.node <- pred_node(grid.mat)
  pred.dt.list[[paste(iteration)]] <- data.table(
    iteration,
    grid.dt,
    pred=as.numeric(grid.node$value))
  loss.node.list$subtrain$backward()#<-back-prop.
  for(layer.i in seq_along(weight.node.list)){
    weight.node <- weight.node.list[[layer.i]]
    weight.node$value <- #learning/param updates:
      weight.node$value-step.size*weight.node$grad
  }
}
loss.dt <- rbindlist(loss.dt.list)
err.dt <- rbindlist(err.dt.list)
pred.dt <- rbindlist(pred.dt.list)
loss.tall <- melt(loss.dt, measure=c("mean.log.loss", "error.percent"))
loss.tall[, log10.iteration := log10(iteration)]
min.dt <- loss.tall[
, .SD[which.min(value)], by=.(set, variable)]
ggplot()+
  facet_grid(variable ~ ., scales="free")+
  scale_y_continuous("")+
  geom_line(aes(
    iteration, value, color=set),
    data=loss.tall)+
  geom_point(aes(
    iteration, value, fill=set),
    shape=21,
    color="black",
    data=min.dt)
```



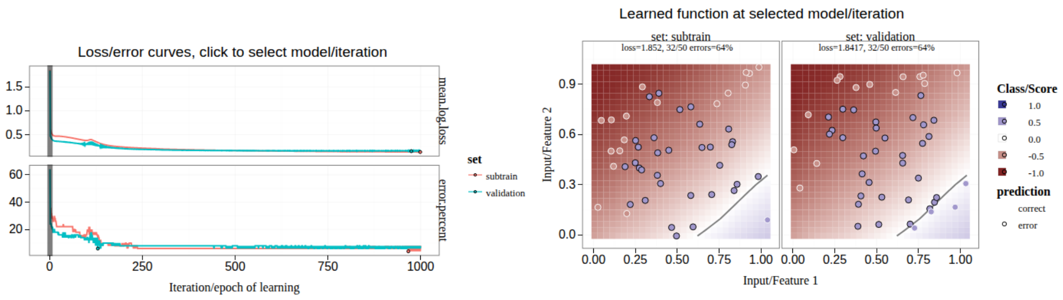
In the code above we saved loss and predictions for all of the iterations of gradient descent, but in the code below we visualize only some of them, due to limited space:

```
some <- function(DT)DT[iteration%in%c(1,5,10,50,100)]
err.dt[, prediction := ifelse(label==pred.num, "correct", "error")]
iteration.contours <- pred.dt[
, get_boundary(pred), by=.(iteration)]
some.loss <- some(loss.dt)
pred.dt[, norm.pred := pred/max(abs(pred)), by=.(iteration)]
some.pred <- some(pred.dt)
some.err <- some(err.dt)
some.contours <- some.pred[
, get_boundary(pred), by=.(iteration)]
ggplot()+
  facet_grid(set ~ iteration, labeller="label_both")+
  geom_tile(aes(
    V1, V2, fill=norm.pred),
    color=NA,
    data=some.pred)+
  geom_path(aes(
    x, y, group=contour.i),
    color="grey50",
    data=some.contours)+
  scale_fill_gradient2()+
  geom_point(aes(
    V1, V2, color=prediction, fill=label/2),
    shape=21,
    size=2,
    data=some.err)+
  scale_color_manual(
    values=c(correct="white", error="black"))+
  coord_equal()+
  scale_y_continuous(breaks=seq(0,1,by=0.5))+
  scale_x_continuous(breaks=seq(0,1,by=0.5))
```



From the plot above we can see that as the number of iterations increases, the predictions get more accurate. Finally, we conclude with an interactive plot where you can click the loss plot to select an iteration of gradient descent, for which the corresponding decision boundary is shown on the predictions plot.

```
  n.subtrain <- sum(is.set.list$subtrain)
  loss.dt[, n.set := ifelse(
    set=="subtrain", n.subtrain, sim.row-n.subtrain
  )][, error.count := n.set*error.percent/100]
  it.by <- 10
  some <- function(DT)DT[iteration %in% as.integer(
    seq(1, max.iterations, by=it.by))]
  animint(
    title="Neural network vs linear model",
    out.dir="neural-networks-sim",
    loss=ggplot()+
      ggtitle("Loss/error curves, click to select model/iteration")+
      theme_bw()+
      theme_animint(width=600, height=350)+
      theme(panel.margin=grid::unit(1, "lines"))+
      facet_grid(variable ~ ., scales="free")+
      scale_y_continuous("")+
      scale_x_continuous(
        "Iteration/epoch of learning")+
      geom_line(aes(
        iteration, value, color=set, group=set),
        data=loss.tall)+
      geom_point(aes(
        iteration, value, fill=set),
        shape=21,
        color="black",
        data=min.dt)+
      geom_tallrect(aes(
        xmin=iteration-it.by/2,
        xmax=iteration+it.by/2),
        alpha=0.5,
        clickSelects="iteration",
        data=some(loss.tall[set=="subtrain"])),
    data=ggplot()+
      ggtitle("Learned function at selected model/iteration")+
      theme_bw()+
      theme_animint(width=600)+
      facet_grid(. ~ set, labeller="label_both")+
      geom_tile(aes(
        V1, V2, fill=norm.pred),
        color=NA,
        showSelected="iteration",
        data=some(pred.dt))+
      geom_text(aes(
        0.5, 1.1, label=paste0(
          "loss=", round(mean.log.loss, 4),
          ", ", error.count, "/", n.set,
          " errors=", error.percent, "%")),
        showSelected="iteration",
        data=loss.dt)+
      geom_path(aes(
        x, y, group=contour.i),
        showSelected="iteration",
        color="grey50",
        data=some(iteration.contours))+
      geom_point(aes(
        V1, V2, fill=label/2, color=prediction),
        showSelected=c("iteration", "set"),
        size=4,
        data=some(err.dt))+
      scale_fill_gradient2(
        "Class/Score")+
      scale_color_manual(
        values=c(correct="white", error="black"))+
      scale_x_continuous(
        "Input/Feature 1")+
      scale_y_continuous(
        "Input/Feature 2")+
      coord_equal())
```



18.5 Chapter summary and exercises

Exercises:

- Add animation over the number of iterations.
- Add smooth transitions when changing the selected number of iterations.
- Add a for loop over random seeds (or cross-validation folds) in the data splitting step, and create a visualization that shows how that affects the results.
- Add a for loop over random seeds at the weight matrix initialization step, and create a visualization that shows how that affects the results.
- Compute results for another neural network architecture (and/or linear model, by adding a for loop over different values of `units.per.layer`). Add another plot or facet which allows selecting the neural network architecture, and allows easy comparison of the min validation loss between models (for example, add facet columns to loss plot, and add horizontal lines to emphasize min loss).
- Modify the learning algorithm to use line search rather than constant step size, and then create a visualization which compares the two approaches in terms of min validation loss.

Next, Chapter 99 explains some R programming idioms that are generally useful for interactive data visualization design.

A

Useful R programming idioms

This appendix describes several R programming idioms which are useful for creating animints.

A.1 Space saving facets

To emphasize the plotted data in facetted ggplots, eliminate the space between facets using the following idiom.

```
ggplot()+
  geom_point(aes(Petal.Width, Sepal.Width), iris)+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(. ~ Species)
```

There are three parts of this idiom:

- `panel.margin=0` eliminates space between panels.
- `theme_bw` activates a black and white theme (black panel borders and white panel backgrounds). This is necessary in order to see the boundaries between panels, since the ggplot default `theme_grey` uses grey panel backgrounds and no panel borders.
- `facet_*` creates a multi-panel ggplot.

Note that we use the grid unit `lines`, which equals the height of one line of text at the default size. This is the only grid unit which animint knows how to translate. It is not recommended to use other units such as `cm`.

A.2 List of data tables

The list of data tables idiom is very useful for creating interactive data visualizations of arbitrary complexity. The general form looks like

```
library(data.table)
outer.data.list <- list()
inner.data.list <- list()
for(outer in outer.vec){
  outer.dt <- computeOuter(outer)
```

```
    outer.data.list[[paste(outer)]] <- data.table(outer, outer.dt)
    for(inner in inner.vec){
      inner.dt <- computeInner(outer.dt, inner)
      inner.data.list[[paste(outer, inner)]] <-
        data.table(outer, inner, inner.dt)
    }
  }
  outer.data <- do.call(rbind, outer.data.list)
  inner.data <- do.call(rbind, inner.data.list)
```

Some comments:

- The first part of the idiom involves initializing empty lists. Here there are two, `outer.data.list` and `inner.data.list`. However there can be as many as necessary.
- The second part of the idiom is a bunch of nested for loops that assign data tables to elements of those lists.
  - Functions like `computeOuter` and `computeInner` can be used, or you can just do the computations directly inside the for loop.
  - To ensure that your code will run as fast as possible, use matrix-vector or vector-scalar operations in the innermost for loop. If you only do scalar-scalar operations in your innermost for loop, then you can definitely improve the performance of your code by removing that for loop and re-writing the computation in terms of vector-scalar operations.
  - The `paste` function is used to assign a `data.table` to a named list element. Although in principle one could use either `data.frame` or `data.table`, in practice `data.table` is often much faster during the last combination step.
- The last part of the idiom uses `do.call` with `rbind` to combine the data tables stored during the for loops.

A.3 addColumn then facet

This idiom is useful for creating multi-panel ggplots with aligned axes. First, define a function which takes as input a data table and one or more values which will be used to add factors to that data table.

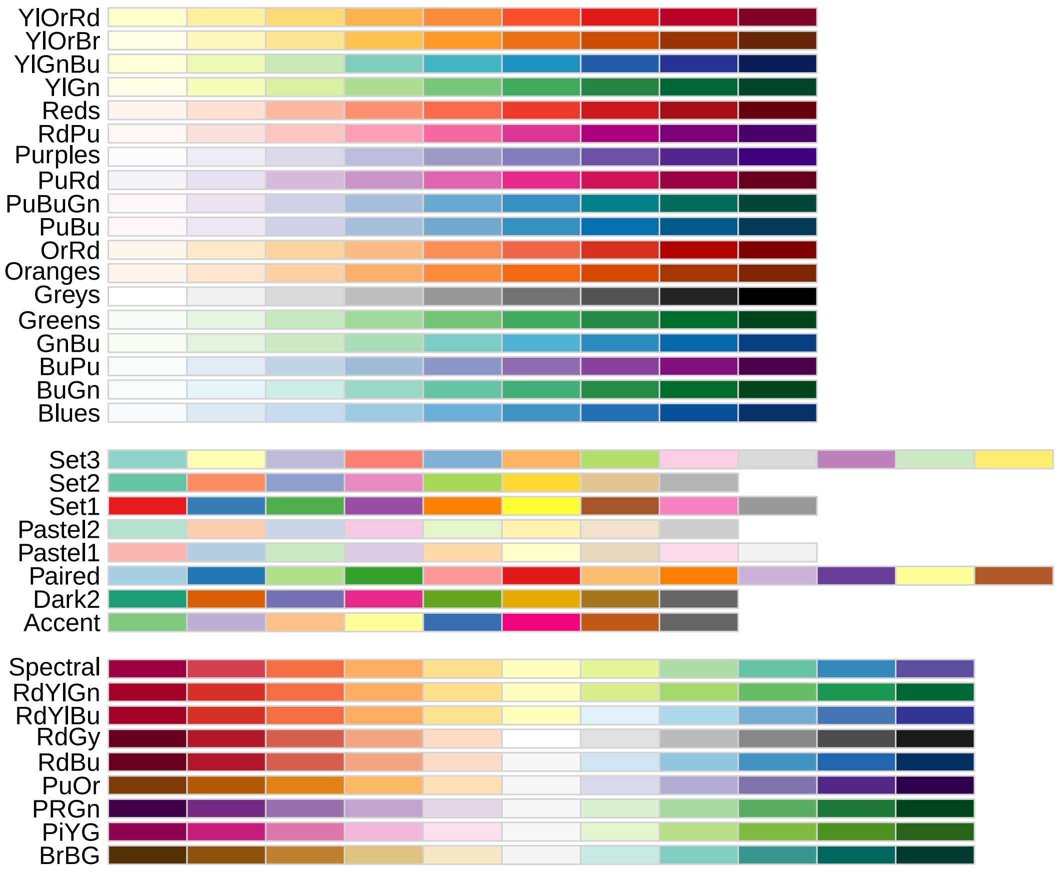
```
addColumn <- function(df, time.period)data.frame(
  df, time.period=factor(time.period, c("1975", "1960-2010")))
animint(
  ggplot()+
  geom_point(aes(
    x=life.expectancy, y=fertility.rate, color=region),
    data=addColumn(WorldBank1975, "1975"))+
  geom_path(aes(
    x=life.expectancy, y=fertility.rate, color=region,
    group=country),
    data=addColumn(WorldBankBefore1975, "1975"))+
  geom_line(aes(
    x=year, y=fertility.rate, color=region, group=country),
    data=addColumn(WorldBank, "1960-2010"))+
  facet_grid(. ~ time.period, scales="free")+
  xlab(""))
```

Note that `scales="free"` and `xlab("")` are used since the x axes now have very different units (year and life expectancy).

A.4 Manual color legends

Color and fill legends in `ggplot2` can be manually specified via `scale_color_manual` and `scale_fill_manual`. Typically we will choose one of the ColorBrewer palettes:

```
RColorBrewer::display.brewer.all()
```



For example to get the R code for the Set1 palette, we can write

```
dput(RColorBrewer::brewer.pal(7, "Set1"))
```

```
c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3", "#FF7F00", "#FFFF33",
"#A65628")
```

We can then copy that R code from the terminal and paste it into our text editor

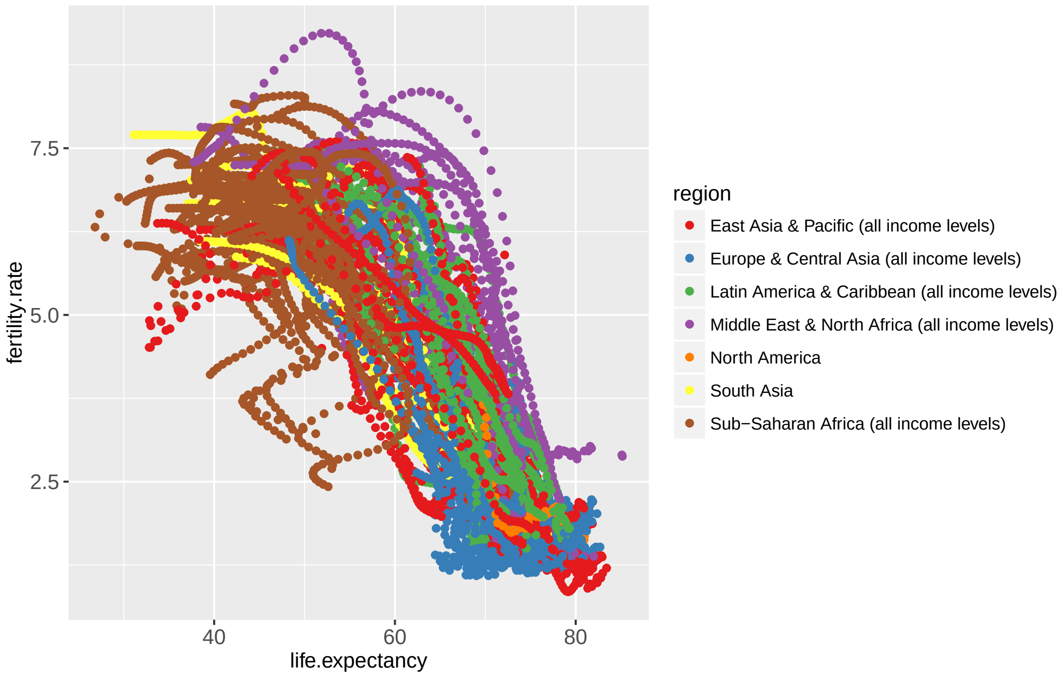
```
data(WorldBank, package="animint2")
region.colors <- c(
  "#E41A1C", "#377EB8", "#4DAF4A", "#984EA3", "#FF7F00", "#FFFF33",
  "#A65628")
names(region.colors) <- levels(WorldBank$region)
region.colors
```

```
       East Asia & Pacific (all income levels)
                                     "#E41A1C"
     Europe & Central Asia (all income levels)
                                     "#377EB8"
 Latin America & Caribbean (all income levels)
                                     "#4DAF4A"
Middle East & North Africa (all income levels)
                                     "#984EA3"
                                 North America
                                     "#FF7F00"
                                    South Asia
                                     "#FFFF33"
        Sub-Saharan Africa (all income levels)
                                     "#A65628"
```

Then we can use it with `scale_color_manual`

```
library(animint2)
ggplot()+
  scale_color_manual(values=region.colors)+
  geom_point(aes(
    x=life.expectancy, y=fertility.rate, color=region),
    data=WorldBank)
```

```
Warning: Removed 1490 rows containing missing values (geom_point).
```


7.5
5.0
2.5
fertility.rate
40
60
80
life.expectancy
region
East Asia & Pacific (all income levels)
Europe & Central Asia (all income levels)
Latin America & Caribbean (all income levels)
Middle East & North Africa (all income levels)
North America
South Asia
Sub−Saharan Africa (all income levels)

B

Contributing

This manual has been developed in the open, and it wouldn't be nearly as good without outside contributions. There are two primary ways you can help make the manual even better.

B.1 Reporting issues

If you don't understand something in this manual, please let us know! Your feedback on what is confusing or hard to understand is valuable.

- For most issues, please report an issue in public to animint developers in this GitHub repo. The link also appears on the right sidebar of each web page.
- For issues which you would prefer to discuss privately, please write me an email.

B.2 Submitting edits

If you spot a typo, or you know how to fix another issue, then feel free to edit the underlying page, and send a pull request:

- Click "Edit this page" on the right sidebar.
- Make the changes using github's in-page editor and save.
- Submit a pull request including a brief description of your changes: "fixing typos" is perfectly adequate.
- If you make significant changes, include the phrase "I assign the copyright of this contribution to Toby Dylan Hocking" - I need this so I can publish the printed manual.