Data Visualization with R

Rob Kabacoff

2018-09-03

Contents

Welcome

R is an amazing platform for data analysis, capable of creating almost any type of graph. This book helps you create the most popular visualizations - from quick and dirty plots to publication-ready graphs. The text relies heavily on the ggplot2 package for graphics, but other approaches are covered as well.

My goal is make this book as helpful and user-friendly as possible. Any feedback is both welcome and appreciated.

Preface

How to use this book

You don't need to read this book from start to finish in order to start building effective graphs. Feel free to jump to the section that you need and then explore others that you find interesting.

Graphs are organized by

- the number of variables to be plotted
- the type of variables to be plotted
- the purpose of the visualization

Chapter	Description
Ch 1	provides a quick overview of how to get your data into R and how to prepare it for analysis.
Ch 2	provides an overview of the `ggplot2` package.
Ch 3	describes graphs for visualizing the distribution of a single categorical (e.g. race) or quantitative (e.g. income) variable.
Ch 4	describes graphs that display the relationship between two variables.
Ch 5	describes graphs that display the relationships among 3 or more variables. It is helpful to read chapters 3 and 4 before this chapter.
Ch 6	provides a brief introduction to displaying data geographically.
Ch 7	describes graphs that display change over time.
Ch 8	describes graphs that can help you interpret the results of statistical models.
Ch 9	covers graphs that do not fit neatly elsewhere (every book needs a miscellaneous chapter).
Ch 10	describes how to customize the look and feel of your graphs. If you are going to share your graphs with others, be sure to skim this chapter.
Ch 11	covers how to save your graphs. Different formats are optimized for different purposes.
Ch 12	provides an introduction to interactive graphics.
Ch 13	gives advice on creating effective graphs and where to go to learn more. It's worth a look.
The Appendices	describe each of the datasets used in this book, and provides a short blurb about the author and the Wesleyan Quantitative Analysis Center.

There is **no one right graph** for displaying data. Check out the examples, and see which type best fits your needs.

Prequisites

It's assumed that you have some experience with the R language and that you have already installed R and RStudio. If not, here are some resources for getting started:

- A (very) short introduction to R
- DataCamp - Introduction to R with Jonathon Cornelissen
- Quick-R
- Getting up to speed with R

Setup

In order to create the graphs in this guide, you'll need to install some optional R packages. To install **all** of the necessary packages, run the following code in the RStudio console window.

```
pkgs <- c("ggplot2", "dplyr", "tidyr",
          "mosaicData", "carData",
          "VIM", "scales", "treemapify",
          "gapminder", "ggmap", "choroplethr",
          "choroplethrMaps", "CGPfunctions",
          "ggcorrplot", "visreg",
          "gcookbook", "forcats",
          "survival", "survminer",
          "ggalluvial", "ggridges",
          "GGally", "superheat",
          "waterfalls", "factoextra",
          "networkD3", "ggthemes",
          "hrbrthemes", "ggpol",
          "ggbeeswarm")
install.packages(pkgs)
```

Alternatively, you can install a given package the first time it is needed.

For example, if you execute

`library(gapminder)`

and get the message

`Error in library(gapminder) : there is no package called ‘gapminder’`

you know that the package has never been installed. Simply execute

`install.packages("gapminder")`

once and

`library(gapminder)`

will work from that point on.

Chapter 1

Data Preparation

Before you can visualize your data, you have to get it into R. This involves importing the data from an external source and massaging it into a useful format.

1.1 Importing data

R can import data from almost any source, including text files, excel spreadsheets, statistical packages, and database management systems. We'll illustrate these techniques using the Salaries dataset, containing the 9 month academic salaries of college professors at a single institution in 2008-2009.

1.1.1 Text files

The `readr` package provides functions for importing delimited text files into R data frames.

```
library(readr)

# import data from a comma delimited file
Salaries <- read_csv("salaries.csv")

# import data from a tab delimited file
Salaries <- read_tsv("salaries.txt")
```

These function assume that the first line of data contains the variable names, values are separated by commas or tabs respectively, and that missing data are represented by blanks. For example, the first few lines of the comma delimited file looks like this.

```
"rank","discipline","yrs.since.phd","yrs.service","sex","salary"
"Prof","B",19,18,"Male",139750
"Prof","B",20,16,"Male",173200
"AsstProf","B",4,3,"Male",79750
"Prof","B",45,39,"Male",115000
"Prof","B",40,41,"Male",141500
"AssocProf","B",6,6,"Male",97000
```

Options allow you to alter these assumptions. See the documentation for more details.

1.1.2 Excel spreadsheets

The `readxl` package can import data from Excel workbooks. Both xls and xlsx formats are supported.

```
library(readxl)

# import data from an Excel workbook
Salaries <- read_excel("salaries.xlsx", sheet=1)
```

Since workbooks can have more than one worksheet, you can specify the one you want with the `sheet` option. The default is `sheet=1`.

1.1.3 Statistical packages

The `haven` package provides functions for importing data from a variety of statistical packages.

```
library(haven)

# import data from Stata
Salaries <- read_dta("salaries.dta")

# import data from SPSS
Salaries <- read_sav("salaries.sav")

# import data from SAS
Salaries <- read_sas("salaries.sas7bdat")
```

1.1.4 Databases

Importing data from a database requires additional steps and is beyond the scope of this book. Depending on the database containing the data, the following packages can help: `RODBC`, `RMySQL`, `ROracle`, `RPostgreSQL`, `RSQLite`, and `RMongo`. In the newest versions of RStudio, you can use the Connections pane to quickly access the data stored in database management systems.

1.2 Cleaning data

The processes of cleaning your data can be the most time-consuming part of any data analysis. The most important steps are considered below. While there are many approaches, those using the `dplyr` and `tidyr` packages are some of the quickest and easiest to learn.

Package	Function	Use
dplyr	select	select variables/columns
dplyr	filter	select observations/rows
dplyr	mutate	transform or recode variables
dplyr	summarize	summarize data
dplyr	group_by	identify subgroups for further processing
tidyr	gather	convert wide format dataset to long format
tidyr	spread	convert long format dataset to wide format

Examples in this section will use the starwars dataset from the `dplyr` package. The dataset provides descriptions of 87 characters from the Starwars universe on 13 variables. (I actually prefer StarTrek, but we work with what we have.)

1.2.1 Selecting variables

The `select` function allows you to limit your dataset to specified variables (columns).

```
library(dplyr)

# keep the variables name, height, and gender
newdata <- select(starwars, name, height, gender)

# keep the variables name and all variables
# between mass and species inclusive
newdata <- select(starwars, name, mass:species)

# keep all variables except birth_year and gender
newdata <- select(starwars, -birth_year, -gender)
```

1.2.2 Selecting observations

The `filter` function allows you to limit your dataset to observations (rows) meeting a specific criteria. Multiple criteria can be combined with the `&` (AND) and `|` (OR) symbols.

```
library(dplyr)

# select females
newdata <- filter(starwars,
                  gender == "female")

# select females that are from Alderaan
newdata <- select(starwars,
                  gender == "female" &
                  homeworld == "Alderaan")


# select individuals that are from
# Alderaan, Coruscant, or Endor
newdata <- select(starwars,
                  homeworld == "Alderaan" |
                  homeworld == "Coruscant" |
                  homeworld == "Endor")

# this can be written more succinctly as
newdata <- select(starwars,
                  homeworld %in% c("Alderaan", "Coruscant", "Endor"))
```

1.2.3 Creating/Recoding variables

The `mutate` function allows you to create new variables or transform existing ones.

```
library(dplyr)

# convert height in centimeters to inches,
# and mass in kilograms to pounds
newdata <- mutate(starwars,
                  height = height * 0.394,
                  mass   = mass   * 2.205)
```

The `ifelse` function (part of base R) can be used for recoding data. The format is `ifelse(test, return if TRUE, return if FALSE)`.

```
library(dplyr)

# if height is greater than 180
# then heightcat = "tall",
# otherwise heightcat = "short"

newdata <- mutate(starwars,
                  heightcat = ifelse(height > 180,
                                     "tall",
                                     "short")

# convert any eye color that is not
# black, blue or brown, to other
newdata <- mutate(starwars,
                  eye_color = ifelse(eye_color %in% c("black", "blue", "brown"),
                                     eye_color,
                                     "other")

# set heights greater than 200 or
# less than 75 to missing
newdata <- mutate(starwars,
                  height = ifelse(height < 75 | height > 200,
                                     NA,
                                     height)
```

1.2.4 Summarizing data

The `summarize` function can be used to reduce multiple values down to a single value (such as a mean). It is often used in conjunction with the `by_group` function, to calculate statistics by group. In the code below, the `na.rm=TRUE` option is used to drop missing values before calculating the means.

```
library(dplyr)

# calculate mean height and mass
newdata <- summarize(starwars,
                     mean_ht = mean(height, na.rm=TRUE),
                     mean_mass = mean(mass, na.rm=TRUE))
newdata
```

```
## # A tibble: 1 x 2
##   mean_ht mean_mass
```

```
##     <dbl>     <dbl>
## 1    174.      97.3
```

```
# calculate mean height and weight by gender
newdata <- group_by(starwars, gender)
newdata <- summarize(newdata,
                     mean_ht = mean(height, na.rm=TRUE),
                     mean_wt = mean(mass, na.rm=TRUE))
newdata
```

```
## # A tibble: 5 x 3
##   gender        mean_ht mean_wt
##   <chr>           <dbl>   <dbl>
## 1 female           165.    54.0
## 2 hermaphrodite    175.  1358.
## 3 male             179.    81.0
## 4 none             200.   140.
## 5 <NA>             120.    46.3
```

1.2.5 Using pipes

Packages like `dplyr` and `tidyr` allow you to write your code in a compact format using the pipe `%>%` operator. Here is an example.

```
library(dplyr)

# calculate the mean height for women by species
newdata <- filter(starwars,
                  gender == "female")
newdata <- group_by(species)
newdata <- summarize(newdata,
                     mean_ht = mean(height, na.rm = TRUE))

# this can be written as
newdata <- starwars %>%
  filter(gender == "female") %>%
  group_by(species) %>%
  summarize(mean_ht = mean(height, na.rm = TRUE))
```

The `%>%` operator passes the result on the left to the first parameter of the function on the right.

1.2.6 Reshaping data

Some graphs require the data to be in wide format, while some graphs require the data to be in long format.

You can convert a wide dataset to a long dataset using

```
library(tidyr)
long_data <- gather(wide_data,
                    key="variable",
                    value="value",
                    sex:income)
```

Table 1.2: Wide data

id	name	sex	age	income
01	Bill	Male	22	55000
02	Bob	Male	25	75000
03	Mary	Female	18	90000

Table 1.3: Long data

id	name	variable	value
01	Bill	sex	Male
02	Bob	sex	Male
03	Mary	sex	Female
01	Bill	age	22
02	Bob	age	25
03	Mary	age	18
01	Bill	income	55000
02	Bob	income	75000
03	Mary	income	90000

Conversely, you can convert a long dataset to a wide dataset using

```
library(tidyr)
wide_data <- spread(long_data, variable, value)
```

1.2.7 Missing data

Real data are likely to contain missing values. There are three basic approaches to dealing with missing data: feature selection, listwise deletion, and imputation. Let's see how each applies to the msleep dataset from the `ggplot2` package. The `msleep` dataset describes the sleep habits of mammals and contains missing values on several variables.

1.2.7.1 Feature selection

In feature selection, you delete variables (columns) that contain too many missing values.

```
data(msleep, package="ggplot2")

# what is the proportion of missing data for each variable?
pctmiss <- colSums(is.na(msleep))/nrow(msleep)
round(pctmiss, 2)
```

```
##         name        genus         vore        order conservation
##         0.00         0.00         0.08         0.00         0.35
##  sleep_total    sleep_rem  sleep_cycle        awake      brainwt
##         0.00         0.27         0.61         0.00         0.33
##       bodywt
##         0.00
```

Sixty-one percent of the sleep_cycle values are missing. You may decide to drop it.

1.2.7.2 Listwise deletion

Listwise deletion involves deleting observations (rows) that contain missing values on *any* of the variables of interest.

```
# Create a dataset containing genus, vore, and conservation.
# Delete any rows containing missing data.
newdata <- select(msleep, genus, vore, conservation)
newdata <- na.omit(newdata)
```

1.2.7.3 Imputation

Imputation involves replacing missing values with "reasonable" guesses about what the values would have been if they had not been missing. There are several approaches, as detailed in such packages as `VIM`, `mice`, `Amelia` and `missForest`. Here we will use the `kNN` function from the `VIM` package to replace missing values with imputed values.

```
# Impute missing values using the 5 nearest neighbors
library(VIM)
newdata <- kNN(msleep, k=5)
```

Basically, for each case with a missing value, the k most similar cases not having a missing value are selected. If the missing value is numeric, the mean of those k cases is used as the imputed value. If the missing value is categorical, the most frequent value from the k cases is used. The process iterates over cases and variables until the results converge (become stable). This is a bit of an oversimplification - see Imputation with R Package VIM for the actual details.

> Important caveate: Missing values can bias the results of studies (sometimes severely). If you have a significant amount of missing data, it is probably a good idea to consult a statistician or data scientist before deleting cases or imputing missing values.

Chapter 2

Introduction to ggplot2

This section provides an brief overview of how the `ggplot2` package works. If you are simply seeking code to make a specific type of graph, feel free to skip this section. However, the material can help you understand how the pieces fit together.

2.1 A worked example

The functions in the `ggplot2` package build up a graph in layers. We'll build a a complex graph by starting with a simple graph and adding additional elements, one at a time.

The example uses data from the 1985 Current Population Survey to explore the relationship between wages (*wage*) and experience (*expr*).

```
# load data
data(CPS85 , package = "mosaicData")
```

In building a `ggplot2` graph, only the first two functions described below are required. The other functions are optional and can appear in any order.

2.1.1 ggplot

The first function in building a graph is the `ggplot` function. It specifies the

- data frame containing the data to be plotted
- the mapping of the variables to visual properties of the graph. The mappings are placed within the `aes` function (where *aes* stands for aesthetics).

```
# specify dataset and mapping
library(ggplot2)
ggplot(data = CPS85,
       mapping = aes(x = exper, y = wage))
```

Why is the graph empty? We specified that the *exper* variable should be mapped to the x-axis and that the *wage* should be mapped to the y-axis, but we haven't yet specified what we wanted placed on the graph.

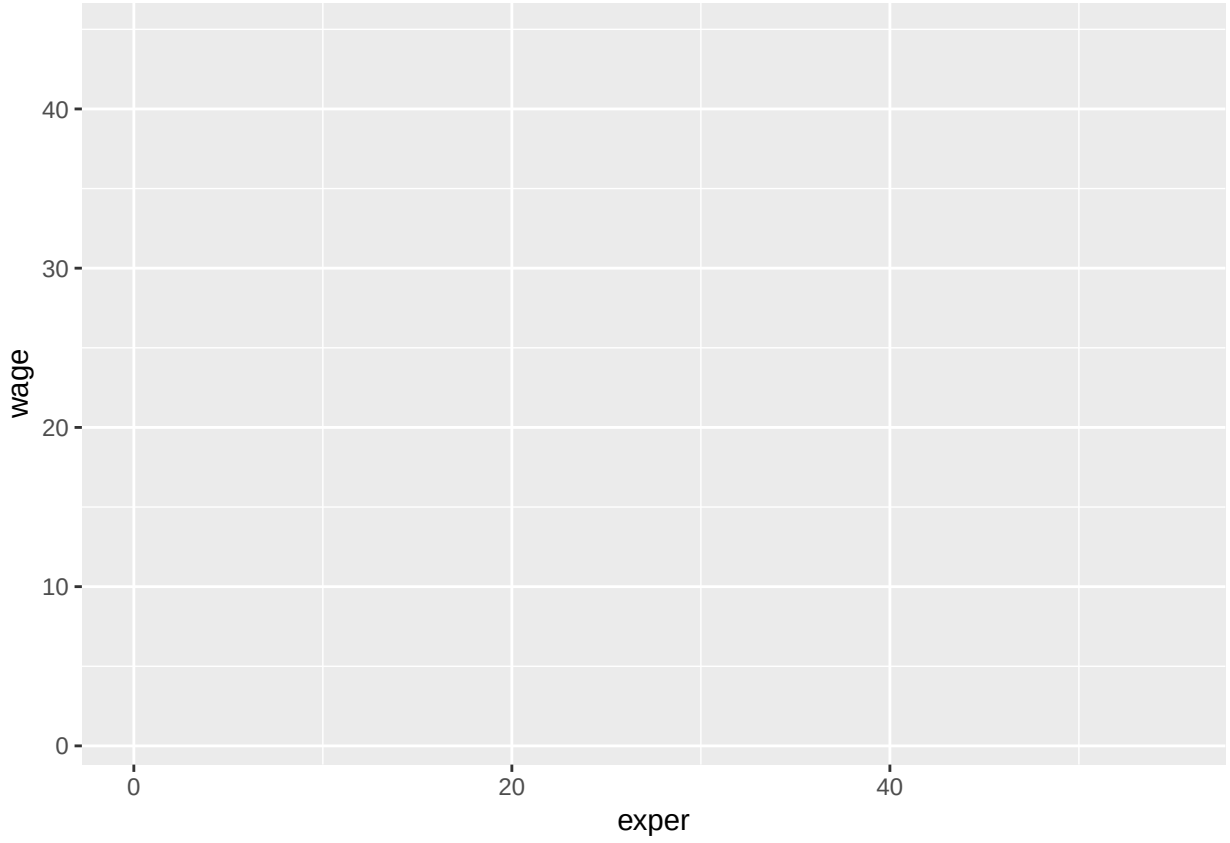


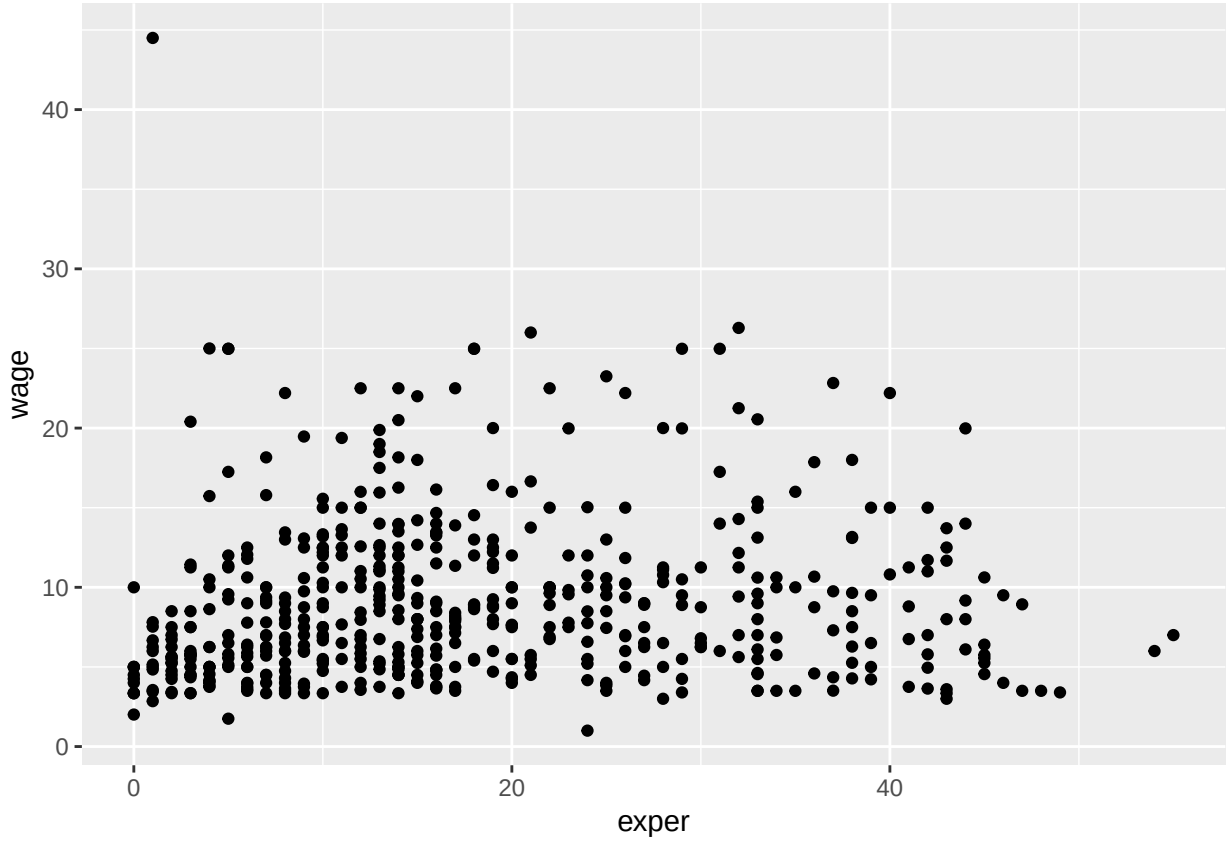
Figure 2.1: Map variables

2.1.2 geoms

Geoms are the geometric objects (points, lines, bars, etc.) that can be placed on a graph. They are added using functions that start with `geom_`. In this example, we'll add points using the `geom_point` function, creating a scatterplot.

In `ggplot2` graphs, functions are chained together using the `+` sign to build a final plot.

```
# add points
ggplot(data = CPS85,
       mapping = aes(x = exper, y = wage)) +
  geom_point()
```



The graph indicates that there is an outlier. One individual has a wage much higher than the rest. We'll delete this case before continuing.

```
# delete outlier
library(dplyr)
plotdata <- filter(CPS85, wage < 40)

# redraw scatterplot
ggplot(data = plotdata,
       mapping = aes(x = exper, y = wage)) +
  geom_point()
```

A number of parameters (options) can be specified in a `geom_` function. Options for the `geom_point` function include `color`, `size`, and `alpha`. These control the point color, size, and transparency, respectively. Trans-

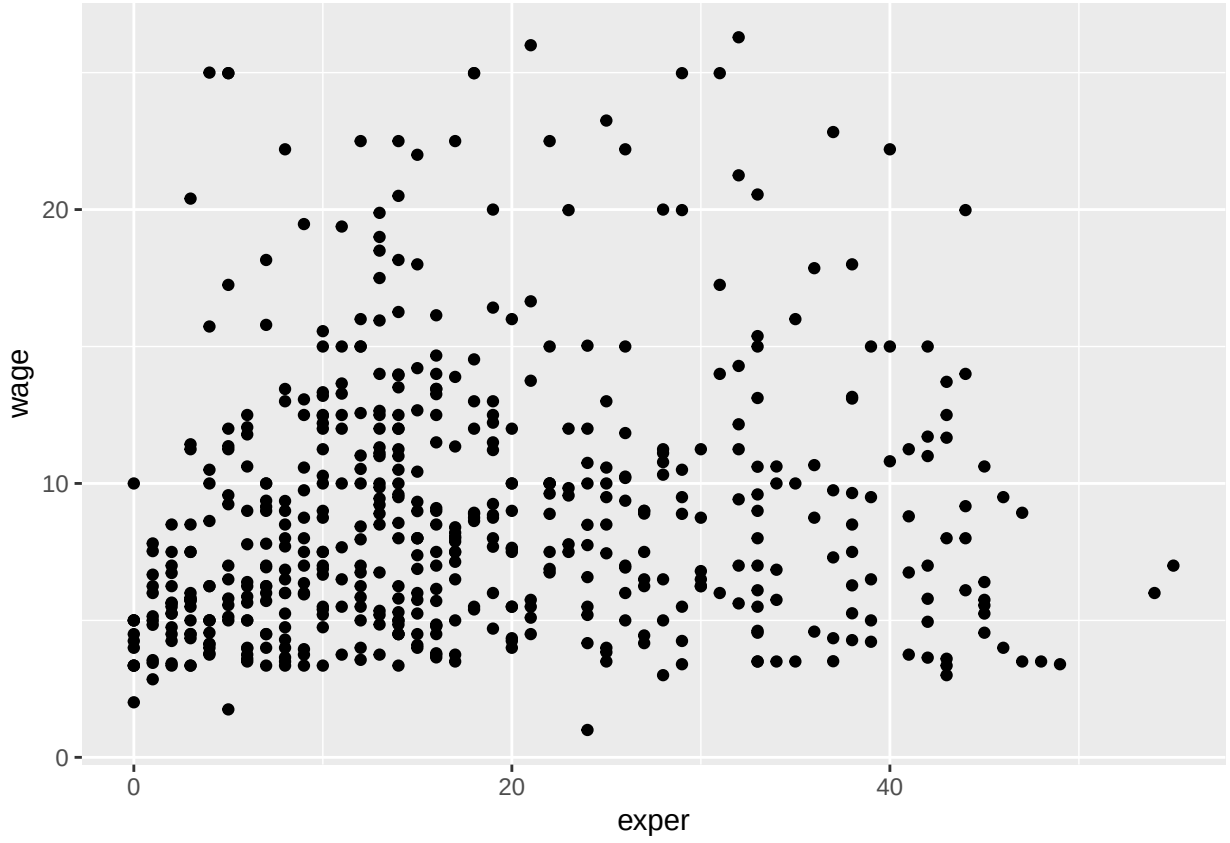


Figure 2.2: Remove outlier

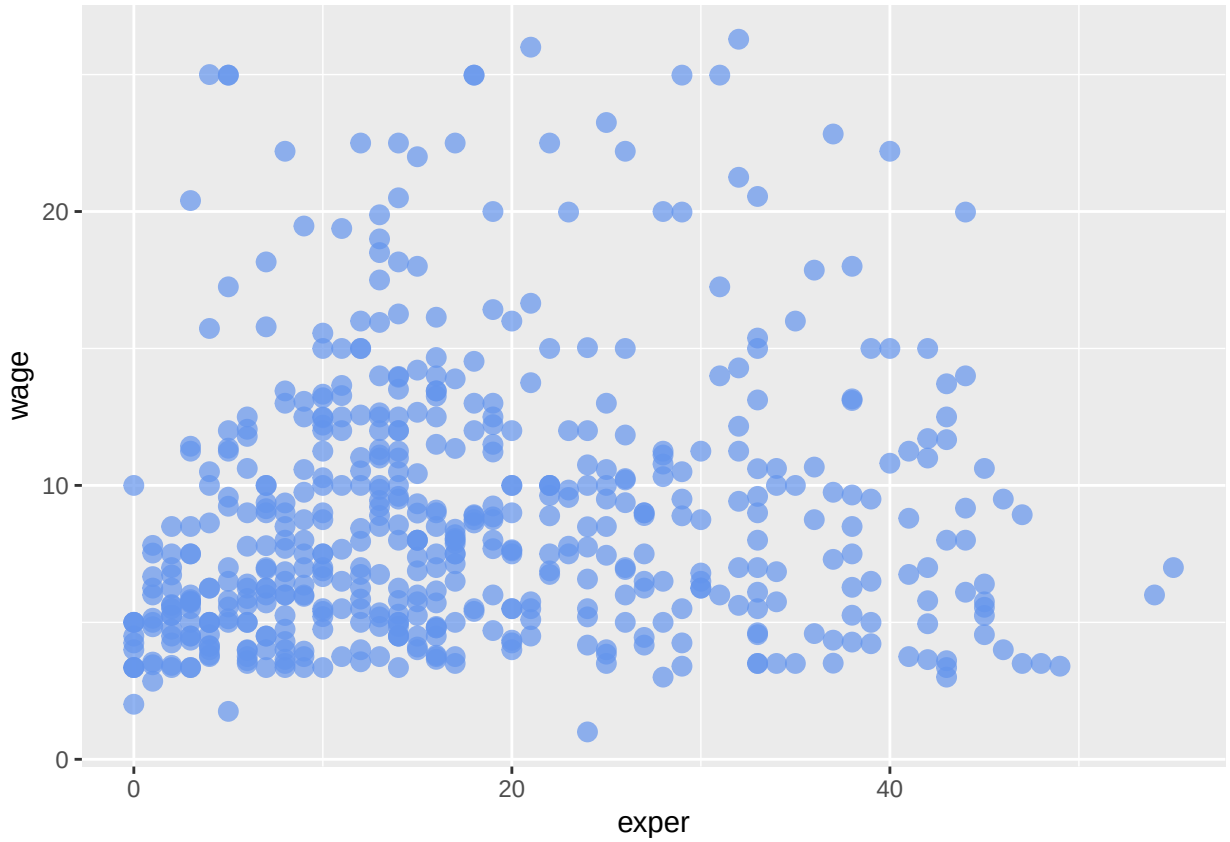


Figure 2.3: Modify point color, transparency, and size

parency ranges from 0 (completely transparent) to 1 (completely opaque). Adding a degree of transparency can help visualize overlapping points.

```
# make points blue, larger, and semi-transparent
ggplot(data = plotdata,
       mapping = aes(x = exper, y = wage)) +
  geom_point(color = "cornflowerblue",
             alpha = .7,
             size = 3)
```

Next, let's add a line of best fit. We can do this with the `geom_smooth` function. Options control the type of line (linear, quadratic, nonparametric), the thickness of the line, the line's color, and the presence or absence of a confidence interval. Here we request a linear regression (`method = lm`) line (where *lm* stands for linear model).

```
# add a line of best fit.
ggplot(data = plotdata,
       mapping = aes(x = exper, y = wage)) +
  geom_point(color = "cornflowerblue",
             alpha = .7,
             size = 3) +
  geom_smooth(method = "lm")
```

Wages appears to increase with experience.

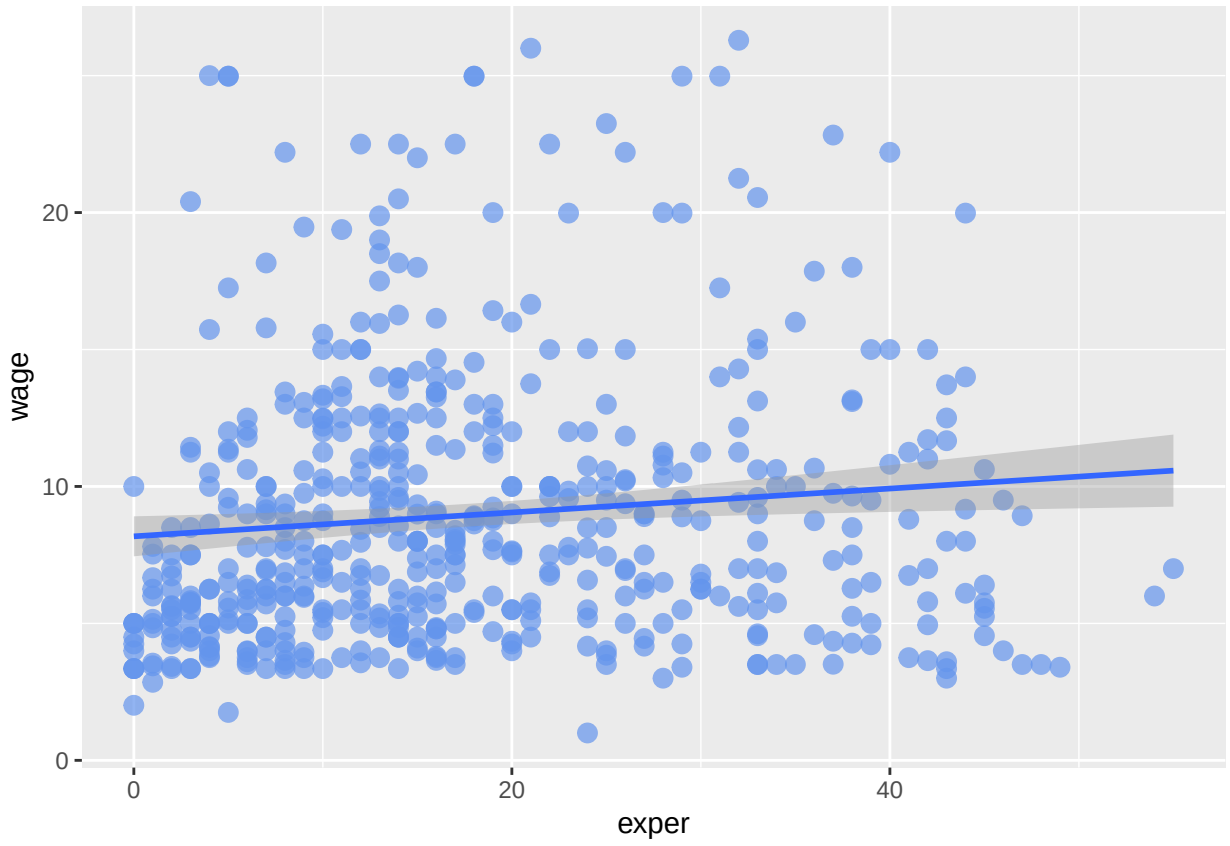


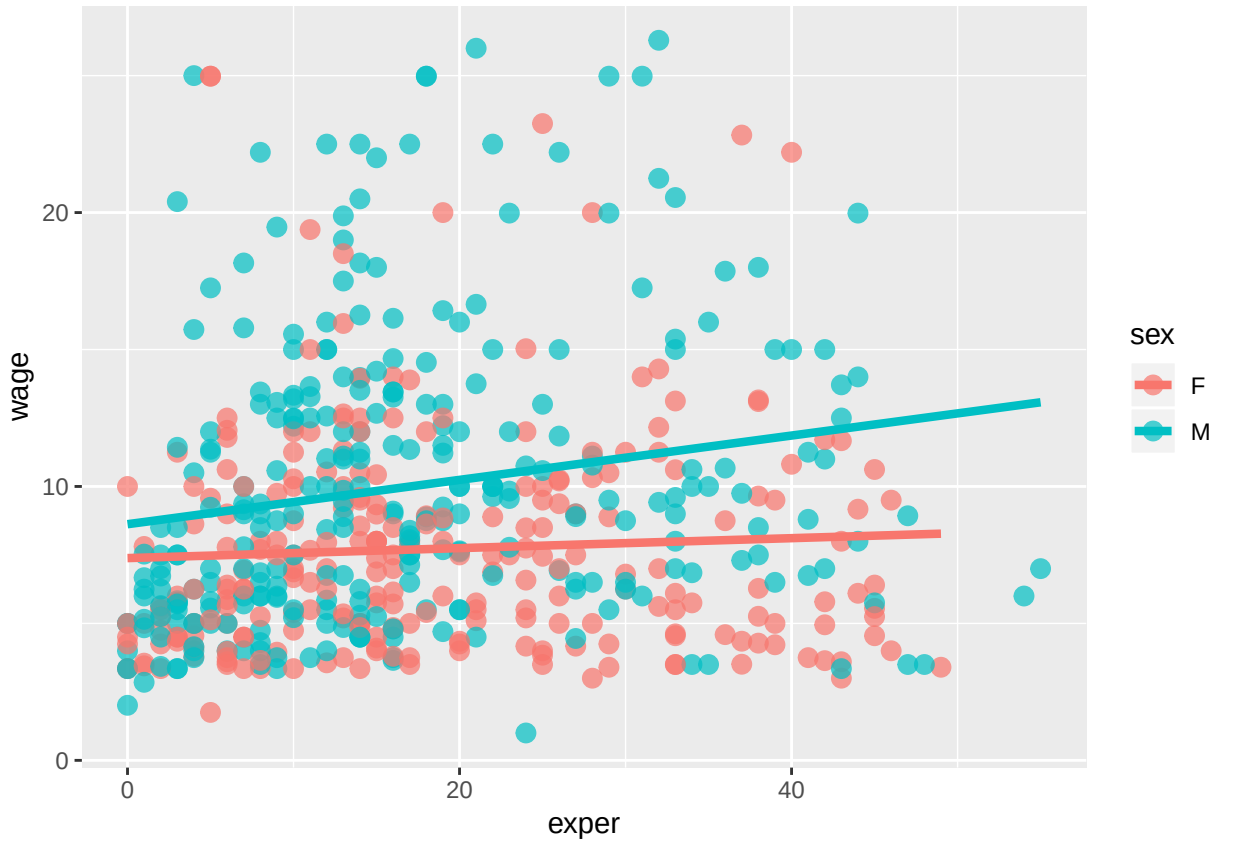
Figure 2.4: Add line of best fit

2.1.3 grouping

In addition to mapping variables to the x and y axes, variables can be mapped to the color, shape, size, transparency, and other visual characteristics of geometric objects. This allows groups of observations to be superimposed in a single graph.

Let's add sex to the plot and represent it by color.

```
# indicate sex using color
ggplot(data = plotdata,
       mapping = aes(x = exper,
                     y = wage,
                     color = sex)) +
  geom_point(alpha = .7,
             size = 3) +
  geom_smooth(method = "lm",
              se = FALSE,
              size = 1.5)
```



The `color = sex` option is placed in the `aes` function, because we are mapping a variable to an aesthetic. The `geom_smooth` option (`se = FALSE`) was added to suppresses the confidence intervals.

It appears that men tend to make more money than women. Additionally, there may be a stronger relationship between experience and wages for men than than for women.

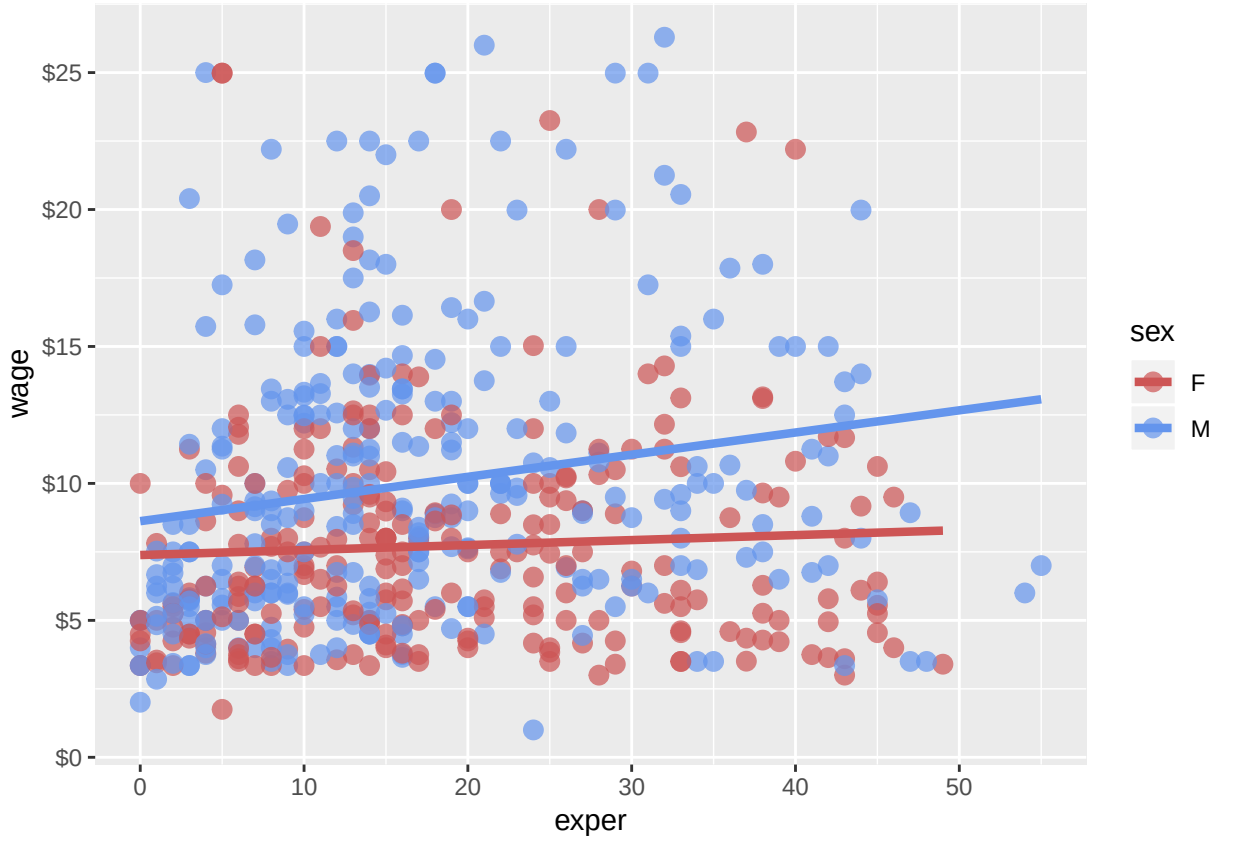


Figure 2.5: Change colors and axis labels

2.1.4 scales

Scales control how variables are mapped to the visual characteristics of the plot. Scale functions (which start with `scale_`) allow you to modify this mapping. In the next plot, we'll change the x and y axis scaling, and the colors employed.

```
# modify the x and y axes and specify the colors to be used
ggplot(data = plotdata,
       mapping = aes(x = exper,
                     y = wage,
                     color = sex)) +
  geom_point(alpha = .7,
             size = 3) +
  geom_smooth(method = "lm",
              se = FALSE,
              size = 1.5) +
  scale_x_continuous(breaks = seq(0, 60, 10)) +
  scale_y_continuous(breaks = seq(0, 30, 5),
                     label = scales::dollar) +
  scale_color_manual(values = c("indianred3",
                                "cornflowerblue"))
```

We're getting there. The numbers on the x and y axes are better, the y axis uses dollar notation, and the

colors are more attractive (IMHO).

Here is a question. Is the relationship between experience, wages and sex the same for each job sector? Let's repeat this graph once for each job sector in order to explore this.

2.1.5 facets

Facets reproduce a graph for each level a given variable (or combination of variables). Facets are created using functions that start with `facet_`. Here, facets will be defined by the eight levels of the *sector* variable.

```
# reproduce plot for each level of job sector
ggplot(data = plotdata,
       mapping = aes(x = exper,
                     y = wage,
                     color = sex)) +
  geom_point(alpha = .7) +
  geom_smooth(method = "lm",
              se = FALSE) +
  scale_x_continuous(breaks = seq(0, 60, 10)) +
  scale_y_continuous(breaks = seq(0, 30, 5),
                     label = scales::dollar) +
  scale_color_manual(values = c("indianred3",
                                "cornflowerblue")) +
  facet_wrap(~sector)
```

It appears that the differences between mean and women depend on the job sector under consideration.

2.1.6 labels

Graphs should be easy to interpret and informative labels are a key element in achieving this goal. The `labs` function provides customized labels for the axes and legends. Additionally, a custom title, subtitle, and caption can be added.

```
# add informative labels
ggplot(data = plotdata,
       mapping = aes(x = exper,
                     y = wage,
                     color = sex)) +
  geom_point(alpha = .7) +
  geom_smooth(method = "lm",
              se = FALSE) +
  scale_x_continuous(breaks = seq(0, 60, 10)) +
  scale_y_continuous(breaks = seq(0, 30, 5),
                     label = scales::dollar) +
  scale_color_manual(values = c("indianred3",
                                "cornflowerblue")) +
  facet_wrap(~sector) +
  labs(title = "Relationship between wages and experience",
       subtitle = "Current Population Survey",
       caption = "source: http://mosaic-web.org/",
       x = " Years of Experience",
       y = "Hourly Wage",
       color = "Gender")
```

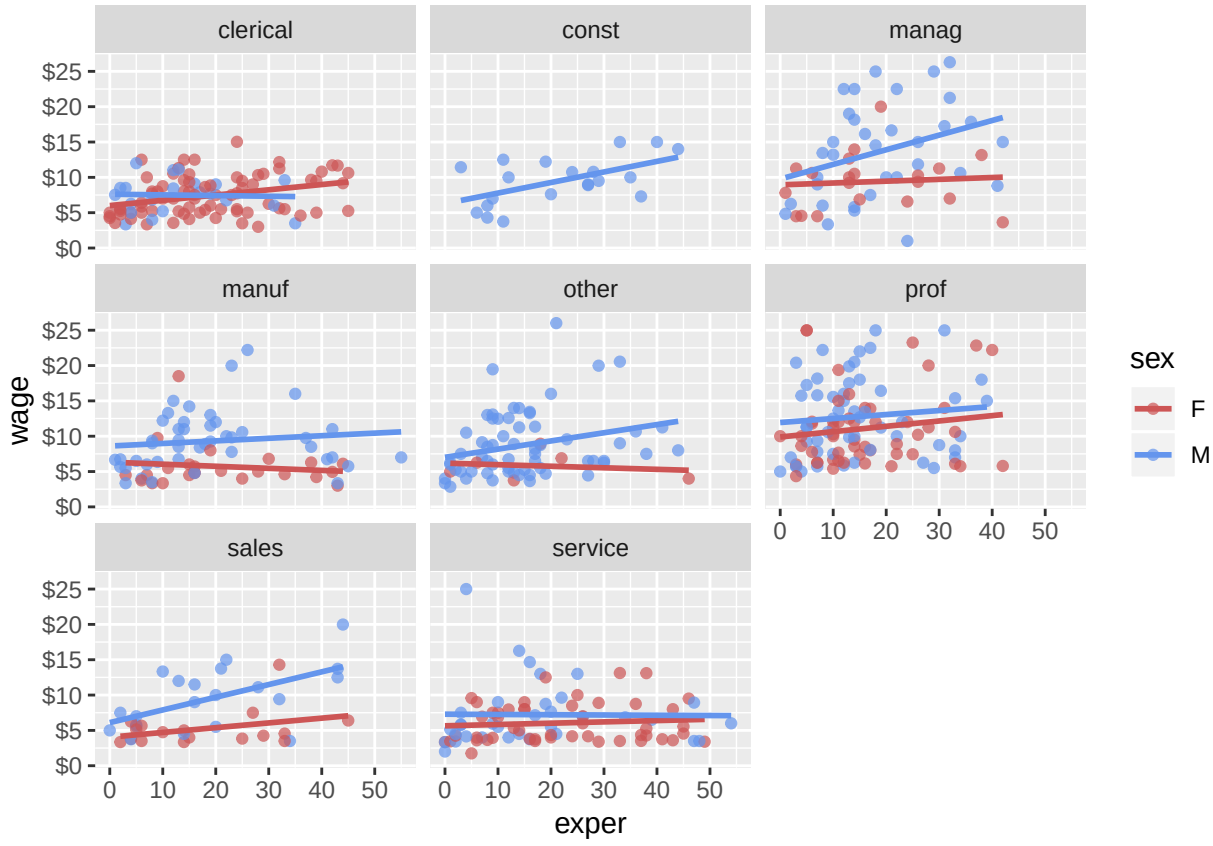
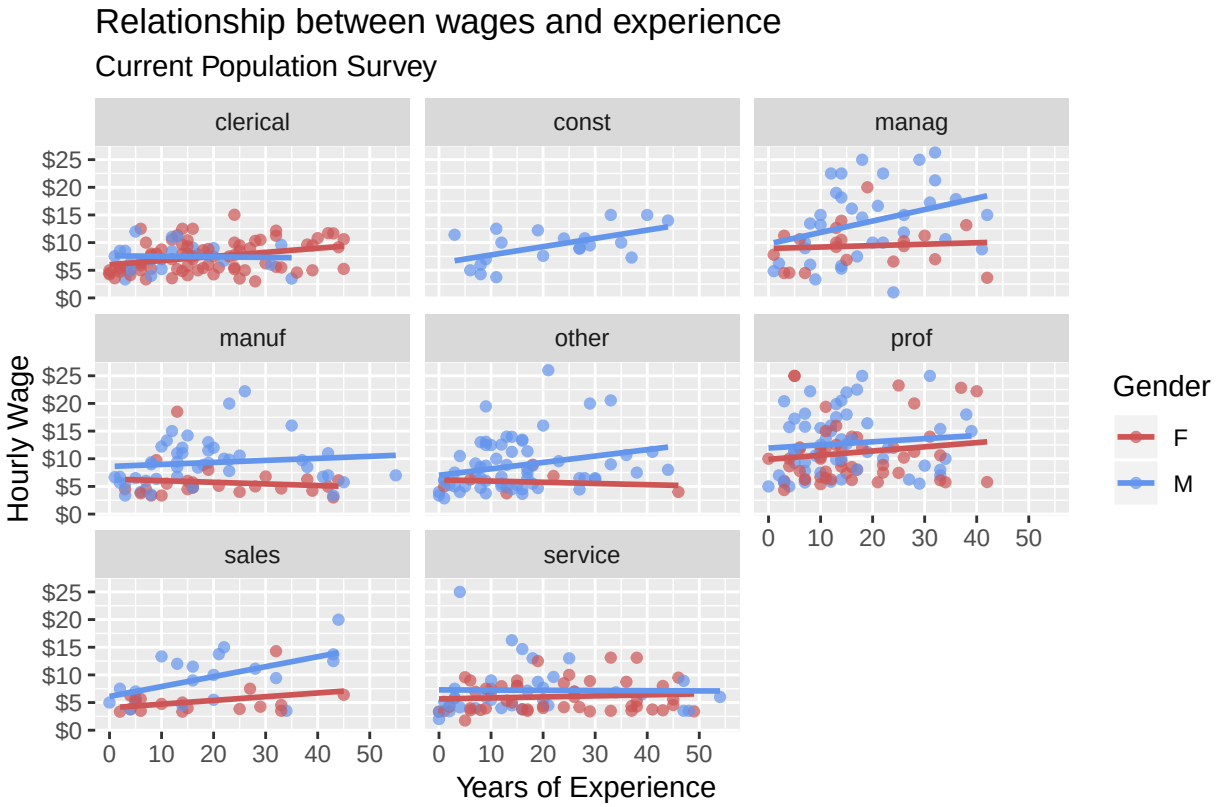


Figure 2.6: Add job sector, using faceting



Now a viewer doesn't need to guess what the labels *expr* and *wage* mean, or where the data come from.

2.1.7 themes

Finally, we can fine tune the appearance of the graph using themes. Theme functions (which start with `theme_`) control background colors, fonts, grid-lines, legend placement, and other non-data related features of the graph. Let's use a cleaner theme.

```
# use a minimalist theme
ggplot(data = plotdata,
       mapping = aes(x = exper,
                     y = wage,
                     color = sex)) +
  geom_point(alpha = .6) +
  geom_smooth(method = "lm",
              se = FALSE) +
  scale_x_continuous(breaks = seq(0, 60, 10)) +
  scale_y_continuous(breaks = seq(0, 30, 5),
                     label = scales::dollar) +
  scale_color_manual(values = c("indianred3",
                                "cornflowerblue")) +
  facet_wrap(~sector) +
  labs(title = "Relationship between wages and experience",
       subtitle = "Current Population Survey",
       caption = "source: http://mosaic-web.org/",
       x = " Years of Experience",
```

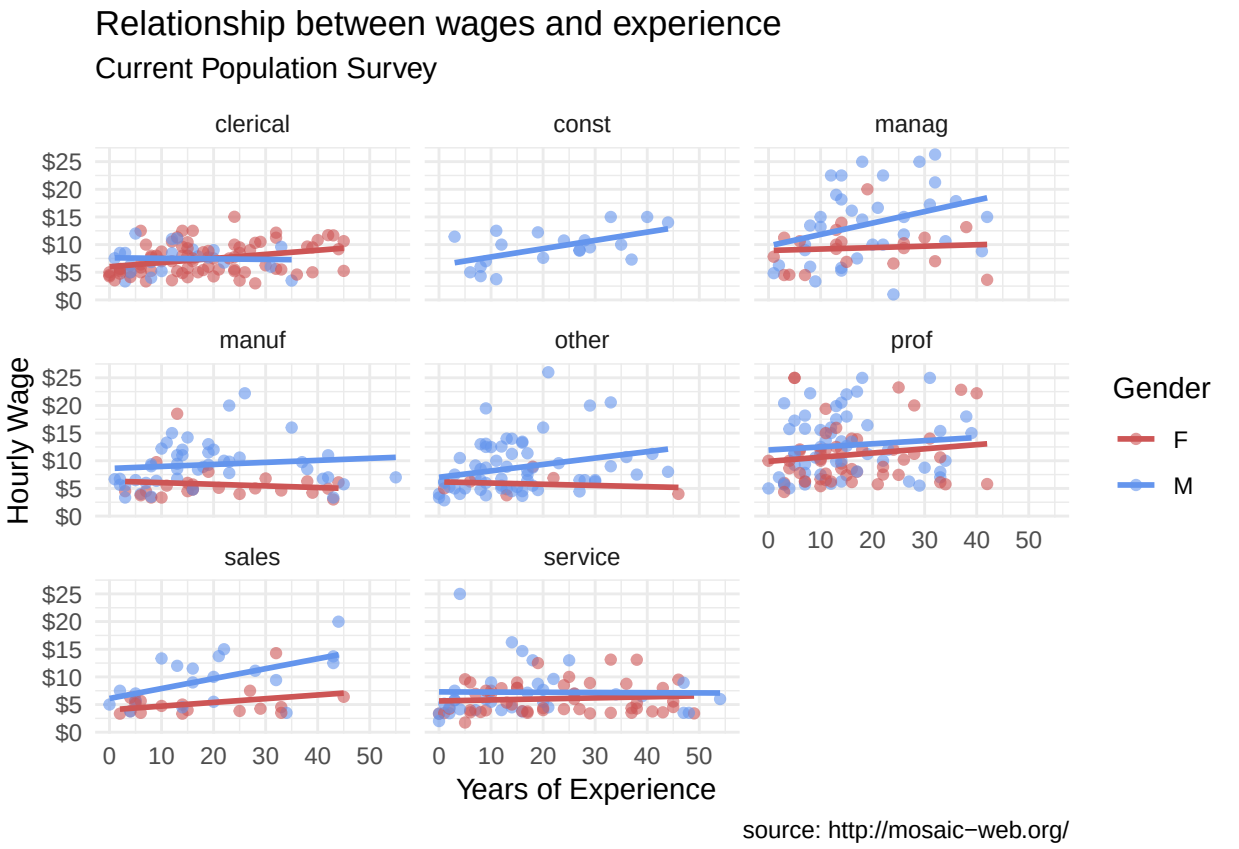


Figure 2.7: Use a simpler theme

```
       y = "Hourly Wage",
       color = "Gender") +
  theme_minimal()
```

Now we have something. It appears that men earn more than women in management, manufacturing, sales, and the "other" category. They are most similar in clerical, professional, and service positions. The data contain no women in the construction sector. For management positions, wages appear to be related to experience for men, but not for women (this may be the most interesting finding). This also appears to be true for sales.

Of course, these findings are tentative. They are based on a limited sample size and do not involve statistical testing to assess whether differences may be due to chance variation.

2.2 Placing the `data` and `mapping` options

Plots created with `ggplot2` always start with the `ggplot` function. In the examples above, the `data` and `mapping` options were placed in this function. In this case they apply to each `geom_` function that follows.

You can also place these options directly within a `geom`. In that case, they only apply only to that specific geom.

Consider the following graph.

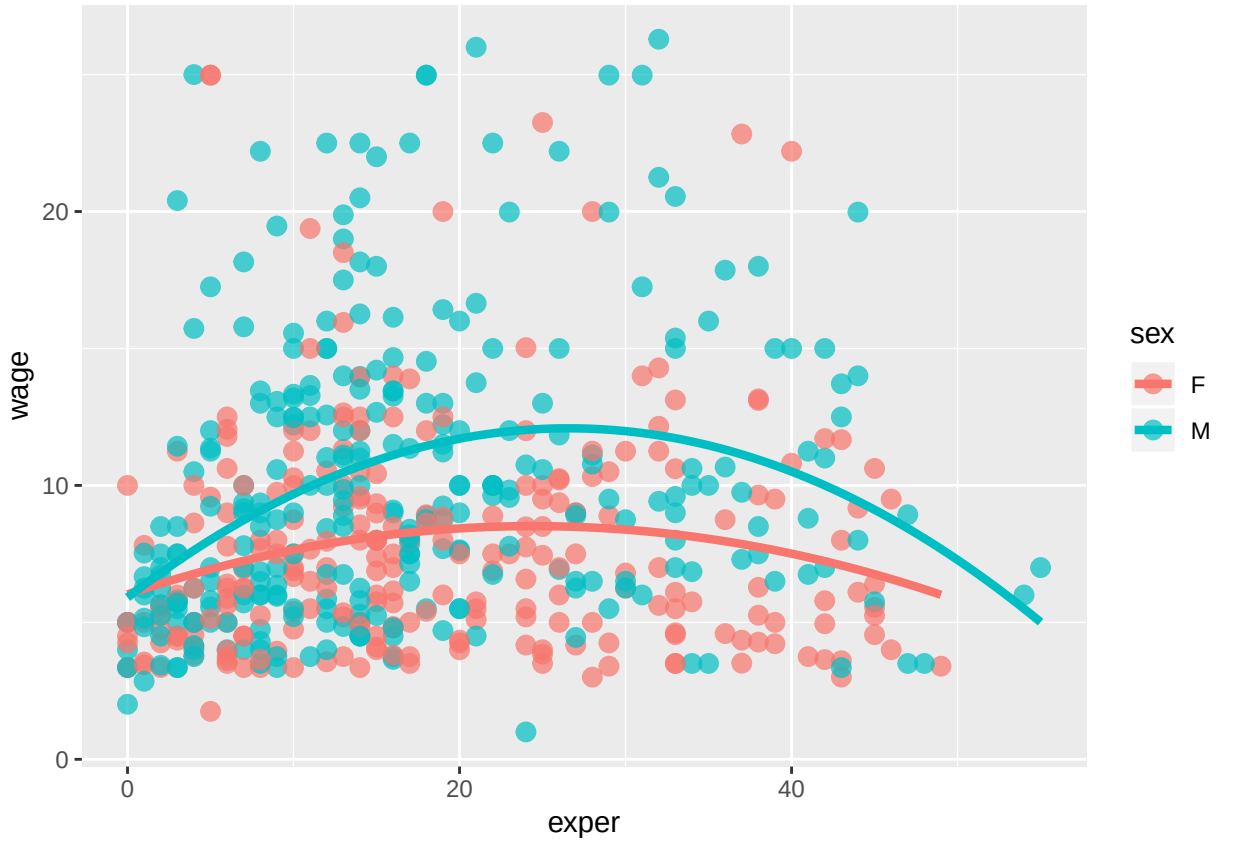


Figure 2.8: Color mapping in ggplot function

```
# placing color mapping in the ggplot function
ggplot(plotdata,
       aes(x = exper,
           y = wage,
           color = sex)) +
  geom_point(alpha = .7,
             size = 3) +
  geom_smooth(method = "lm",
              formula = y ~ poly(x,2),
              se = FALSE,
              size = 1.5)
```

Since the mapping of sex to color appears in the `ggplot` function, it applies to *both* `geom_point` and `geom_smooth`. The color of the point indicates the sex, and a separate colored trend line is produced for men and women. Compare this to

```
# placing color mapping in the geom_point function
ggplot(plotdata,
       aes(x = exper,
           y = wage)) +
  geom_point(aes(color = sex),
             alpha = .7,
             size = 3) +
```

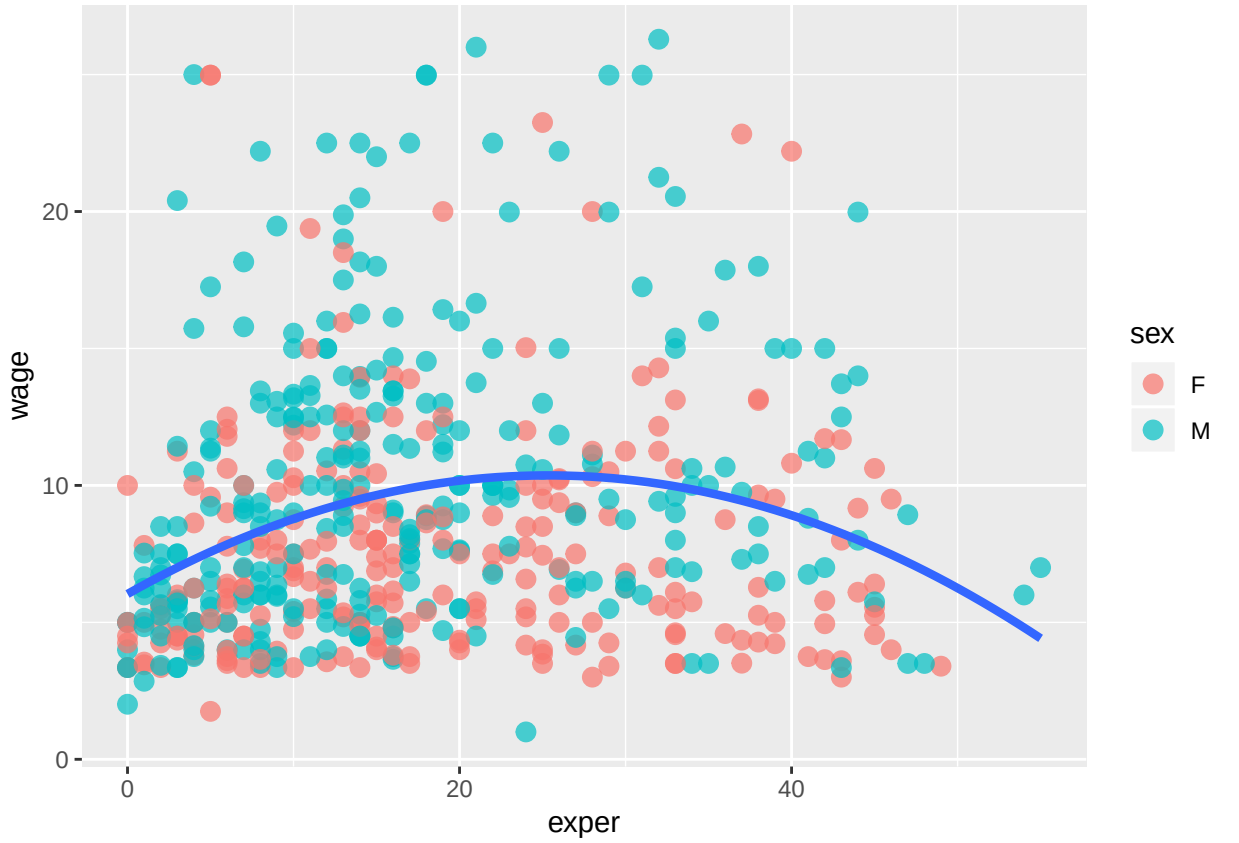


Figure 2.9: Color mapping in ggplot function

```
  geom_smooth(method = "lm",
              formula = y ~ poly(x,2),
              se = FALSE,
              size = 1.5)
```

Since the sex to color mapping only appears in the `geom_point` function, it is only used there. A single trend line is created for all observations.

Most of the examples in this book place the data and mapping options in the `ggplot` function. Additionally, the phrases *data=* and *mapping=* are omitted since the first option always refers to data and the second option always refers to mapping.

2.3 Graphs as objects

A `ggplot2` graph can be saved as a named R object (like a data frame), manipulated further, and then printed or saved to disk.

```
# prepare data
data(CPS85 , package = "mosaicData")
plotdata <- CPS85[CPS85$wage < 40,]
```

```
# create scatterplot and save it
myplot <- ggplot(data = plotdata,
                 aes(x = exper, y = wage)) +
            geom_point()

# print the graph
myplot

# make the points larger and blue
# then print the graph
myplot <- myplot + geom_point(size = 3, color = "blue")
myplot

# print the graph with a title and line of best fit
# but don't save those changes
myplot + geom_smooth(method = "lm") +
  labs(title = "Mildly interesting graph")

# print the graph with a black and white theme
# but don't save those changes
myplot + theme_bw()
```

This can be a real time saver (and help you avoid carpal tunnel syndrome). It is also handy when saving graphs programmatically.

Now it's time to try out other types of graphs.

Chapter 3

Univariate Graphs

Univariate graphs plot the distribution of data from a single variable. The variable can be categorical (e.g., race, sex) or quantitative (e.g., age, weight).

3.1 Categorical

The distribution of a single categorical variable is typically plotted with a bar chart, a pie chart, or (less commonly) a tree map.

3.1.1 Bar chart

The Marriage dataset contains the marriage records of 98 individuals in Mobile County, Alabama. Below, a bar chart is used to display the distribution of wedding participants by race.

```
library(ggplot2)
data(Marriage, package = "mosaicData")

# plot the distribution of race
ggplot(Marriage, aes(x = race)) +
  geom_bar()
```

The majority of participants are white, followed by black, with very few Hispanics or American Indians.

You can modify the bar fill and border colors, plot labels, and title by adding options to the `geom_bar` function.

```
# plot the distribution of race with modified colors and labels
ggplot(Marriage, aes(x = race)) +
  geom_bar(fill = "cornflowerblue",
           color="black") +
  labs(x = "Race",
       y = "Frequency",
       title = "Participants by race")
```

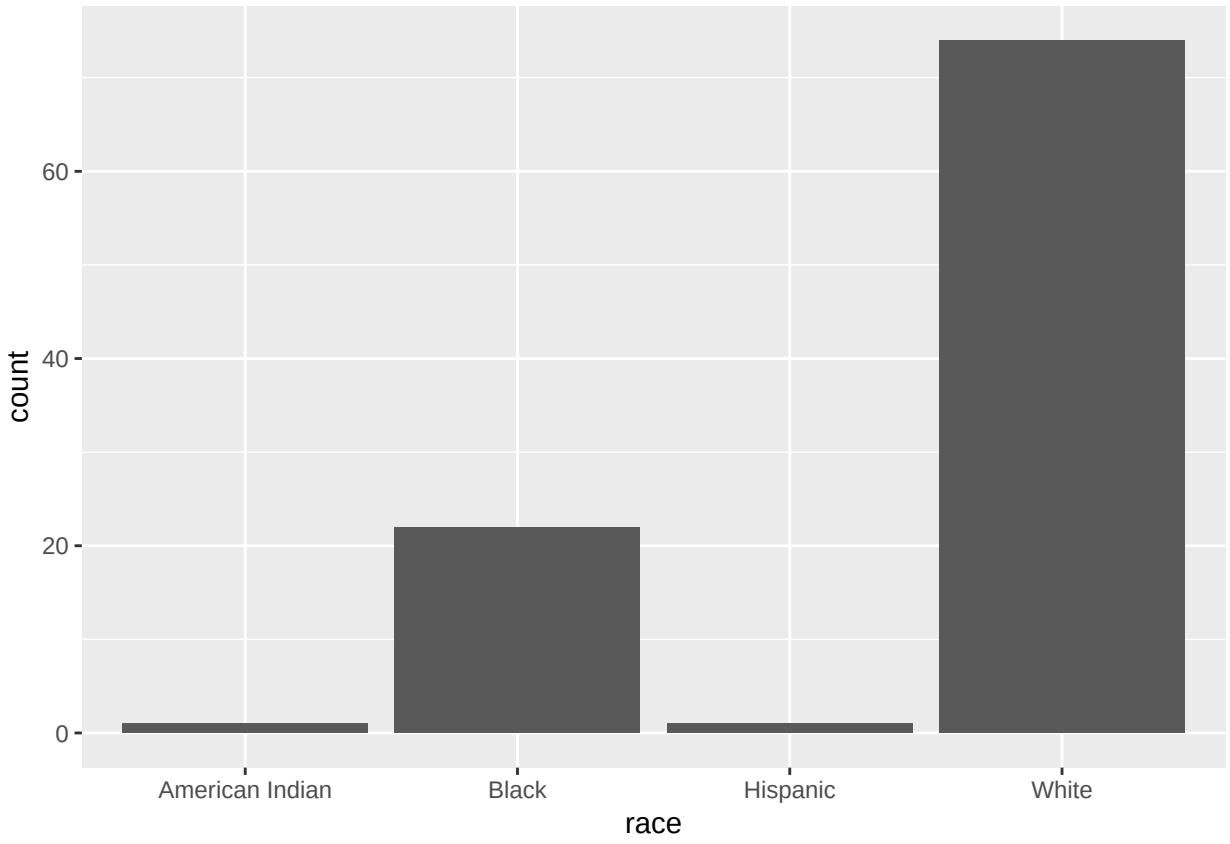


Figure 3.1: Simple barchart

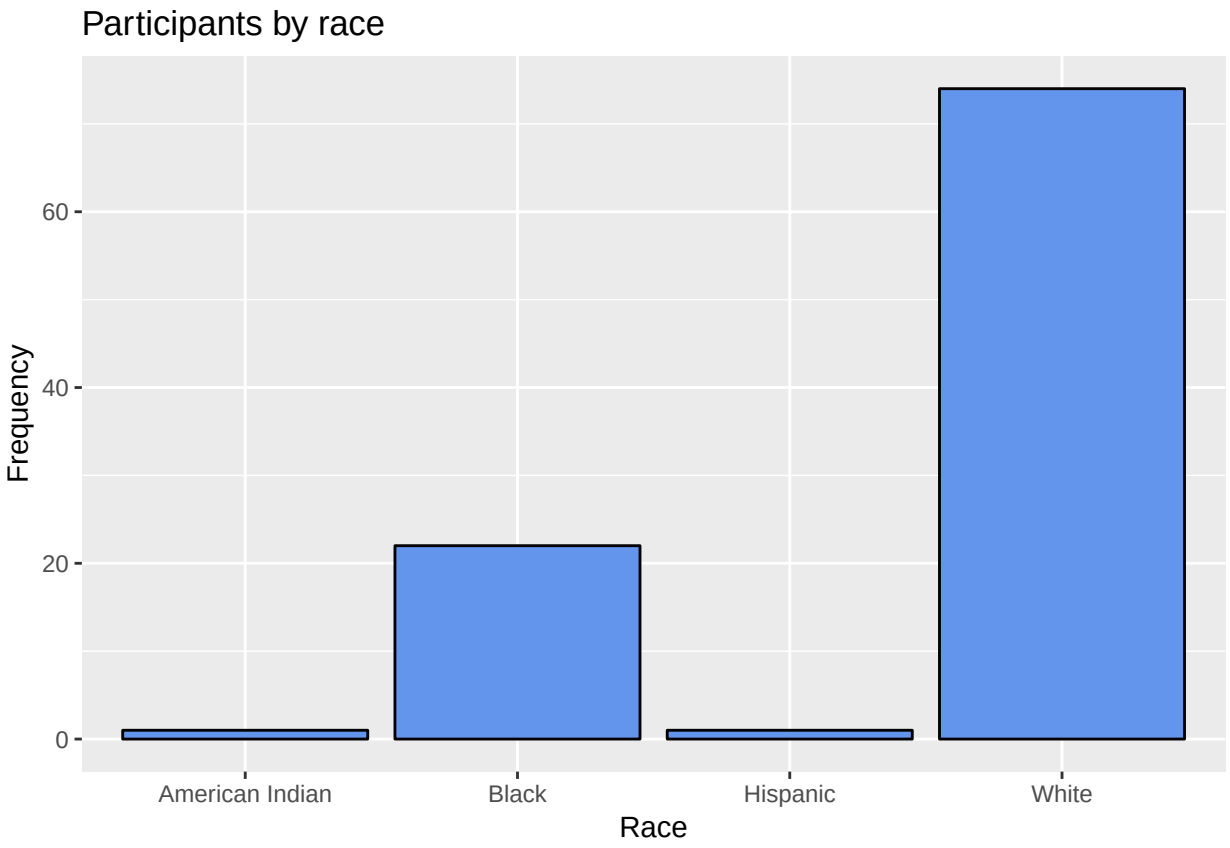


Figure 3.2: Barchart with modified colors, labels, and title

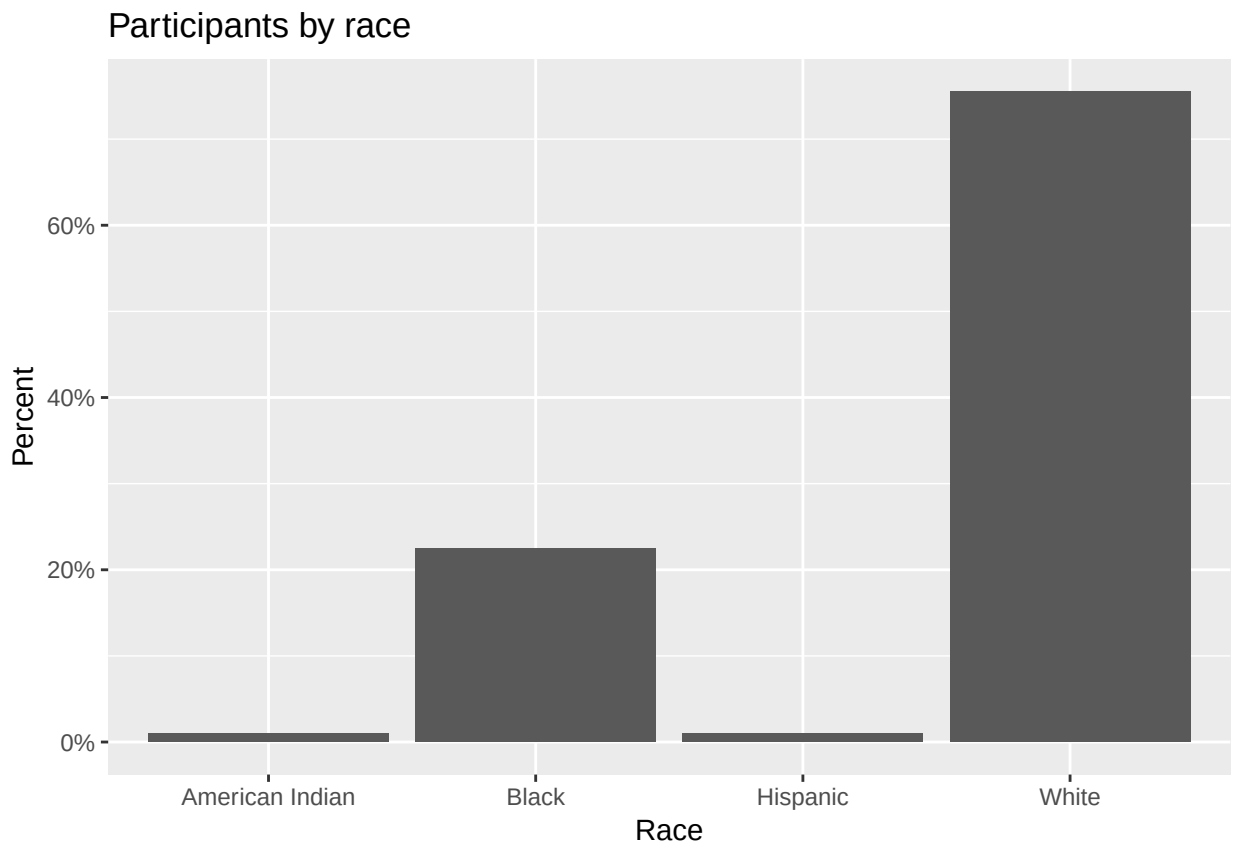


Figure 3.3: Barchart with percentages

3.1.1.1 Percents

Bars can represent percents rather than counts. For bar charts, the code `aes(x=race)` is actually a shortcut for `aes(x = race, y = ..count..)`, where `..count..` is a special variable representing the frequency within each category. You can use this to calculate percentages, by specifying the `y` variable explicitly.

```
# plot the distribution as percentages
ggplot(Marriage,
       aes(x = race,
           y = ..count.. / sum(..count..))) +
  geom_bar() +
  labs(x = "Race",
       y = "Percent",
       title  = "Participants by race") +
  scale_y_continuous(labels = scales::percent)
```

In the code above, the `scales` package is used to add % symbols to the y-axis labels.

3.1.1.2 Sorting categories

It is often helpful to sort the bars by frequency. In the code below, the frequencies are calculated explicitly. Then the `reorder` function is used to sort the categories by the frequency. The option `stat="identity"` tells the plotting function not to calculate counts, because they are supplied directly.

Table 3.1: plotdata

race	n
American Indian	1
Black	22
Hispanic	1
White	74

```
# calculate number of participants in
# each race category
library(dplyr)
plotdata <- Marriage %>%
 count(race)
```

The resulting dataset is give below.

This new dataset is then used to create the graph.

```
# plot the bars in ascending order
ggplot(plotdata,
       aes(x = reorder(race, n),
           y = n)) +
  geom_bar(stat = "identity") +
  labs(x = "Race",
       y = "Frequency",
       title  = "Participants by race")
```

The graph bars are sorted in ascending order. Use `reorder(race, -n)` to sort in descending order.

3.1.1.3 Labeling bars

Finally, you may want to label each bar with its numerical value.

```
# plot the bars with numeric labels
ggplot(plotdata,
       aes(x = race,
           y = n)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = n),
            vjust=-0.5) +
  labs(x = "Race",
       y = "Frequency",
       title  = "Participants by race")
```

Here `geom_text` adds the labels, and `vjust` controls vertical justification. See Annotations for more details.

Putting these ideas together, you can create a graph like the one below. The minus sign in `reorder(race, -pct)` is used to order the bars in descending order.

```
library(dplyr)
library(scales)
```

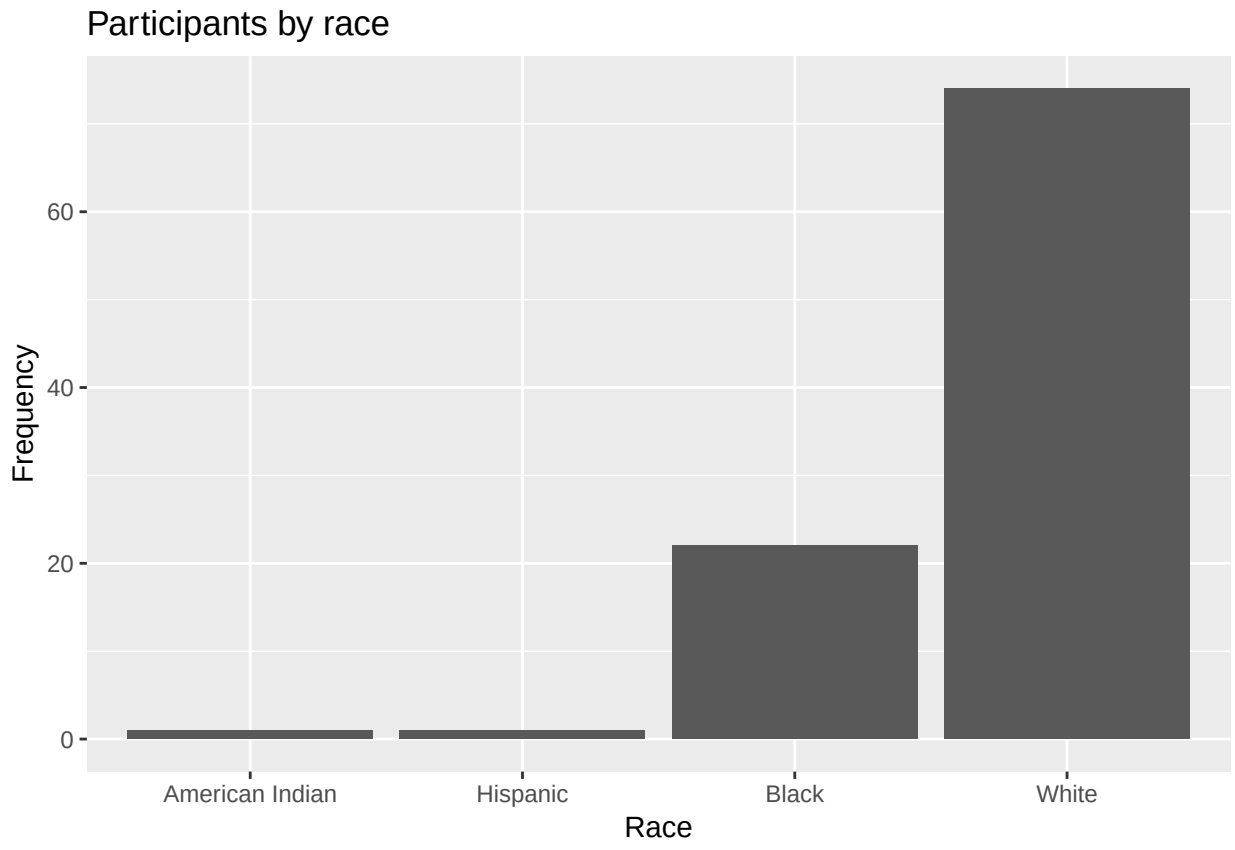


Figure 3.4: Sorted bar chart

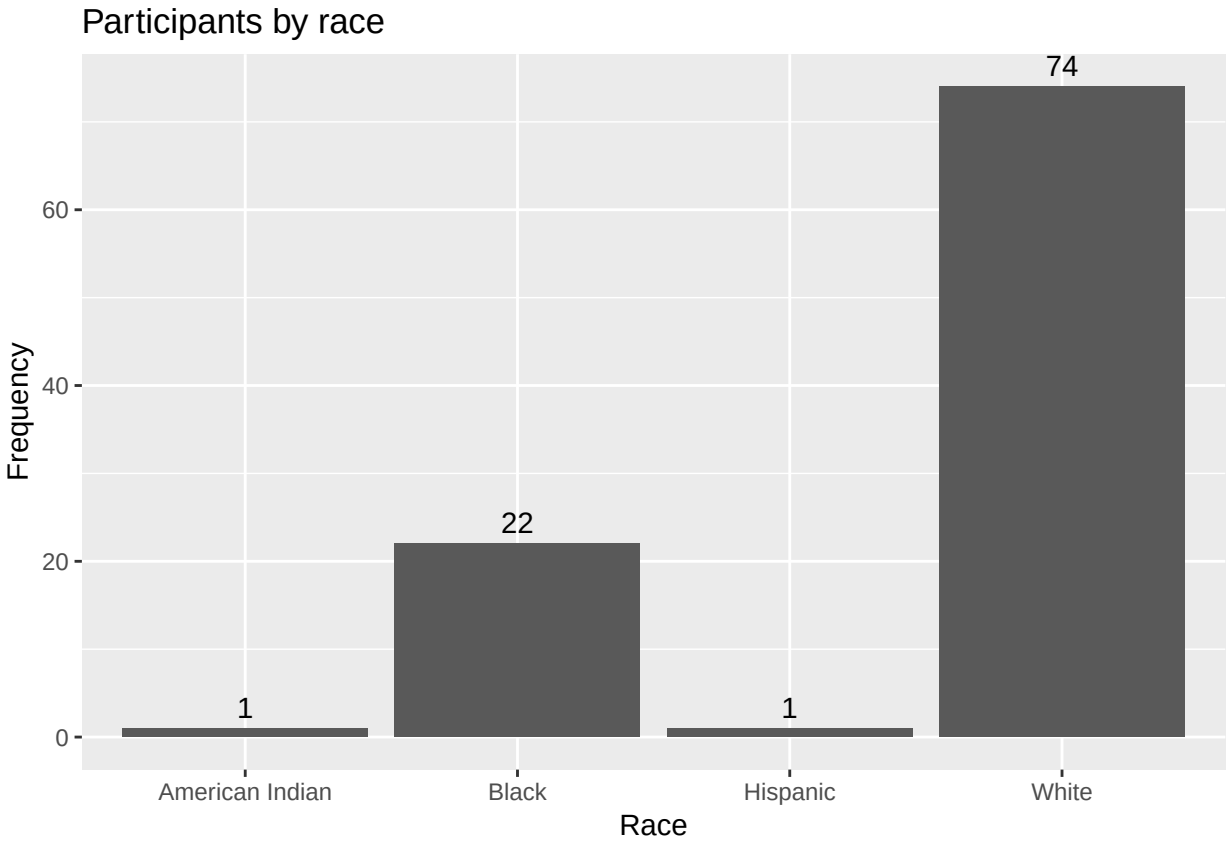


Figure 3.5: Bar chart with numeric labels

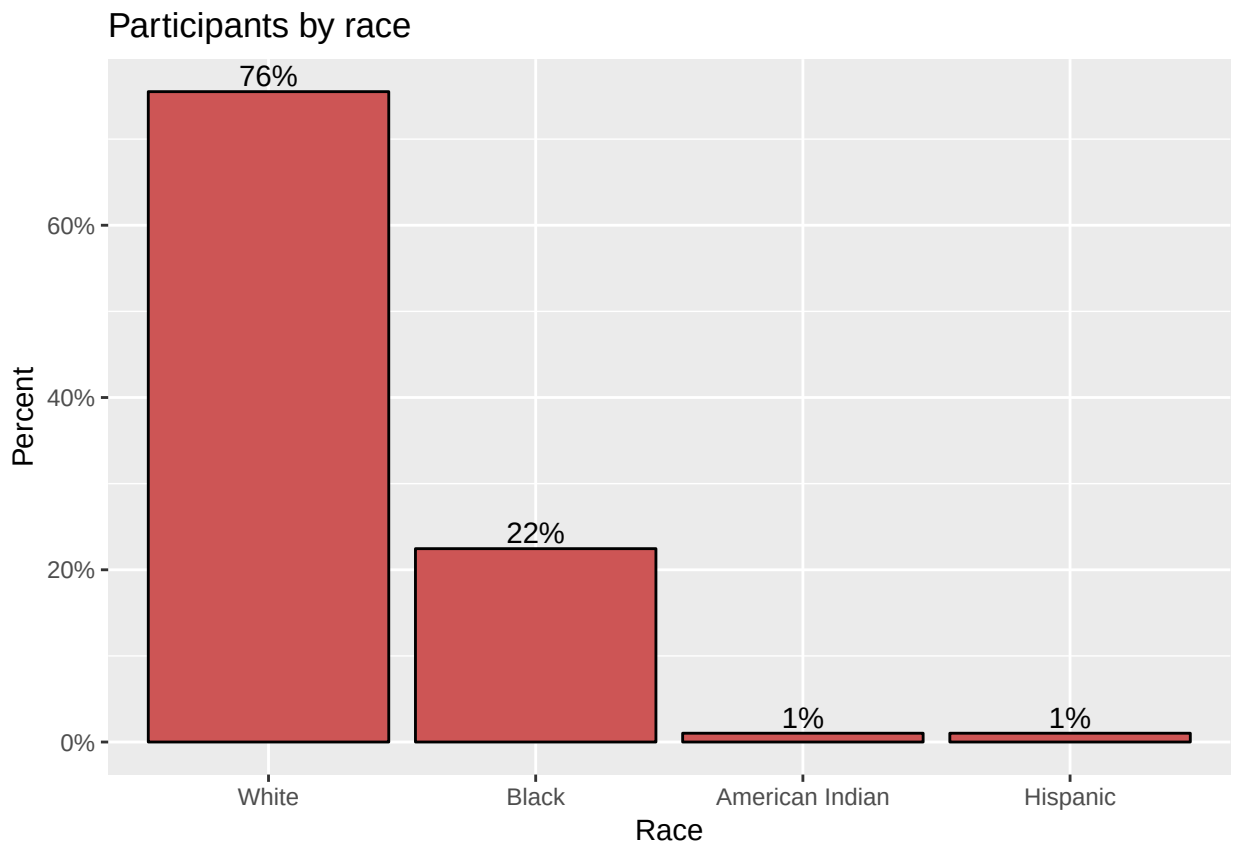


Figure 3.6: Sorted bar chart with percent labels

```
plotdata <- Marriage %>%
  count(race) %>%
  mutate(pct = n / sum(n),
         pctlabel = paste0(round(pct*100), "%"))

# plot the bars as percentages,
# in decending order with bar labels
ggplot(plotdata,
       aes(x = reorder(race, -pct),
           y = pct)) +
  geom_bar(stat = "identity",
           fill = "indianred3",
           color = "black") +
  geom_text(aes(label = pctlabel),
            vjust = -0.25) +
  scale_y_continuous(labels = percent) +
  labs(x = "Race",
       y = "Percent",
       title  = "Participants by race")
```

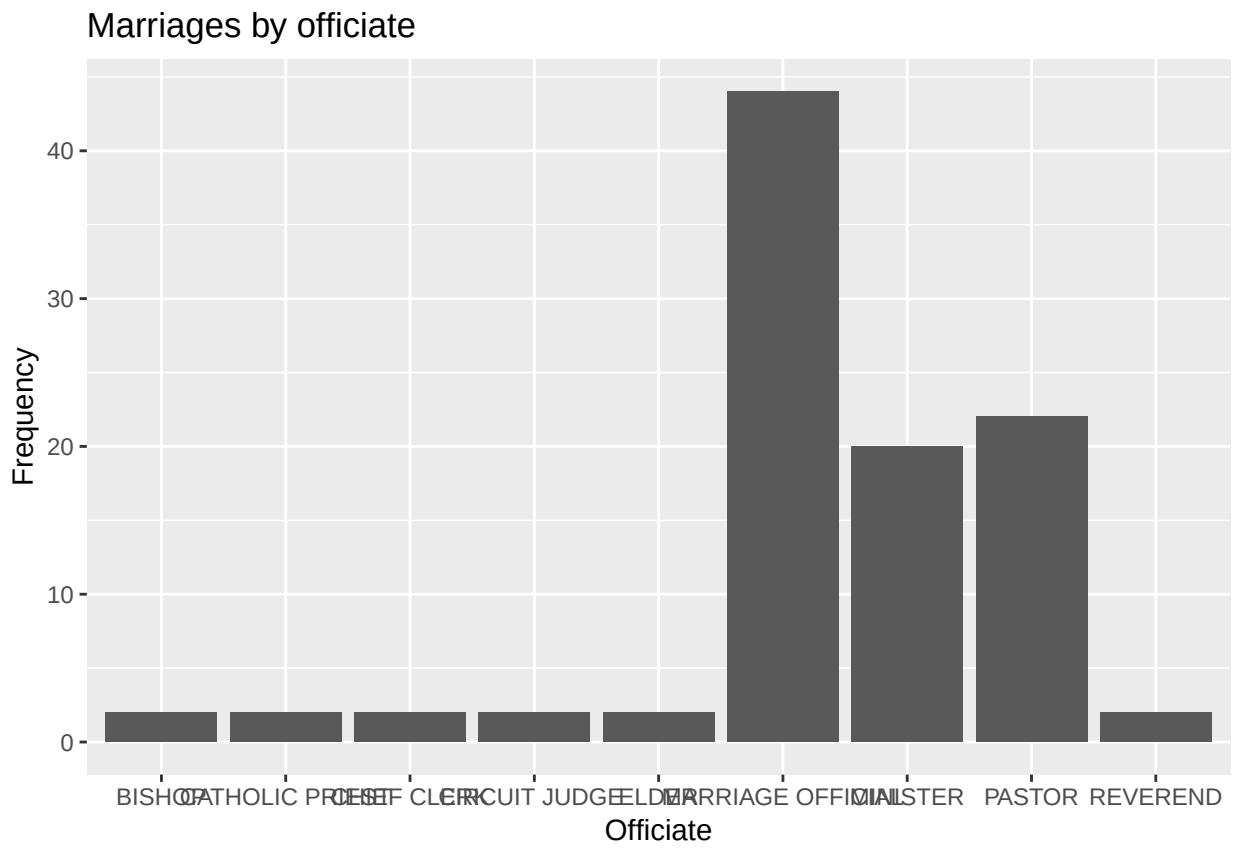


Figure 3.7: Barchart with problematic labels

3.1.1.4 Overlapping labels

Category labels may overlap if (1) there are many categories or (2) the labels are long. Consider the distribution of marriage officials.

```
# basic bar chart with overlapping labels
ggplot(Marriage, aes(x = officialTitle)) +
  geom_bar() +
  labs(x = "Officiate",
       y = "Frequency",
       title = "Marriages by officiate")
```

In this case, you can flip the x and y axes.

```
# horizontal bar chart
ggplot(Marriage, aes(x = officialTitle)) +
  geom_bar() +
  labs(x = "",
       y = "Frequency",
       title = "Marriages by officiate") +
  coord_flip()
```

Alternatively, you can rotate the axis labels.

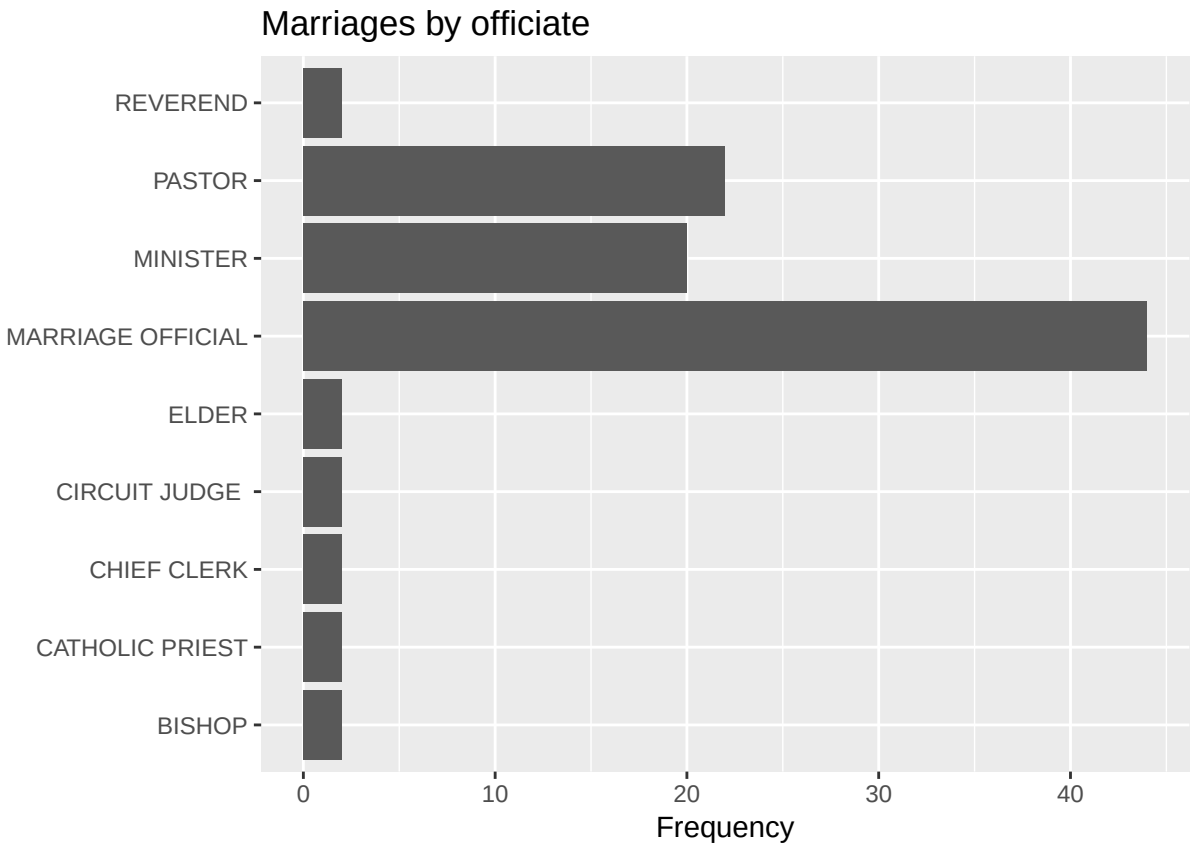


Figure 3.8: Horizontal barchart

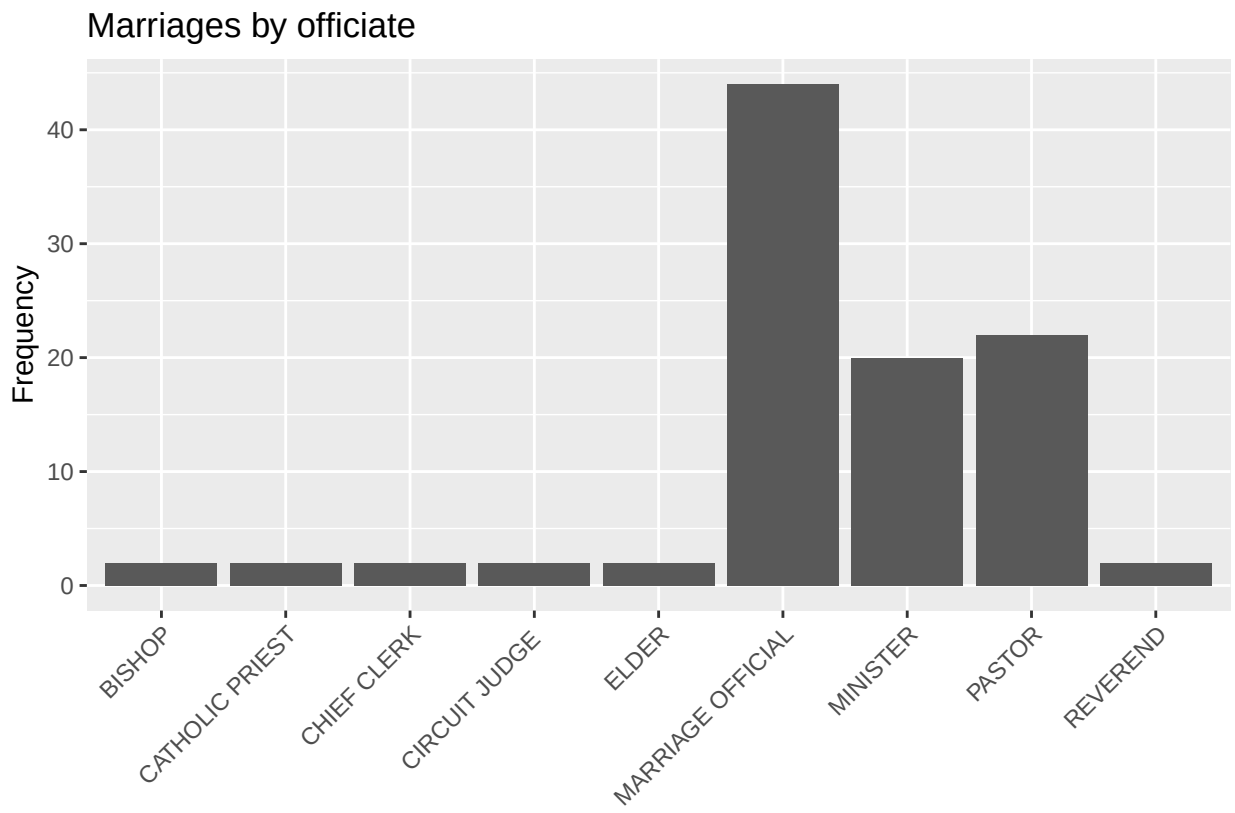
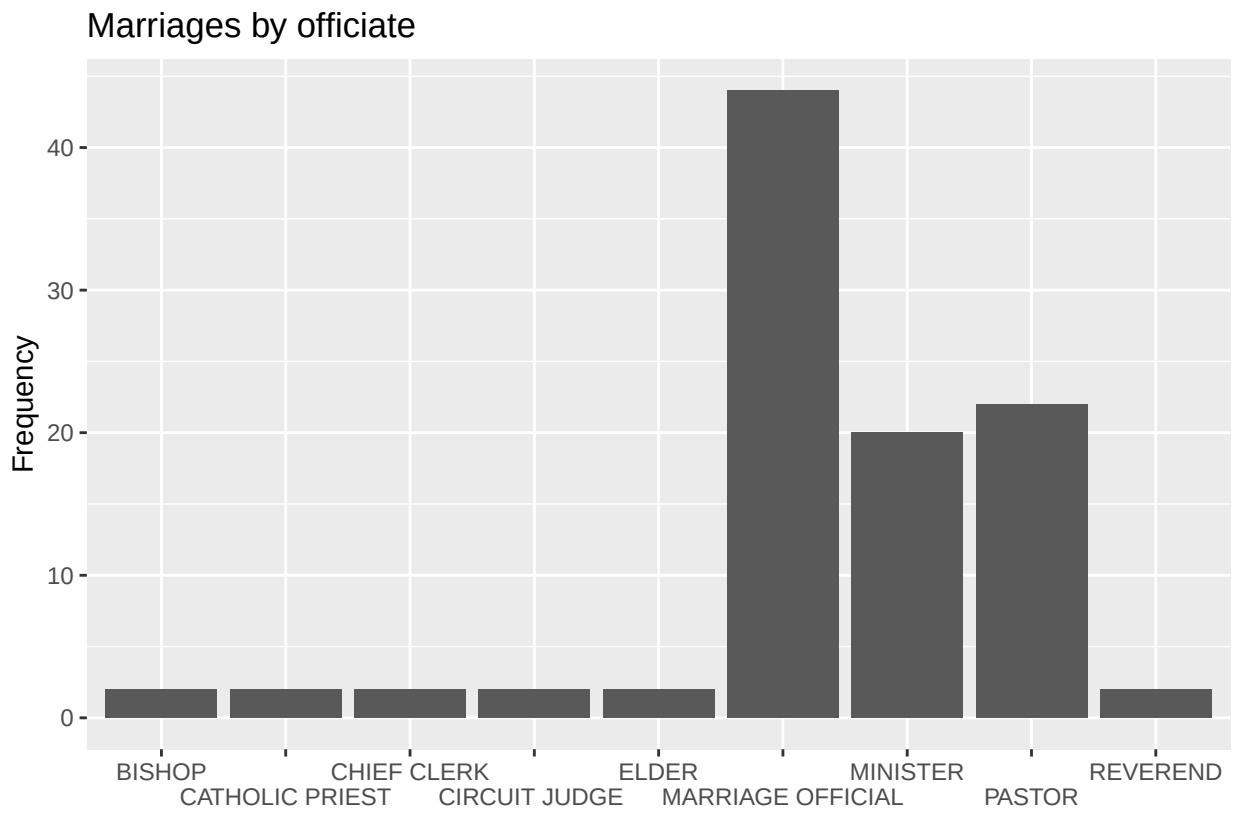


Figure 3.9: Barchart with rotated labels

```
# bar chart with rotated labels
ggplot(Marriage, aes(x = officialTitle)) +
  geom_bar() +
  labs(x = "",
       y = "Frequency",
       title = "Marriages by officiate") +
  theme(axis.text.x = element_text(angle = 45,
                                   hjust = 1))
```

Finally, you can try staggering the labels. The trick is to add a newline `\n` to every other label.

```
# bar chart with staggered labels
lbls <- paste0(c("", "\n"),
               levels(Marriage$officialTitle))
ggplot(Marriage,
       aes(x=factor(officialTitle,
                    labels = lbls))) +
  geom_bar() +
  labs(x = "",
       y = "Frequency",
       title = "Marriages by officiate")
```



Pie chart

Pie charts are controversial in statistics. If your goal is to compare the frequency of categories, you are better off with bar charts (humans are better at judging the length of bars than the volume of pie slices). If your goal is compare each category with the the whole (e.g., what portion of participants are Hispanic compared to all participants), and the number of categories is small, then pie charts may work for you. It takes a bit more code to make an attractive pie chart in R.

```
# create a basic ggplot2 pie chart
 plotdata <- Marriage %>%
    count(race) %>%
    arrange(desc(race)) %>%
    mutate(prop = round(n * 100 / sum(n), 1),
           lab.ypos = cumsum(prop) - 0.5  *prop)

  ggplot(plotdata,
         aes(x = "",
             y = prop,
             fill = race)) +
    geom_bar(width = 1,
             stat = "identity",
             color = "black") +
    coord_polar("y",
                start = 0,
                direction = -1) +
    theme_void()
```

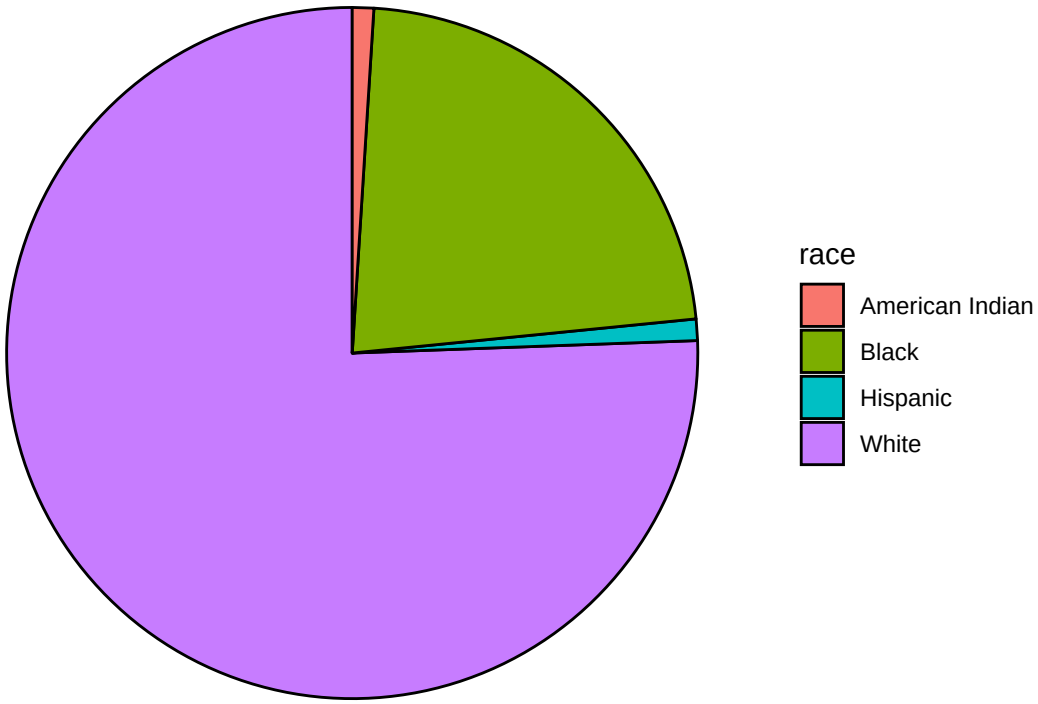


Figure 3.10: Basic pie chart

Now let's get fancy and add labels, while removing the legend.

```
# create a pie chart with slice labels
plotdata <- Marriage %>%
  count(race) %>%
  arrange(desc(race)) %>%
  mutate(prop = round(n*100/sum(n), 1),
         lab.ypos = cumsum(prop) - 0.5*prop)

plotdata$label <- paste0(plotdata$race, "\n",
                         round(plotdata$prop), "%")

ggplot(plotdata,
       aes(x = "",
           y = prop,
           fill = race)) +
  geom_bar(width = 1,
           stat = "identity",
           color = "black") +
  geom_text(aes(y = lab.ypos, label = label),
            color = "black") +
  coord_polar("y",
              start = 0,
              direction = -1) +
  theme_void() +
  theme(legend.position = "FALSE") +
  labs(title = "Participants by race")
```

The pie chart makes it easy to compare each slice with the whole. For example, Back is seen to roughly a quarter of the total participants.

3.1.2 Tree map

An alternative to a pie chart is a tree map. Unlike pie charts, it can handle categorical variables that have *many* levels.

```
library(treemapify)

# create a treemap of marriage officials
plotdata <- Marriage %>%
  count(officialTitle)

ggplot(plotdata,
       aes(fill = officialTitle,
           area = n)) +
  geom_treemap() +
  labs(title = "Marriages by officiate")
```

Here is a more useful version with labels.

```
# create a treemap with tile labels
ggplot(plotdata,
```

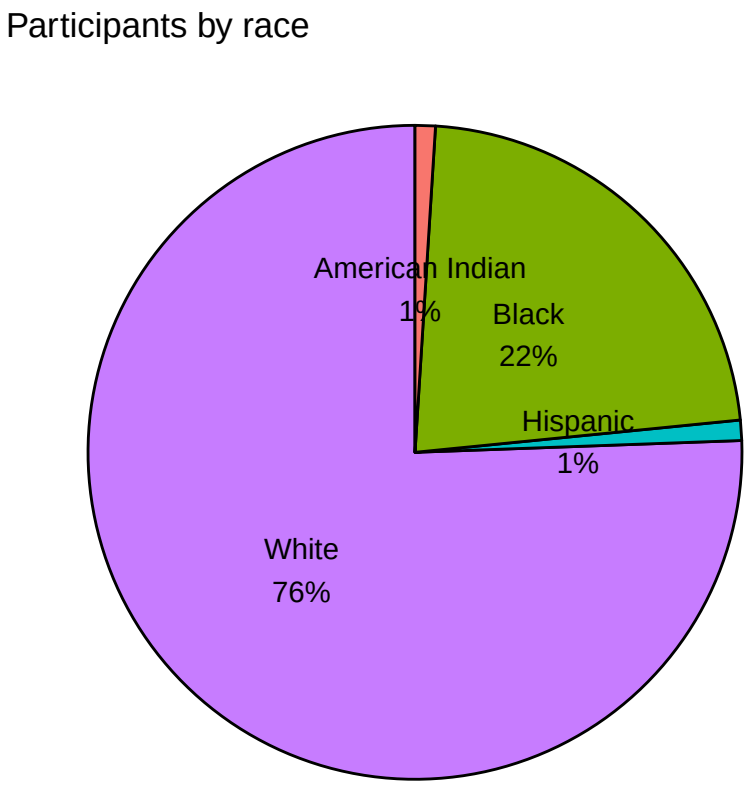


Figure 3.11: Pie chart with percent labels

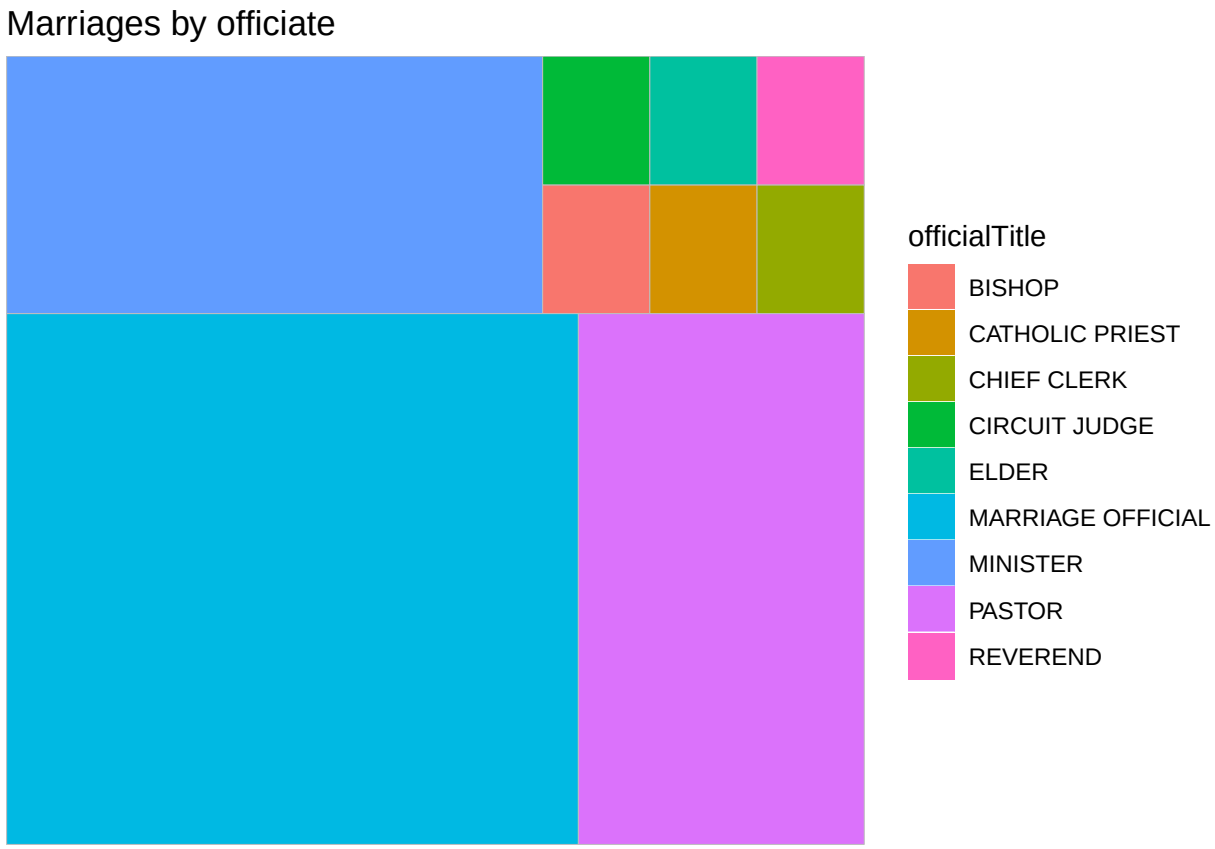


Figure 3.12: Basic treemap

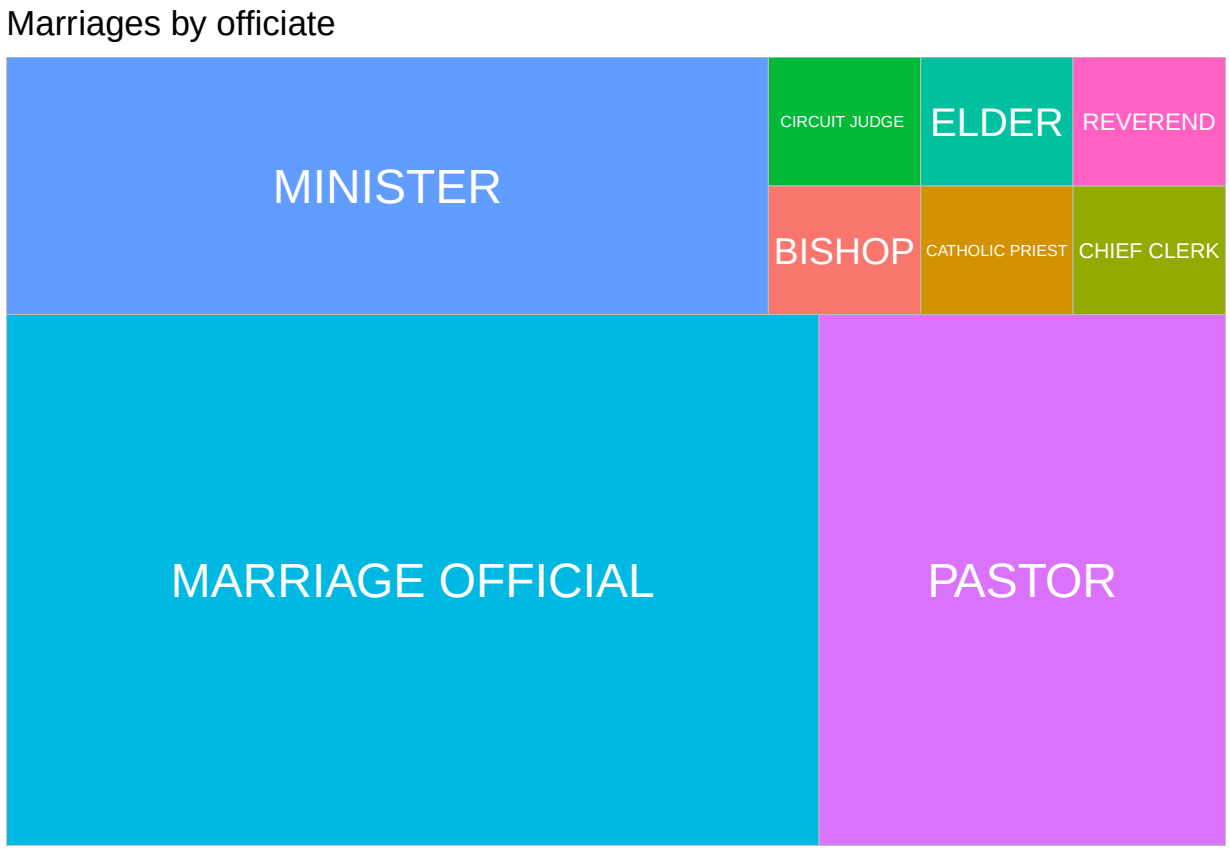


Figure 3.13: Treemap with labels

```
      aes(fill = officialTitle,
          area = n,
          label = officialTitle)) +
geom_treemap() +
geom_treemap_text(colour = "white",
                  place = "centre") +
labs(title = "Marriages by officiate") +
theme(legend.position = "none")
```

3.2 Quantitative

The distribution of a single quantitative variable is typically plotted with a histogram, kernel density plot, or dot plot.

3.2.1 Histogram

Using the Marriage dataset, let's plot the ages of the wedding participants.

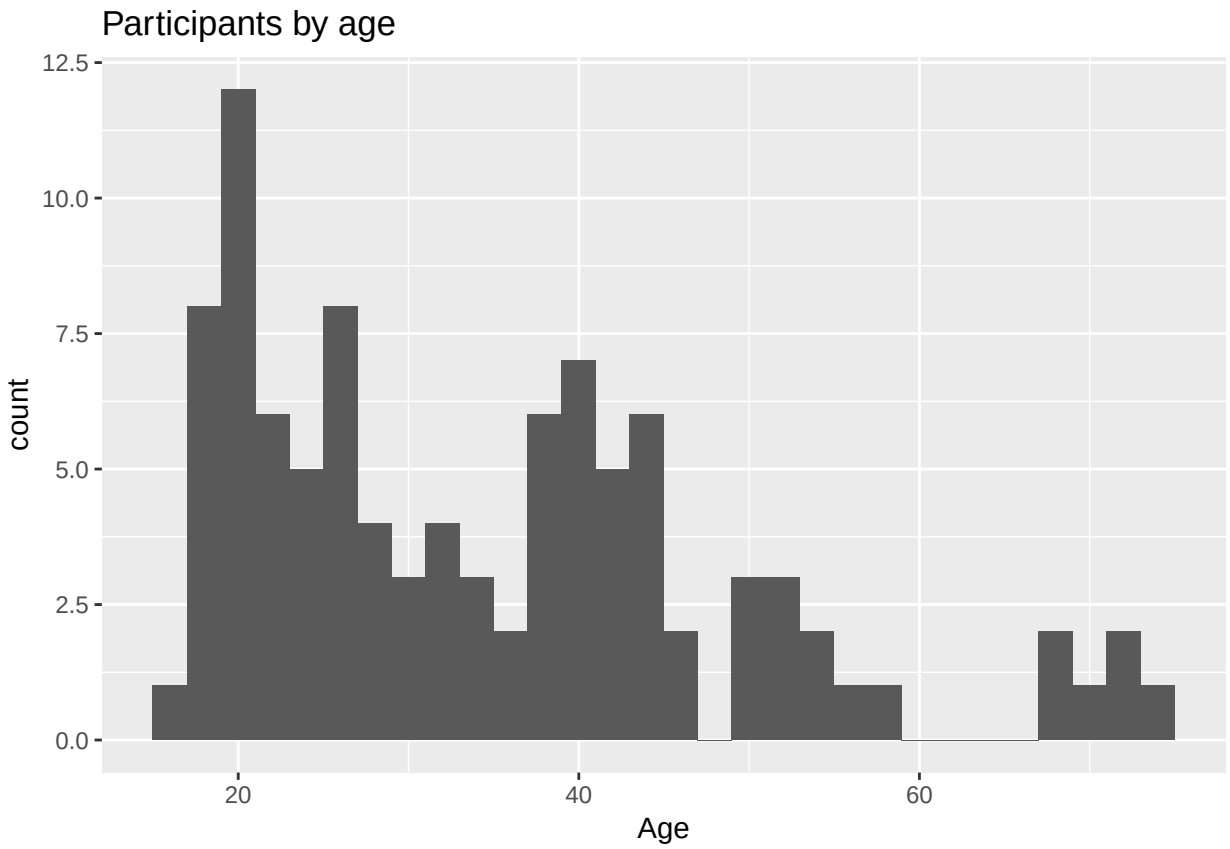


Figure 3.14: Basic histogram

```
library(ggplot2)

# plot the age distribution using a histogram
ggplot(Marriage, aes(x = age)) +
  geom_histogram() +
  labs(title = "Participants by age",
       x = "Age")
```

Most participants appear to be in their early 20's with another group in their 40's, and a much smaller group in their later sixties and early seventies. This would be a *multimodal* distribution.

Histogram colors can be modified using two options

- `fill` - fill color for the bars
- `color` - border color around the bars

```
# plot the histogram with blue bars and white borders
ggplot(Marriage, aes(x = age)) +
  geom_histogram(fill = "cornflowerblue",
                 color = "white") +
  labs(title="Participants by age",
       x = "Age")
```

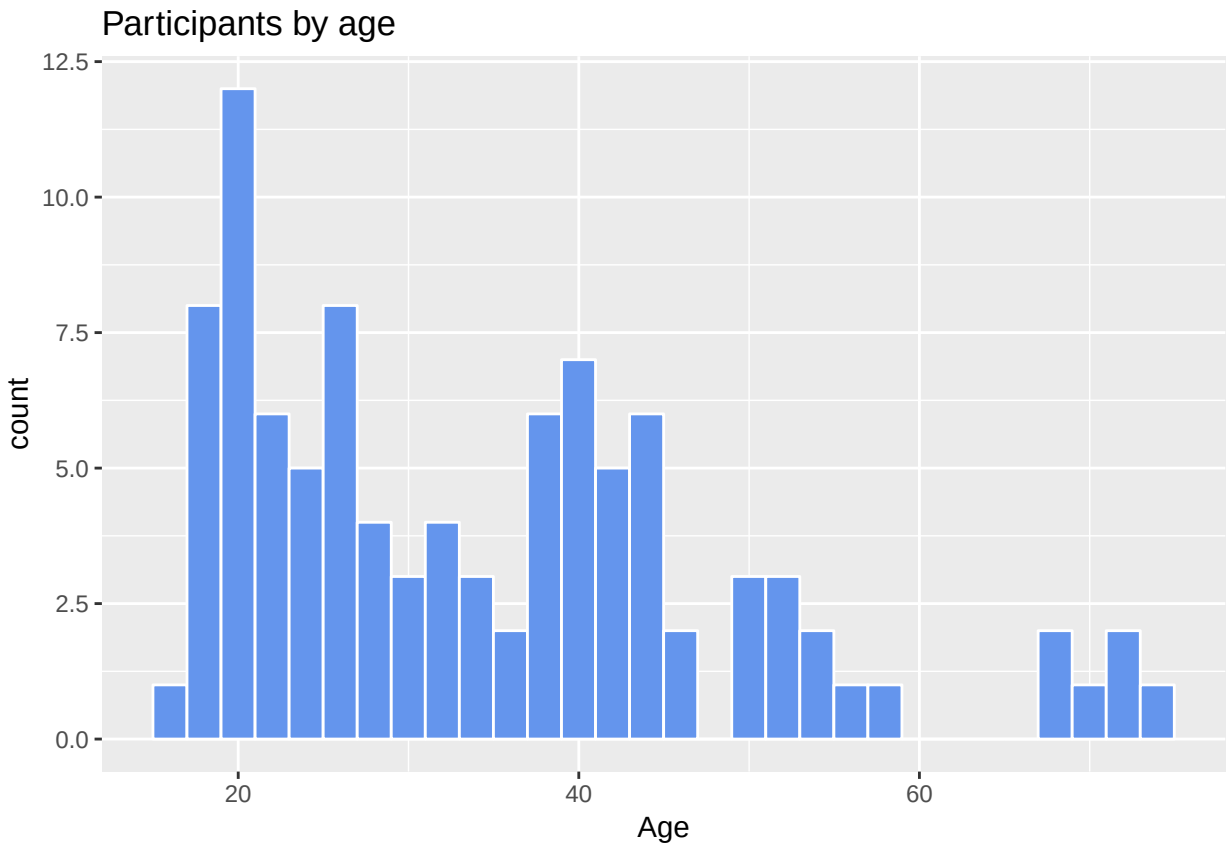


Figure 3.15: Histogram with specified fill and border colors

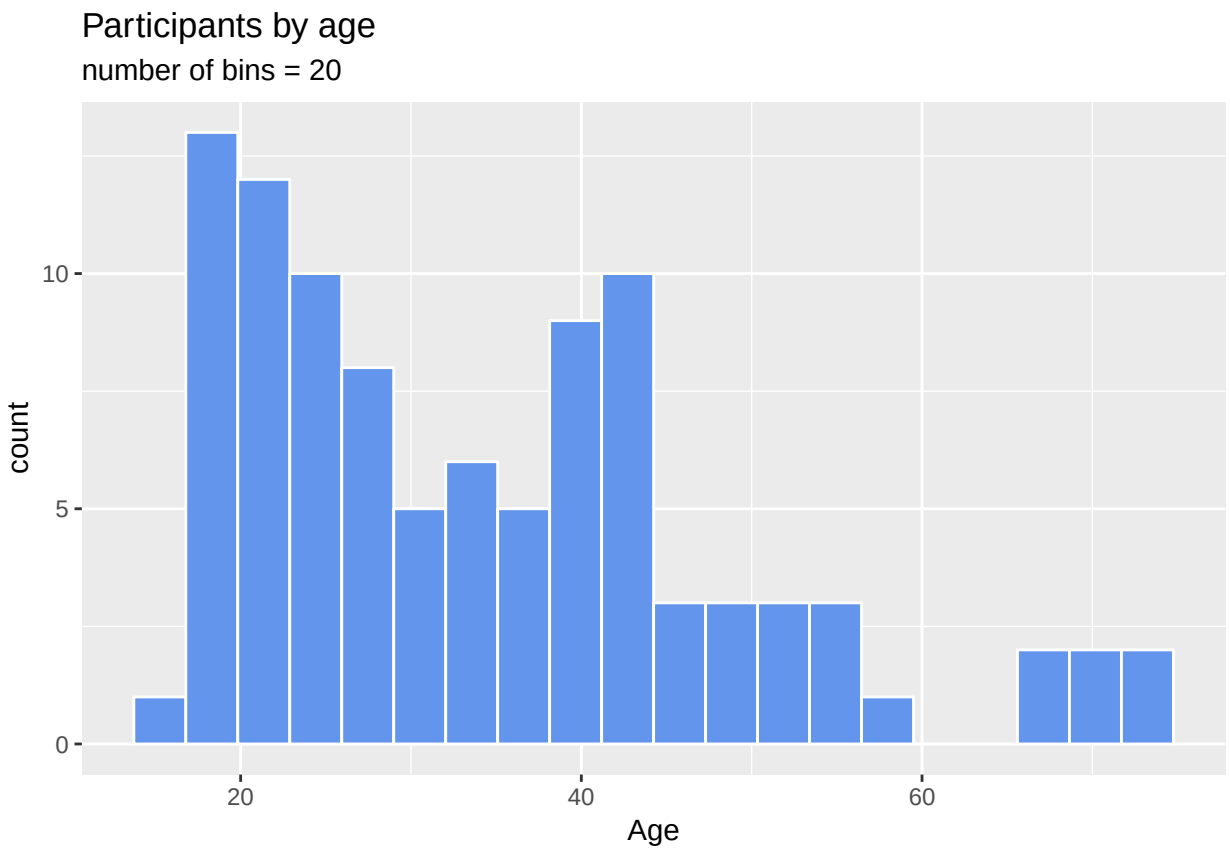


Figure 3.16: Histogram with a specified number of bins

3.2.1.1 Bins and bandwidths

One of the most important histogram options is `bins`, which controls the number of bins into which the numeric variable is divided (i.e., the number of bars in the plot). The default is 30, but it is helpful to try smaller and larger numbers to get a better impression of the shape of the distribution.

```
# plot the histogram with 20 bins
ggplot(Marriage, aes(x = age)) +
  geom_histogram(fill = "cornflowerblue",
                 color = "white",
                 bins = 20) +
  labs(title="Participants by age",
       subtitle = "number of bins = 20",
       x = "Age")
```

Alternatively, you can specify the `binwidth`, the width of the bins represented by the bars.

```
# plot the histogram with a binwidth of 5
ggplot(Marriage, aes(x = age)) +
  geom_histogram(fill = "cornflowerblue",
                 color = "white",
                 binwidth = 5) +
  labs(title="Participants by age",
```

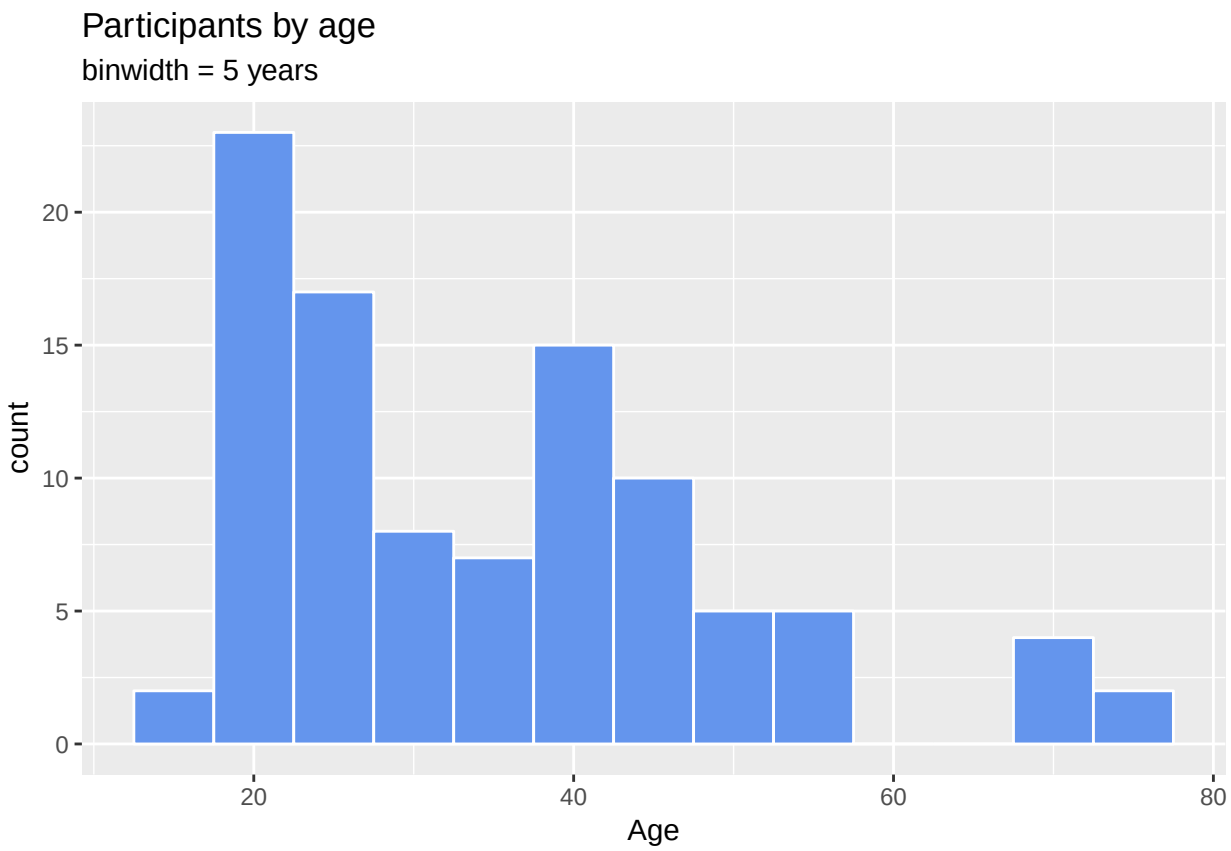
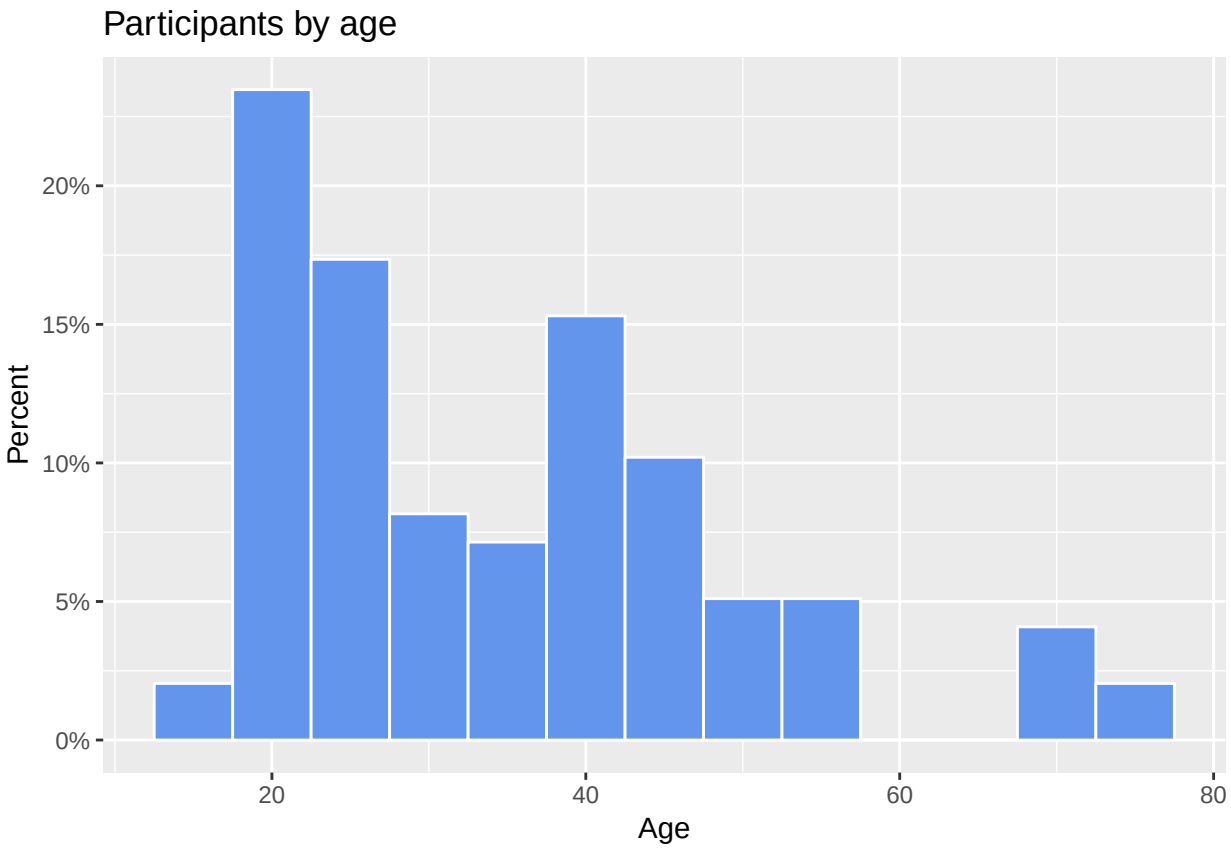


Figure 3.17: Histogram with specified a bin width

```
       subtitle = "binwidth = 5 years",
       x = "Age")
```

As with bar charts, the y-axis can represent counts or percent of the total.

```
# plot the histogram with percentages on the y-axis
library(scales)
ggplot(Marriage,
       aes(x = age,
           y= ..count.. / sum(..count..))) +
  geom_histogram(fill = "cornflowerblue",
                 color = "white",
                 binwidth = 5) +
  labs(title="Participants by age",
       y = "Percent",
       x = "Age") +
  scale_y_continuous(labels = percent)
```



Kernel Density plot {#Kernel}

An alternative to a histogram is the kernel density plot. Technically, kernel density estimation is a nonparametric method for estimating the probability density function of a continuous random variable. (What??) Basically, we are trying to draw a smoothed histogram, where the area under the curve equals one.

```
# Create a kernel density plot of age
ggplot(Marriage, aes(x = age)) +
  geom_density() +
  labs(title = "Participants by age")
```

The graph shows the distribution of scores. For example, the proportion of cases between 20 and 40 years old would be represented by the area under the curve between 20 and 40 on the x-axis.

As with previous charts, we can use `fill` and `color` to specify the fill and border colors.

```
# Create a kernel density plot of age
ggplot(Marriage, aes(x = age)) +
  geom_density(fill = "indianred3") +
  labs(title = "Participants by age")
```

3.2.1.2 Smoothing parameter

The degree of smoothness is controlled by the bandwidth parameter `bw`. To find the default value for a particular variable, use the `bw.nrd0` function. Values that are larger will result in more smoothing, while values that are smaller will produce less smoothing.

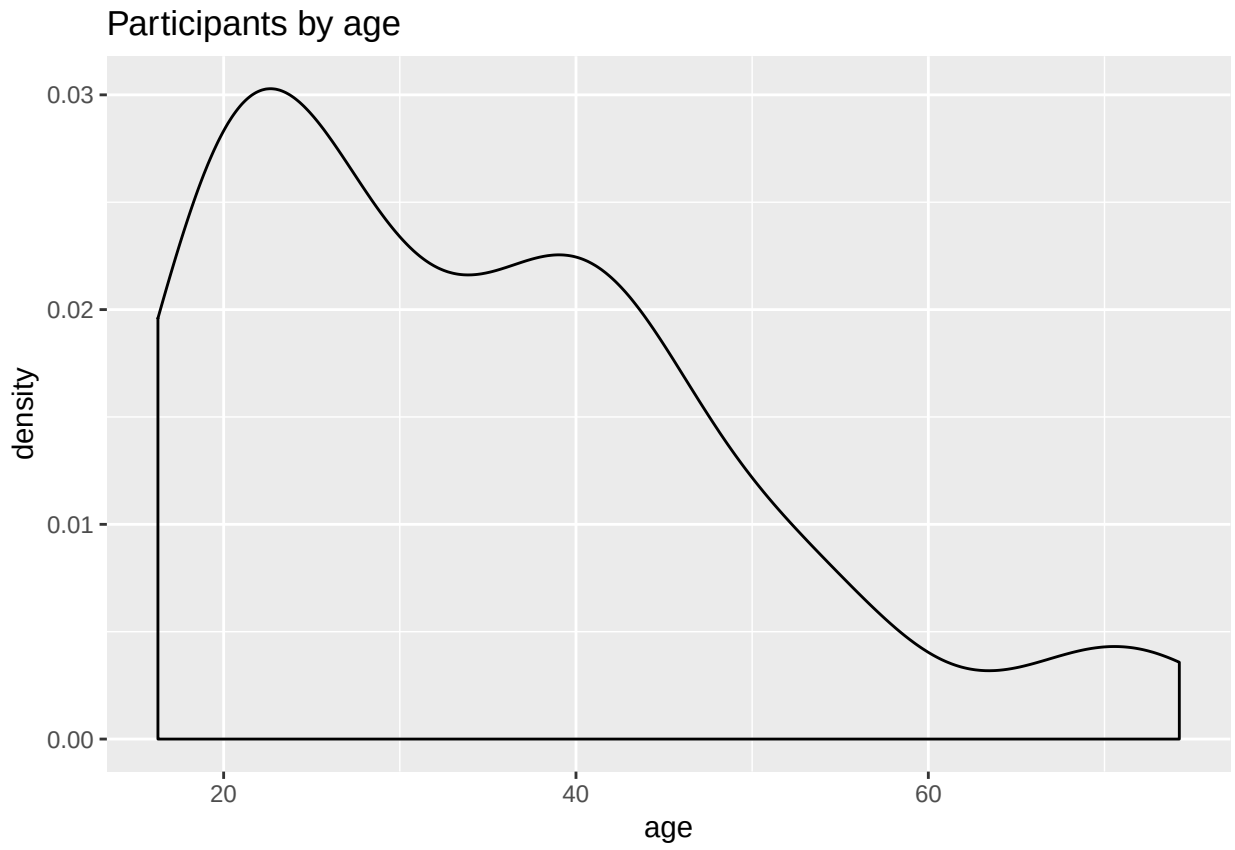


Figure 3.18: Basic kernel density plot

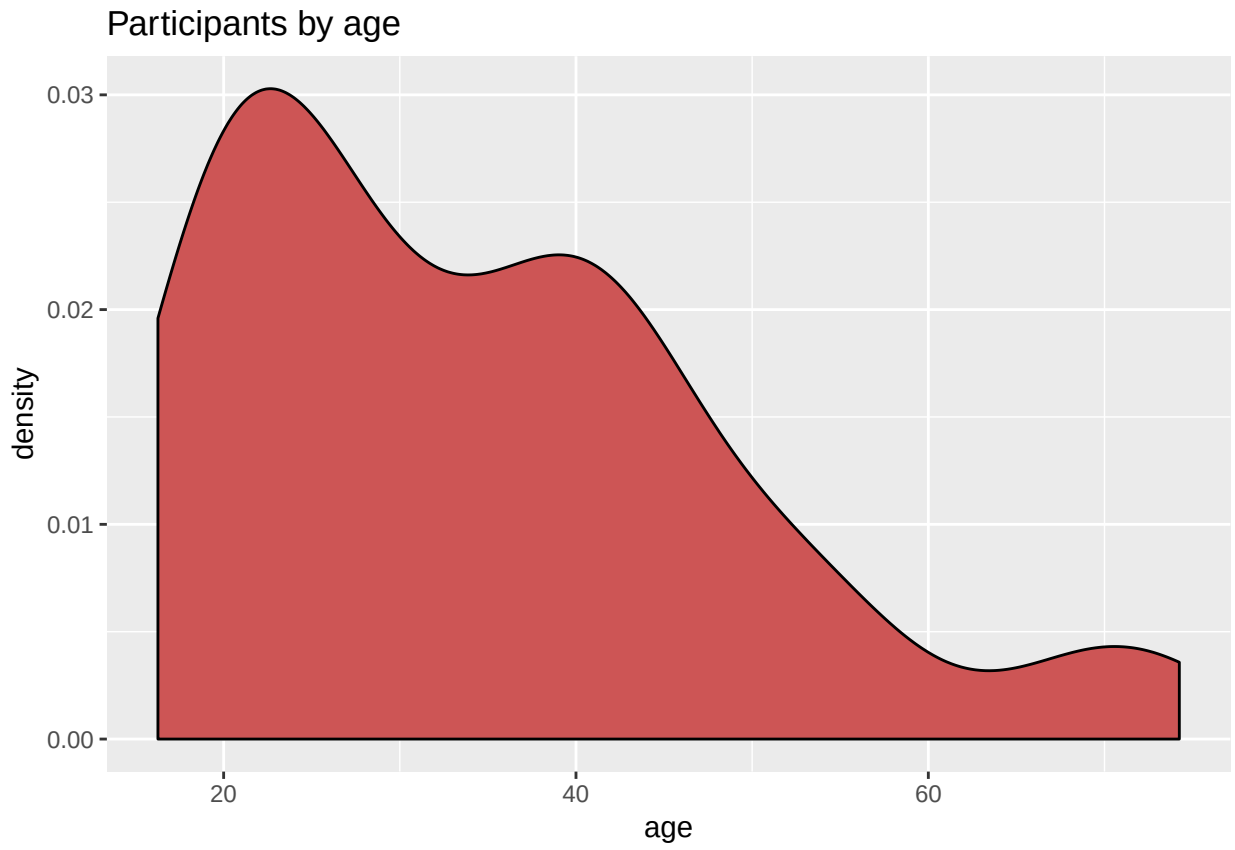


Figure 3.19: Kernel density plot with fill

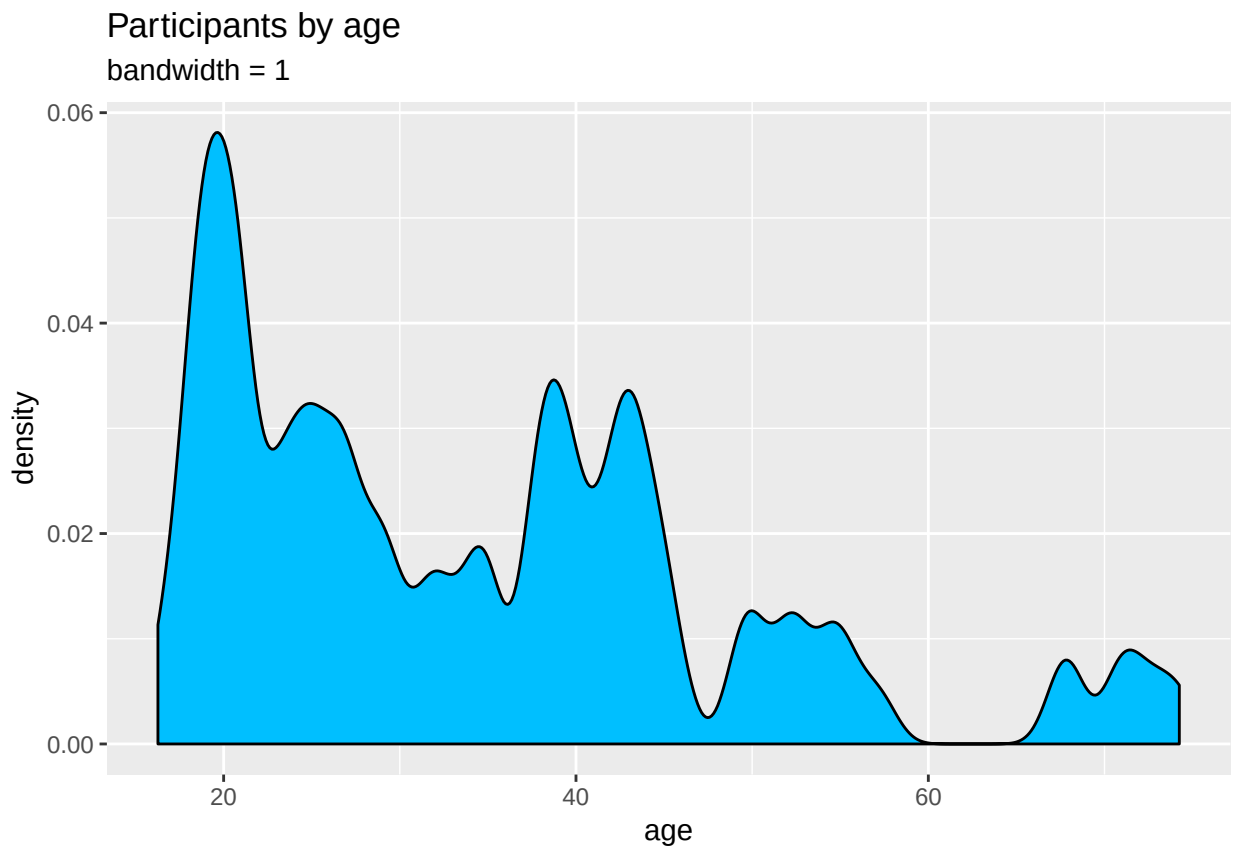


Figure 3.20: Kernel density plot with a specified bandwidth

```
# default bandwidth for the age variable
bw.nrd0(Marriage$age)
```

```
## [1] 5.181946
```

```
# Create a kernel density plot of age
ggplot(Marriage, aes(x = age)) +
  geom_density(fill = "deepskyblue",
               bw = 1) +
  labs(title = "Participants by age",
       subtitle = "bandwidth = 1")
```

In this example, the default bandwidth for age is 5.18. Choosing a value of 1 resulted in less smoothing and more detail.

Kernel density plots allow you to easily see which scores are most frequent and which are relatively rare. However it can be difficult to explain the meaning of the y-axis to a non-statistician. (But it will make you look really smart at parties!)

3.2.2 Dot Chart

Another alternative to the histogram is the dot chart. Again, the quantitative variable is divided into bins, but rather than summary bars, each observation is represented by a dot. By default, the width of a dot

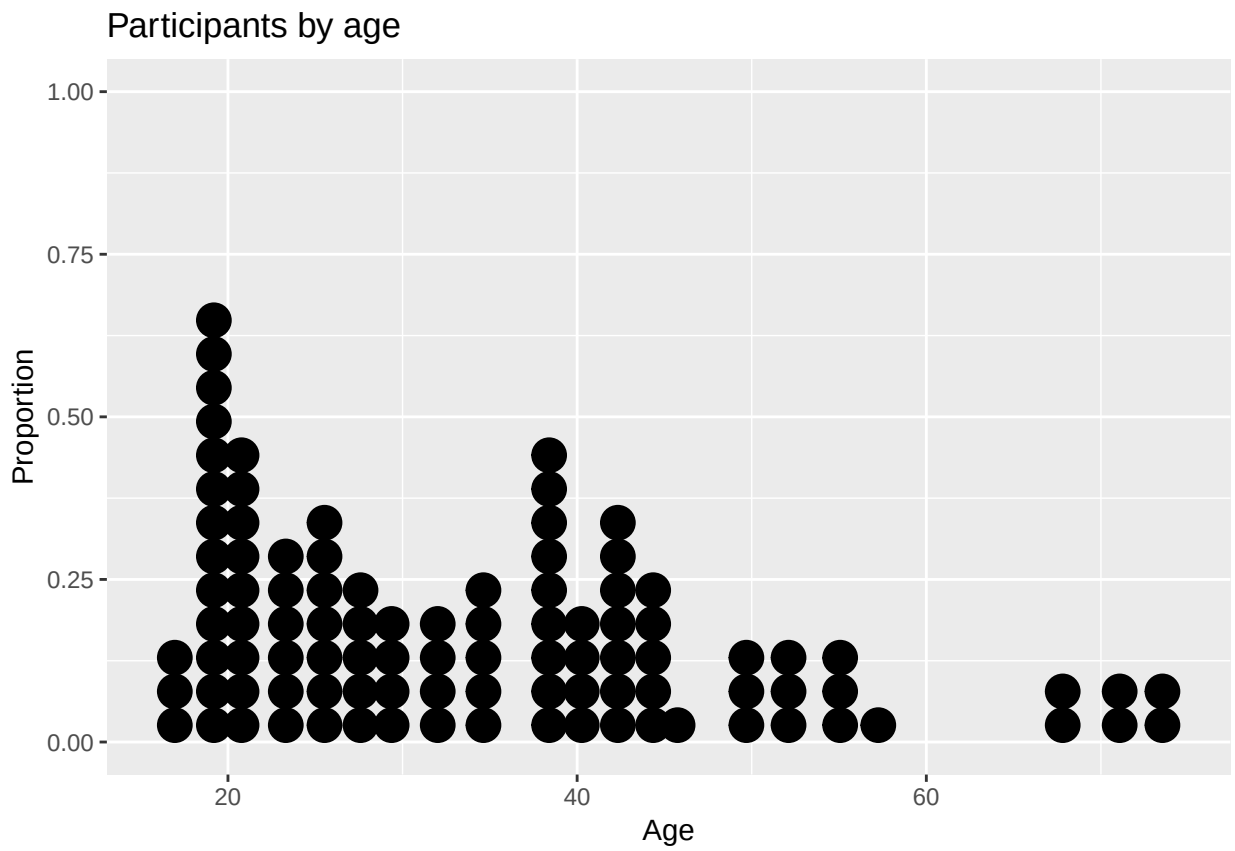


Figure 3.21: Basic dotplot

corresponds to the bin width, and dots are stacked, with each dot representing one observation. This works best when the number of observations is small (say, less than 150).

```
# plot the age distribution using a dotplot
ggplot(Marriage, aes(x = age)) +
  geom_dotplot() +
  labs(title = "Participants by age",
       y = "Proportion",
       x = "Age")
```

The `fill` and `color` options can be used to specify the fill and border color of each dot respectively.

```
# Plot ages as a dot plot using
# gold dots with black borders
ggplot(Marriage, aes(x = age)) +
  geom_dotplot(fill = "gold",
               color = "black") +
  labs(title = "Participants by age",
       y = "Proportion",
       x = "Age")
```

There are many more options available. See the help for details and examples.

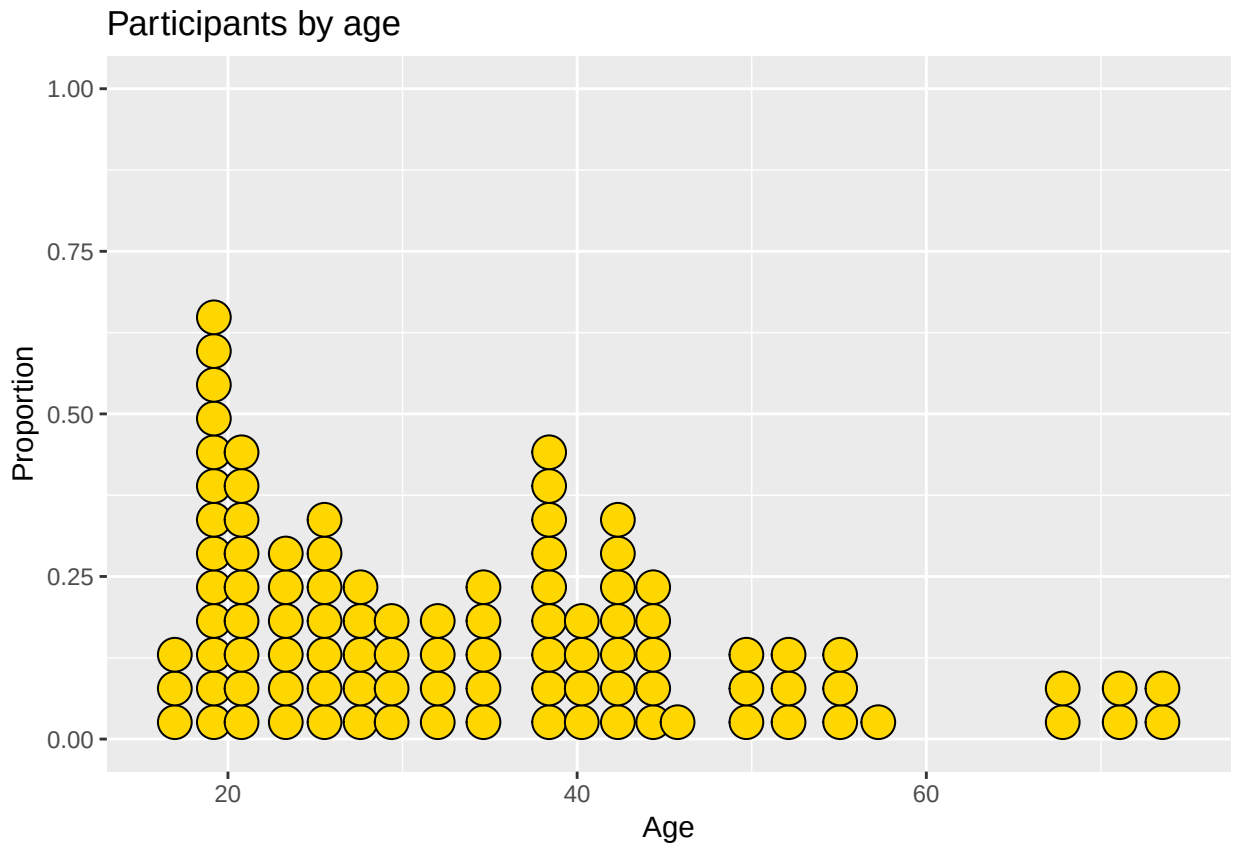


Figure 3.22: Dotplot with a specified color scheme

Chapter 4

Bivariate Graphs

Bivariate graphs display the relationship between two variables. The type of graph will depend on the measurement level of the variables (categorical or quantitative).

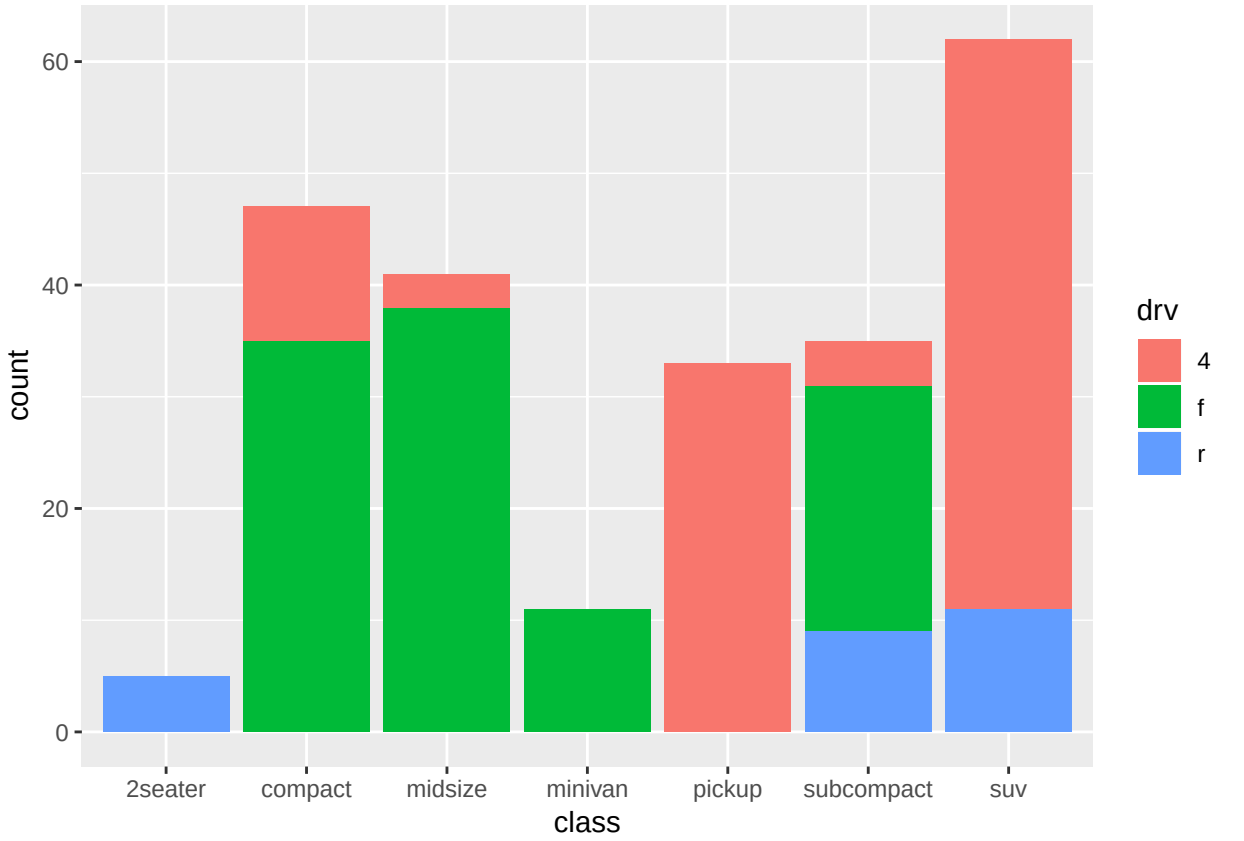
4.1 Categorical vs. Categorical

When plotting the relationship between two categorical variables, stacked, grouped, or segmented bar charts are typically used. A less common approach is the mosaic chart.

4.1.1 Stacked bar chart

Let's plot the relationship between automobile class and drive type (front-wheel, rear-wheel, or 4-wheel drive) for the automobiles in the Fuel economy dataset.

```
library(ggplot2)

# stacked bar chart
ggplot(mpg,
       aes(x = class,
           fill = drv)) +
  geom_bar(position = "stack")
```



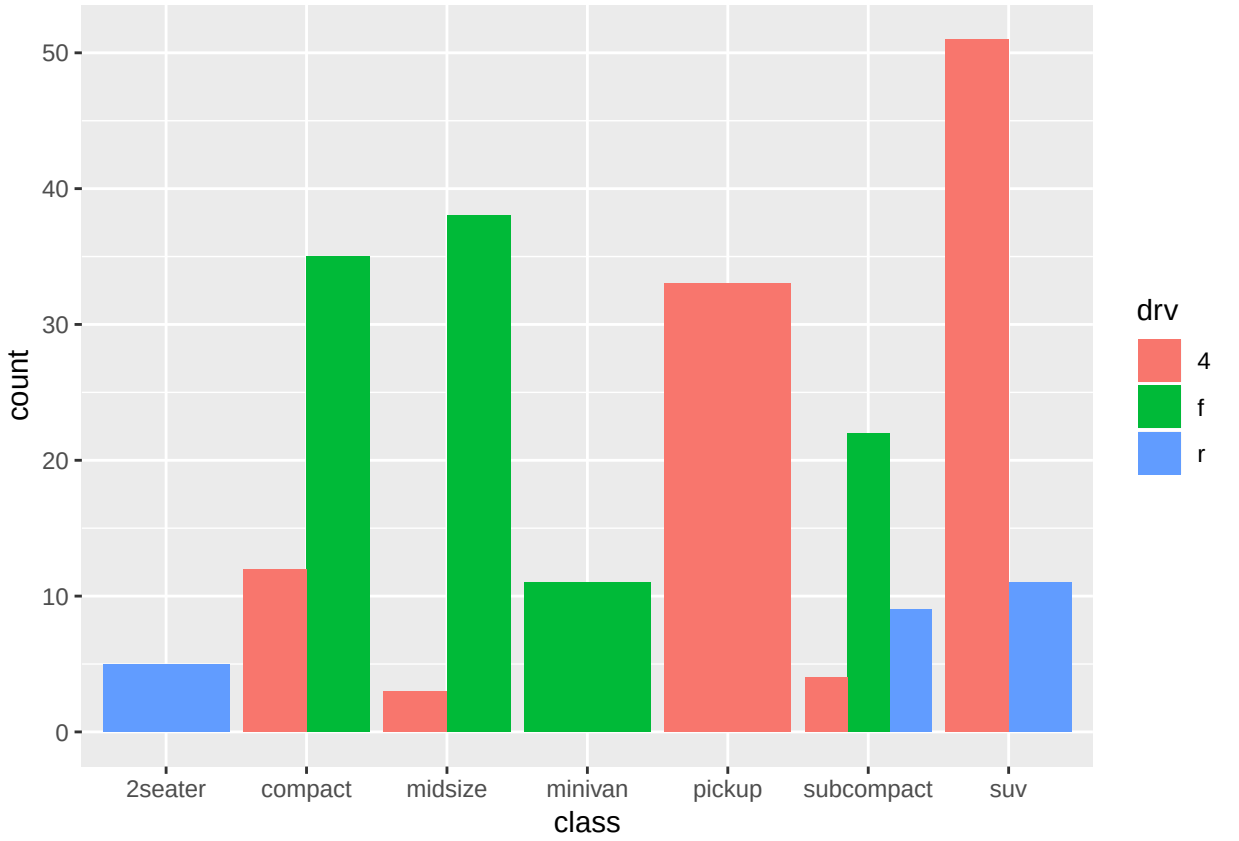
From the chart, we can see for example, that the most common vehicle is the SUV. All 2seater cars are rear wheel drive, while most, but not all SUVs are 4-wheel drive.

Stacked is the default, so the last line could have also been written as `geom_bar()`.

4.1.2 Grouped bar chart

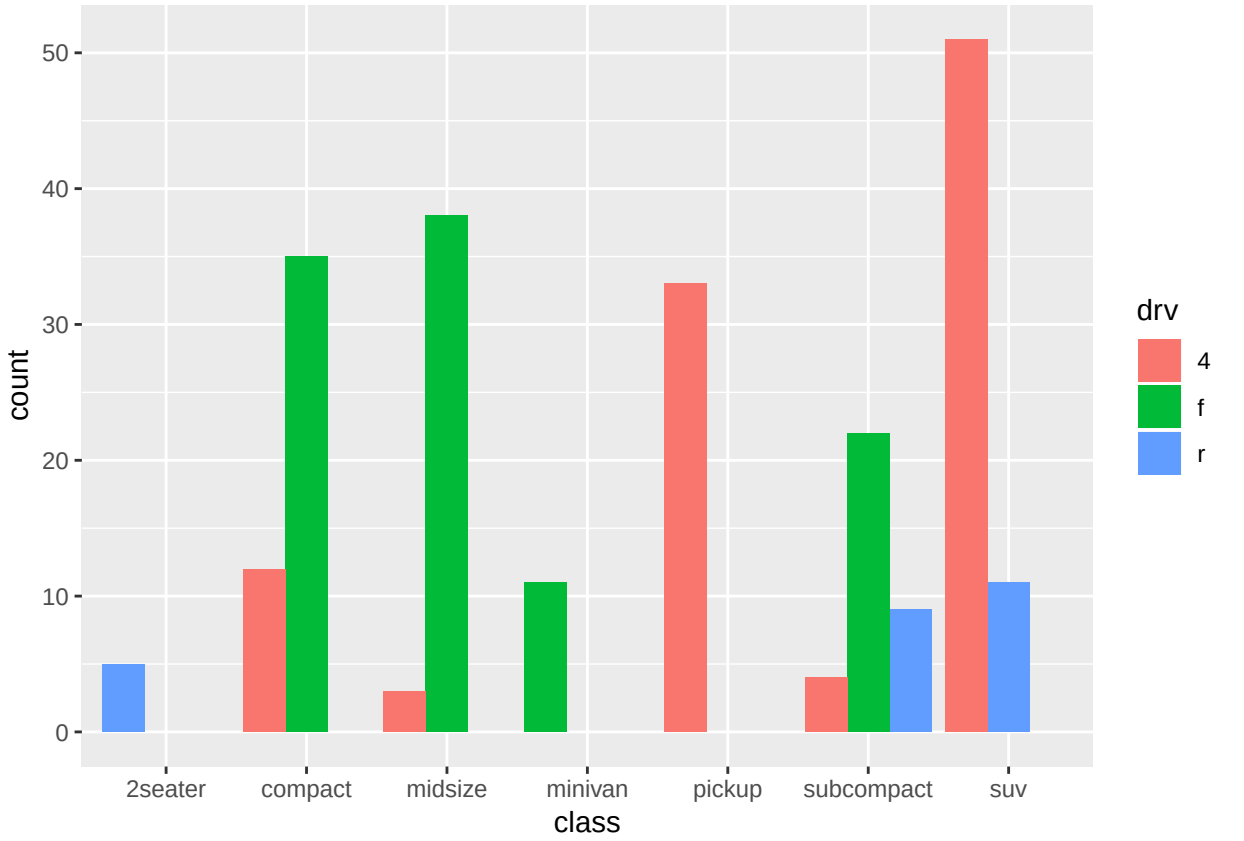
Grouped bar charts place bars for the second categorical variable side-by-side. To create a grouped bar plot use the `position = "dodge"` option.

```
library(ggplot2)

# grouped bar plot
ggplot(mpg,
       aes(x = class,
           fill = drv)) +
  geom_bar(position = "dodge")
```



Notice that all Minivans are front-wheel drive. By default, zero count bars are dropped and the remaining bars are made wider. This may not be the behavior you want. You can modify this using the `position = position_dodge(preserve = "single")` option.

```
library(ggplot2)

# grouped bar plot preserving zero count bars
ggplot(mpg,
       aes(x = class,
           fill = drv)) +
  geom_bar(position = position_dodge(preserve = "single"))
```



Note that this option is only available in the latest development version of `ggplot2`, but should should be generally available shortly.

4.1.3 Segmented bar chart

A segmented bar plot is a stacked bar plot where each bar represents 100 percent. You can create a segmented bar chart using the `position = "filled"` option.

```
library(ggplot2)

# bar plot, with each bar representing 100%
ggplot(mpg,
       aes(x = class,
           fill = drv)) +
  geom_bar(position = "fill") +
  labs(y = "Proportion")
```

This type of plot is particularly useful if the goal is to compare the percentage of a category in one variable across each level of another variable. For example, the proportion of front-wheel drive cars go up as you move from compact, to midsize, to minivan.

4.1.4 Improving the color and labeling

You can use additional options to improve color and labeling. In the graph below

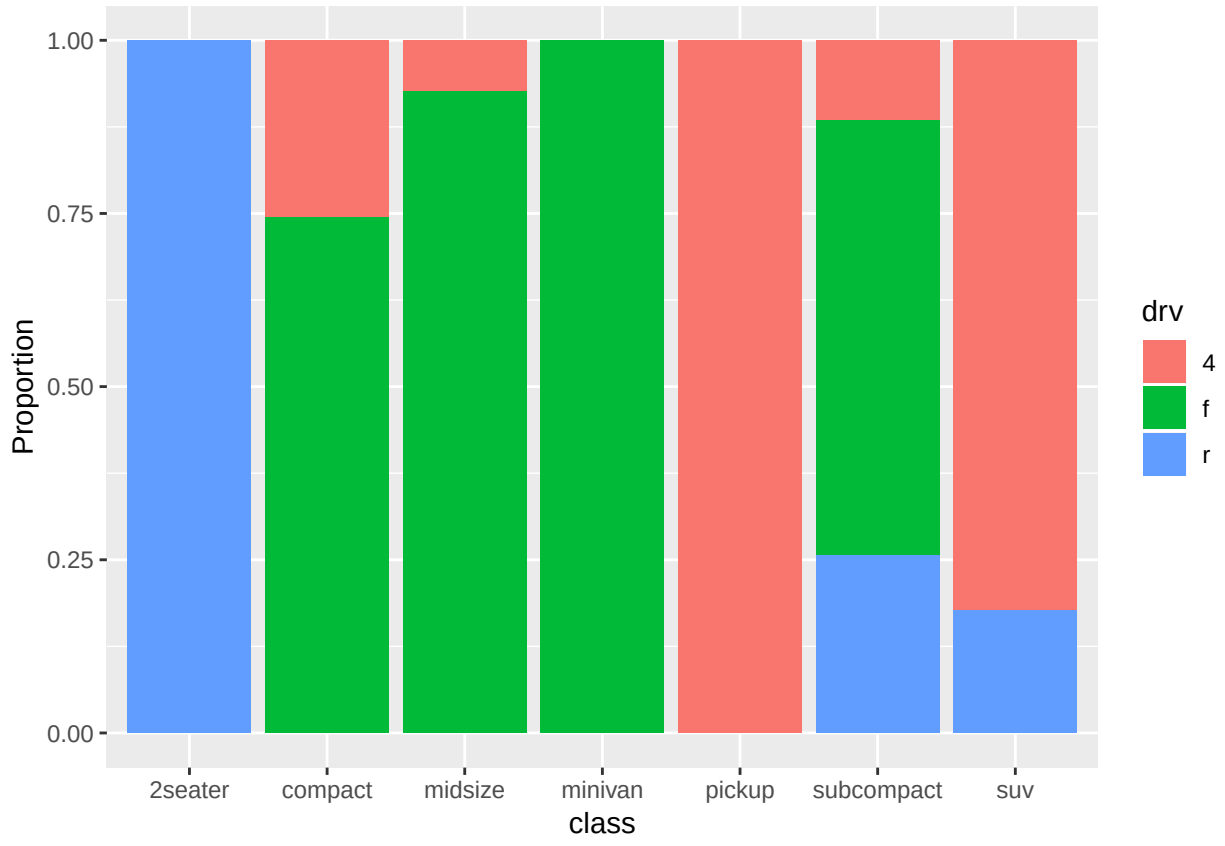
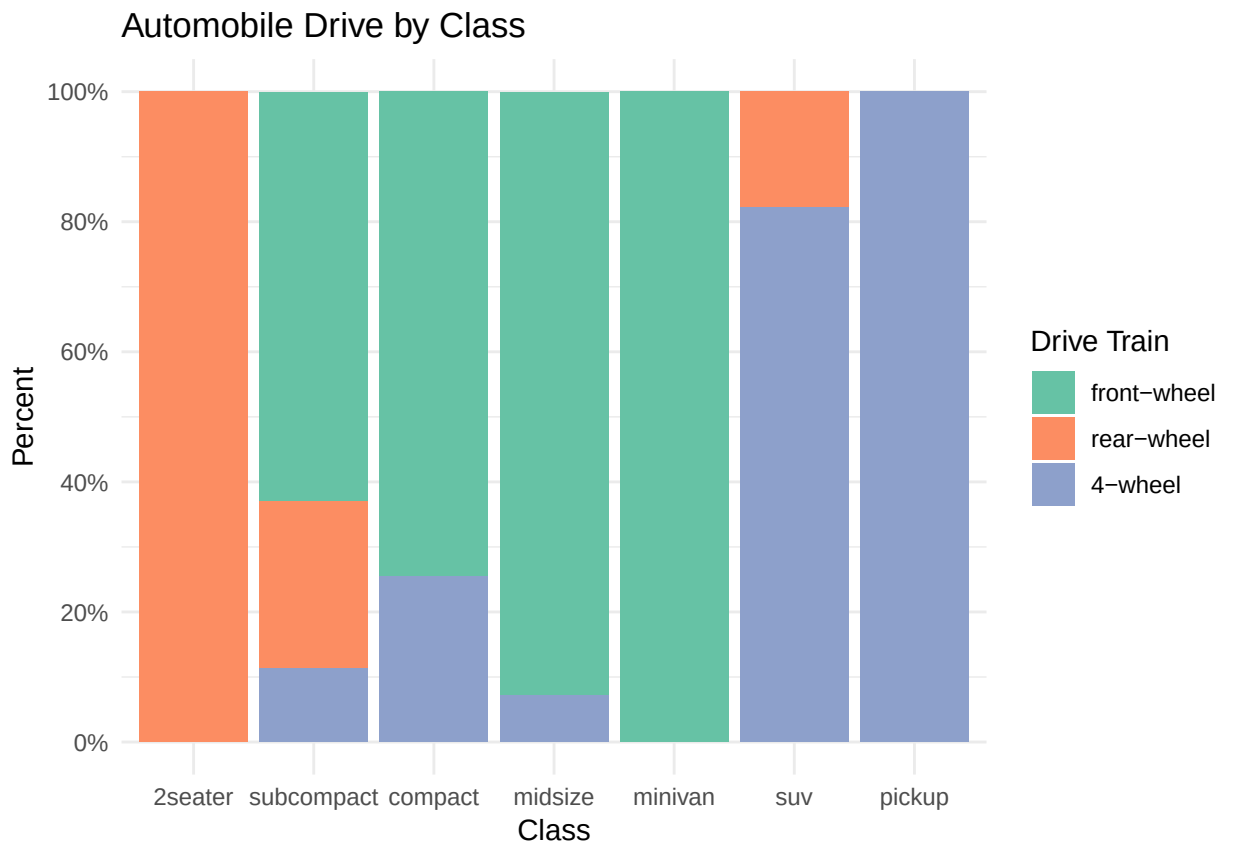


Figure 4.1: Segmented bar chart

- `factor` modifies the order of the categories for the class variable and both the order and the labels for the drive variable
- `scale_y_continuous` modifies the y-axis tick mark labels
- `labs` provides a title and changed the labels for the x and y axes and the legend
- `scale_fill_brewer` changes the fill color scheme
- `theme_minimal` removes the grey background and changed the grid color

```
library(ggplot2)

# bar plot, with each bar representing 100%,
# reordered bars, and better labels and colors
library(scales)
ggplot(mpg,
       aes(x = factor(class,
                      levels = c("2seater", "subcompact",
                                 "compact", "midsize",
                                 "minivan", "suv", "pickup")),
           fill = factor(drv,
                         levels = c("f", "r", "4"),
                         labels = c("front-wheel",
                                    "rear-wheel",
                                    "4-wheel")))) +
  geom_bar(position = "fill") +
  scale_y_continuous(breaks = seq(0, 1, .2),
                     label = percent) +
  scale_fill_brewer(palette = "Set2") +
  labs(y = "Percent",
       fill = "Drive Train",
       x = "Class",
       title = "Automobile Drive by Class") +
  theme_minimal()
```



In the graph above, the `factor` function was used to reorder and/or rename the levels of a categorical variable. You could also apply this to the original dataset, making these changes permanent. It would then apply to all future graphs using that dataset. For example:

```
# change the order the levels for the categorical variable "class"
mpg$class = factor(mpg$class,
                   levels = c("2seater", "subcompact",
                              "compact", "midsize",
                              "minivan", "suv", "pickup")
```

I placed the `factor` function within the `ggplot` function to demonstrate that, if desired, you can change the order of the categories and labels for the categories for a single graph.

The other functions are discussed more fully in the section on Customizing graphs.

Next, let's add percent labels to each segment. First, we'll create a summary dataset that has the necessary labels.

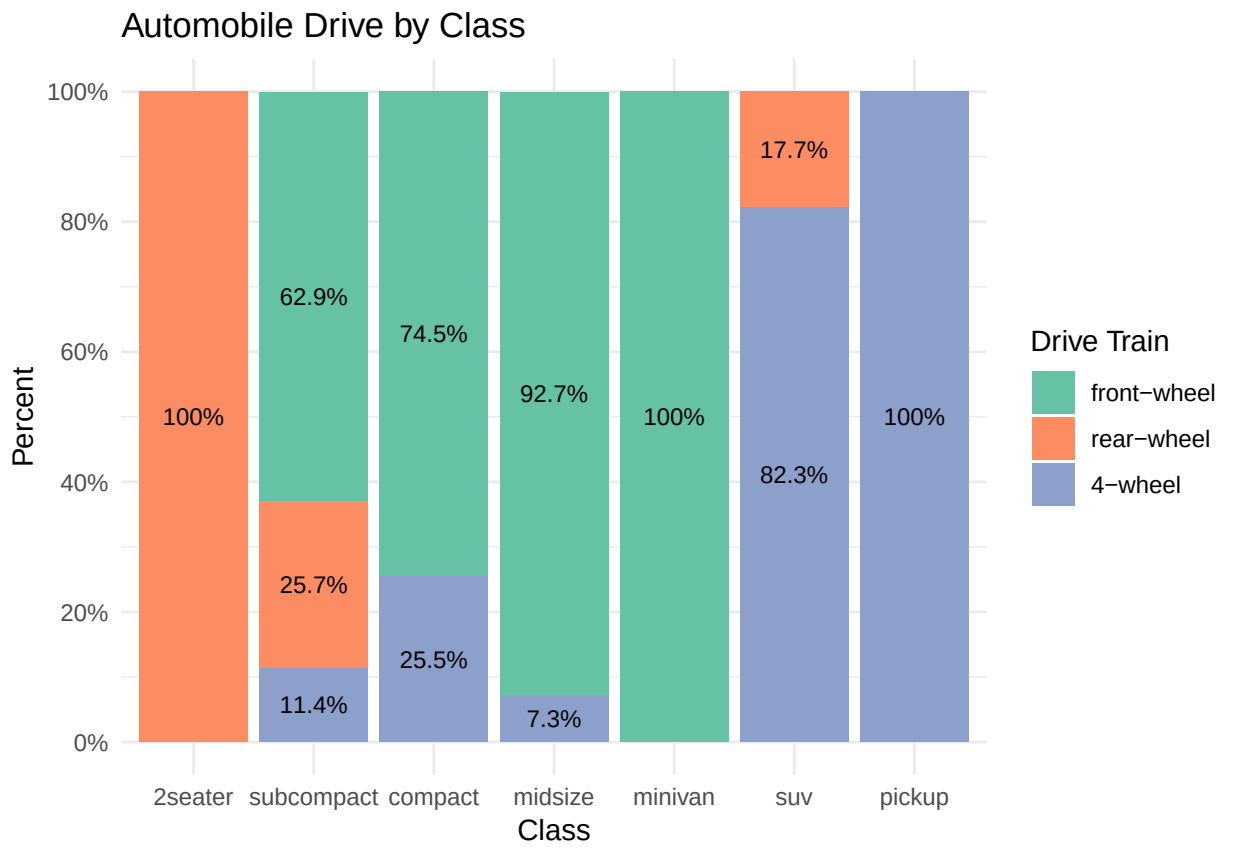
```
# create a summary dataset
library(dplyr)
plotdata <- mpg %>%
  group_by(class, drv) %>%
  summarize(n = n()) %>%
  mutate(pct = n/sum(n),
         lbl = scales::percent(pct))
plotdata
```

```
## # A tibble: 12 x 5
```

```
## # Groups:   class [7]
##    class      drv       n    pct lbl
##    <chr>      <chr> <int>  <dbl> <chr>
##  1 2seater    r         5 1.00   100%
##  2 compact    4        12 0.255  25.5%
##  3 compact    f        35 0.745  74.5%
##  4 midsize    4         3 0.0732 7.3%
##  5 midsize    f        38 0.927  92.7%
##  6 minivan    f        11 1.00   100%
##  7 pickup     4        33 1.00   100%
##  8 subcompact 4         4 0.114  11.4%
##  9 subcompact f        22 0.629  62.9%
## 10 subcompact r         9 0.257  25.7%
## 11 suv        4        51 0.823  82.3%
## 12 suv        r        11 0.177  17.7%
```

Next, we'll use this dataset and the `geom_text` function to add labels to each bar segment.

```
# create segmented bar chart
# adding labels to each segment

ggplot(plotdata,
       aes(x = factor(class,
                      levels = c("2seater", "subcompact",
                                 "compact", "midsize",
                                 "minivan", "suv", "pickup")),
           y = pct,
           fill = factor(drv,
                         levels = c("f", "r", "4"),
                         labels = c("front-wheel",
                                    "rear-wheel",
                                    "4-wheel")))) +
  geom_bar(stat = "identity",
           position = "fill") +
  scale_y_continuous(breaks = seq(0, 1, .2),
                     label = percent) +
  geom_text(aes(label = lbl),
            size = 3,
            position = position_stack(vjust = 0.5)) +
  scale_fill_brewer(palette = "Set2") +
  labs(y = "Percent",
       fill = "Drive Train",
       x = "Class",
       title = "Automobile Drive by Class") +
  theme_minimal()
```



Now we have a graph that is easy to read and interpret.

4.1.5 Other plots

Mosaic plots provide an alternative to stacked bar charts for displaying the relationship between categorical variables. They can also provide more sophisticated statistical information.

4.2 Quantitative vs. Quantitative

The relationship between two quantitative variables is typically displayed using scatterplots and line graphs.

4.2.1 Scatterplot

The simplest display of two quantitative variables is a scatterplot, with each variable represented on an axis. For example, using the Salaries dataset, we can plot experience (*yrs.since.phd*) vs. academic salary (*salary*) for college professors.

```
library(ggplot2)
data(Salaries, package="carData")

# simple scatterplot
ggplot(Salaries,
       aes(x = yrs.since.phd,
```

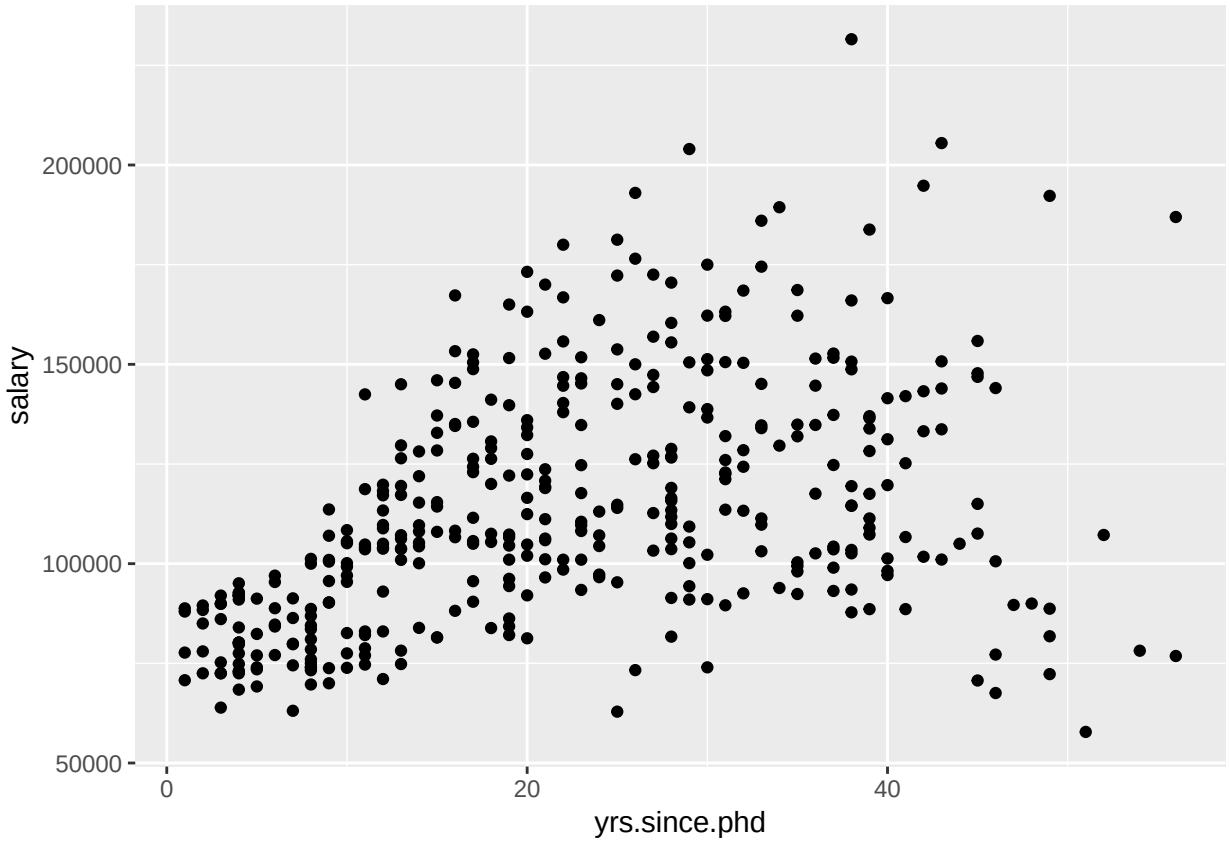


Figure 4.2: Simple scatterplot

```
          y = salary)) +
  geom_point()
```

`geom_point` options can be used to change the

- `color` - point color
- `size` - point size
- `shape` - point shape
- `alpha` - point transparency. Transparency ranges from 0 (transparent) to 1 (opaque), and is a useful parameter when points overlap.

The functions `scale_x_continuous` and `scale_y_continuous` control the scaling on x and y axes respectively.

See Customizing graphs for details.

We can use these options and functions to create a more attractive scatterplot.

```
# enhanced scatter plot
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary)) +
  geom_point(color="cornflowerblue",
```

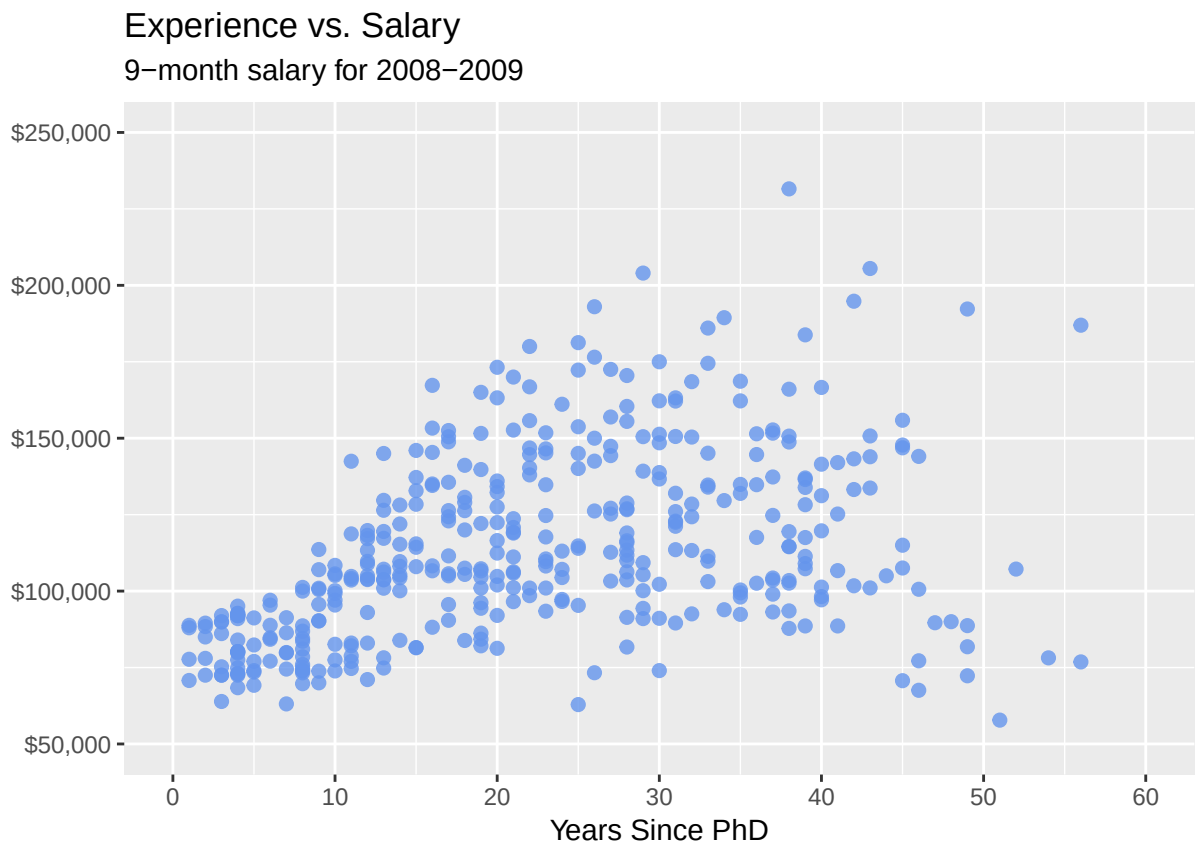


Figure 4.3: Scatterplot with color, transparency, and axis scaling

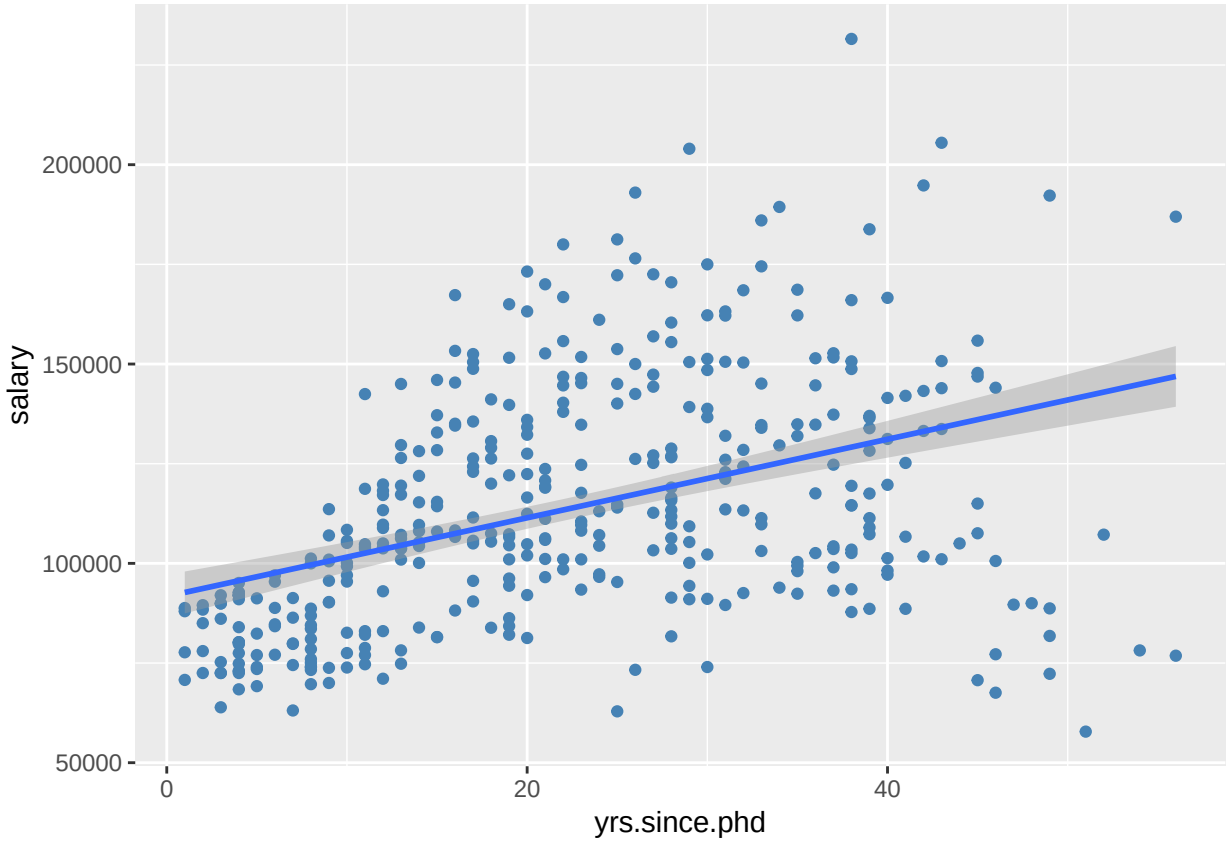
```
            size = 2,
            alpha=.8) +
  scale_y_continuous(label = scales::dollar,
                     limits = c(50000, 250000)) +
  scale_x_continuous(breaks = seq(0, 60, 10),
                     limits=c(0, 60)) +
  labs(x = "Years Since PhD",
       y = "",
       title = "Experience vs. Salary",
       subtitle = "9-month salary for 2008-2009")
```

4.2.1.1 Adding best fit lines

It is often useful to summarize the relationship displayed in the scatterplot, using a best fit line. Many types of lines are supported, including linear, polynomial, and nonparametric (loess). By default, 95% confidence limits for these lines are displayed.

```
# scatterplot with linear fit line
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary)) +
```

```
  geom_point(color= "steelblue") +
  geom_smooth(method = "lm")
```



Clearly, salary increases with experience. However, there seems to be a dip at the right end - professors with significant experience, earning lower salaries. A straight line does not capture this non-linear effect. A line with a bend will fit better here.

A polynomial regression line provides a fit line of the form

$$\hat{y} = \beta_0 + \beta_1 x + \beta 2 x^2 + \beta 3 x^3 + \beta 4 x^4 + \ldots$$

Typically either a quadratic (one bend), or cubic (two bends) line is used. It is rarely necessary to use a higher order(>3) polynomials. Applying a quadratic fit to the salary dataset produces the following result.

```
# scatterplot with quadratic line of best fit
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary)) +
  geom_point(color= "steelblue") +
  geom_smooth(method = "lm",
              formula = y ~ poly(x, 2),
              color = "indianred3")
```

Finally, a smoothed nonparametric fit line can often provide a good picture of the relationship. The default in `ggplot2` is a loess line which stands for for **l**ocally **w**eighted **s**catterplot **s**moothing.

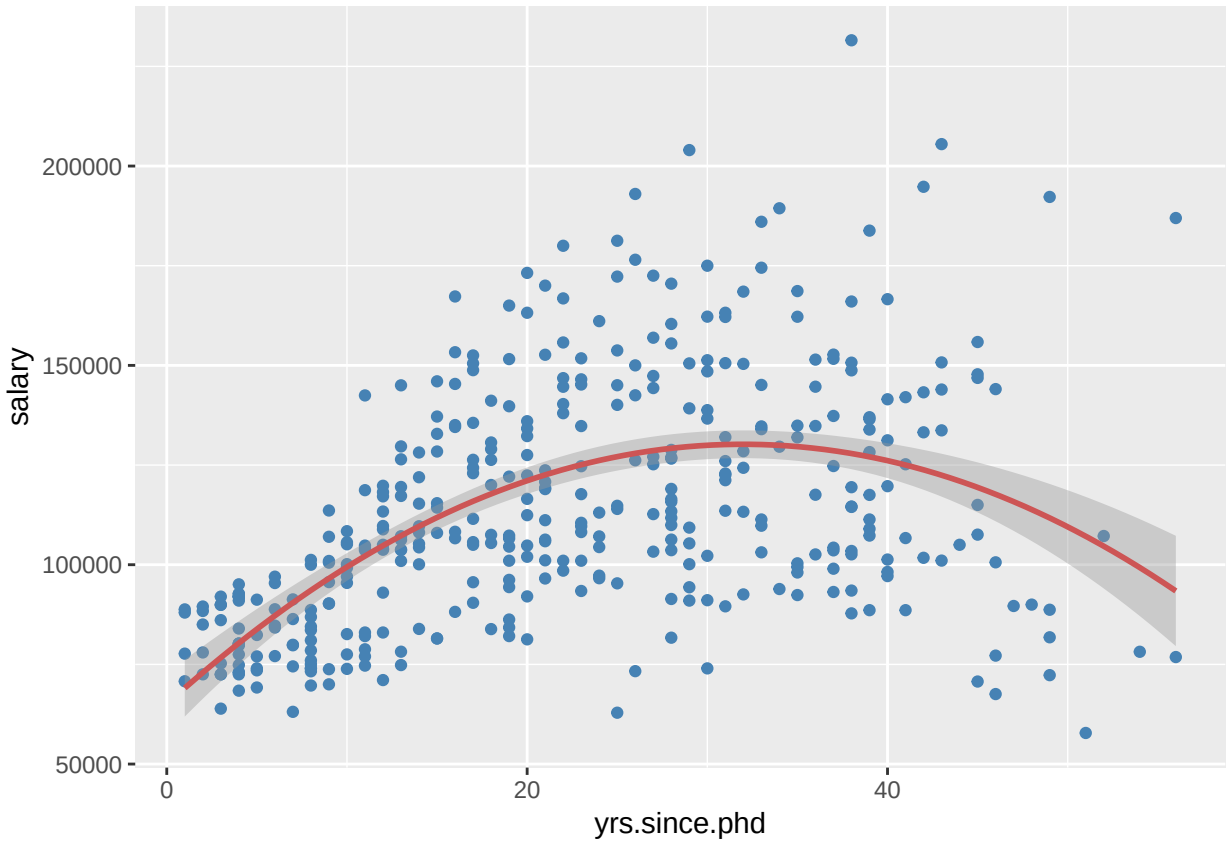


Figure 4.4: Scatterplot with quadratic fit line

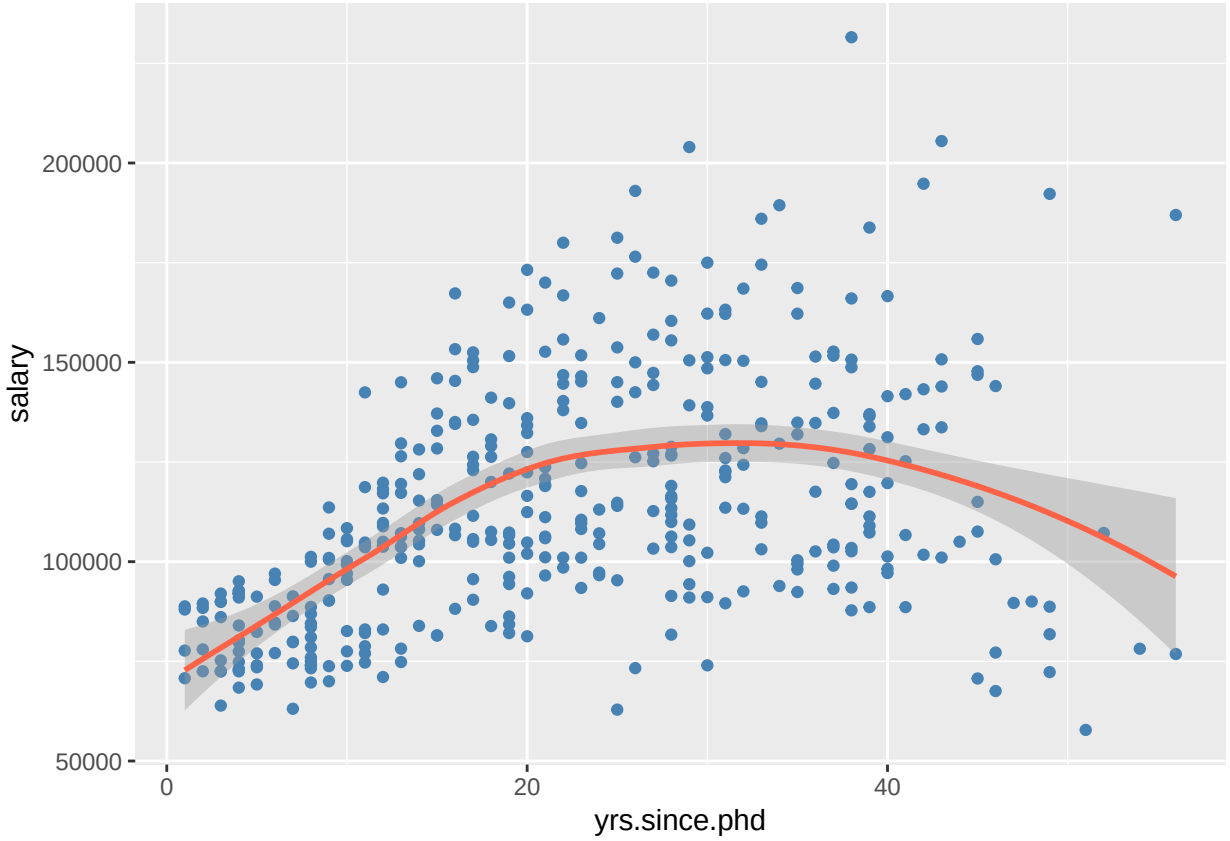


Figure 4.5: Scatterplot with nonparametric fit line

```
# scatterplot with loess smoothed line
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary)) +
  geom_point(color= "steelblue") +
  geom_smooth(color = "tomato")
```

You can suppress the confidence bands by including the option `se = FALSE`.

Here is a complete (and more attractive) plot.

```
# scatterplot with loess smoothed line
# and better labeling and color
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary)) +
  geom_point(color="cornflowerblue",
             size = 2,
             alpha = .6) +
  geom_smooth(size = 1.5,
              color = "darkgrey") +
  scale_y_continuous(label = scales::dollar,
                     limits = c(50000, 250000)) +
```

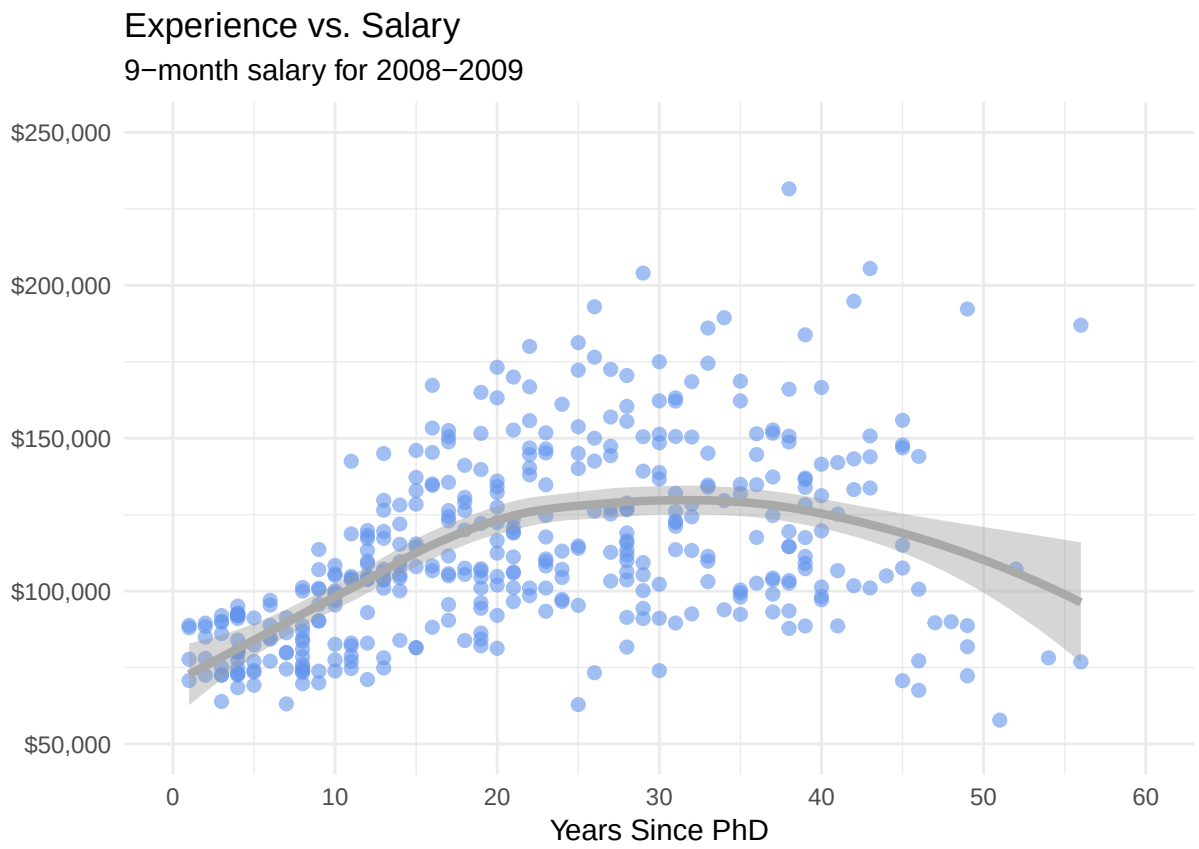


Figure 4.6: Scatterplot with nonparametric fit line

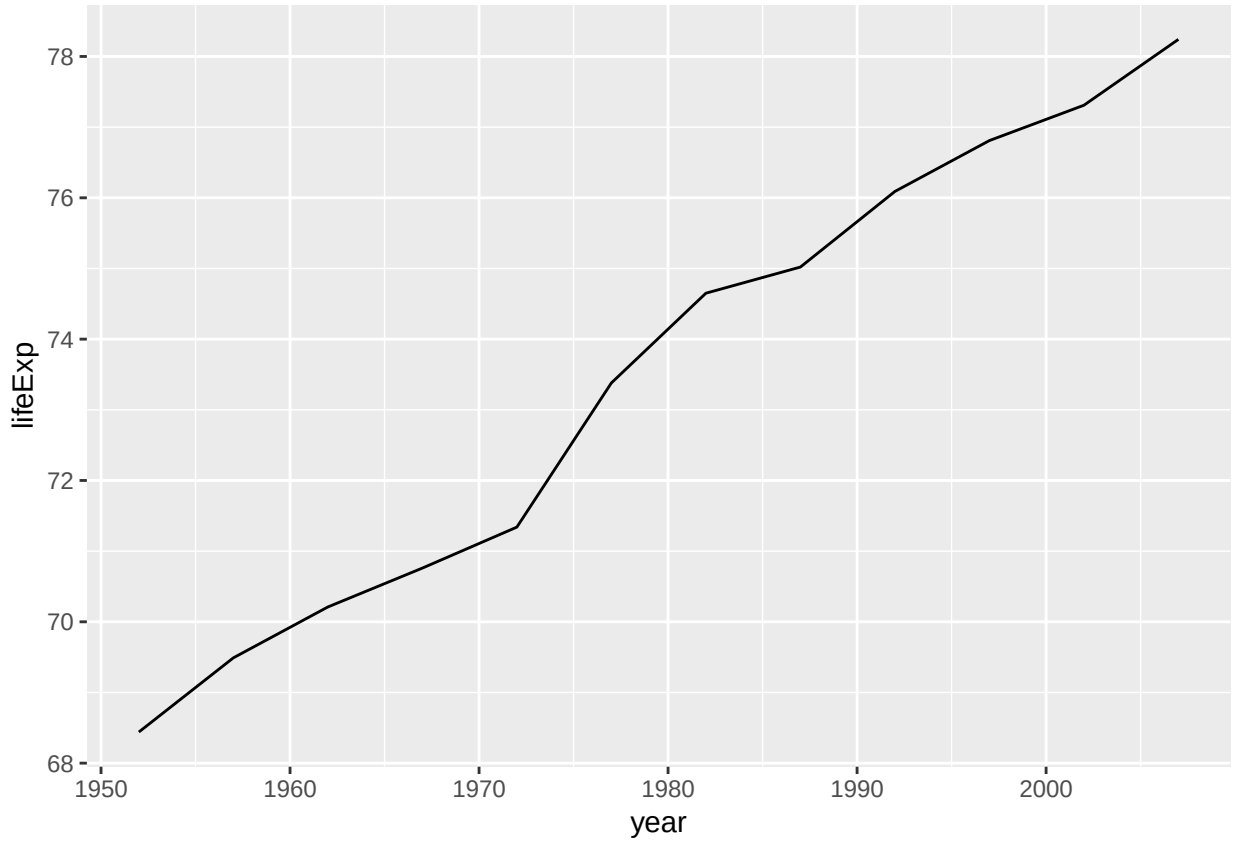
```
  scale_x_continuous(breaks = seq(0, 60, 10),
                     limits = c(0, 60)) +
  labs(x = "Years Since PhD",
       y = "",
       title = "Experience vs. Salary",
       subtitle = "9-month salary for 2008-2009") +
  theme_minimal()
```

4.2.2 Line plot

When one of the two variables represents time, a line plot can be an effective method of displaying relationship. For example, the code below displays the relationship between time (*year*) and life expectancy (*lifeExp*) in the United States between 1952 and 2007. The data comes from the gapminder dataset.

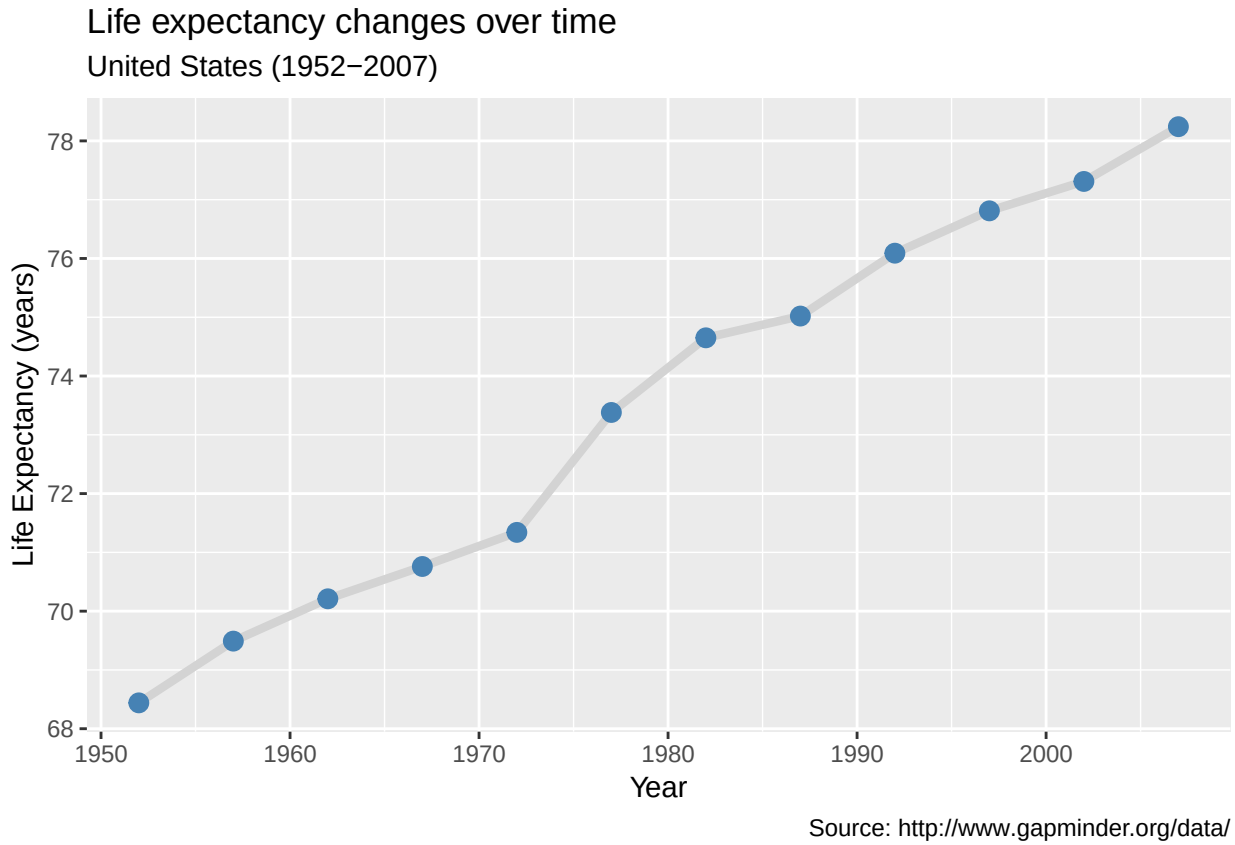
```
data(gapminder, package="gapminder")

# Select US cases
library(dplyr)
plotdata <- filter(gapminder,
                   country == "United States")
```

```
# simple line plot
ggplot(plotdata,
       aes(x = year,
           y = lifeExp)) +
  geom_line()
```



It is hard to read individual values in the graph above. In the next plot, we'll add points as well.

```
# line plot with points
# and improved labeling
ggplot(plotdata,
       aes(x = year,
           y = lifeExp)) +
  geom_line(size = 1.5,
            color = "lightgrey") +
  geom_point(size = 3,
             color = "steelblue") +
  labs(y = "Life Expectancy (years)",
       x = "Year",
       title = "Life expectancy changes over time",
       subtitle = "United States (1952-2007)",
       caption = "Source: http://www.gapminder.org/data/")
```



Time dependent data is covered in more detail under Time series. Customizing line graphs is covered in the Customizing graphs section.

4.3 Categorical vs. Quantitative

When plotting the relationship between a categorical variable and a quantitative variable, a large number of graph types are available. These include bar charts using summary statistics, grouped kernel density plots, side-by-side box plots, side-by-side violin plots, mean/sem plots, ridgeline plots, and Cleveland plots.

4.3.1 Bar chart (on summary statistics)

In previous sections, bar charts were used to display the number of cases by category for a single variable or for two variables. You can also use bar charts to display other summary statistics (e.g., means or medians) on a quantitative variable for each level of a categorical variable.

For example, the following graph displays the mean salary for a sample of university professors by their academic rank.

```
data(Salaries, package="carData")

# calculate mean salary for each rank
library(dplyr)
plotdata <- Salaries %>%
  group_by(rank) %>%
  summarize(mean_salary = mean(salary))
```

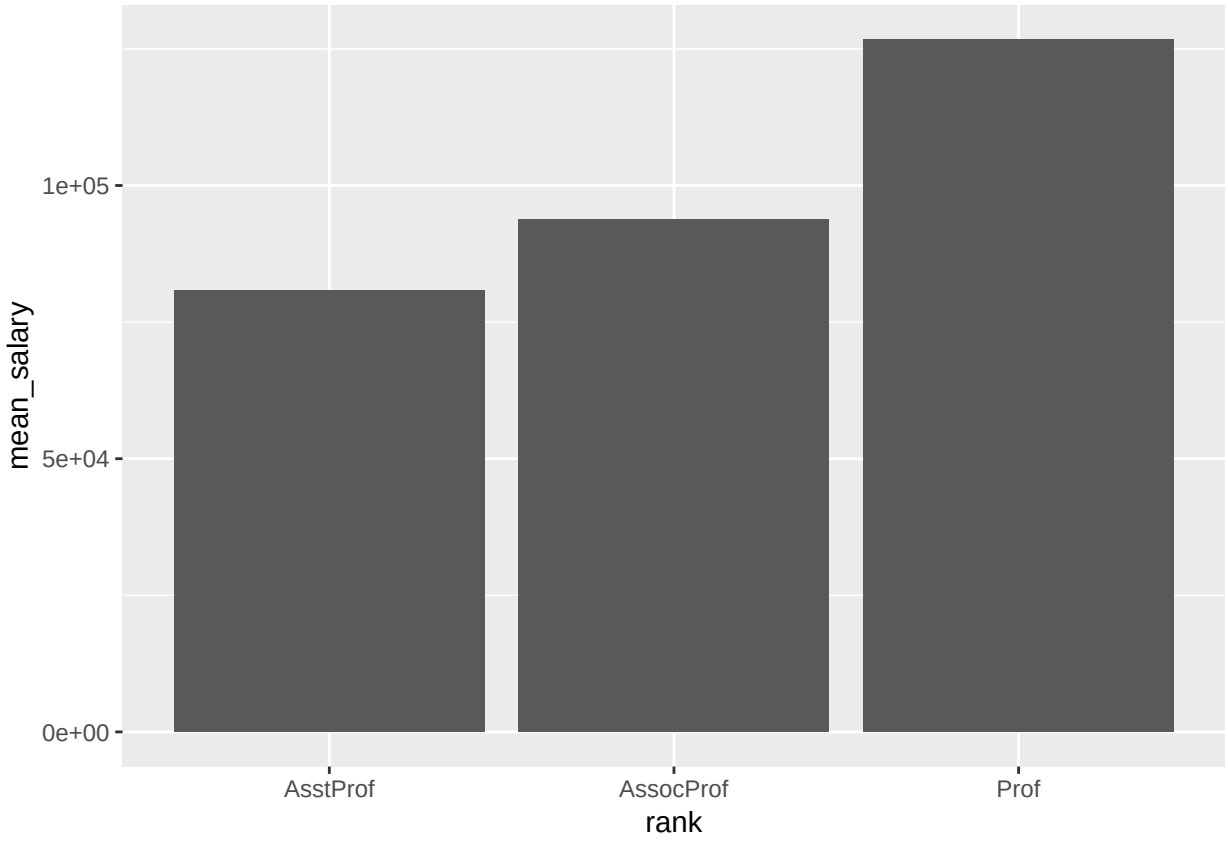


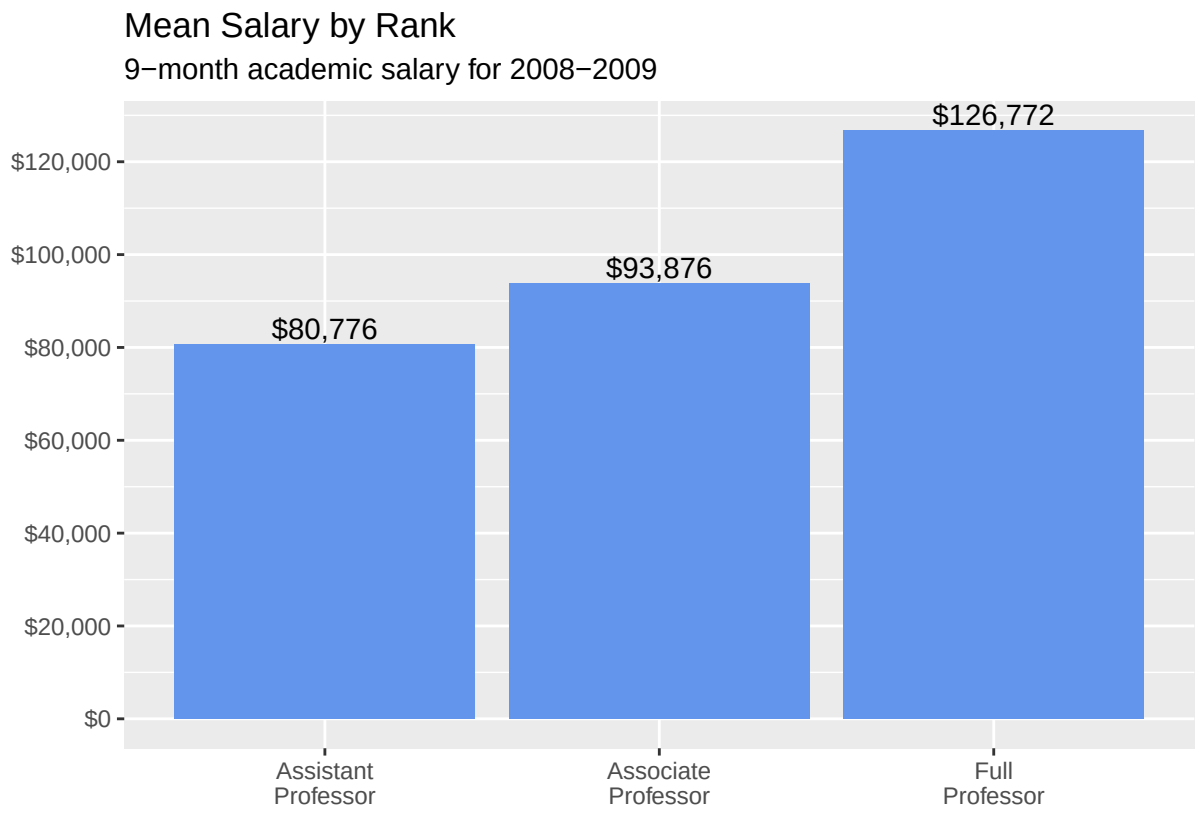
Figure 4.7: Bar chart displaying means

```
# plot mean salaries
ggplot(plotdata,
       aes(x = rank,
           y = mean_salary)) +
  geom_bar(stat = "identity")
```

We can make it more attractive with some options.

```
# plot mean salaries in a more attractive fashion
library(scales)
ggplot(plotdata,
       aes(x = factor(rank,
                      labels = c("Assistant\nProfessor",
                                 "Associate\nProfessor",
                                 "Full\nProfessor")),
                      y = mean_salary)) +
  geom_bar(stat = "identity",
           fill = "cornflowerblue") +
  geom_text(aes(label = dollar(mean_salary)),
            vjust = -0.25) +
  scale_y_continuous(breaks = seq(0, 130000, 20000),
                     label = dollar) +
```

```
  labs(title = "Mean Salary by Rank",
       subtitle = "9-month academic salary for 2008-2009",
       x = "",
       y = "")
```

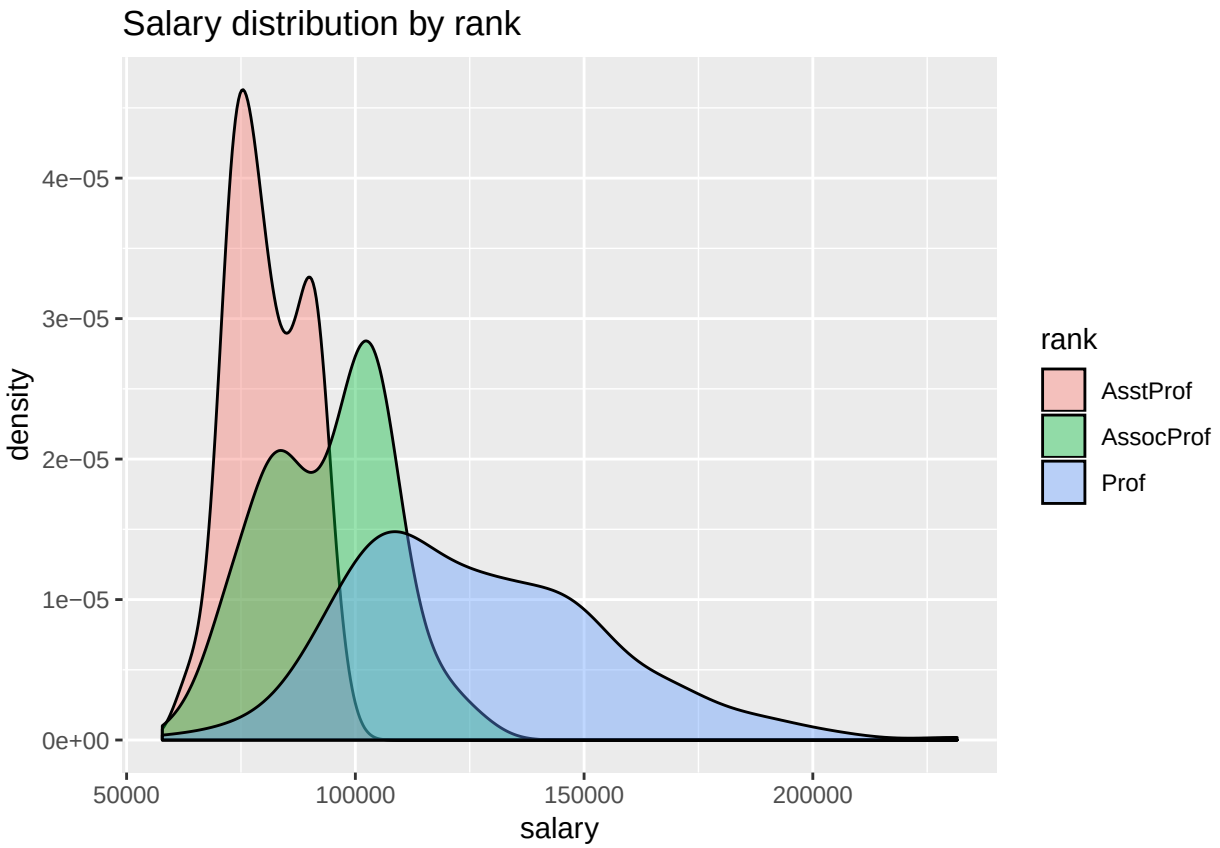


One limitation of such plots is that they do not display the distribution of the data - only the summary statistic for each group. The plots below correct this limitation to some extent.

4.3.2 Grouped kernel density plots

One can compare groups on a numeric variable by superimposing kernel density plots in a single graph.

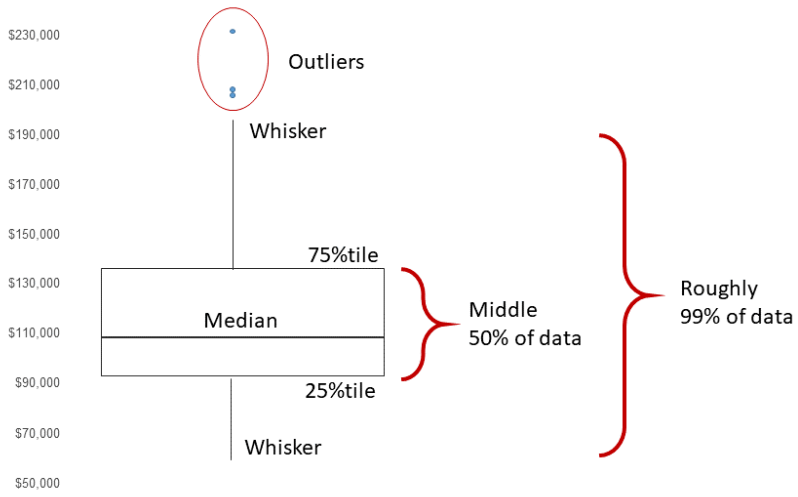
```
# plot the distribution of salaries
# by rank using kernel density plots
ggplot(Salaries,
       aes(x = salary,
           fill = rank)) +
  geom_density(alpha = 0.4) +
  labs(title = "Salary distribution by rank")
```



The `alpha` option makes the density plots partially transparent, so that we can see what is happening under the overlaps. Alpha values range from 0 (transparent) to 1 (opaque). The graph makes clear that, in general, salary goes up with rank. However, the salary range for full professors is *very* wide.

4.3.3 Box plots

A boxplot displays the 25^{th} percentile, median, and 75^{th} percentile of a distribution. The whiskers (vertical lines) capture roughly 99% of a normal distribution, and observations outside this range are plotted as points representing outliers (see the figure below).



Side-by-side box plots are very useful for comparing groups (i.e., the levels of a categorical variable) on a numerical variable.

```
# plot the distribution of salaries by rank using boxplots
ggplot(Salaries,
       aes(x = rank,
           y = salary)) +
  geom_boxplot() +
  labs(title = "Salary distribution by rank")
```

Notched boxplots provide an approximate method for visualizing whether groups differ. Although not a formal test, if the notches of two boxplots do not overlap, there is strong evidence (95% confidence) that the medians of the two groups differ.

```
# plot the distribution of salaries by rank using boxplots
ggplot(Salaries, aes(x = rank,
                     y = salary)) +
  geom_boxplot(notch = TRUE,
               fill = "cornflowerblue",
               alpha = .7) +
  labs(title = "Salary distribution by rank")
```

In the example above, all three groups appear to differ.

One of the advantages of boxplots is that their widths are not usually meaningful. This allows you to compare the distribution of many groups in a single graph.

4.3.4 Violin plots

Violin plots are similar to kernel density plots, but are mirrored and rotated 90°.

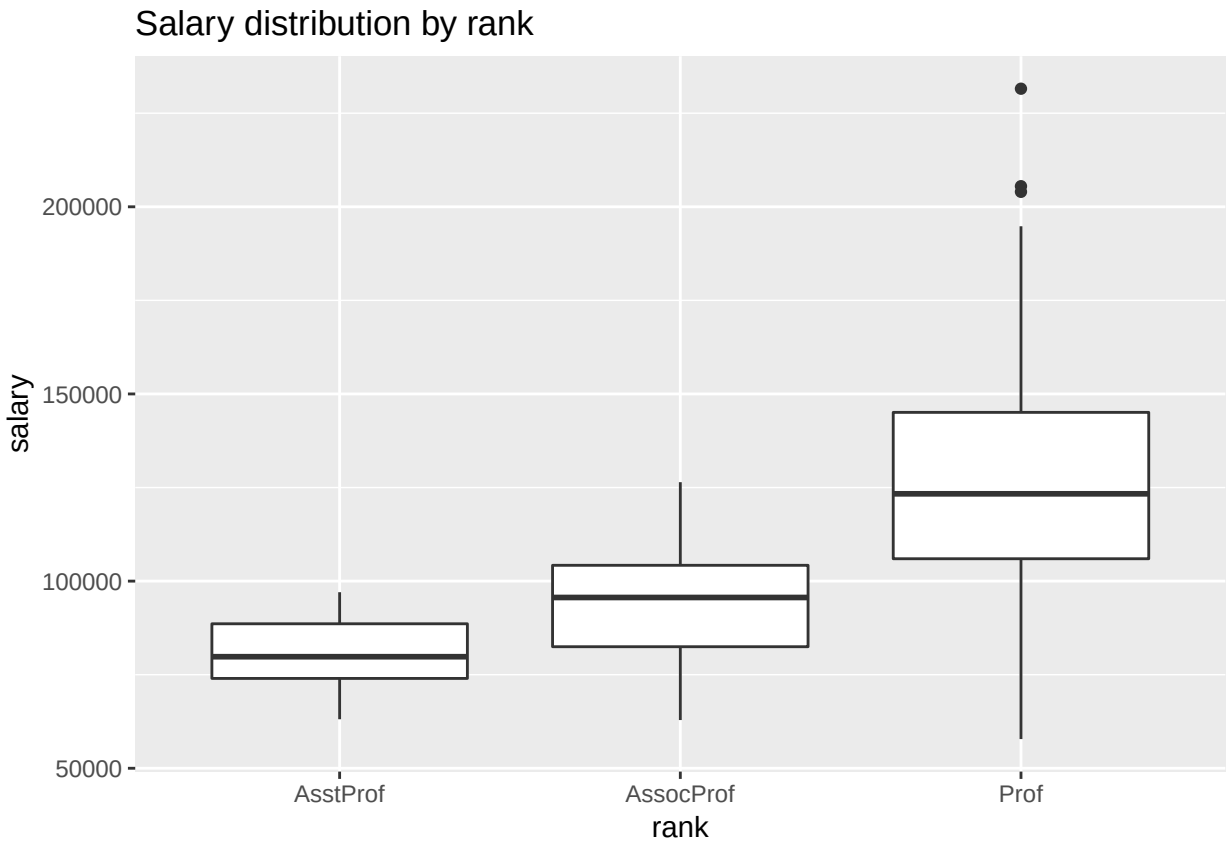


Figure 4.8: Side-by-side boxplots

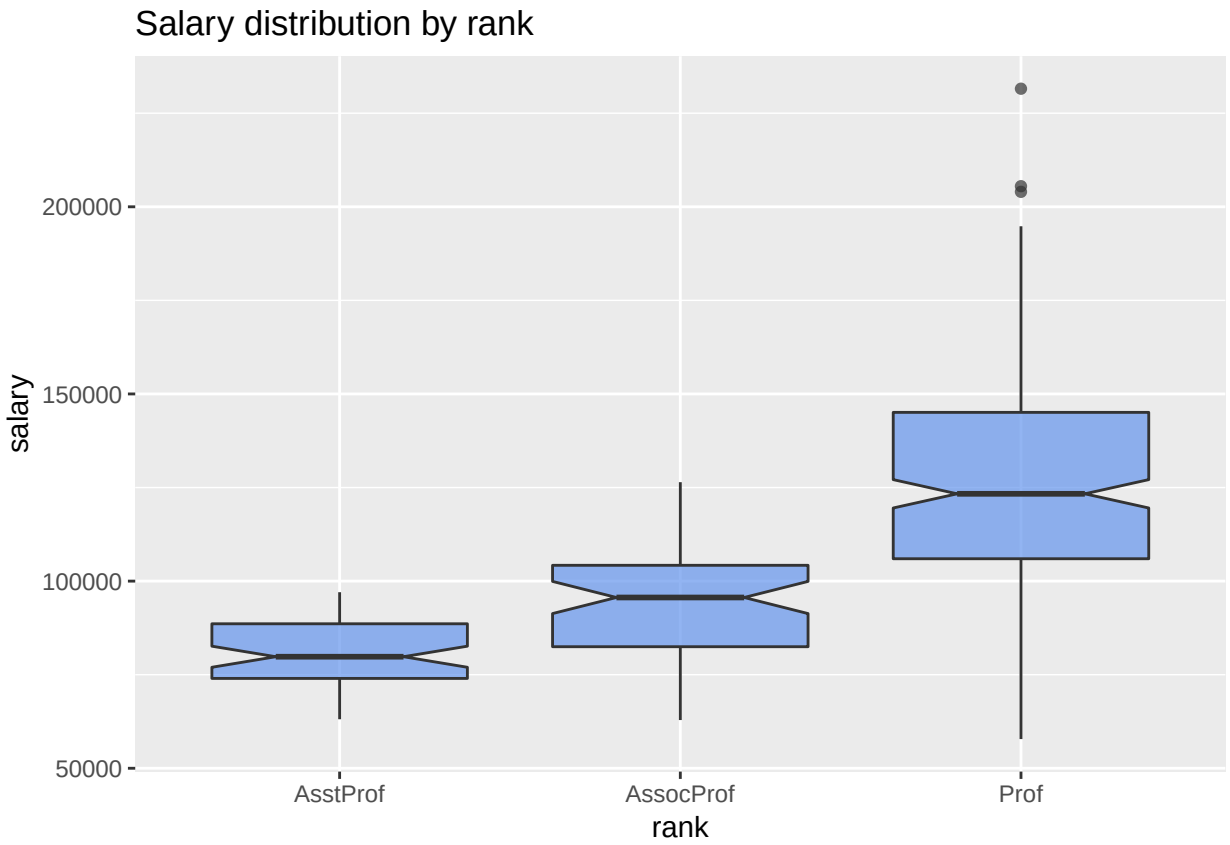
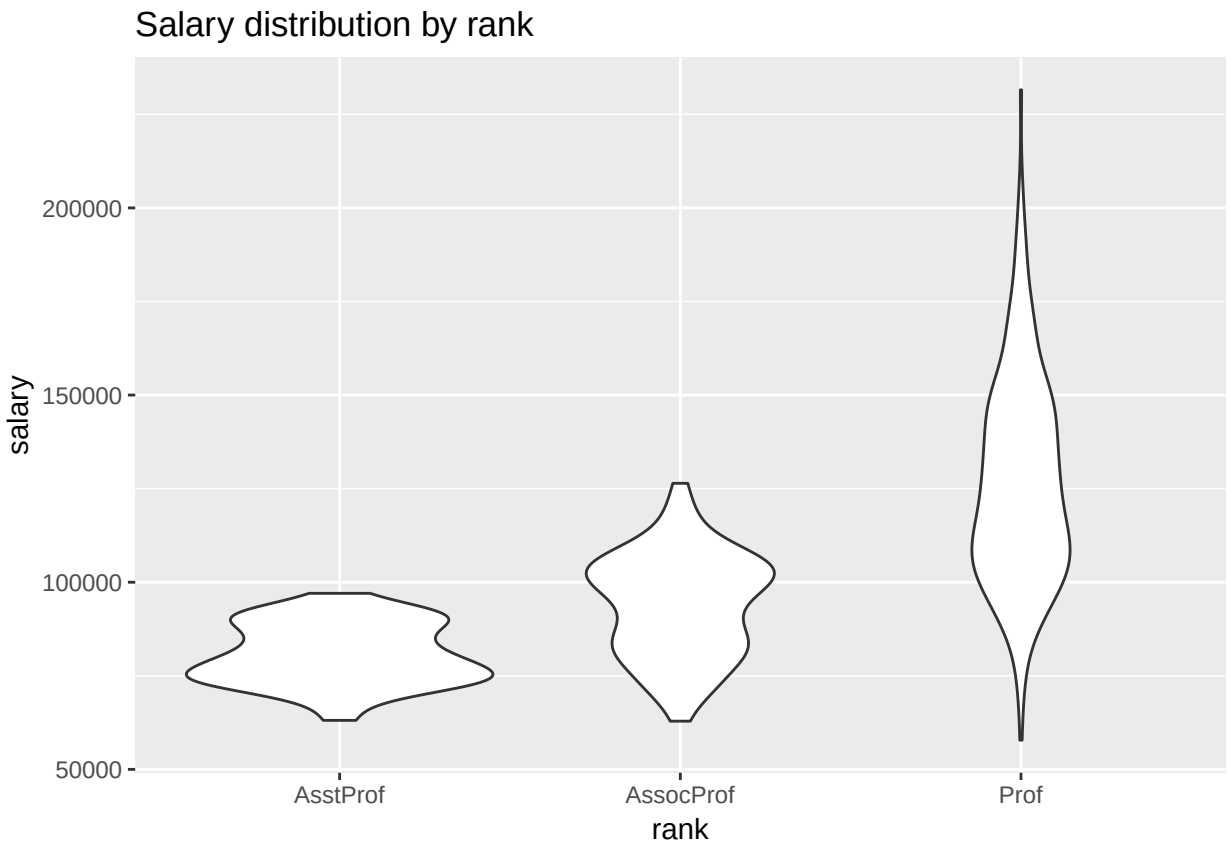


Figure 4.9: Side-by-side notched boxplots

```
# plot the distribution of salaries
# by rank using violin plots
ggplot(Salaries,
       aes(x = rank,
           y = salary)) +
  geom_violin() +
  labs(title = "Salary distribution by rank")
```



A useful variation is to superimpose boxplots on violin plots.

```
# plot the distribution using violin and boxplots
ggplot(Salaries,
       aes(x = rank,
           y = salary)) +
  geom_violin(fill = "cornflowerblue") +
  geom_boxplot(width = .2,
               fill = "orange",
               outlier.color = "orange",
               outlier.size = 2) +
  labs(title = "Salary distribution by rank")
```

4.3.5 Ridgeline plots

A ridgeline plot (also called a joyplot) displays the distribution of a quantitative variable for several groups. They're similar to kernel density plots with vertical faceting, but take up less room. Ridgeline plots are

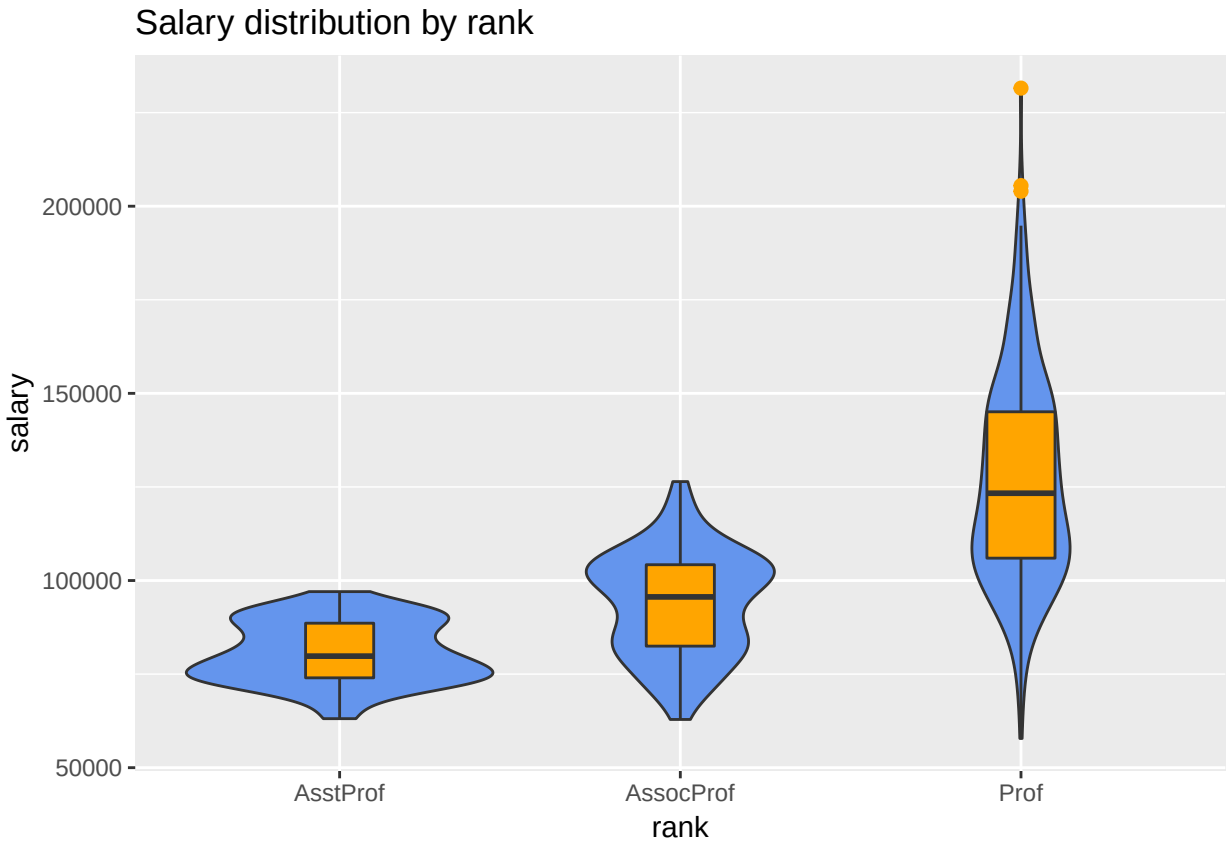


Figure 4.10: Side-by-side violin/box plots

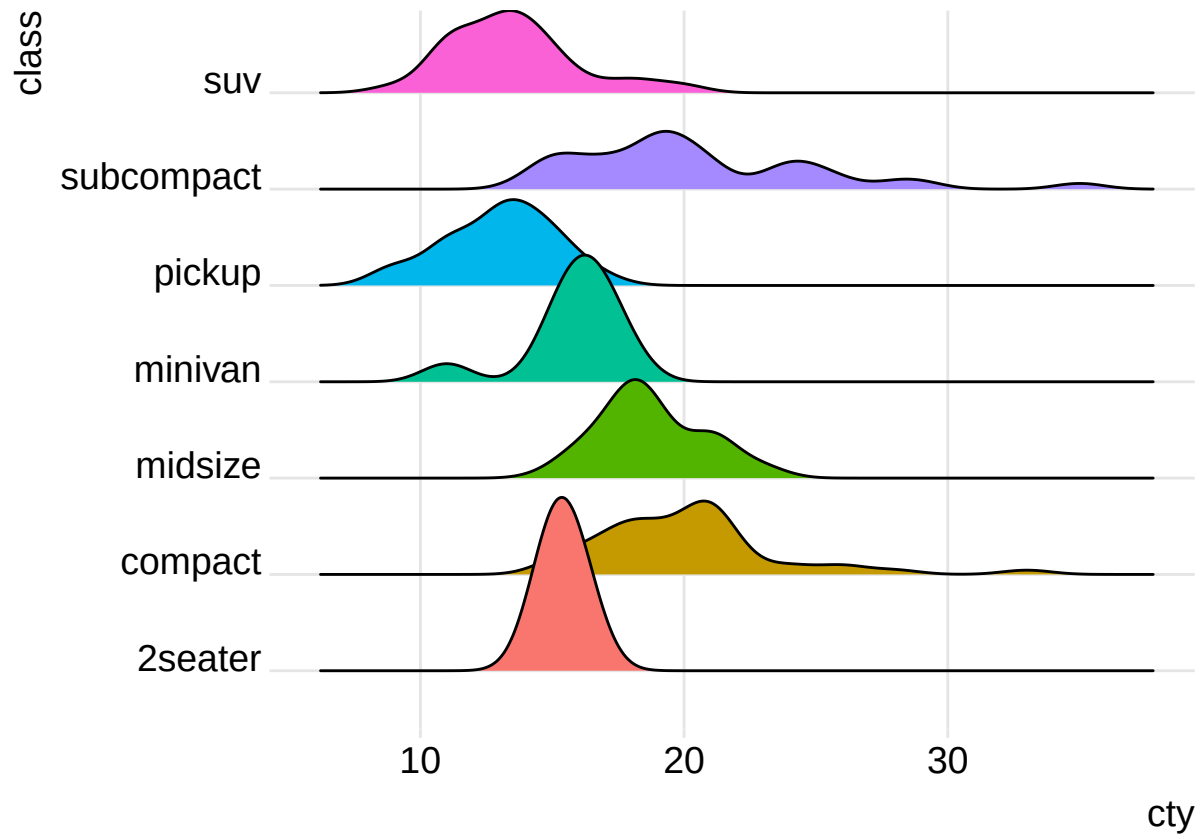


Figure 4.11: Ridgeline graph with color fill

created with the `ggridges` package.

Using the Fuel economy dataset, let's plot the distribution of city driving miles per gallon by car class.

```
# create ridgeline graph
library(ggplot2)
library(ggridges)

ggplot(mpg,
       aes(x = cty,
           y = class,
           fill = class)) +
  geom_density_ridges() +
  theme_ridges() +
  labs("Highway mileage by auto class") +
  theme(legend.position = "none")
```

I've suppressed the legend here because it's redundant (the distributions are already labeled on the y-axis). Unsurprisingly, pickup trucks have the poorest mileage, while subcompacts and compact cars tend to achieve ratings. However, there is a very wide range of gas mileage scores for these smaller cars.

Note the the possible overlap of distributions is the trade-off for a more compact graph. You can add transparency if the the overlap is severe using `geom_density_ridges(alpha = n)`, where n ranges from 0 (transparent) to 1 (opaque). See the package vingnette for more details.

Table 4.1: Plot data

rank	n	mean	sd	se	ci
AsstProf	67	80775.99	8174.113	998.6268	1993.823
AssocProf	64	93876.44	13831.700	1728.9625	3455.056
Prof	266	126772.11	27718.675	1699.5410	3346.322

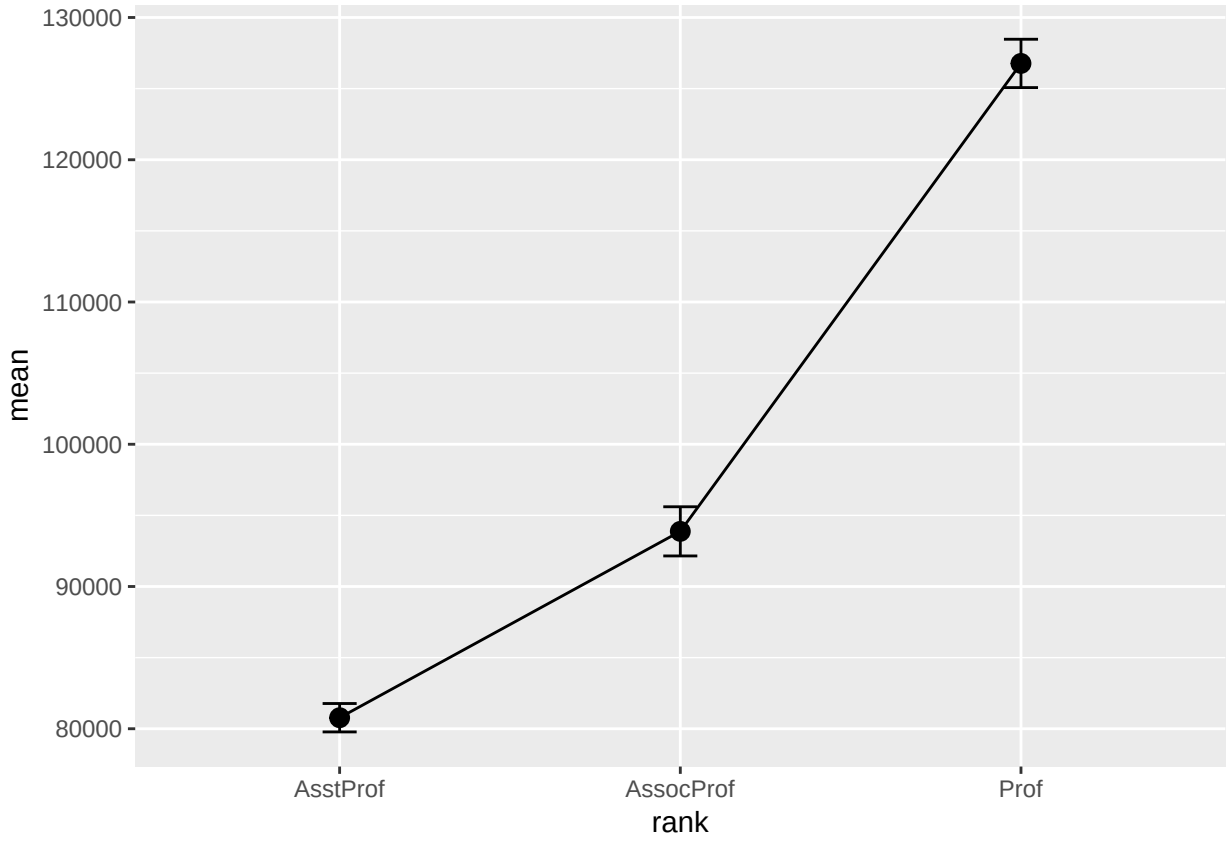
4.3.6 Mean/SEM plots

A popular method for comparing groups on a numeric variable is the mean plot with error bars. Error bars can represent standard deviations, standard error of the mean, or confidence intervals. In this section, we'll plot means and standard errors.

```
# calculate means, standard deviations,
# standard errors, and 95% confidence
# intervals by rank
library(dplyr)
plotdata <- Salaries %>%
  group_by(rank) %>%
  summarize(n = n(),
            mean = mean(salary),
            sd = sd(salary),
            se = sd / sqrt(n),
            ci = qt(0.975, df = n - 1) * sd / sqrt(n))
```

The resulting dataset is given below.

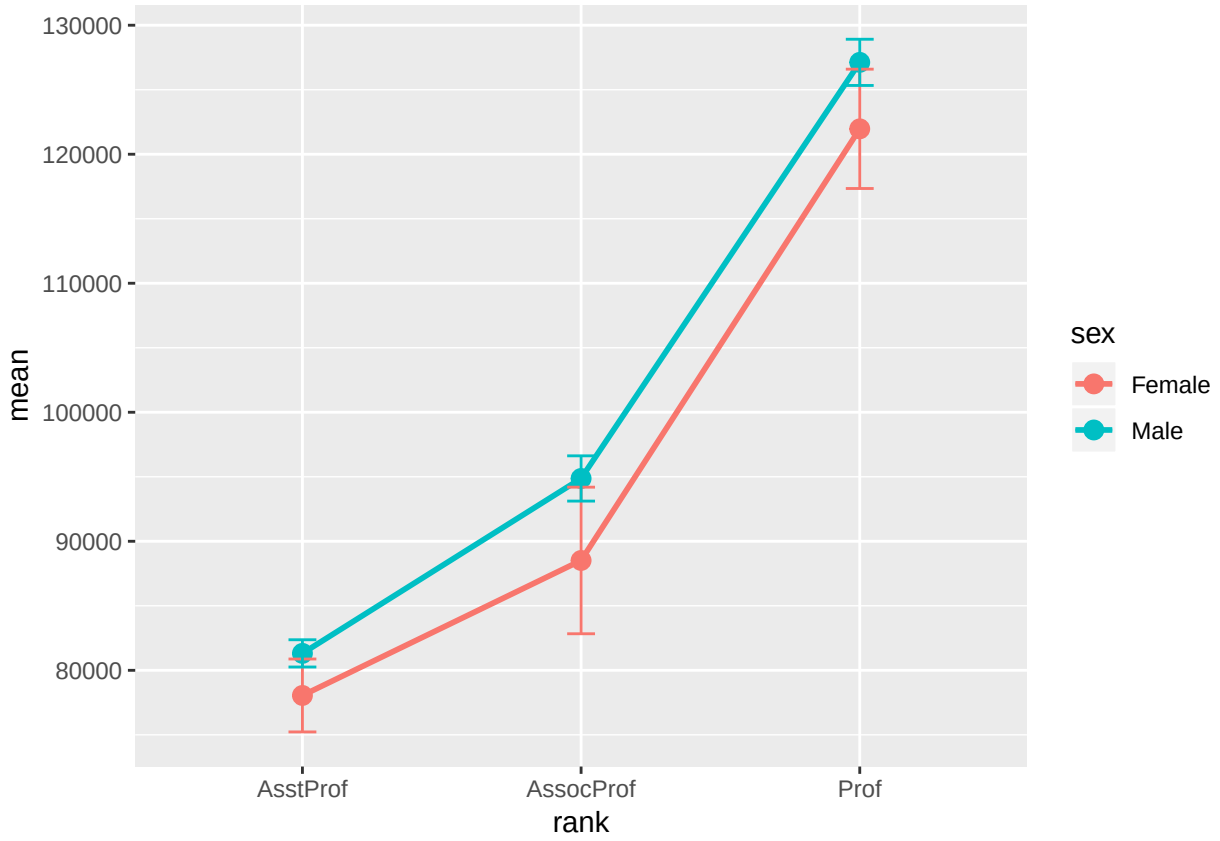
```
# plot the means and standard errors
ggplot(plotdata,
       aes(x = rank,
           y = mean,
           group = 1)) +
  geom_point(size = 3) +
  geom_line() +
  geom_errorbar(aes(ymin = mean - se,
                    ymax = mean + se),
                width = .1)
```



Although we plotted error bars representing the standard error, we could have plotted standard deviations or 95% confidence intervals. Simply replace `se` with `sd` or `error` in the `aes` option.

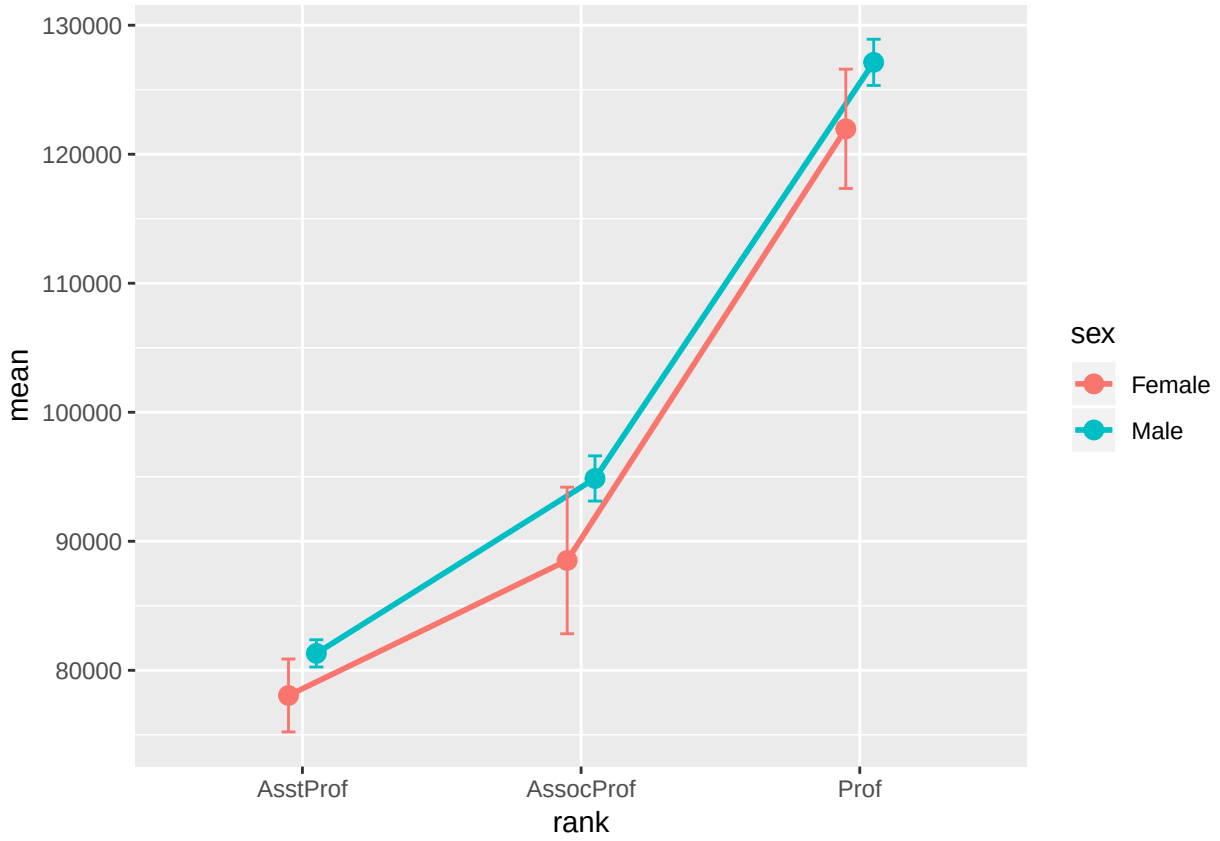
We can use the same technique to compare salary across rank and sex. (Technically, this is not bivariate since we're plotting rank, sex, and salary, but it seems to fit here)

```
# calculate means and standard errors by rank and sex
plotdata <- Salaries %>%
  group_by(rank, sex) %>%
  summarize(n = n(),
            mean = mean(salary),
            sd = sd(salary),
            se = sd/sqrt(n))

# plot the means and standard errors by sex
ggplot(plotdata, aes(x = rank,
                     y = mean,
                     group=sex,
                     color=sex)) +
  geom_point(size = 3) +
  geom_line(size = 1) +
  geom_errorbar(aes(ymin  =mean - se,
                    ymax = mean+se),
                width = .1)
```



Unfortunately, the error bars overlap. We can dodge the horizontal positions a bit to overcome this.

```
# plot the means and standard errors by sex (dodged)
pd <- position_dodge(0.2)
ggplot(plotdata,
       aes(x = rank,
           y = mean,
           group=sex,
           color=sex)) +
  geom_point(position = pd,
             size = 3) +
  geom_line(position = pd,
            size = 1) +
  geom_errorbar(aes(ymin = mean - se,
                    ymax = mean + se),
                width = .1,
                position= pd)
```



Finally, lets add some options to make the graph more attractive.

```
# improved means/standard error plot
pd <- position_dodge(0.2)
ggplot(plotdata,
       aes(x = factor(rank,
                      labels = c("Assistant\nProfessor",
                                 "Associate\nProfessor",
                                 "Full\nProfessor")),
           y = mean,
           group=sex,
           color=sex)) +
  geom_point(position=pd,
             size = 3) +
  geom_line(position = pd,
            size = 1) +
  geom_errorbar(aes(ymin = mean - se,
                    ymax = mean + se),
                width = .1,
                position = pd,
                size = 1) +
  scale_y_continuous(label = scales::dollar) +
  scale_color_brewer(palette="Set1") +
  theme_minimal() +
  labs(title = "Mean salary by rank and sex",
       subtitle = "(mean +/- standard error)",
       x = "",
```

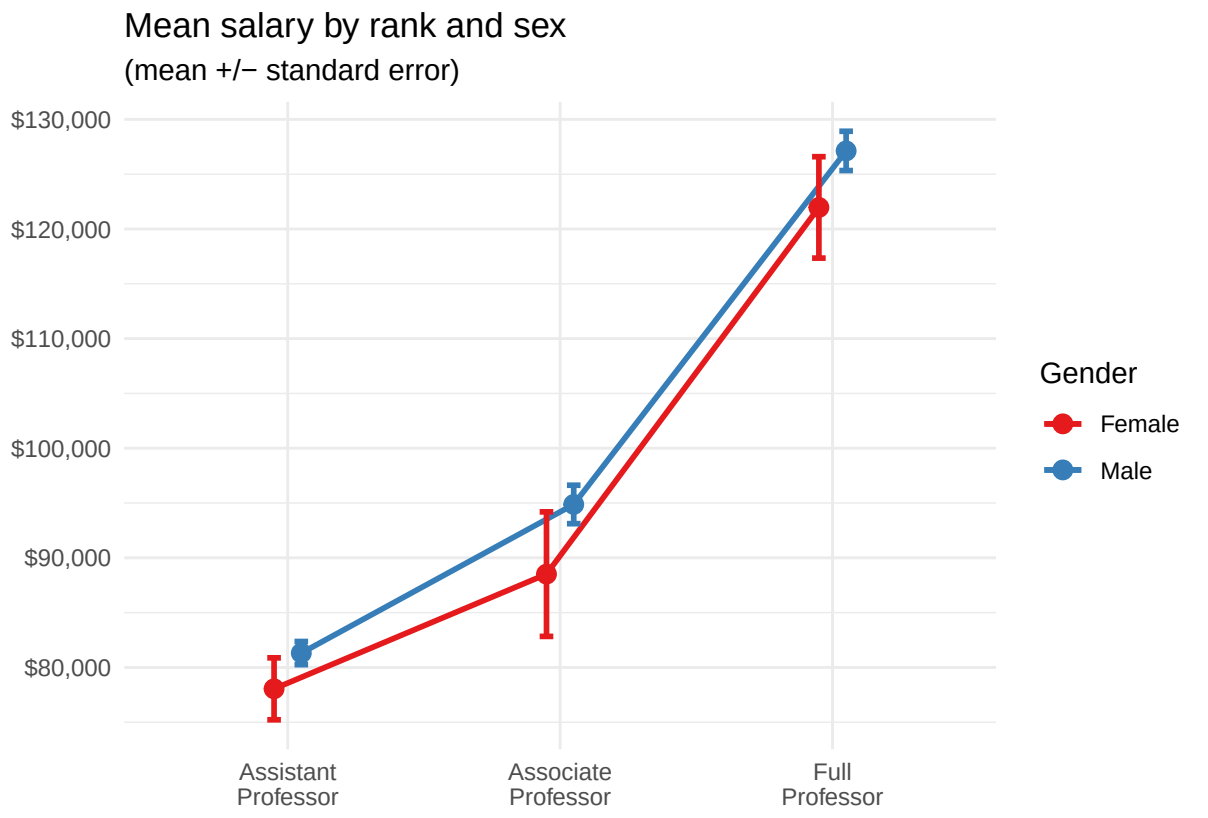


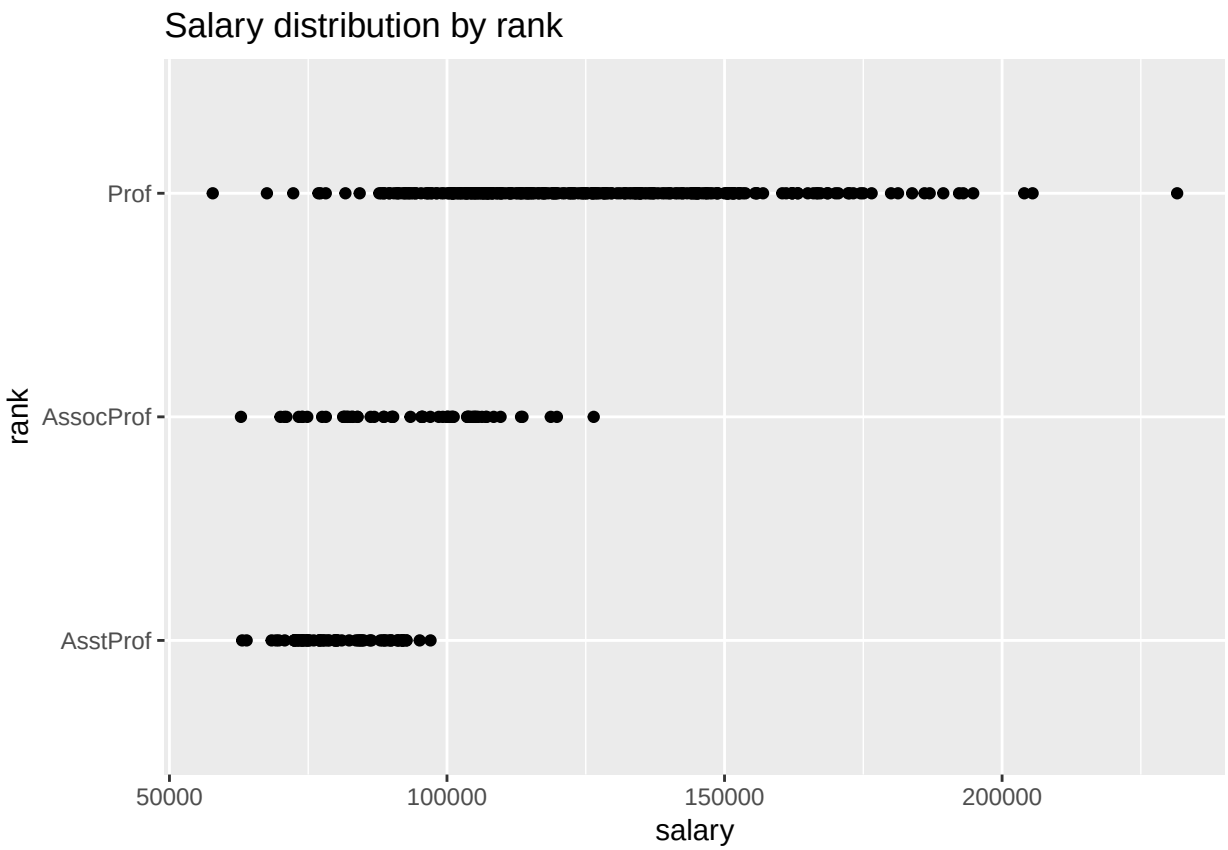
Figure 4.12: Mean/se plot with better labels and colors

```
       y = "",
       color = "Gender")
```

4.3.7 Strip plots

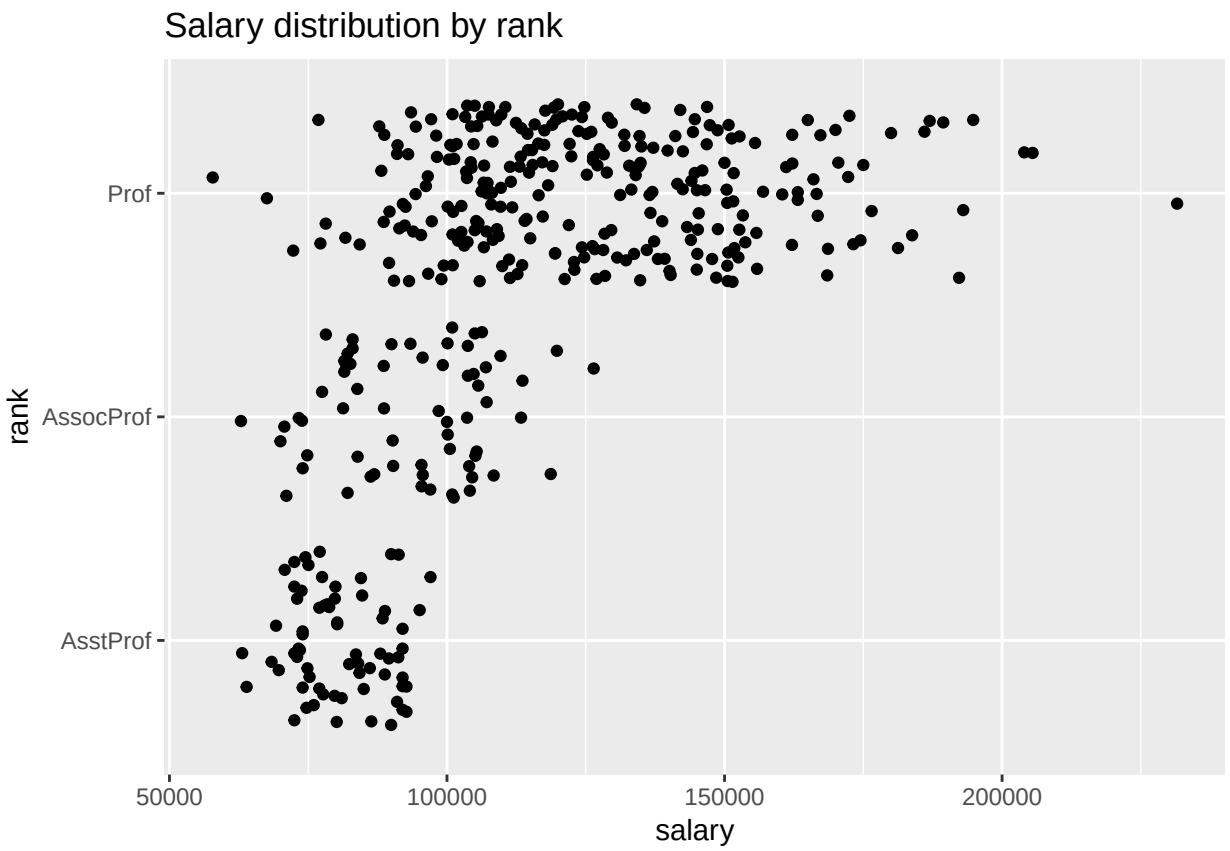
The relationship between a grouping variable and a numeric variable can be displayed with a scatter plot. For example

```
# plot the distribution of salaries
# by rank using strip plots
ggplot(Salaries,
       aes(y = rank,
           x = salary)) +
  geom_point() +
  labs(title = "Salary distribution by rank")
```



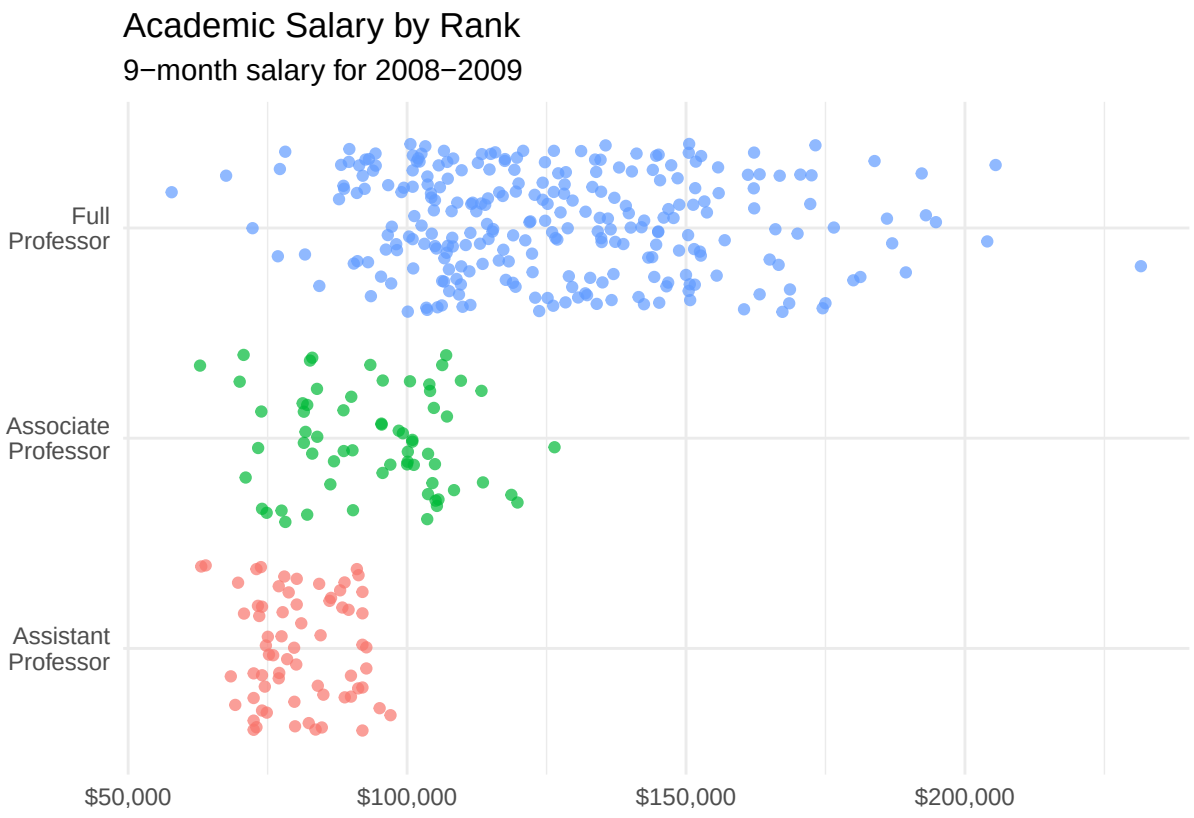
These one-dimensional scatterplots are called strip plots. Unfortunately, overprinting of points makes interpretation difficult. The relationship is easier to see if the the points are jittered. Basically a small random number is added to each y-coordinate.

```
# plot the distribution of salaries
# by rank using jittering
ggplot(Salaries,
       aes(y = rank,
           x = salary)) +
  geom_jitter() +
  labs(title = "Salary distribution by rank")
```



It is easier to compare groups if we use color.

```
# plot the distribution of salaries
# by rank using jittering
library(scales)
ggplot(Salaries,
       aes(y = factor(rank,
                      labels = c("Assistant\nProfessor",
                                 "Associate\nProfessor",
                                 "Full\nProfessor")),
           x = salary,
           color = rank)) +
  geom_jitter(alpha = 0.7,
              size = 1.5) +
  scale_x_continuous(label = dollar) +
  labs(title = "Academic Salary by Rank",
       subtitle = "9-month salary for 2008-2009",
       x = "",
       y = "") +
  theme_minimal() +
  theme(legend.position = "none")
```



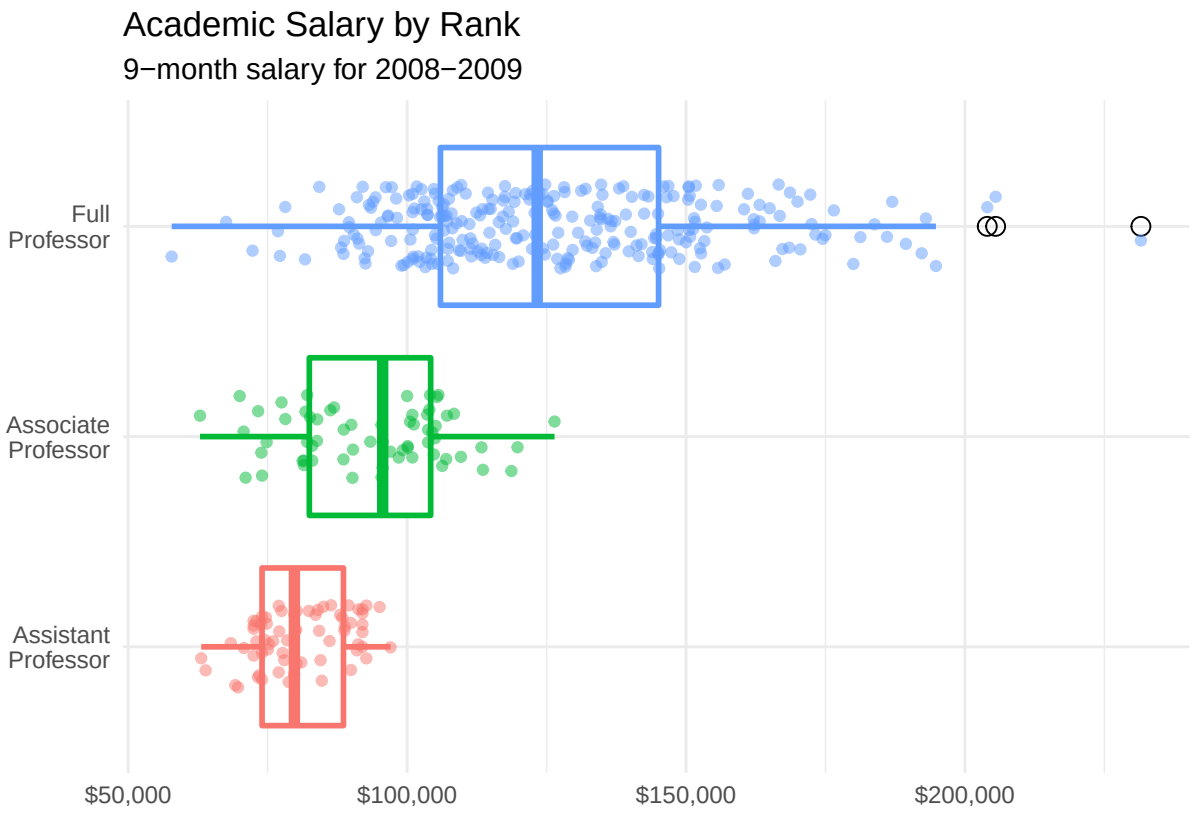
The option `legend.position = "none"` is used to suppress the legend (which is not needed here). Jittered plots work well when the number of points in not overly large.

4.3.7.1 Combining jitter and boxplots

It may be easier to visualize distributions if we add boxplots to the jitter plots.

```
# plot the distribution of salaries
# by rank using jittering
library(scales)
ggplot(Salaries,
       aes(x = factor(rank,
                      labels = c("Assistant\nProfessor",
                                 "Associate\nProfessor",
                                 "Full\nProfessor")),
           y = salary,
           color = rank)) +
  geom_boxplot(size=1,
               outlier.shape = 1,
               outlier.color = "black",
               outlier.size  = 3) +
  geom_jitter(alpha = 0.5,
              width=.2) +
  scale_y_continuous(label = dollar) +
  labs(title = "Academic Salary by Rank",
       subtitle = "9-month salary for 2008-2009",
```

```
      x = "",
      y = "") +
theme_minimal() +
theme(legend.position = "none") +
coord_flip()
```



Several options were added to create this plot.

For the boxplot

- `size = 1` makes the lines thicker

- `outlier.color = "black"` makes outliers black
- `outlier.shape = 1` specifies circles for outliers
- `outlier.size = 3` increases the size of the outlier symbol

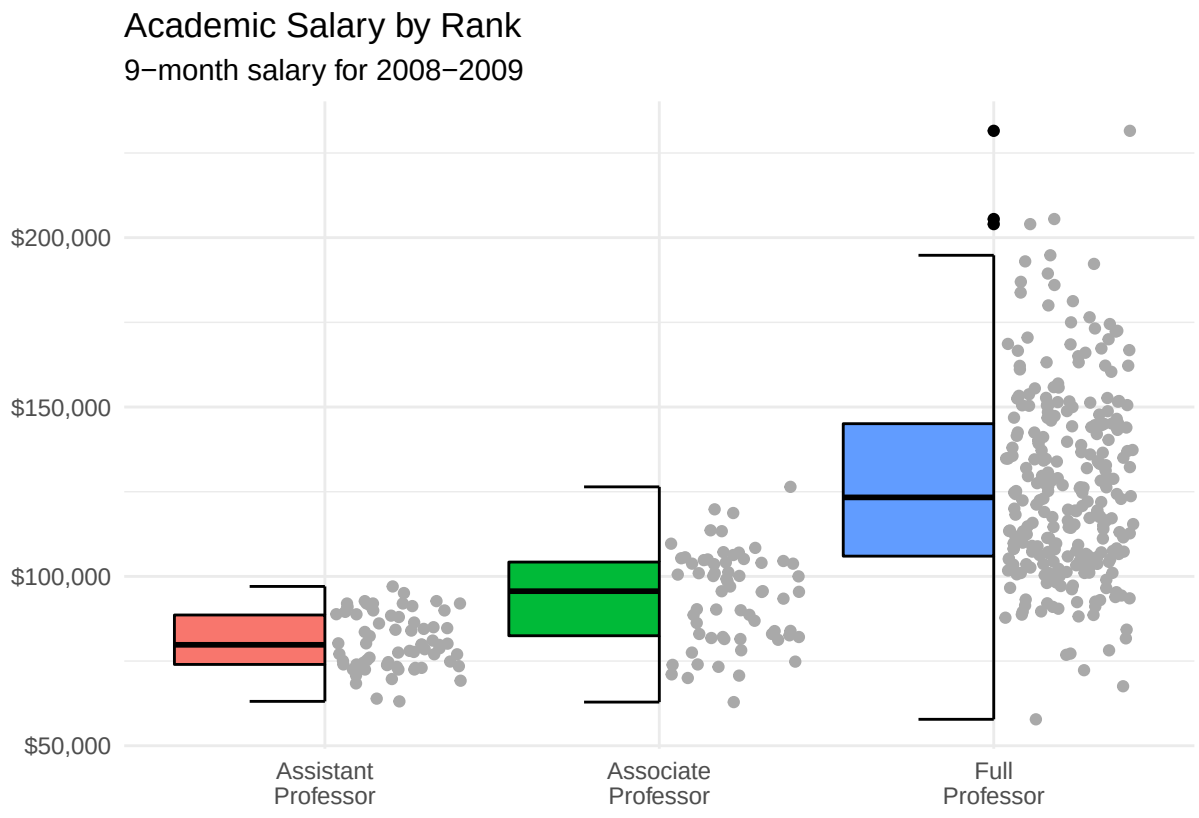
For the jitter

- `alpha = 0.5` makes the points more transparent
- `width = .2` decreases the amount of jitter (.4 is the default)

Finally, the x and y axes are revered using the `coord_flip` function (i.e., the graph is turned on its side).

Before moving on, it is worth mentioning the `geom_boxjitter` function provided in the `ggpol` package. It creates a hybrid boxplot - half boxplot, half scatterplot.

```
# plot the distribution of salaries
# by rank using jittering
library(ggpol)
library(scales)
ggplot(Salaries,
       aes(x = factor(rank,
                      labels = c("Assistant\nProfessor",
                                 "Associate\nProfessor",
                                 "Full\nProfessor")),
           y = salary,
           fill=rank)) +
  geom_boxjitter(color="black",
                 jitter.color = "darkgrey",
                 errorbar.draw = TRUE) +
  scale_y_continuous(label = dollar) +
  labs(title = "Academic Salary by Rank",
       subtitle = "9-month salary for 2008-2009",
       x = "",
       y = "") +
  theme_minimal() +
  theme(legend.position = "none")
```

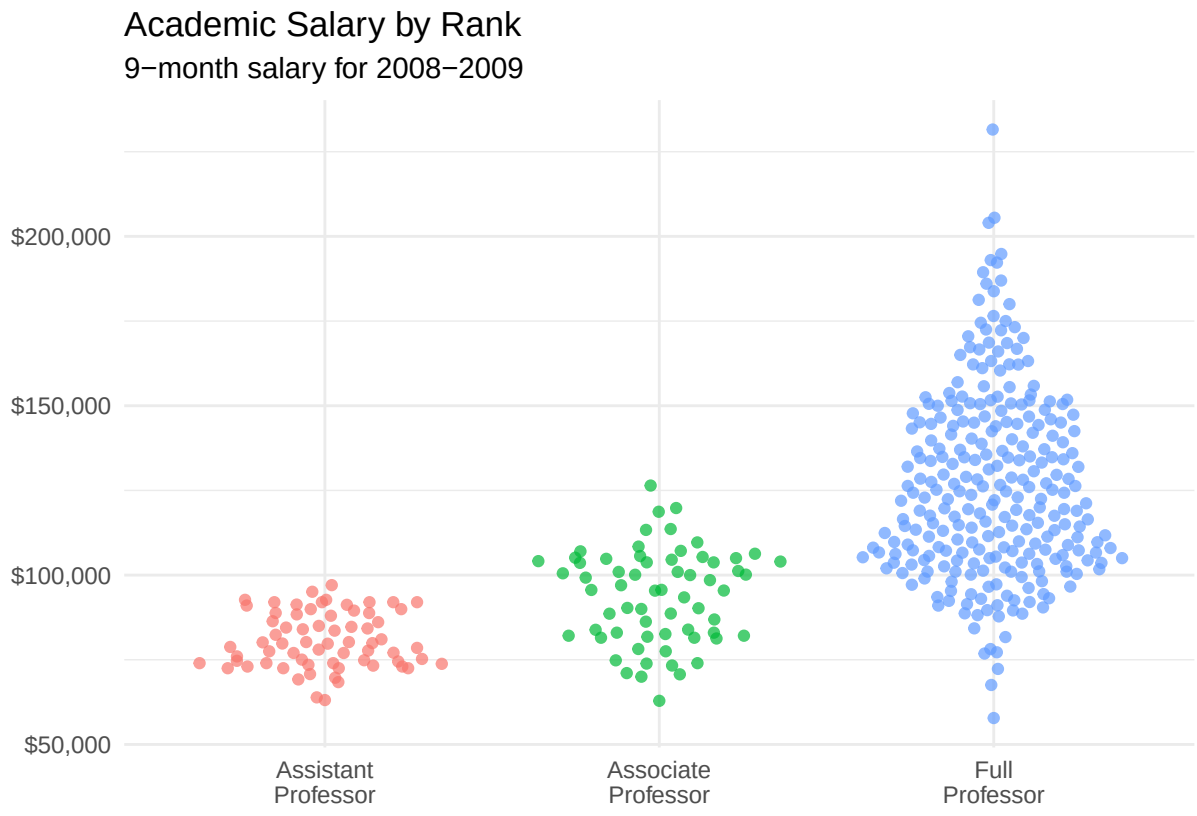


Beeswarm Plots

Beeswarm plots (also called violin scatter plots) are similar to jittered scatterplots, in that they display the distribution of a quantitative variable by plotting points in way that reduces overlap. In addition, they also help display the density of the data at each point (in a manner that is similar to a violin plot). Continuing

the previous example

```
# plot the distribution of salaries
# by rank using beewarm-syle plots
library(ggbeeswarm)
library(scales)
ggplot(Salaries,
       aes(x = factor(rank,
                      labels = c("Assistant\nProfessor",
                                 "Associate\nProfessor",
                                 "Full\nProfessor")),
           y = salary,
           color = rank)) +
  geom_quasirandom(alpha = 0.7,
                   size = 1.5) +
  scale_y_continuous(label = dollar) +
  labs(title = "Academic Salary by Rank",
       subtitle = "9-month salary for 2008-2009",
       x = "",
       y = "") +
  theme_minimal() +
  theme(legend.position = "none")
```



The plots are create using the `geom_quasirandom` function. These plots can be easier to read than simple jittered strip plots. To learn more about these plots, see Beeswarm-style plots with ggplot2.

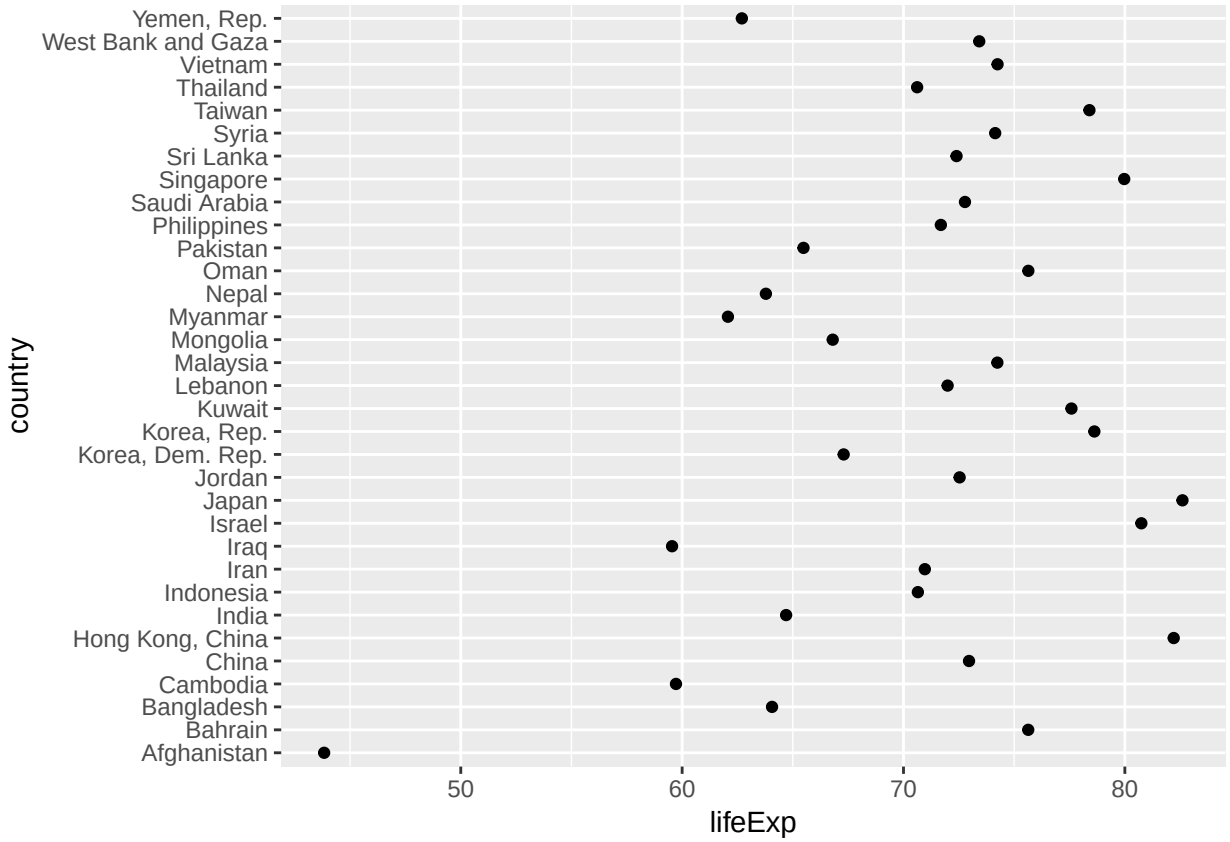


Figure 4.13: Basic Cleveland dot plot

4.3.8 Cleveland Dot Charts

Cleveland plots are useful when you want to compare a numeric statistic for a large number of groups. For example, say that you want to compare the 2007 life expectancy for Asian country using the gapminder dataset.

```
data(gapminder, package="gapminder")

# subset Asian countries in 2007
library(dplyr)
plotdata <- gapminder %>%
  filter(continent == "Asia" &
         year == 2007)

# basic Cleveland plot of life expectancy by country
ggplot(plotdata,
       aes(x= lifeExp, y = country)) +
  geom_point()
```

Comparisons are usually easier if the y-axis is sorted.

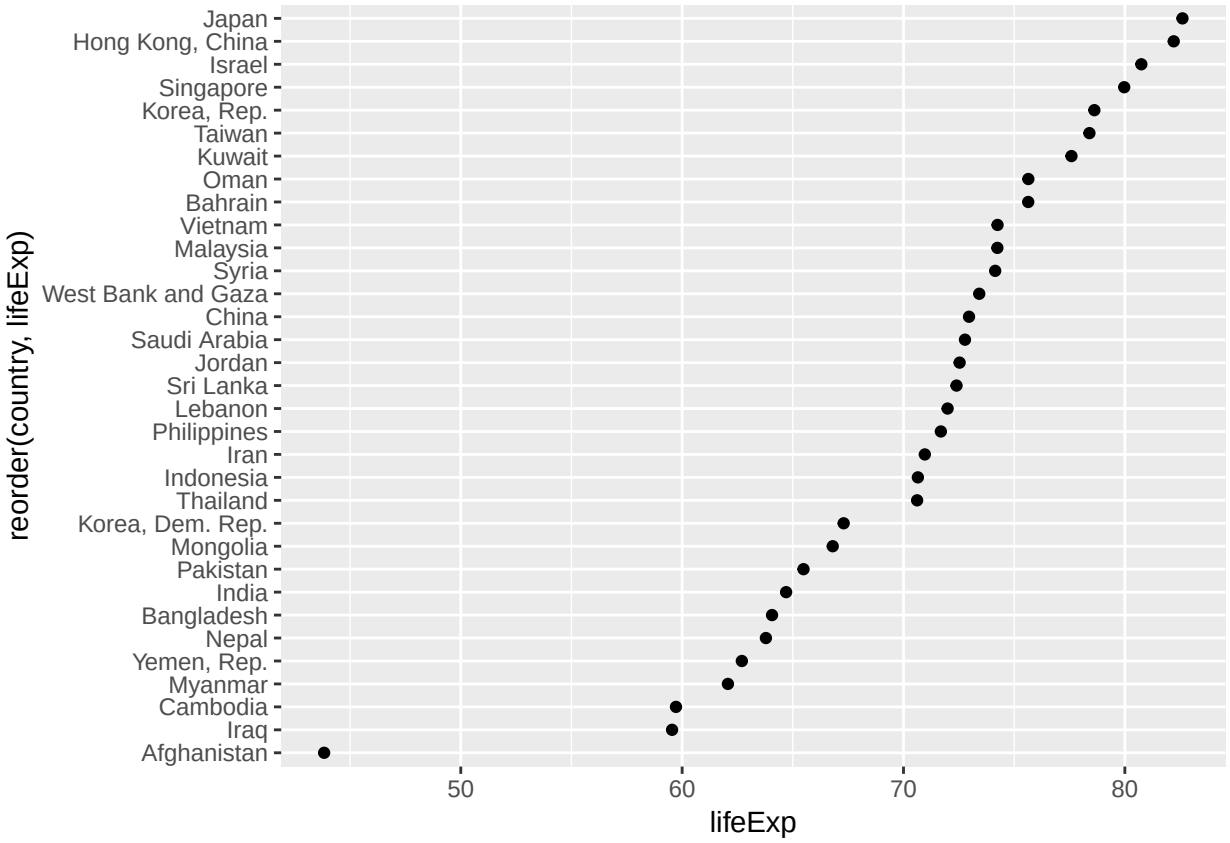


Figure 4.14: Sorted Cleveland dot plot

```
# Sorted Cleveland plot
ggplot(plotdata,
       aes(x=lifeExp,
           y=reorder(country, lifeExp))) +
  geom_point()
```

Finally, we can use options to make the graph more attractive.

```
# Fancy Cleveland plot
ggplot(plotdata,
       aes(x=lifeExp,
           y=reorder(country, lifeExp))) +
  geom_point(color="blue",
             size = 2) +
  geom_segment(aes(x = 40,
                   xend = lifeExp,
                   y = reorder(country, lifeExp),
                   yend = reorder(country, lifeExp)),
                   color = "lightgrey") +
  labs (x = "Life Expectancy (years)",
        y = "",
        title = "Life Expectancy by Country",
        subtitle = "GapMinder data for Asia - 2007") +
```

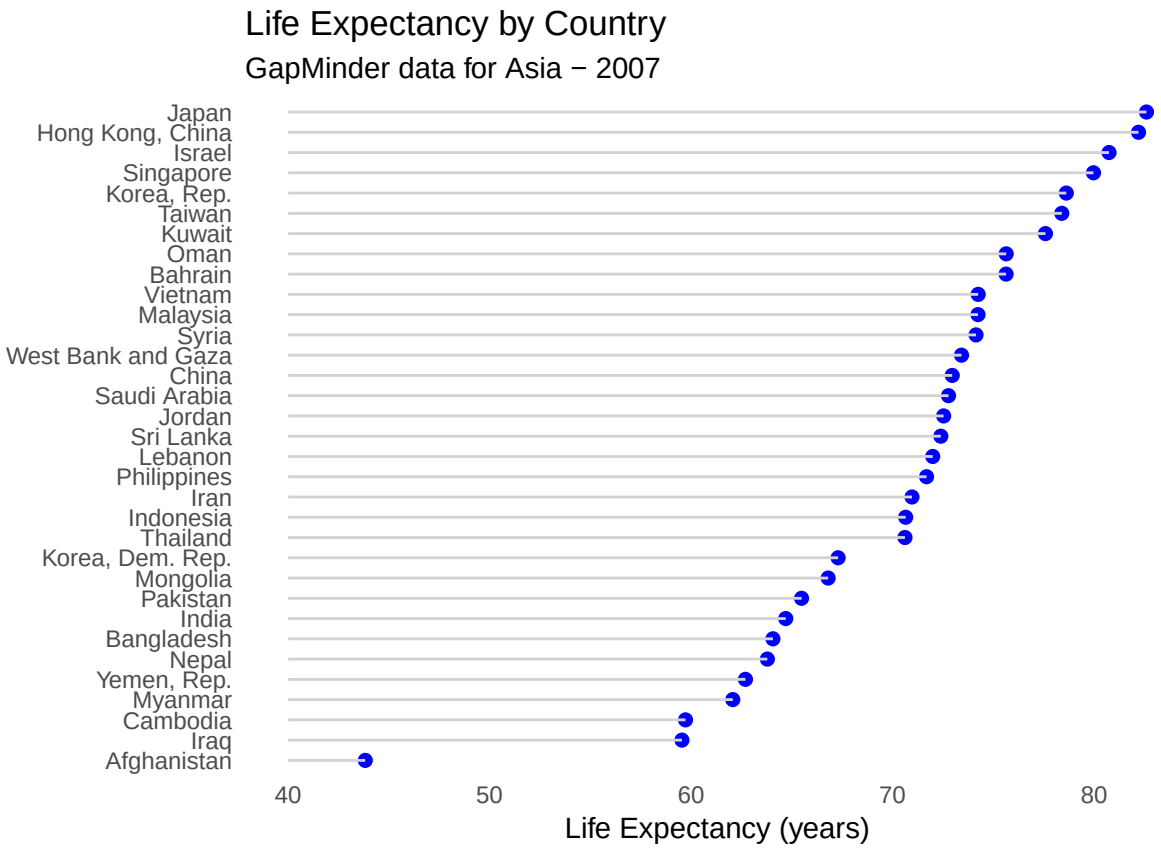


Figure 4.15: Fancy Cleveland plot

```
  theme_minimal() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

Japan clearly has the highest life expectancy, while Afghanistan has the lowest by far. This last plot is also called a lollipop graph (you can see why).

Chapter 5

Multivariate Graphs

Multivariate graphs display the relationships among three or more variables. There are two common methods for accommodating multiple variables: grouping and faceting.

5.1 Grouping

In grouping, the values of the first two variables are mapped to the x and y axes. Then additional variables are mapped to other visual characteristics such as color, shape, size, line type, and transparency. Grouping allows you to plot the data for multiple groups in a single graph.

Using the Salaries dataset, let's display the relationship between *yrs.since.phd* and *salary*.

```
library(ggplot2)
data(Salaries, package="carData")

# plot experience vs. salary
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary)) +
  geom_point() +
  labs(title = "Academic salary by years since degree")
```

Next, let's include the rank of the professor, using color.

```
# plot experience vs. salary (color represents rank)
ggplot(Salaries, aes(x = yrs.since.phd,
                     y = salary,
                     color=rank)) +
  geom_point() +
  labs(title = "Academic salary by rank and years since degree")
```

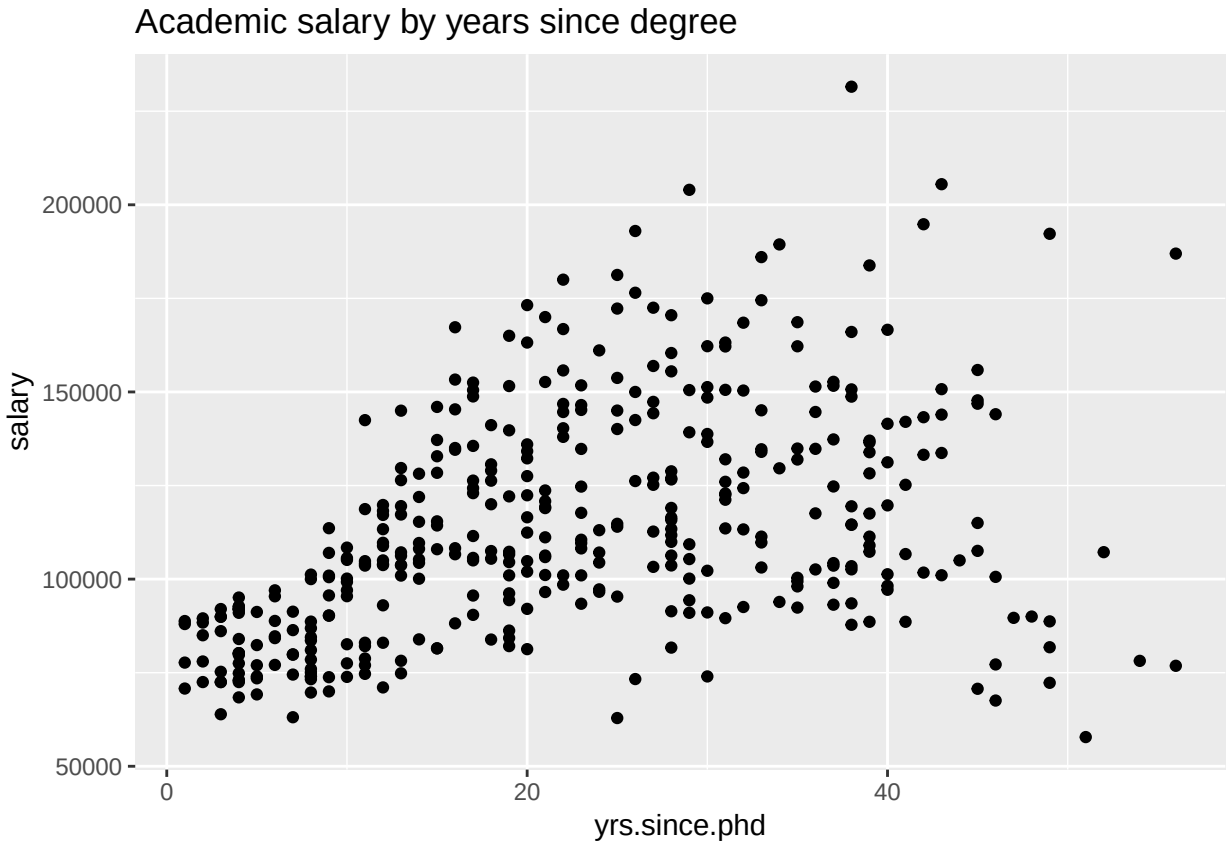
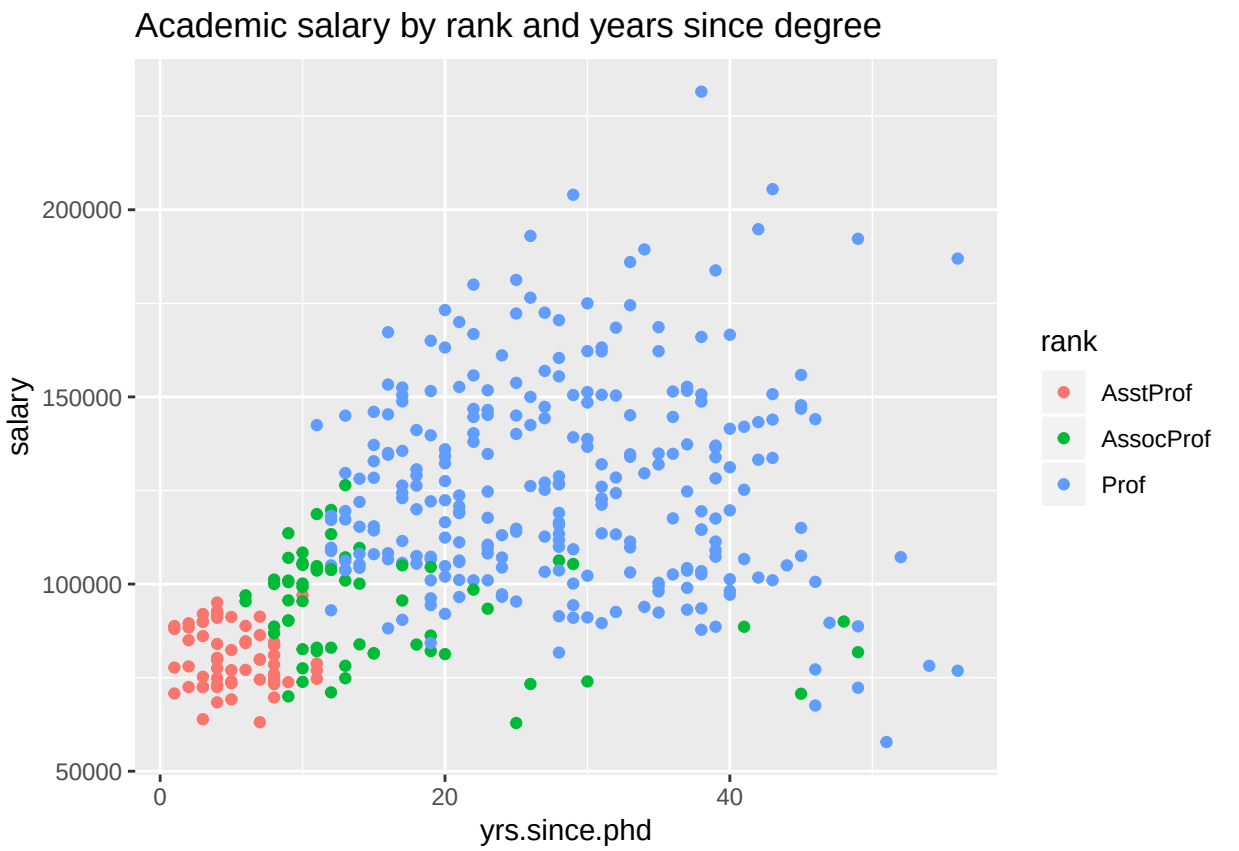
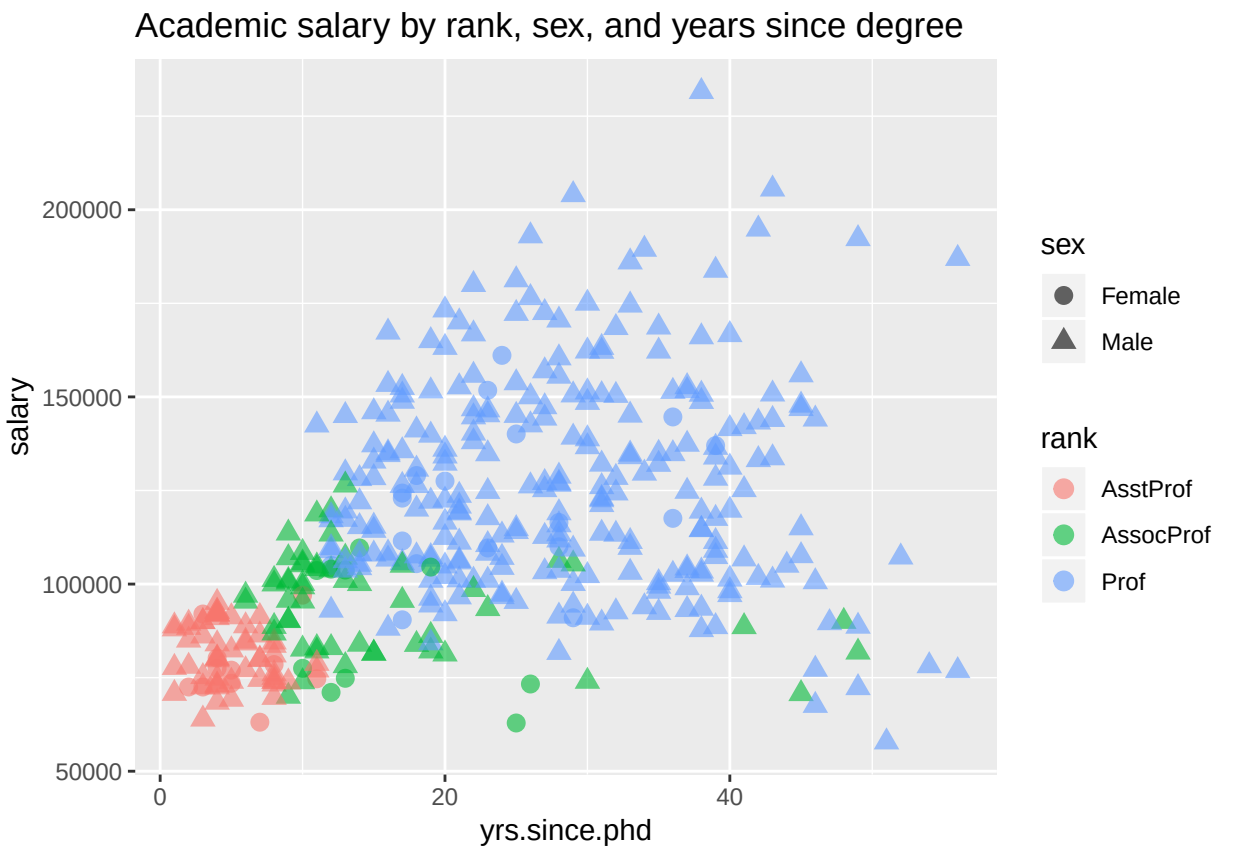


Figure 5.1: Simple scatterplot



Finally, let's add the gender of professor, using the shape of the points to indicate sex. We'll increase the point size and add transparency to make the individual points clearer.

```
# plot experience vs. salary
# (color represents rank, shape represents sex)
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary,
           color = rank,
           shape = sex)) +
  geom_point(size = 3,
             alpha = .6) +
  labs(title = "Academic salary by rank, sex, and years since degree")
```

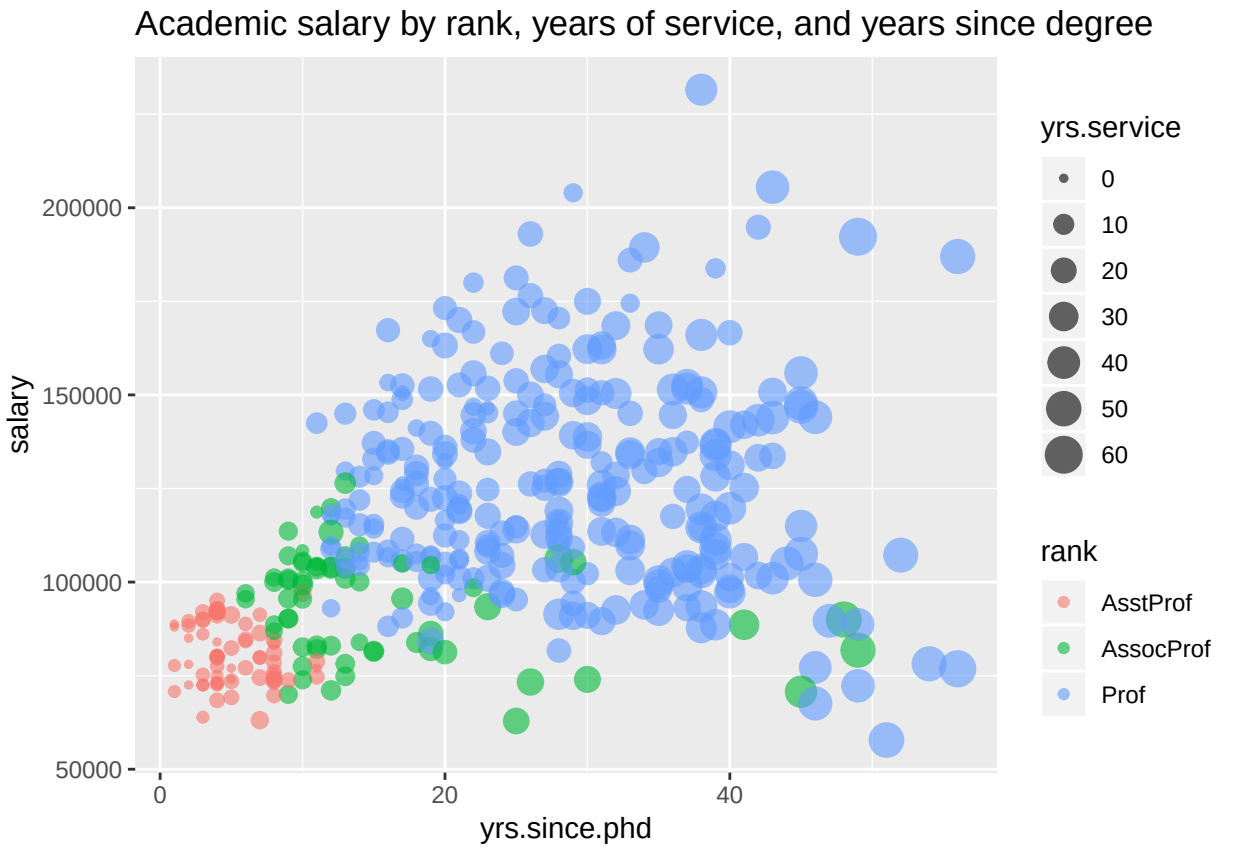


I can't say that this is a great graphic. It is very busy, and it can be difficult to distinguish male from female professors. Faceting (described in the next section) would probably be a better approach.

> Notice the difference between specifying a constant value (such as `size = 3`) and a mapping of a variable to a visual characteristic (e.g., `color = rank`). Mappings are always placed within the `aes` function, while the assignment of a constant value always appear outside of the `aes` function.

Here is a cleaner example. We'll graph the relationship between years since Ph.D. and salary using the size of the points to indicate years of service. This is called a bubble plot.

```
library(ggplot2)
data(Salaries, package="carData")

# plot experience vs. salary
# (color represents rank and size represents service)
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary,
           color = rank,
           size = yrs.service)) +
  geom_point(alpha = .6) +
  labs(title = "Academic salary by rank, years of service, and years since degree")
```



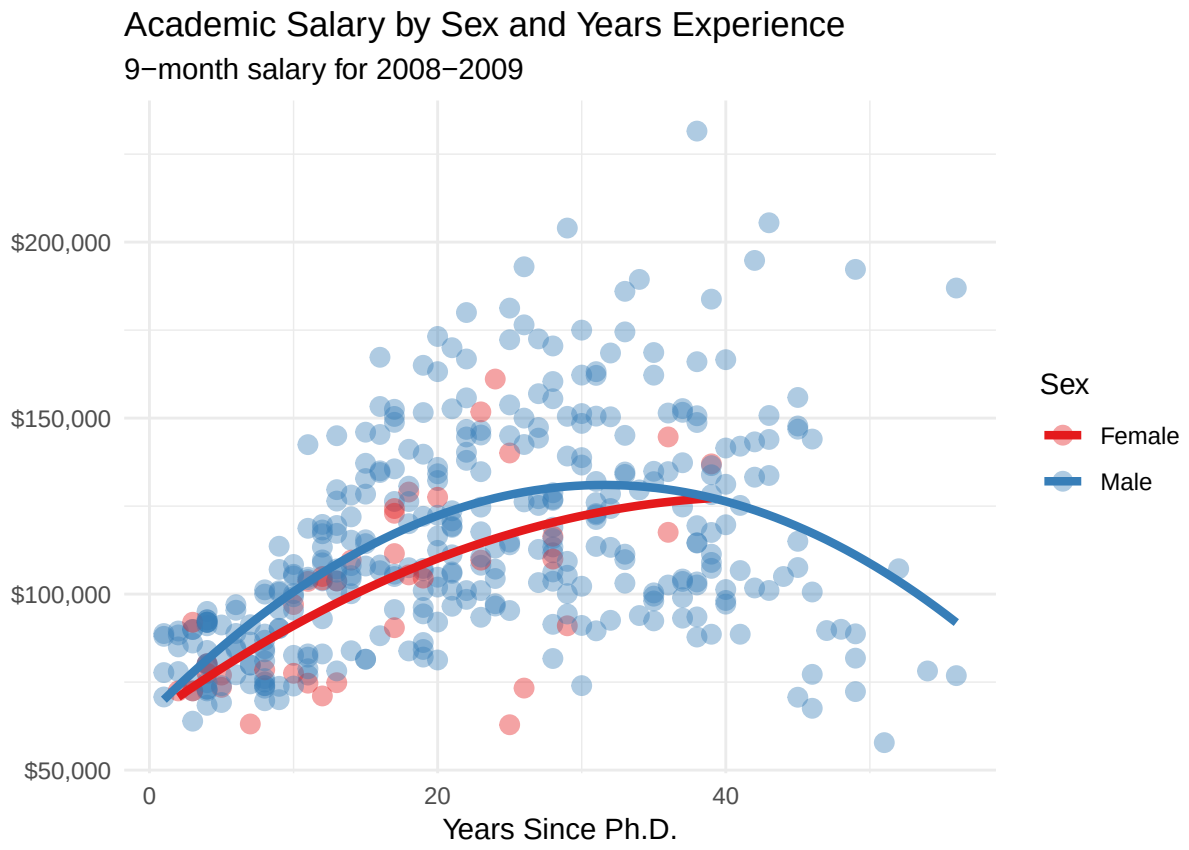
There is obviously a strong positive relationship between years since Ph.D. and year of service. Assistant Professors fall in the 0-11 years since Ph.D. and 0-10 years of service range. Clearly highly experienced professionals don't stay at the Assistant Professor level (they are probably promoted or leave the University). We don't find the same time demarcation between Associate and Full Professors.

Bubble plots are described in more detail in a later chapter.

As a final example, let's look at the *yrs.since.phd* vs *salary* and add *sex* using color and quadratic best fit lines.

```
# plot experience vs. salary with
# fit lines (color represents sex)
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary,
           color = sex)) +
  geom_point(alpha = .4,
             size = 3) +
  geom_smooth(se=FALSE,
              method = "lm",
              formula = y~poly(x,2),
              size = 1.5) +
  labs(x = "Years Since Ph.D.",
       title = "Academic Salary by Sex and Years Experience",
       subtitle = "9-month salary for 2008-2009",
       y = "",
       color = "Sex") +
  scale_y_continuous(label = scales::dollar) +
```

```
  scale_color_brewer(palette = "Set1") +
  theme_minimal()
```

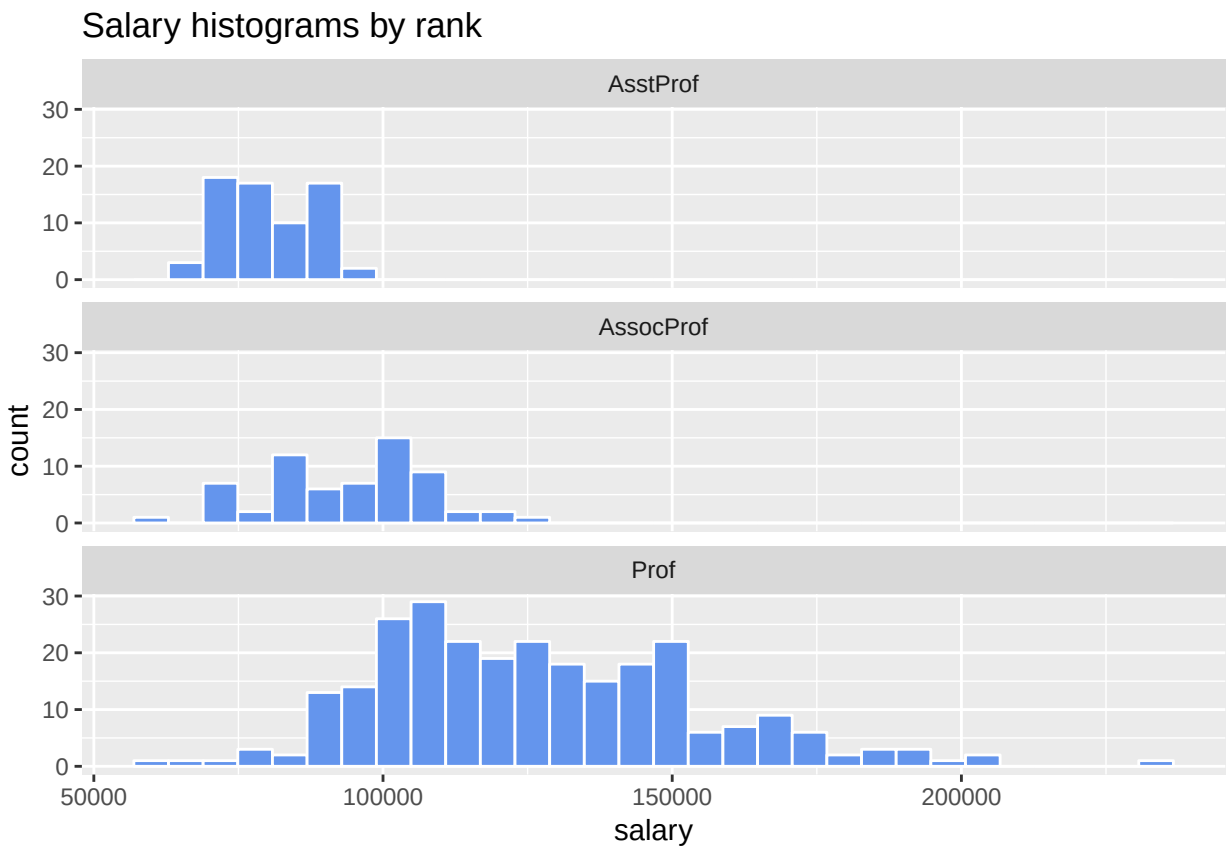


Faceting {#Faceting}

Grouping allows you to plot multiple variables in a single graph, using visual characteristics such as color, shape, and size.

In **faceting**, a graph consists of several separate plots or *small multiples*, one for each level of a third variable, or combination of variables. It is easiest to understand this with an example.

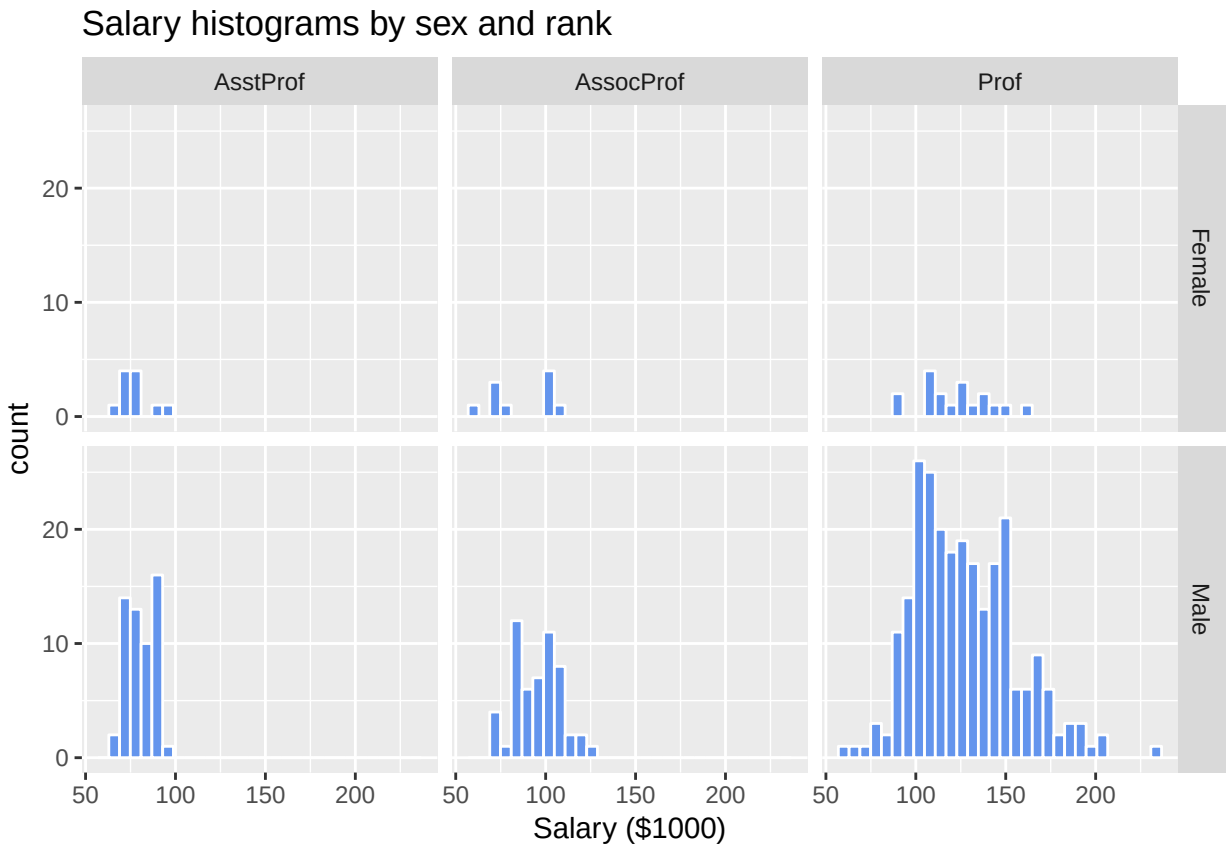
```
# plot salary histograms by rank
ggplot(Salaries, aes(x = salary)) +
  geom_histogram(fill = "cornflowerblue",
                 color = "white") +
  facet_wrap(~rank, ncol = 1) +
  labs(title = "Salary histograms by rank")
```



The `facet_wrap` function creates a separate graph for each level of `rank`. The `ncol` option controls the number of columns.

In the next example, two variables are used to define the facets.

```
# plot salary histograms by rank and sex
ggplot(Salaries, aes(x = salary / 1000)) +
  geom_histogram(color = "white",
                 fill = "cornflowerblue") +
  facet_grid(sex ~ rank) +
  labs(title = "Salary histograms by sex and rank",
       x = "Salary ($1000)")
```



The format of the `facet_grid` function is

```
facet_grid( row variable(s) ~ column variable(s))
```

Here, the function assigns *sex* to the rows and *rank* to the columns, creating a matrix of 6 plots in one graph.

We can also combine grouping and faceting. Let's use Mean/SE plots and faceting to compare the salaries of male and female professors, within rank and discipline. We'll use color to distinguish sex and faceting to create plots for rank by discipline combinations.

```
# calculate means and standard erroes by sex,
# rank and discipline

library(dplyr)
plotdata <- Salaries %>%
  group_by(sex, rank, discipline) %>%
  summarize(n = n(),
            mean = mean(salary),
            sd = sd(salary),
            se = sd / sqrt(n))

# create better labels for discipline
plotdata$discipline <- factor(plotdata$discipline,
                              labels = c("Theoretical",
                                         "Applied"))
# create plot
ggplot(plotdata,
       aes(x = sex,
```

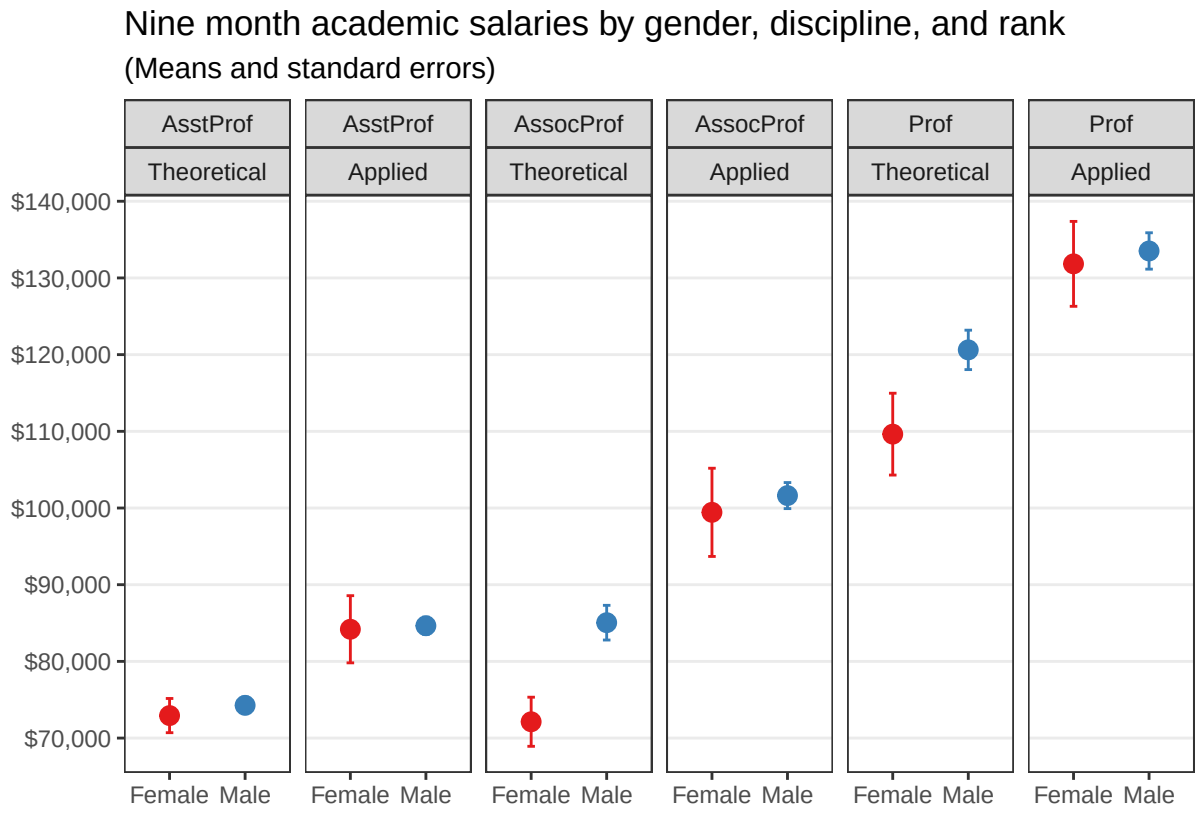


Figure 5.2: Salary by sex, rank, and discipline

```
          y = mean,
          color = sex)) +
geom_point(size = 3) +
geom_errorbar(aes(ymin = mean - se,
                  ymax = mean + se),
              width = .1) +
scale_y_continuous(breaks = seq(70000, 140000, 10000),
                   label = scales::dollar) +
facet_grid(. ~ rank + discipline) +
theme_bw() +
theme(legend.position = "none",
      panel.grid.major.x = element_blank(),
      panel.grid.minor.y = element_blank()) +
labs(x="",
     y="",
     title="Nine month academic salaries by gender, discipline, and rank",
     subtitle = "(Means and standard errors)") +
scale_color_brewer(palette="Set1")
```

The statement `facet_grid(. ~ rank + discipline)` specifies no row variable (.) and columns defined by the combination of *rank* and *discipline*.

The `theme_` functions create create a black and white theme and eliminates vertical grid lines and minor

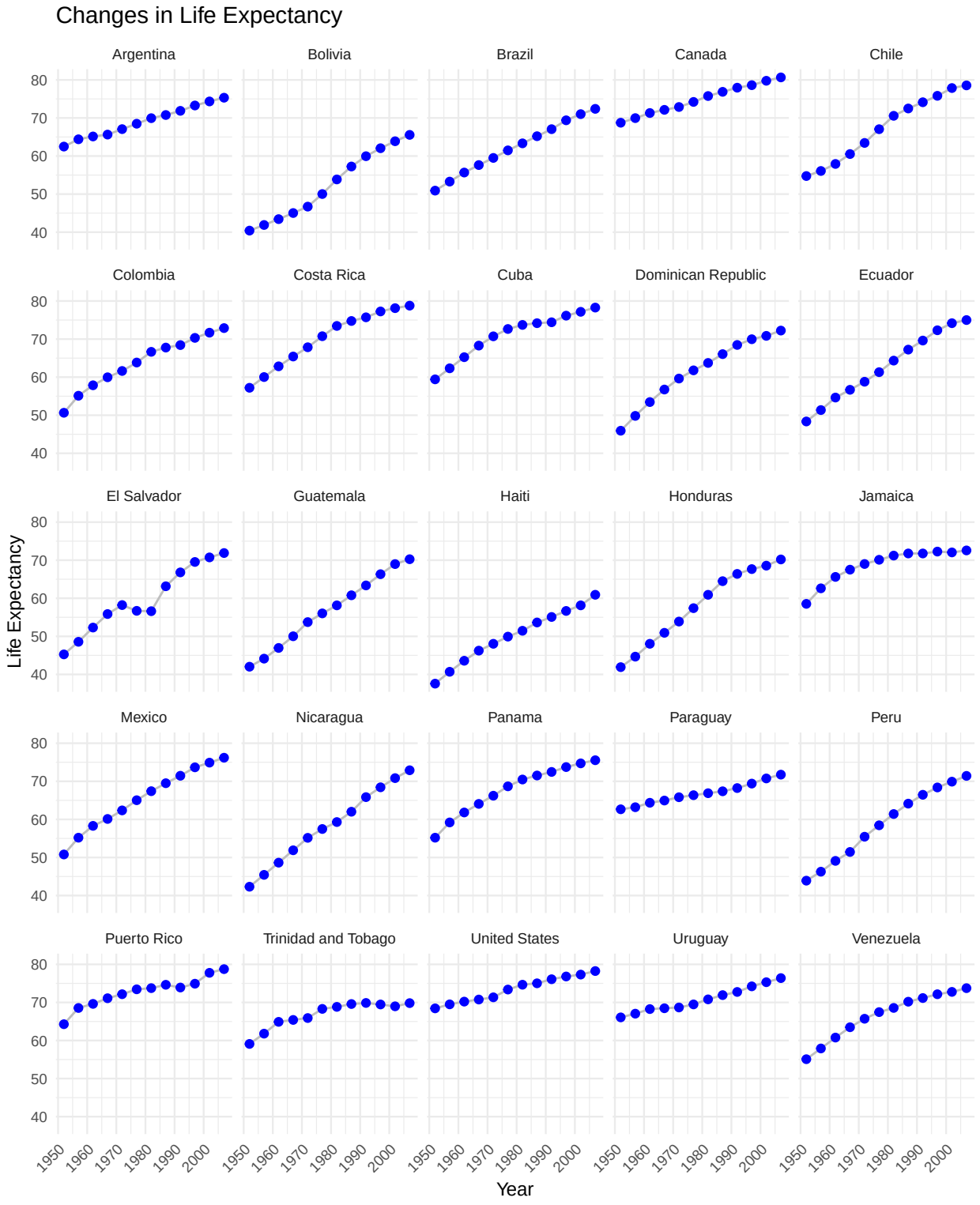
horizontal grid lines. The `scale_color_brewer` function changes the color scheme for the points and error bars.

At first glance, it appears that there might be gender differences in salaries for associate and full professors in theoretical fields. I say "might" because we haven't done any formal hypothesis testing yet (ANCOVA in this case).

See the Customizing section to learn more about customizing the appearance of a graph.

As a final example, we'll shift to a new dataset and plot the change in life expectancy over time for countries in the "Americas". The data comes from the gapminder dataset in the `gapminder` package. Each country appears in its own facet. The **theme** functions are used to simplify the background color, rotate the x-axis text, and make the font size smaller.

```
# plot life expectancy by year separately
# for each country in the Americas
data(gapminder, package = "gapminder")

# Select the Americas data
plotdata <- dplyr::filter(gapminder,
                          continent == "Americas")

# plot life expectancy by year, for each country
ggplot(plotdata, aes(x=year, y = lifeExp)) +
  geom_line(color="grey") +
  geom_point(color="blue") +
  facet_wrap(~country) +
  theme_minimal(base_size = 9) +
  theme(axis.text.x = element_text(angle = 45,
                                   hjust = 1)) +
  labs(title = "Changes in Life Expectancy",
       x = "Year",
       y = "Life Expectancy")
```



We can see that life expectancy is increasing in each country, but that Haiti is lagging behind.

Chapter 6

Maps

R provides a myriad of methods for creating both static and interactive maps containing statistical information. This section focuses on the use of `ggmap` and `choroplethr`.

6.1 Dot density maps

Dot density maps use points on a map to explore spatial relationships.

The Houston crime dataset contains the date, time, and address of six types of criminal offenses reported between January and August 2010. The longitude and latitude of each offence was added using `geocode` function, which takes an address and returns coordinates using the Google Maps API.

We'll use this dataset to plot the locations of rape reports.

```
library(ggmap)

# subset the data
library(dplyr)
rapes <- filter(crime, offense == "rape") %>%
  select(date, offense, address, lon, lat)

# view data
head(rapes)
```

```
##       date offense            address       lon      lat
## 1 1/1/2010    rape   5950 glenmont dr -95.48498 29.72007
## 2 1/1/2010    rape    2350 sperber ln -95.34817 29.75505
## 3 1/1/2010    rape   5850 mackinaw rd -95.47353 29.60021
## 4 1/1/2010    rape 5850 southwest fwy -95.48174 29.72603
## 5 1/2/2010    rape  7550 corporate dr -95.55224 29.69836
## 6 1/2/2010    rape   1150 fidelity st -95.25535 29.74147
```

Let's set up the map.

(1) Find the center coordinates for Houston, TX

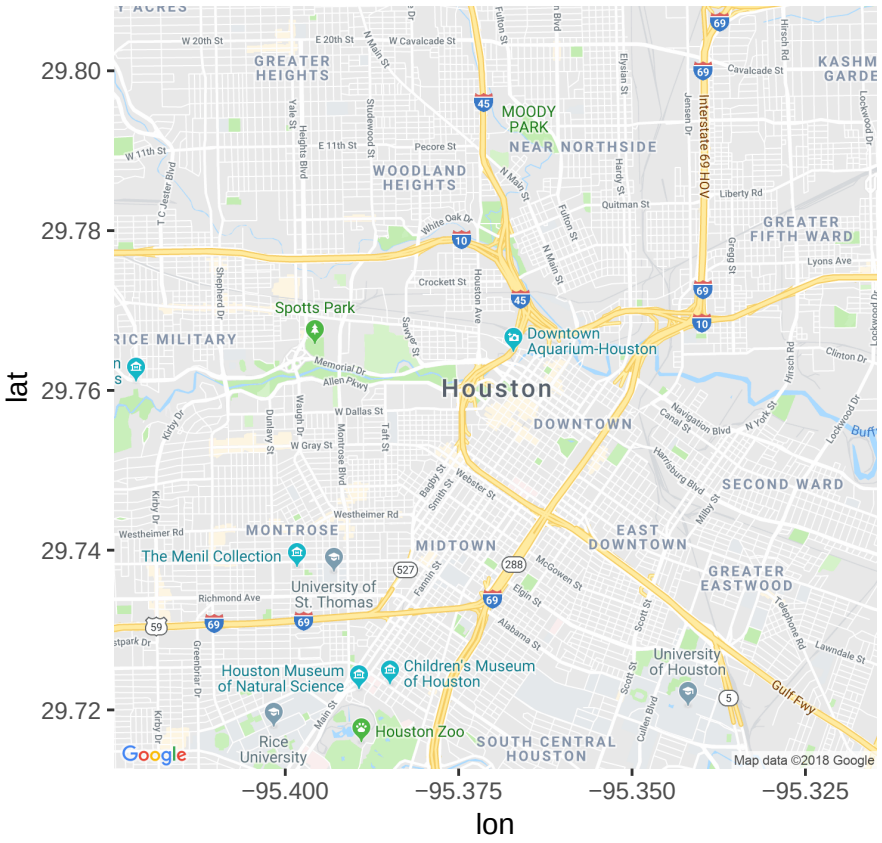


Figure 6.1: Houston map

```
# using geocode function returns
# lon=-95.3698, lat=29.76043
houston_center <- geocode("Houston, TX")
```

(2) Get the background map image.

- Specify a zoom factor from 3 (continent) to 21 (building). The default is 10 (city).
- Specify a map type. Types include terrain, terrain-background, satellite, roadmap, hybrid, watercolor, and toner.

```
# get Houston map
houston_map <- get_map(houston_center,
                       zoom = 13,
                       maptype = "roadmap")
ggmap(houston_map)
```

(3) Add crime locations to the map.

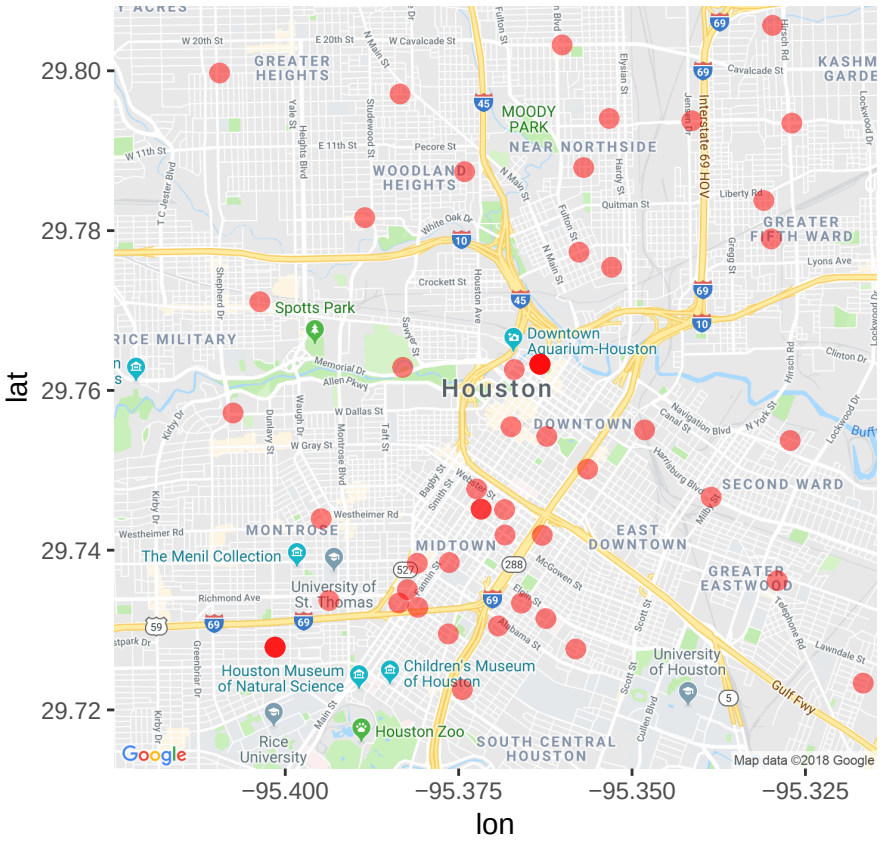
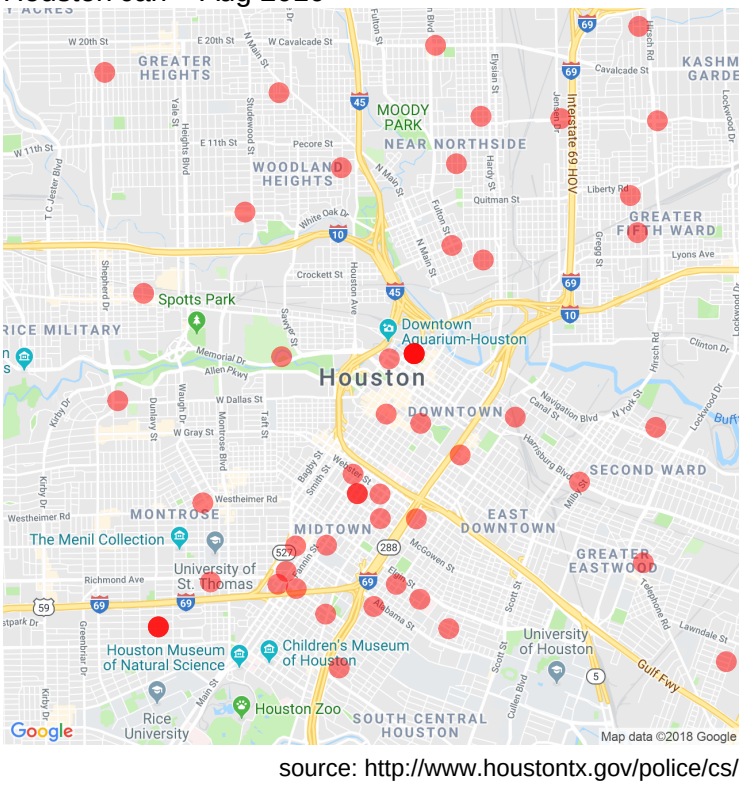


Figure 6.2: Crime locations

```
# add incident locations
ggmap(houston_map,
      base_layer = ggplot(data = rapes,
                          aes(x=lon, y = lat))) +
  geom_point(color = "red",
             size = 3,
             alpha = 0.5)
```

(4) Clean up the plot and add labels.

```
# remove long and lat numbers and add titles
ggmap(houston_map,
      base_layer = ggplot(aes(x=lon, y = lat),
                          data = rapes)) +
  geom_point(color = "red",
             size = 3,
             alpha = 0.5) +
  theme_void() +
  labs(title = "Location of reported rapes",
       subtitle = "Houston Jan - Aug 2010",
       caption = "source: http://www.houstontx.gov/police/cs/")
```



source: http://www.houstontx.gov/police/cs/

Figure 6.3: Crime locations with titles, and without longitude and latitude

There seems to be a concentration of rape reports in midtown.

To learn more about ggmap, see ggmap: Spatial Visualization with ggplot2.

6.2 Choropleth maps

Choropleth maps use color or shading on predefined areas to indicate average values of a numeric variable in that area. In this section we'll use the `choroplethr` package to create maps that display information by country, US state, and US county.

6.2.1 Data by country

Let's create a world map and color the countries by life expectancy using the 2007 gapminder data.

The `choroplethr` package has numerous functions that simplify the task of creating a choropleth map. To plot the life expectancy data, we'll use the `country_choropleth` function.

The function requires that the data frame to be plotted has a column named *region* and a column named *value*. Additionally, the entries in the *region* column must exactly match how the entries are named in the *region* column of the dataset `country.map` from the `choroplethrMaps` package.

```
# view the first 12 region names in country.map
data(country.map, package = "choroplethrMaps")
head(unique(country.map$region), 12)
```

```
##  [1] "afghanistan" "angola"      "azerbaijan"  "moldova"     "madagascar"
##  [6] "mexico"      "macedonia"   "mali"        "myanmar"     "montenegro"
## [11] "mongolia"    "mozambique"
```

Note that the region entries are all lower case.

To continue, we need to make some edits to our `gapminder` dataset. Specifically, we need to

1. select the 2007 data
2. rename the *country* variable to *region*
3. rename the *lifeExp* variable to *value*
4. recode *region* values to lower case
5. recode some *region* values to match the region values in the `country.map` data frame. The `recode` function in the `dplyr` package take the form `recode(variable, oldvalue1 = newvalue1, oldvalue2 = newvalue2, ...)`

```
# prepare dataset
data(gapminder, package = "gapminder")
plotdata <- gapminder %>%
  filter(year == 2007) %>%
  rename(region = country,
         value = lifeExp) %>%
  mutate(region = tolower(region)) %>%
  mutate(region = recode(region,
```

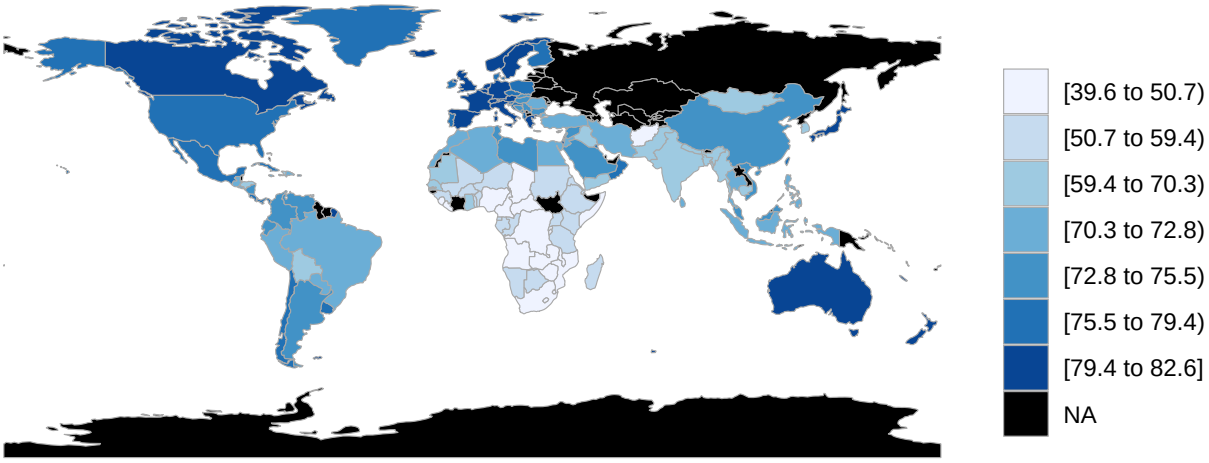


Figure 6.4: Choropleth map of life expectancy

```
                          "united states"    = "united states of america",
                          "congo, dem. rep." = "democratic republic of the congo",
                          "congo, rep."      = "republic of congo",
                          "korea, dem. rep." = "south korea",
                          "korea. rep."      = "north korea",
                          "tanzania"         = "united republic of tanzania",
                          "serbia"           = "republic of serbia",
                          "slovak republic"  = "slovakia",
                          "yemen, rep."      = "yemen"))
```

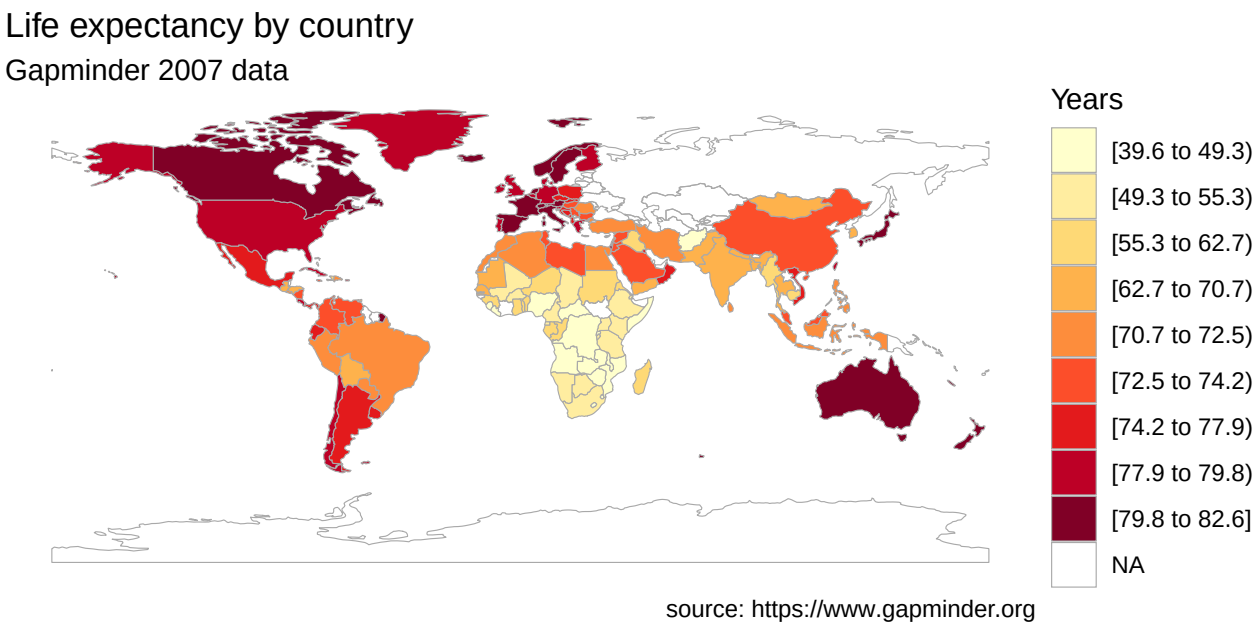
Now lets create the map.

```
library(choroplethr)
country_choropleth(plotdata)
```

`choroplethr` functions return `ggplot2` graphs. Let's make it a bit more attractive by modifying the code with additional `ggplot2` functions.

```
country_choropleth(plotdata,
                   num_colors=9) +
  scale_fill_brewer(palette="YlOrRd") +
  labs(title = "Life expectancy by country",
```

```
       subtitle = "Gapminder 2007 data",
       caption = "source: https://www.gapminder.org",
       fill = "Years")
```



Data by US state

For US data, the `choroplethr` package provides functions for creating maps by county, state, zip code, and census tract. Additionally, map regions can be labeled.

Let's plot US states by Mexcian American popultion, using the 2010 Census.

To plot the population data, we'll use the `state_choropleth` function. The function requires that the data frame to be plotted has a column named *region* to represent state, and a column named *value* (the quantity to be plotted). Additionally, the entries in the *region* column must exactly match how the entries are named in the *region* column of the dataset `state.map` from the `choroplethrMaps` package.

The `zoom = continental_us_states` option will create a map that excludes Hawaii and Alaska.

```
library(ggplot2)
library(choroplethr)
data(continental_us_states)

# input the data
library(readr)
mex_am <- read_tsv("mexican_american.csv")

# prepare the data
mex_am$region <- tolower(mex_am$state)
```

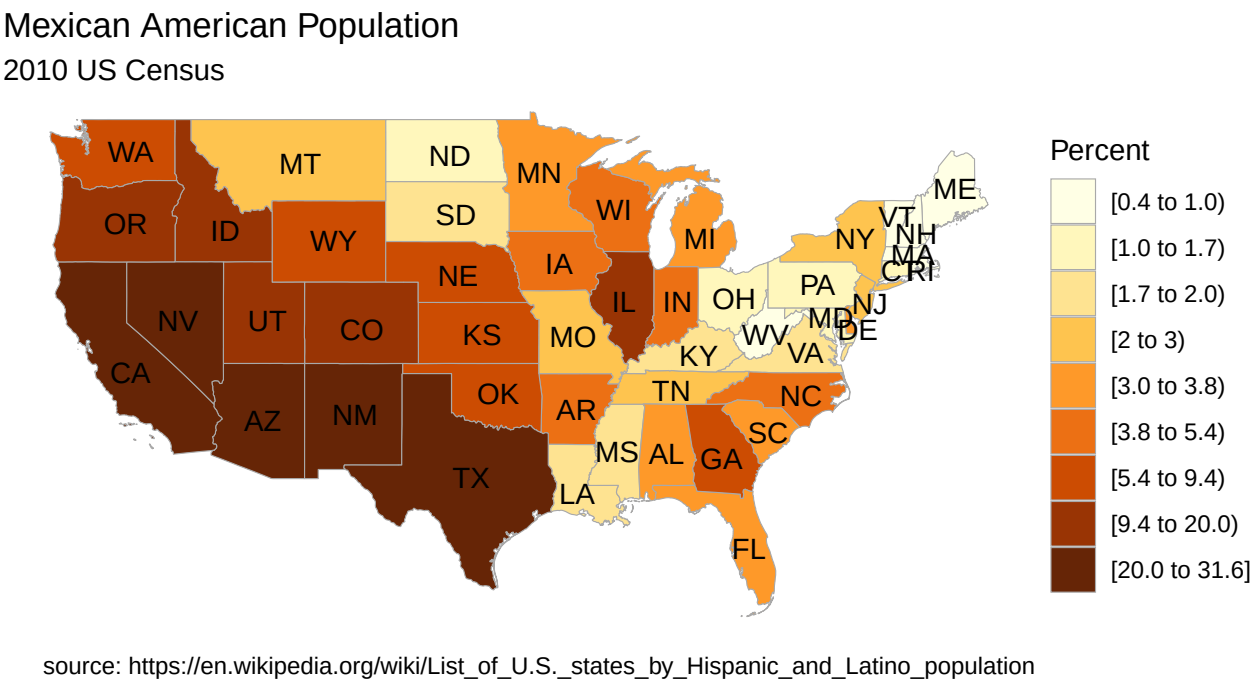


Figure 6.5: Choropleth map of US States

```
mex_am$value <- mex_am$percent

# create the map
state_choropleth(mex_am,
                 num_colors=9,
                 zoom = continental_us_states) +
  scale_fill_brewer(palette="YlOrBr") +
  labs(title = "Mexican American Population",
       subtitle = "2010 US Census",
       caption = "source: https://en.wikipedia.org/wiki/List_of_U.S._states_by_Hispanic_and_Latino_popul
       fill = "Percent")
```

6.2.2 Data by US county

Finally, let's plot data by US counties. We'll plot the violent crime rate per 1000 individuals for Connecticut counties in 2012. Data come from the FBI Uniform Crime Statistics.

We'll use the `county_choropleth` function. Again, the function requires that the data frame to be plotted has a column named *region* and a column named *value*.

Additionally, the entries in the *region* column must be numeric codes and exactly match how the entries are given in the *region* column of the dataset `county.map` from the `choroplethrMaps` package.

Our dataset has country names (e.g. fairfield). However, we need region codes (e.g., 9001). We can use the `county.regions` dataset to lookup the region code for each county name.

Additionally, we'll use the option `reference_map = TRUE` to add a reference map from Google Maps.

```
library(ggplot2)
library(choroplethr)
library(dplyr)

# enter violent crime rates by county
crimes_ct <- data.frame(
  county = c("fairfield", "hartford",
             "litchfield", "middlesex",
             "new haven", "new london",
             "tolland", "windham"),
  value = c(3.00, 3.32,
            1.02, 1.24,
            4.13, 4.61,
            0.16, 1.60)
)

crimes_ct
```

```
##       county value
## 1  fairfield  3.00
## 2   hartford  3.32
## 3 litchfield  1.02
## 4  middlesex  1.24
## 5  new haven  4.13
## 6 new london  4.61
## 7    tolland  0.16
## 8    windham  1.60
```

```
# obtain region codes for connecticut
data(county.regions,
     package = "choroplethrMaps")
region <- county.regions %>%
  filter(state.name == "connecticut")

region
```

```
##   region county.fips.character county.name  state.name
## 1   9001                 09001   fairfield connecticut
## 2   9003                 09003    hartford connecticut
## 3   9005                 09005  litchfield connecticut
## 4   9007                 09007   middlesex connecticut
## 5   9009                 09009   new haven connecticut
## 6   9011                 09011  new london connecticut
## 7   9013                 09013     tolland connecticut
## 8   9015                 09015     windham connecticut
##   state.fips.character state.abb
## 1                   09        CT
## 2                   09        CT
```

```
## 3                    09        CT
## 4                    09        CT
## 5                    09        CT
## 6                    09        CT
## 7                    09        CT
## 8                    09        CT
```

```
# join crime data to region code data
plotdata <- inner_join(crimes_ct,
                       region,
                       by=c("county" = "county.name"))
plotdata
```

```
##       county value region county.fips.character  state.name
## 1  fairfield  3.00   9001                 09001 connecticut
## 2   hartford  3.32   9003                 09003 connecticut
## 3 litchfield  1.02   9005                 09005 connecticut
## 4  middlesex  1.24   9007                 09007 connecticut
## 5  new haven  4.13   9009                 09009 connecticut
## 6 new london  4.61   9011                 09011 connecticut
## 7    tolland  0.16   9013                 09013 connecticut
## 8    windham  1.60   9015                 09015 connecticut
##   state.fips.character state.abb
## 1                    09        CT
## 2                    09        CT
## 3                    09        CT
## 4                    09        CT
## 5                    09        CT
## 6                    09        CT
## 7                    09        CT
## 8                    09        CT
```

```
# create choropleth map
county_choropleth(plotdata,
                  state_zoom = "connecticut",
                  reference_map = TRUE,
                  num_colors = 8) +
  scale_fill_brewer(palette="YlOrRd") +
  labs(title = "Connecticut Violent Crime Rates",
       subtitle = "FBI 2012 data",
       caption = "source: https://ucr.fbi.gov",
       fill = "Violent Crime\n Rate Per 1000")
```

See the choroplethr help for more details.

> R provides *many* ways to create chropleth maps. The `choroplethr` package is just one route. The `tmap` package provides another. A google search is sure to find others.

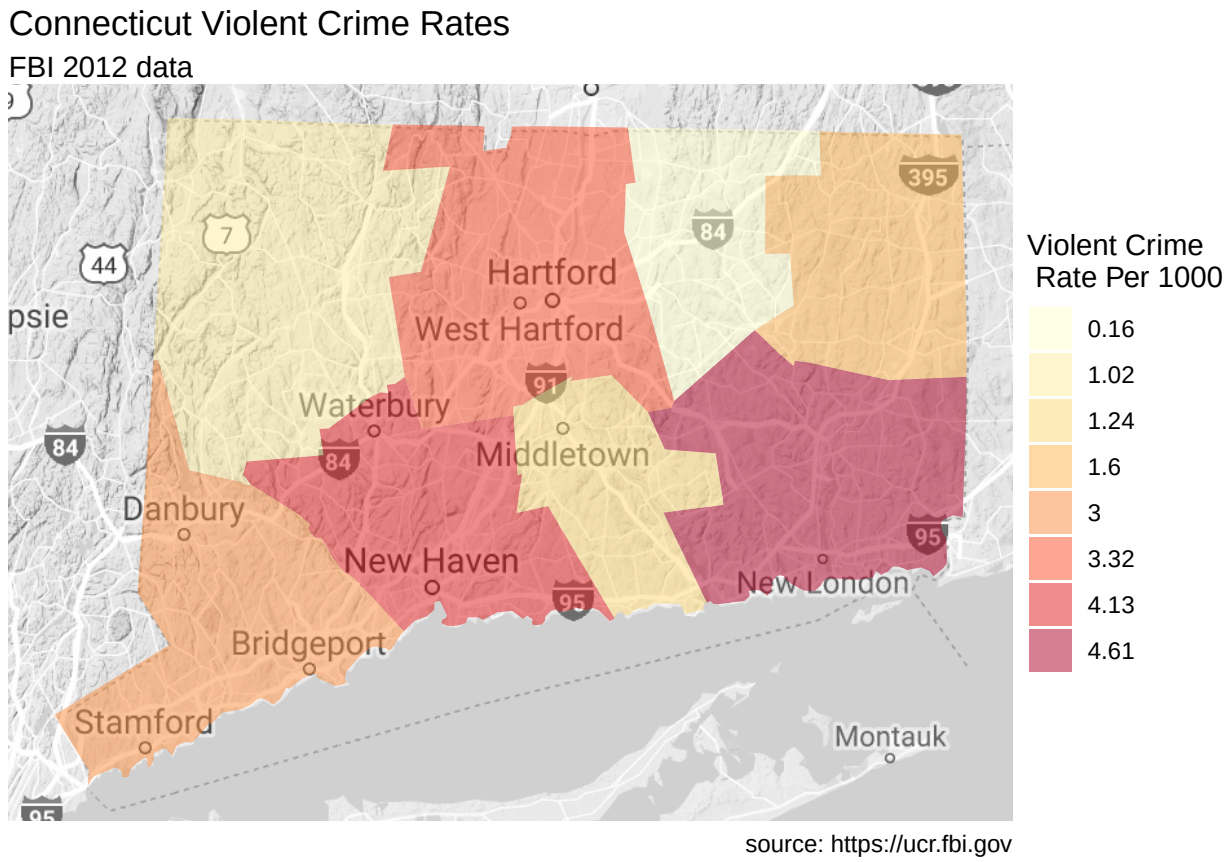


Figure 6.6: Choropleth map of violent crimes by Connecticut counties

Chapter 7

Time-dependent graphs

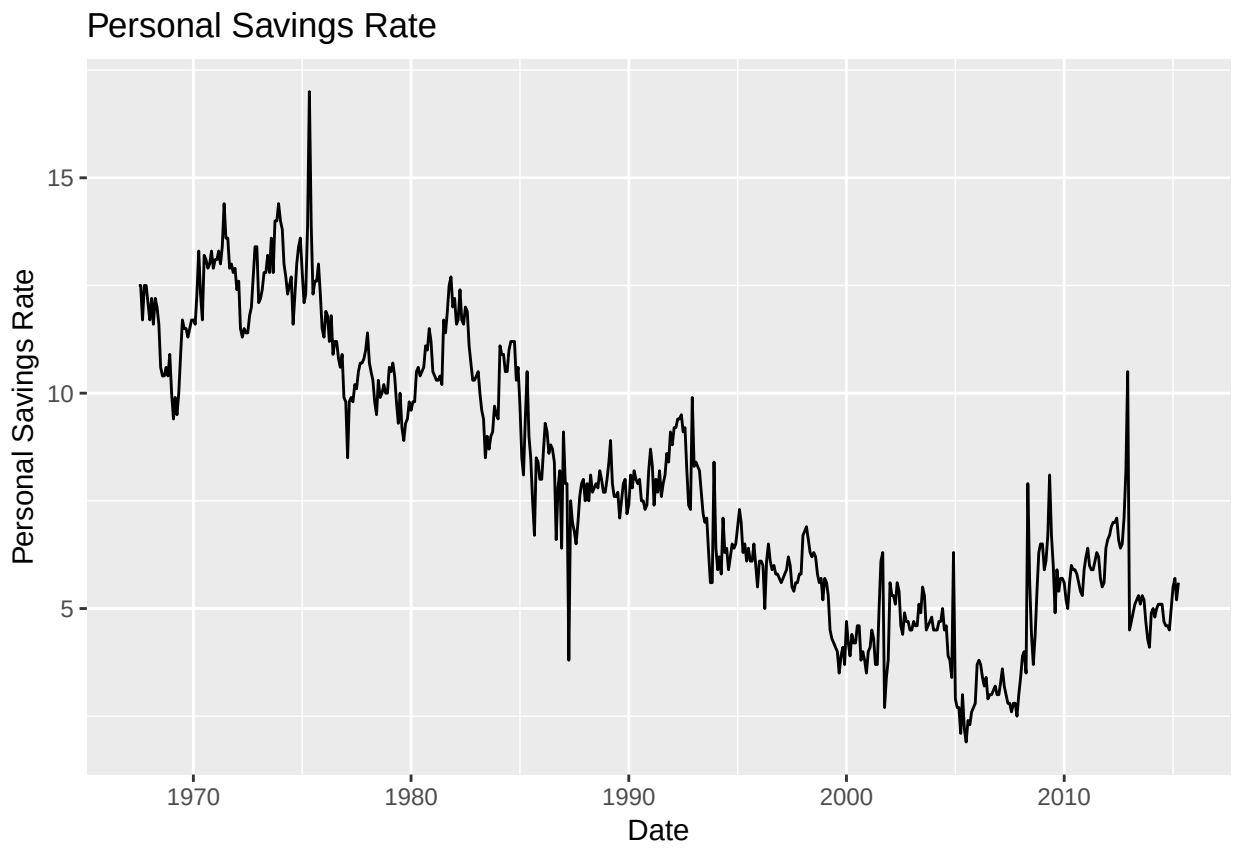
A graph can be a powerful vehicle for displaying change over time. The most common time-dependent graph is the time series line graph. Other options include the dumbbell charts and the slope graph.

7.1 Time series

A time series is a set of quantitative values obtained at successive time points. The intervals between time points (e.g., hours, days, weeks, months, or years) are usually equal.

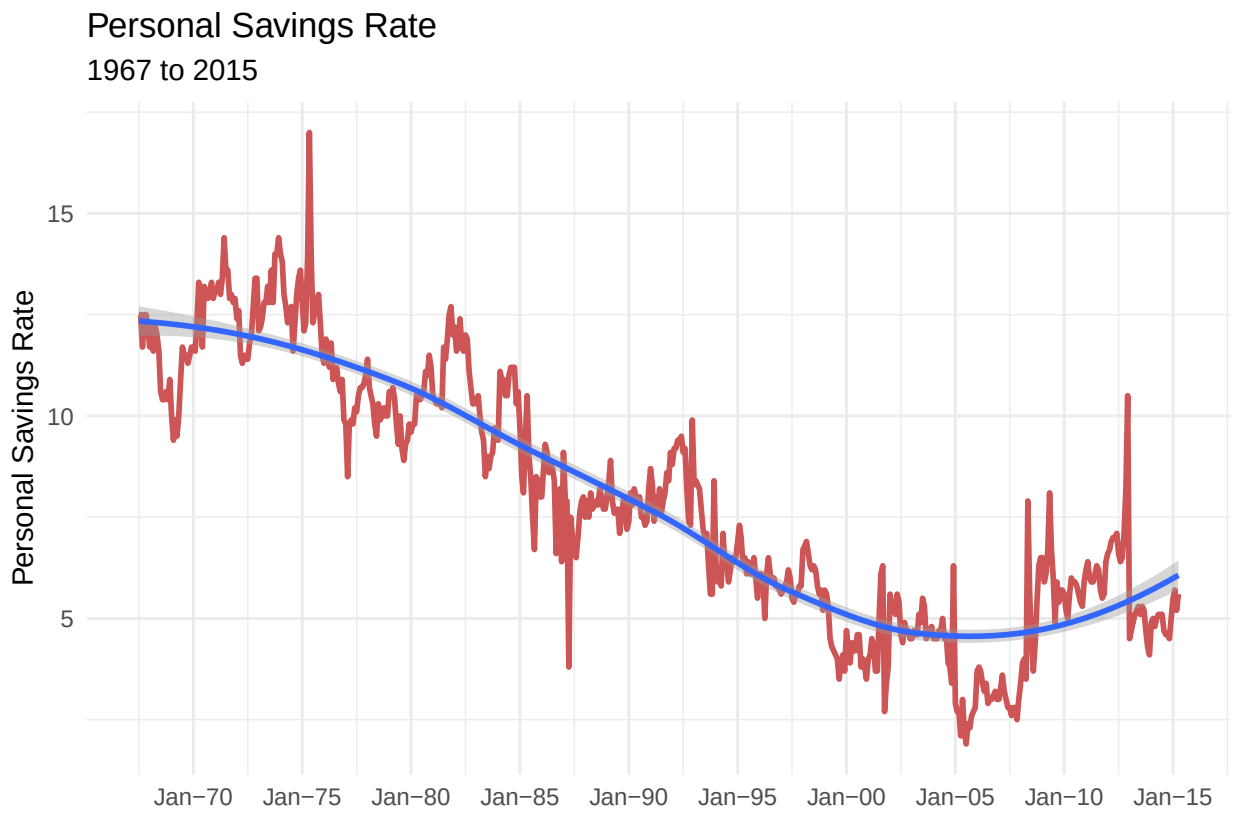
Consider the Economics time series that come with the `ggplot2` package. It contains US monthly economic data collected from January 1967 thru January 2015. Let's plot personal savings rate (*psavert*). We can do this with a simple line plot.

```
library(ggplot2)
ggplot(economics, aes(x = date, y = psavert)) +
  geom_line() +
  labs(title = "Personal Savings Rate",
       x = "Date",
       y = "Personal Savings Rate")
```



The `scale_x_date` function can be used to reformat dates. In the graph below, tick marks appear every 5 years and dates are presented in MMM-YY format. Additionally, the time series line is given an off-red color and made thicker, a trend line (loess) and titles are added, and the theme is simplified.

```
library(ggplot2)
library(scales)
ggplot(economics, aes(x = date, y = psavert)) +
  geom_line(color = "indianred3",
            size=1 ) +
  geom_smooth() +
  scale_x_date(date_breaks = '5 years',
               labels = date_format("%b-%y")) +
  labs(title = "Personal Savings Rate",
       subtitle = "1967 to 2015",
       x = "",
       y = "Personal Savings Rate") +
  theme_minimal()
```



When plotting time series, be sure that the date variable is class `date` and not class `character`. See date values for more details.

Let's close this section with a multivariate time series (more than one series). We'll compare closing prices for Apple and Facebook from Jan 1, 2018 to July 31, 2018.

```
# multivariate time series

# one time install
# install.packages("quantmod")

library(quantmod)
library(dplyr)

# get apple (AAPL) closing prices
apple <- getSymbols("AAPL",
                    return.class = "data.frame",
                    from="2018-01-01")

apple <- AAPL %>%
  mutate(Date = as.Date(row.names(.))) %>%
  select(Date, AAPL.Close) %>%
  rename(Close = AAPL.Close) %>%
  mutate(Company = "Apple")

# get facebook (FB) closing prices
facebook <- getSymbols("FB",
```

```
                        return.class = "data.frame",
                       from="2018-01-01")

facebook <- FB %>%
  mutate(Date = as.Date(row.names(.))) %>%
  select(Date, FB.Close) %>%
  rename(Close = FB.Close) %>%
  mutate(Company = "Facebook")

# combine data for both companies
mseries <- rbind(apple, facebook)

# plot data
library(ggplot2)
ggplot(mseries,
       aes(x=Date, y= Close, color=Company)) +
  geom_line(size=1) +
  scale_x_date(date_breaks = '1 month',
               labels = scales::date_format("%b")) +
  scale_y_continuous(limits = c(150, 220),
                     breaks = seq(150, 220, 10),
                     labels = scales::dollar) +
  labs(title = "NASDAQ Closing Prices",
       subtitle = "Jan - Aug 2018",
       caption = "source: Yahoo Finance",
       y = "Closing Price") +
  theme_minimal() +
  scale_color_brewer(palette = "Dark2")
```

You can see the huge hit that Facebook took at the end of July.

7.2 Dummbbell charts

Dumbbell charts are useful for displaying change between two time points for several groups or observations. The `geom_dumbbell` function from the `ggalt` package is used.

Using the gapminder dataset let's plot the change in life expectancy from 1952 to 2007 in the Americas. The dataset is in long format. We will need to convert it to wide format in order to create the dumbbell plot

```
library(ggalt)
library(tidyr)
library(dplyr)

# load data
data(gapminder, package = "gapminder")

# subset data
plotdata_long <- filter(gapminder,
                        continent == "Americas" &
                        year %in% c(1952, 2007)) %>%
  select(country, year, lifeExp)
```

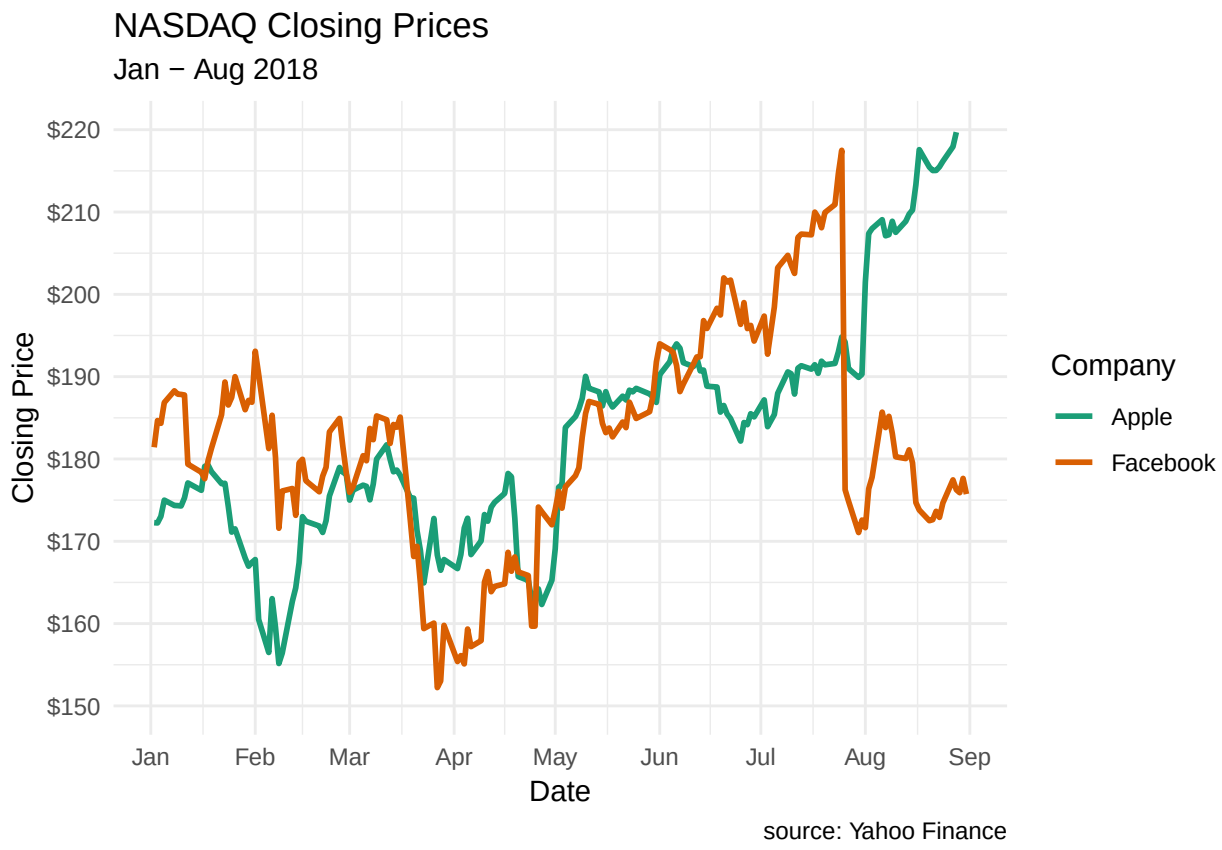


Figure 7.1: Multivariate time series

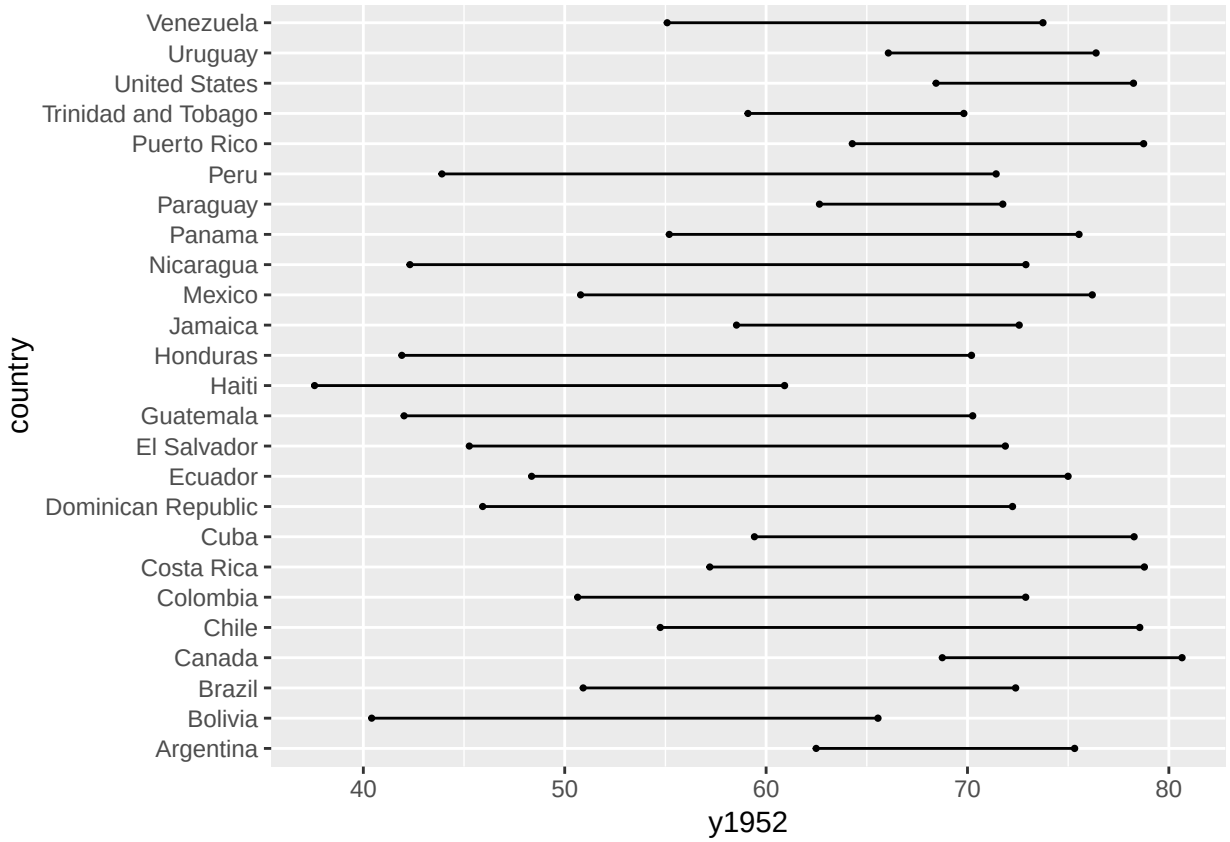


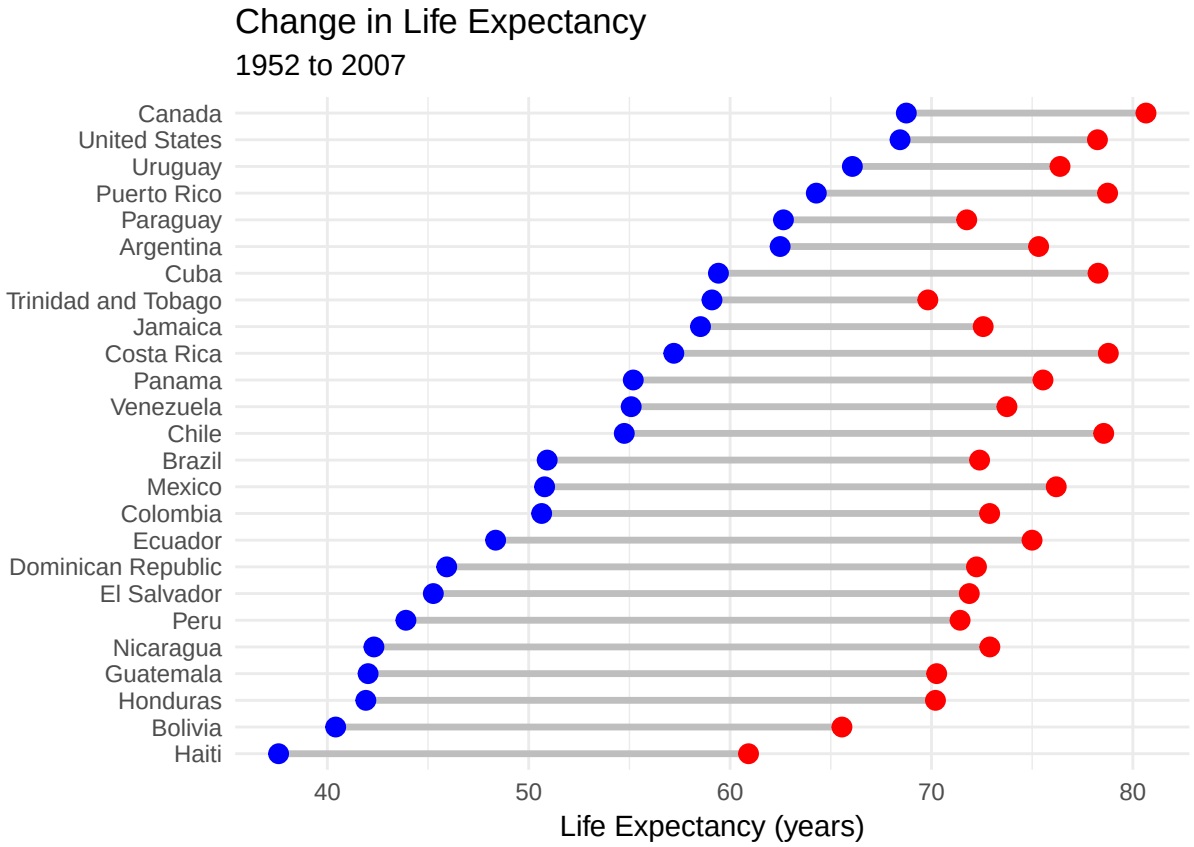
Figure 7.2: Simple dumbbell chart

```
# convert data to wide format
plotdata_wide <- spread(plotdata_long, year, lifeExp)
names(plotdata_wide) <- c("country", "y1952", "y2007")

# create dumbbell plot
ggplot(plotdata_wide, aes(y = country,
                          x = y1952,
                          xend = y2007)) +
  geom_dumbbell()
```

The graph will be easier to read if the countries are sorted and the points are sized and colored. In the next graph, we'll sort by 1952 life expectancy, and modify the line and point size, color the points, add titles and labels, and simplify the theme.

```
# create dumbbell plot
ggplot(plotdata_wide,
       aes(y = reorder(country, y1952),
           x = y1952,
           xend = y2007)) +
  geom_dumbbell(size = 1.2,
                size_x = 3,
                size_xend = 3,
                colour = "grey",
```

```
                colour_x = "blue",
                colour_xend = "red") +
theme_minimal() +
labs(title = "Change in Life Expectancy",
     subtitle = "1952 to 2007",
     x = "Life Expectancy (years)",
     y = "")
```



It is easier to discern patterns here. For example Haiti started with the lowest life expectancy in 1952 and still has the lowest in 2007. Paraguay started relatively high by has made few gains.

7.3 Slope graphs

When there are several groups and several time points, a slope graph can be helpful. Let's plot life expectancy for six Central American countries in 1992, 1997, 2002, and 2007. Again we'll use the gapminder data.

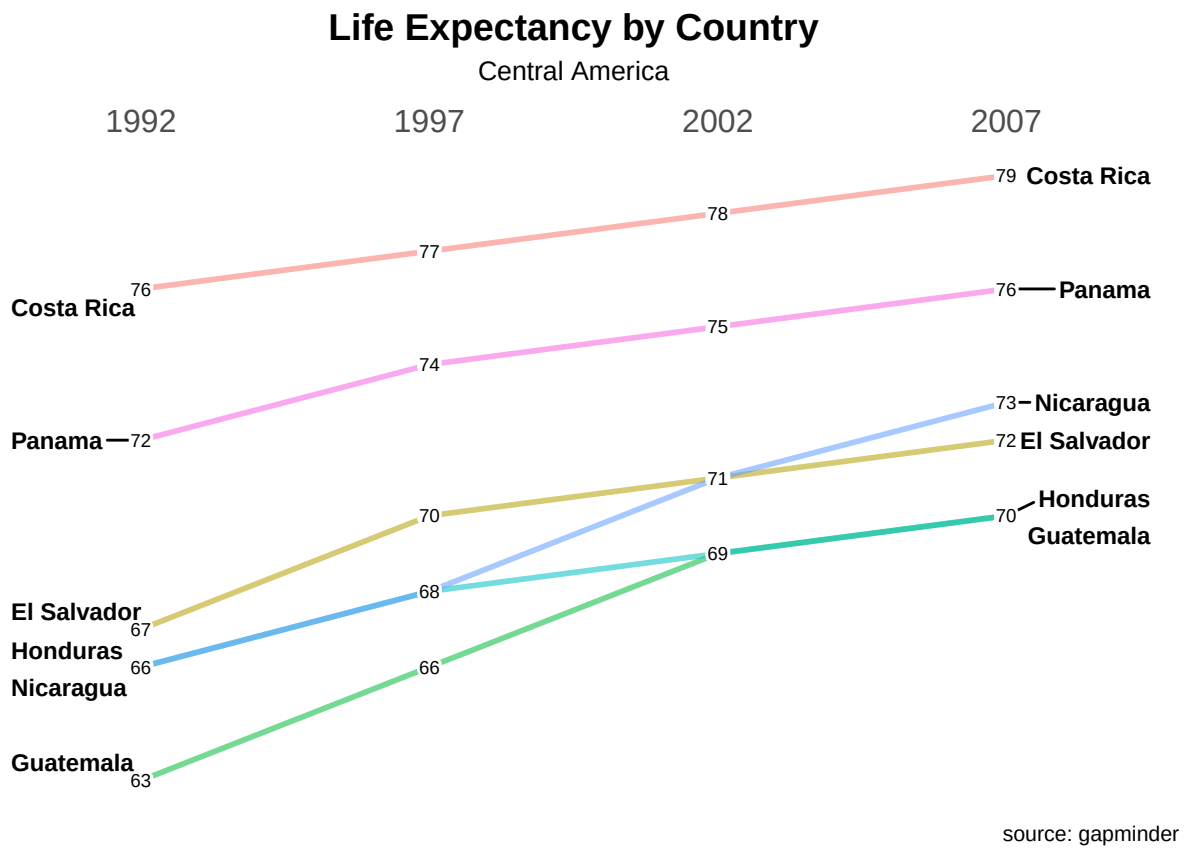
To create a slope graph, we'll use the `newggslopegraph` function from the `CGPfunctions` package.

The `newggslopegraph` function parameters are (in order)

- data frame
- time variable (which must be a factor)
- numeric variable to be plotted

- and grouping variable (creating one line per group).

```
library(CGPfunctions)

# Select Central American countries data
# for 1992, 1997, 2002, and 2007

df <- gapminder %>%
  filter(year %in% c(1992, 1997, 2002, 2007) &
           country %in% c("Panama", "Costa Rica",
                          "Nicaragua", "Honduras",
                          "El Salvador", "Guatemala",
                          "Belize")) %>%
  mutate(year = factor(year),
         lifeExp = round(lifeExp))

# create slope graph

newggslopegraph(df, year, lifeExp, country) +
  labs(title="Life Expectancy by Country",
       subtitle="Central America",
       caption="source: gapminder")
```

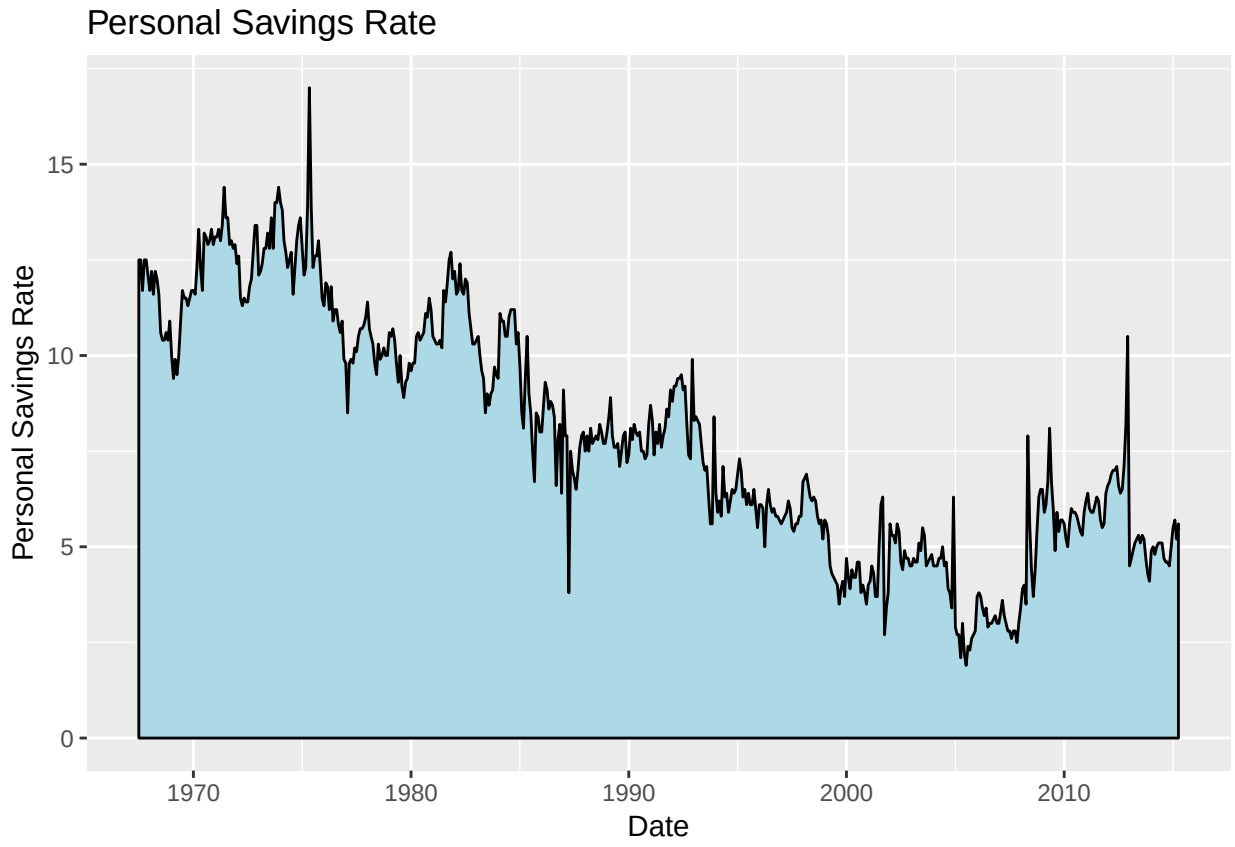


In the graph above, Costa Rica has the highest life expectancy across the range of years studied. Guatemala has the lowest, and caught up with Honduras (also low at 69) in 2002.

7.4 Area Charts

A simple area chart is basically a line graph, with a fill from the line to the x-axis.

```
# basic area chart
ggplot(economics, aes(x = date, y = psavert)) +
  geom_area(fill="lightblue", color="black") +
  labs(title = "Personal Savings Rate",
       x = "Date",
       y = "Personal Savings Rate")
```



A stacked area chart can be used to show differences between groups over time. Consider the `uspopage` dataset from the `gcookbook` package. We'll plot the age distribution of the US population from 1900 and 2002.

```
# stacked area chart
data(uspopage, package = "gcookbook")
ggplot(uspopage, aes(x = Year,
                     y = Thousands,
                     fill = AgeGroup)) +
  geom_area() +
  labs(title = "US Population by age",
       x = "Year",
       y = "Population in Thousands")
```

It is best to avoid scientific notation in your graphs. How likely is it that the average reader will know that

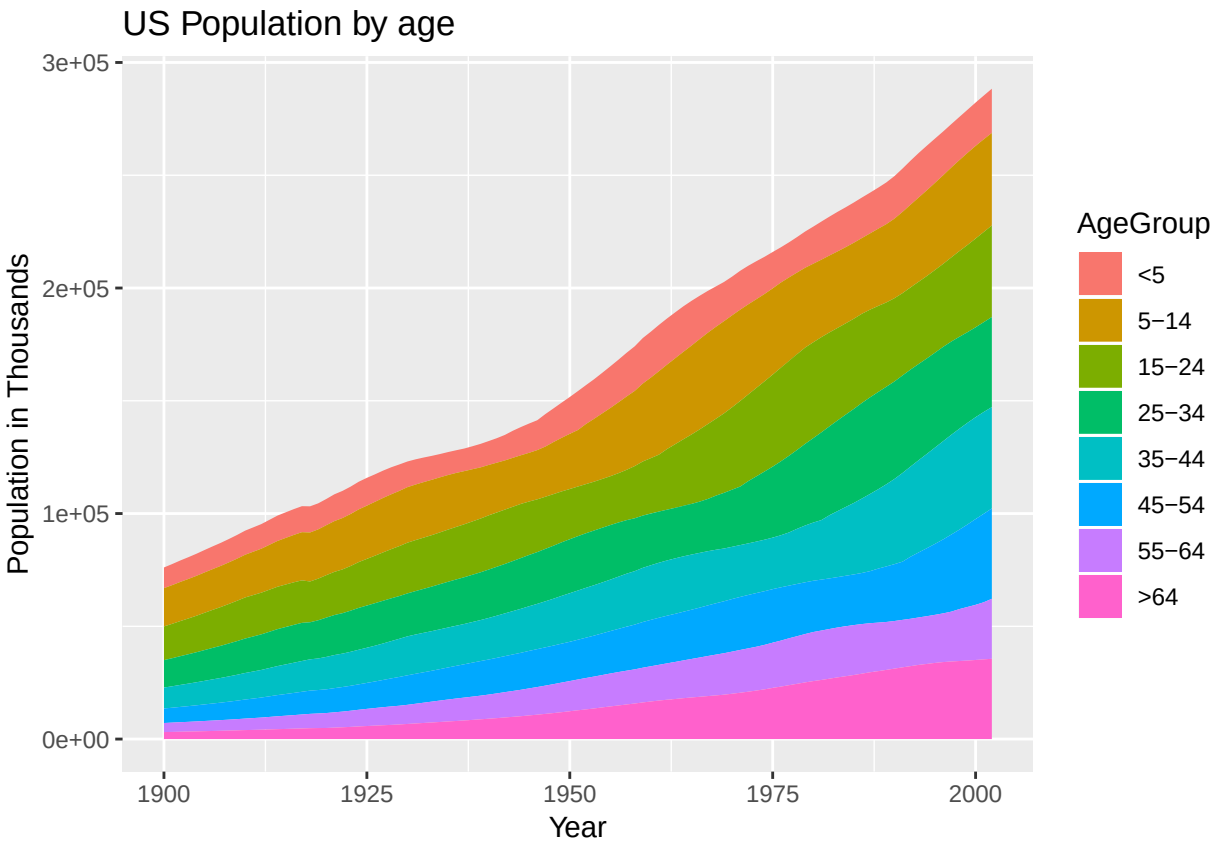


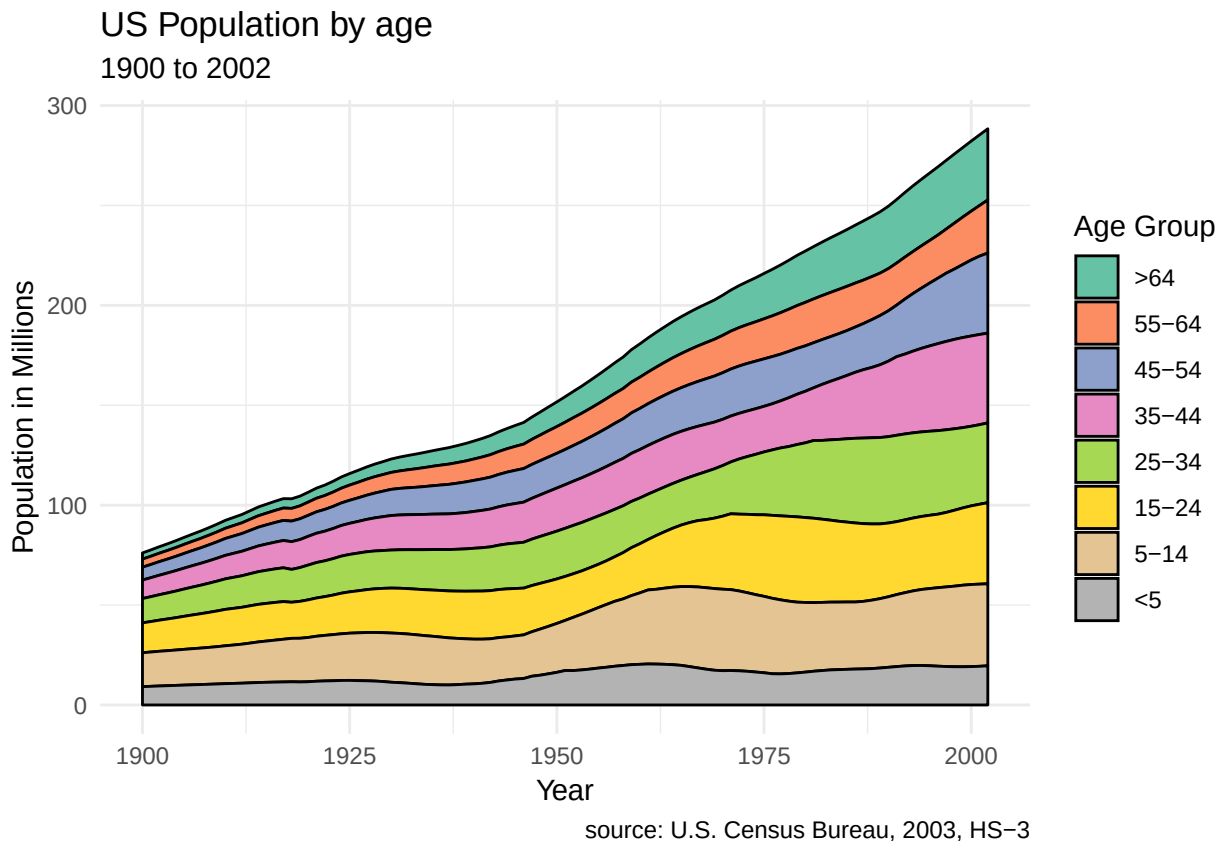
Figure 7.3: Stacked area chart

3e+05 means 300,000,000? It is easy to change the scale in `ggplot2`. Simply divide the Thousands variable by 1000 and report it as Millions. While we are at it, let's

- create black borders to highlight the difference between groups
- reverse the order the groups to match increasing age
- improve labeling
- choose a different color scheme
- choose a simpler theme.

The levels of the *AgeGroup* variable can be reversed using the `fct_rev` function in the `forcats` package.

```
# stacked area chart
data(uspopage, package = "gcookbook")
ggplot(uspopage, aes(x = Year,
                     y = Thousands/1000,
                     fill = forcats::fct_rev(AgeGroup))) +
  geom_area(color = "black") +
  labs(title = "US Population by age",
       subtitle = "1900 to 2002",
       caption = "source: U.S. Census Bureau, 2003, HS-3",
       x = "Year",
       y = "Population in Millions",
       fill = "Age Group") +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()
```



Apparently, the number of young children have not changed very much in the past 100 years.

Stacked area charts are most useful when interest is on both (1) group change over time and (2) overall change over time. Place the most important groups at the bottom. These are the easiest to interpret in this type of plot.

Chapter 8

Statistical Models

A statistical model describes the relationship between one or more explanatory variables and one or more response variables. Graphs can help to visualize these relationships. In this section we'll focus on models that have a single response variable that is either quantitative (a number) or binary (yes/no).

8.1 Correlation plots

Correlation plots help you to visualize the pairwise relationships between a set of quantitative variables by displaying their correlations using color or shading.

Consider the Saratoga Houses dataset, which contains the sale price and characteristics of Saratoga County, NY homes in 2006. In order to explore the relationships among the quantitative variables, we can calculate the Pearson Product-Moment correlation coefficients.

```
data(SaratogaHouses, package="mosaicData")

# select numeric variables
df <- dplyr::select_if(SaratogaHouses, is.numeric)

# calulate the correlations
r <- cor(df, use="complete.obs")
round(r,2)
```

```
##             price lotSize   age landValue livingArea pctCollege bedrooms
## price        1.00    0.16 -0.19      0.58       0.71       0.20     0.40
## lotSize      0.16    1.00 -0.02      0.06       0.16      -0.03     0.11
## age         -0.19   -0.02  1.00     -0.02      -0.17      -0.04     0.03
## landValue    0.58    0.06 -0.02      1.00       0.42       0.23     0.20
## livingArea   0.71    0.16 -0.17      0.42       1.00       0.21     0.66
## pctCollege   0.20   -0.03 -0.04      0.23       0.21       1.00     0.16
## bedrooms     0.40    0.11  0.03      0.20       0.66       0.16     1.00
## fireplaces   0.38    0.09 -0.17      0.21       0.47       0.25     0.28
## bathrooms    0.60    0.08 -0.36      0.30       0.72       0.18     0.46
## rooms        0.53    0.14 -0.08      0.30       0.73       0.16     0.67
##             fireplaces bathrooms rooms
## price             0.38      0.60  0.53
## lotSize           0.09      0.08  0.14
```

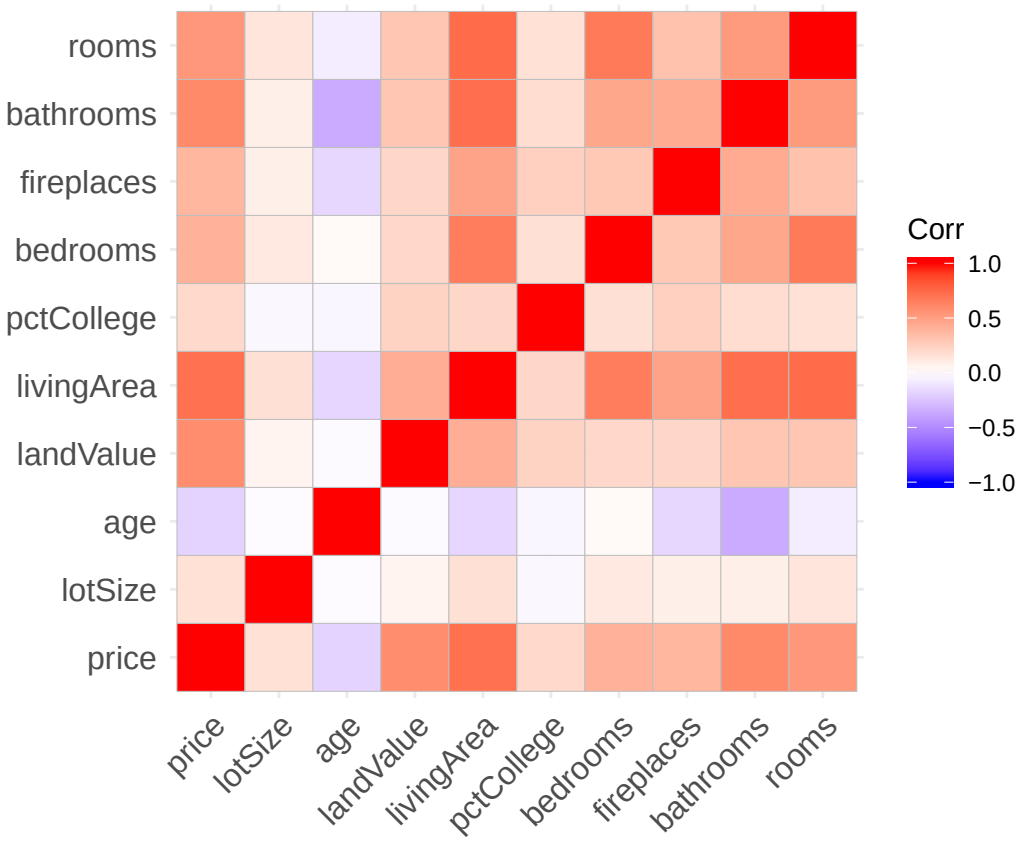


Figure 8.1: Correlation matrix

```
## age             -0.17     -0.36 -0.08
## landValue        0.21      0.30  0.30
## livingArea       0.47      0.72  0.73
## pctCollege       0.25      0.18  0.16
## bedrooms         0.28      0.46  0.67
## fireplaces       1.00      0.44  0.32
## bathrooms        0.44      1.00  0.52
## rooms            0.32      0.52  1.00
```

The `ggcorrplot` function in the `ggcorrplot` package can be used to visualize these correlations. By default, it creates a `ggplot2` graph were darker red indicates stronger positive correlations, darker blue indicates stronger negative correlations and white indicates no correlation.

```
library(ggplot2)
library(ggcorrplot)
ggcorrplot(r)
```

From the graph, an increase in number of bathrooms and living area are associated with increased price, while older homes tend to be less expensive. Older homes also tend to have fewer bathrooms.

The `ggcorrplot` function has a number of options for customizing the output. For example

- `hc.order = TRUE` reorders the variables, placing variables with similar correlation patterns together.

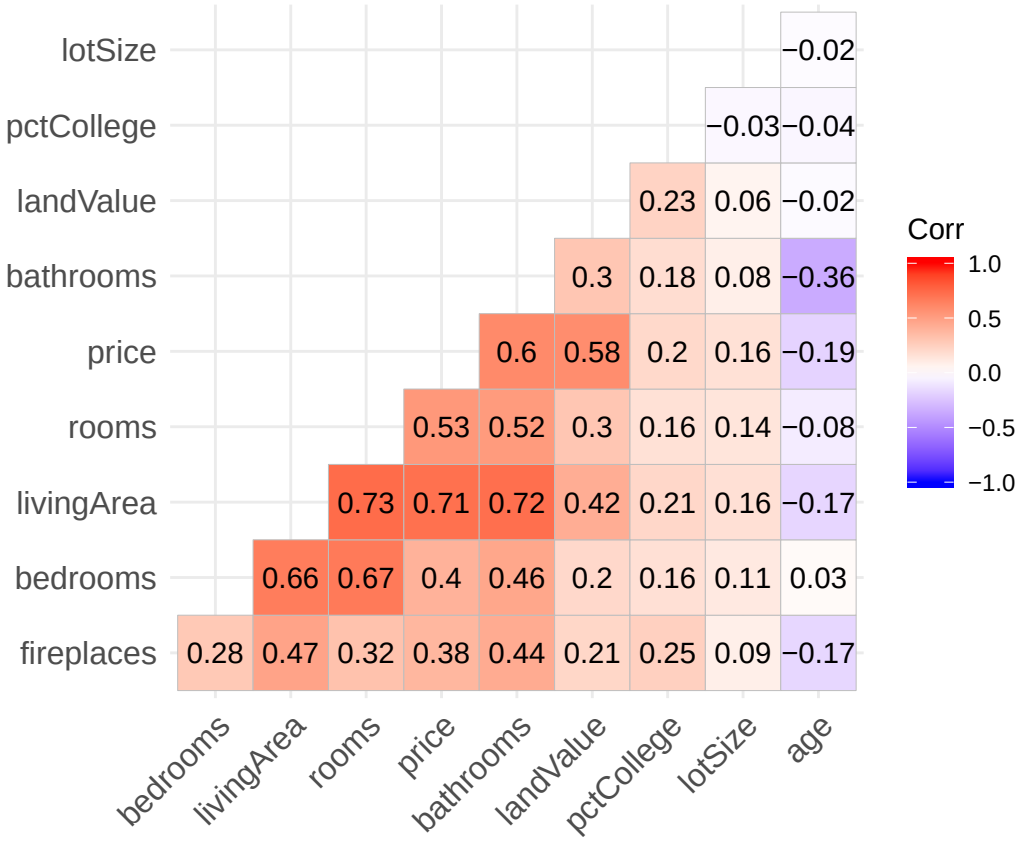


Figure 8.2: Sorted lower triangel correlation matrix with options

- `type = "lower"` plots the lower portion of the correlation matrix.
- `lab = TRUE` overlays the correlation coefficients (as text) on the plot.

```
ggcorrplot(r,
           hc.order = TRUE,
           type = "lower",
           lab = TRUE)
```

These, and other options, can make the graph easier to read and interpret.

8.2 Linear Regression

Linear regression allows us to explore the relationship between a quantitative response variable and an explanatory variable while other variables are held constant.

Consider the prediction of home prices in the Saratoga dataset from lot size (square feet), age (years), land value (1000s dollars), living area (square feet), number of bedrooms and bathrooms and whether the home is on the waterfront or not.

```
data(SaratogaHouses, package="mosaicData")
houses_lm <- lm(price ~ lotSize + age + landValue +
                  livingArea + bedrooms + bathrooms +
```

Table 8.1: Linear Regression results

term	estimate	std.error	statistic	p.value
(Intercept)	139878.80	16472.93	8.49	0.00
lotSize	7500.79	2075.14	3.61	0.00
age	-136.04	54.16	-2.51	0.01
landValue	0.91	0.05	19.84	0.00
livingArea	75.18	4.16	18.08	0.00
bedrooms	-5766.76	2388.43	-2.41	0.02
bathrooms	24547.11	3332.27	7.37	0.00
waterfrontNo	-120726.62	15600.83	-7.74	0.00

```
                  waterfront,
                data = SaratogaHouses)
```

From the results, we can estimate that an increase of one square foot of living area is associated with a home price increase of $75, holding the other variables constant. Additionally, waterfront home cost approximately $120,726 more than non-waterfront home, again controlling for the other variables in the model.

The visreg package provides tools for visualizing these conditional relationships.

The `visreg` function takes (1) the model and (2) the variable of interest and plots the conditional relationship, controlling for the other variables. The option `gg = TRUE` is used to produce a `ggplot2` graph.

```
# conditional plot of price vs. living area
library(ggplot2)
library(visreg)
visreg(houses_lm, "livingArea", gg = TRUE)
```

The graph suggests that, after controlling for lot size, age, living area, number of bedrooms and bathrooms, and waterfront location, sales price increases with living area in a linear fashion.

> **How does `visreg` work?** The fitted model is used to predict values of the response variable, across the range of the chosen explanatory variable. The other variables are set to their median value (for numeric variables) or most frequent category (for categorical variables). The user can override these defaults and chose specific values for any variable in the model.

Continuing the example, the price difference between waterfront and non-waterfront homes is plotted, controlling for the other seven variables. Since a `ggplot2` graph is produced, other `ggplot2` functions can be added to customize the graph.

```
# conditional plot of price vs. waterfront location
visreg(houses_lm, "waterfront", gg = TRUE) +
  scale_y_continuous(label = scales::dollar) +
  labs(title = "Relationship between price and location",
       subtitle = "controlling for lot size, age, land value, bedrooms and bathrooms",
       caption = "source: Saratoga Housing Data (2006)",
       y = "Home Price",
       x = "Waterfront")
```

There are far fewer homes on the water, and they tend to be more expensive (even controlling for size, age, and land value).

The `vizreg` package provides a wide range of plotting capabilities. See Visualization of regression models using visreg for details.

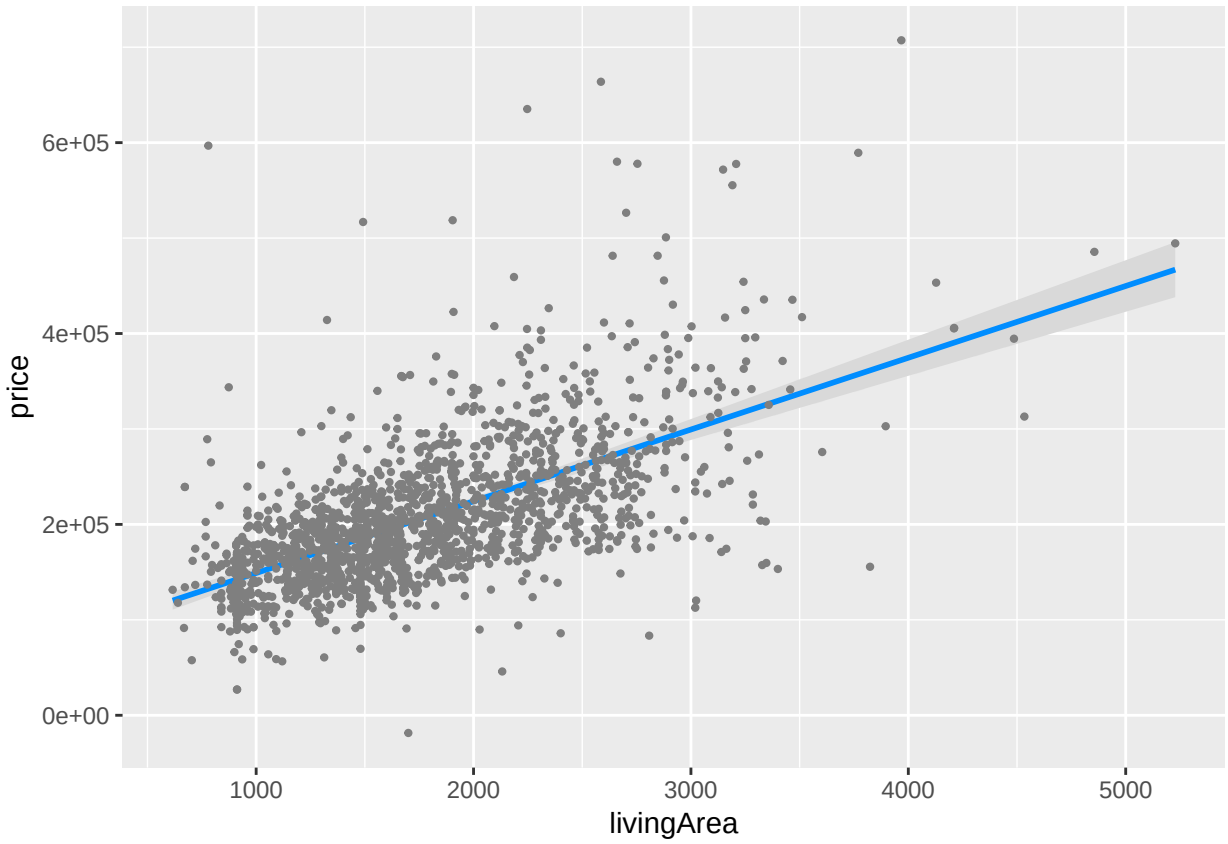


Figure 8.3: Conditional plot of living area and price

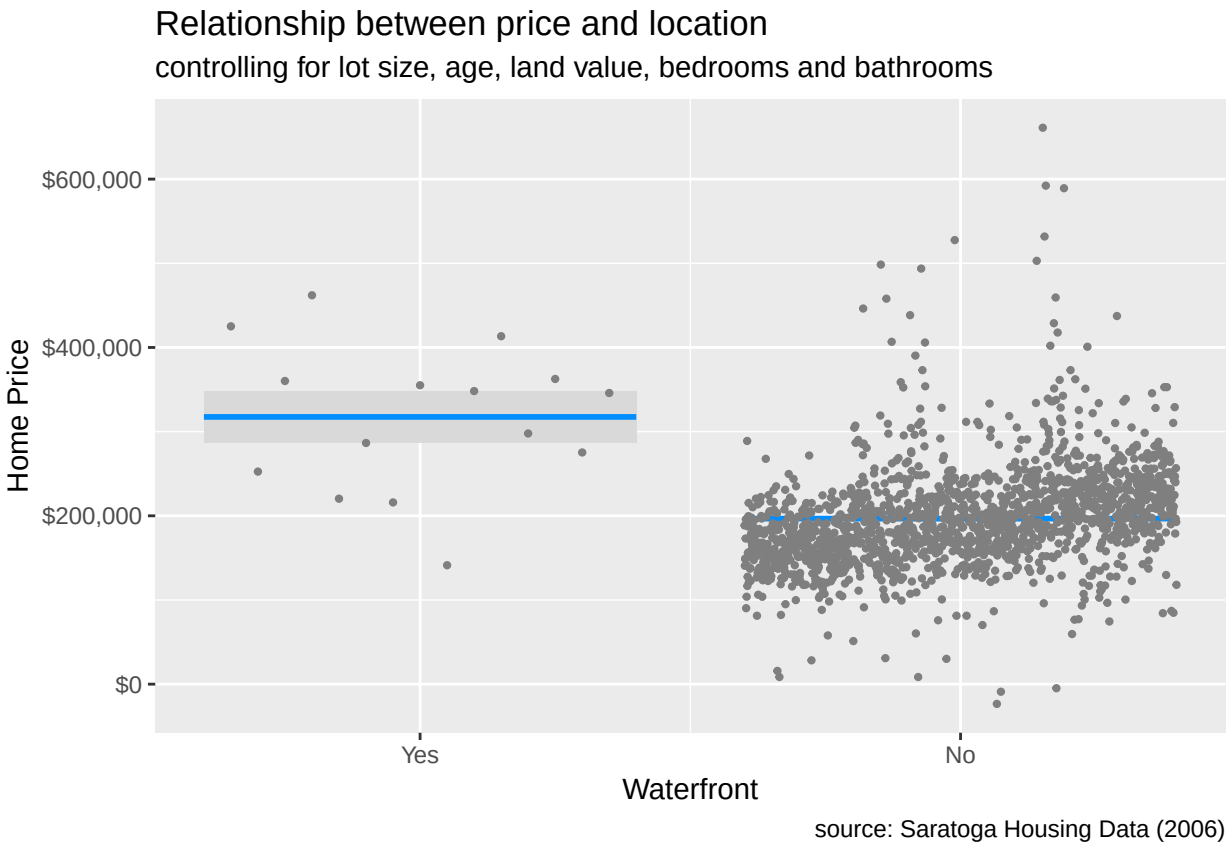


Figure 8.4: Conditional plot of location and price

8.3 Logistic regression

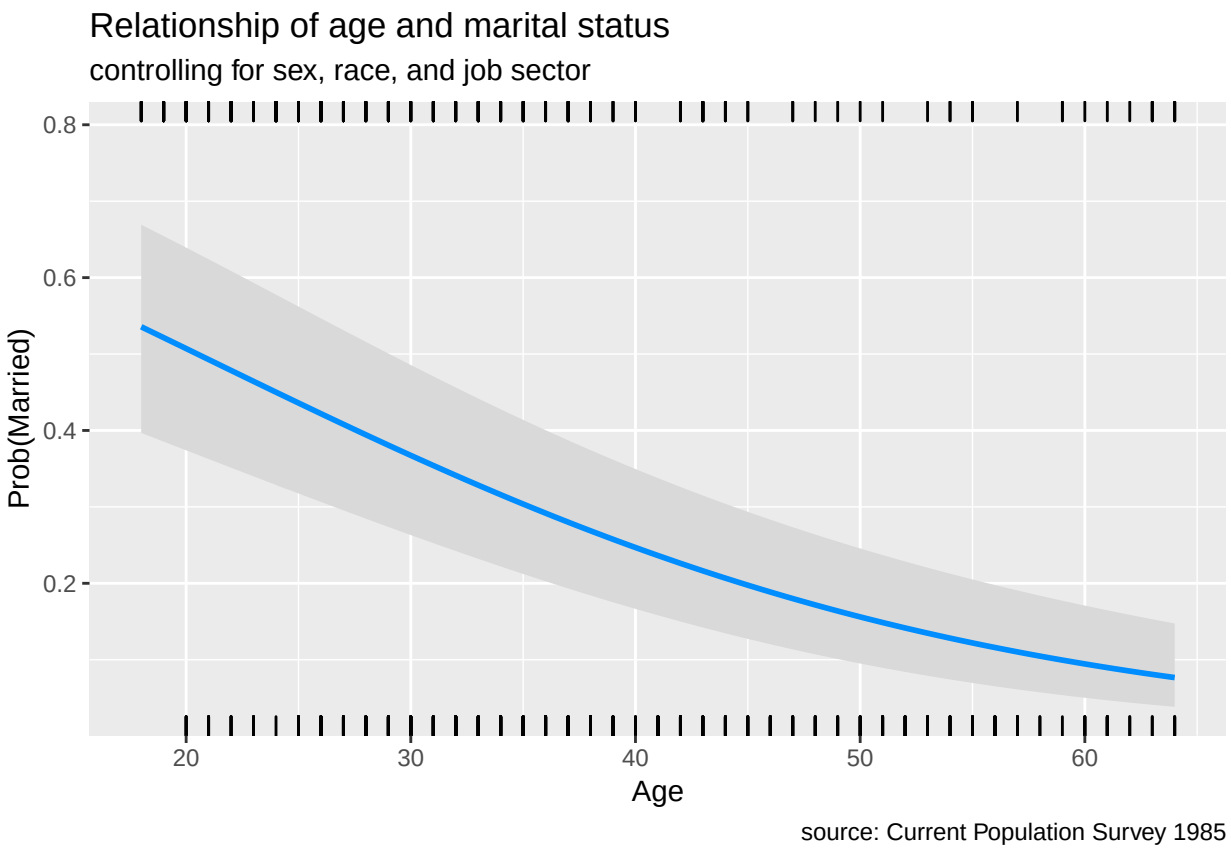
Logistic regression can be used to explore the relationship between a binary response variable and an explanatory variable while other variables are held constant. Binary response variables have two levels (yes/no, lived/died, pass/fail, malignant/benign). As with linear regression, we can use the visreg package to visualize these relationships.

Using the CPS85 data let's predict the log-odds of being married, given one's sex, age, race and job sector.

```
# fit logistic model for predicting
# marital status: married/single
data(CPS85, package = "mosaicData")
cps85_glm <- glm(married ~ sex + age + race + sector,
                 family="binomial",
                 data=CPS85)
```

Using the fitted model, let's visualize the relationship between age and the probability of being married, holding the other variables constant. Again, the `visreg` function takes the model and the variable of interest and plots the conditional relationship, controlling for the other variables. The option `gg = TRUE` is used to produce a `ggplot2` graph. The `scale = "response"` option creates a plot based on a probability (rather than log-odds) scale.

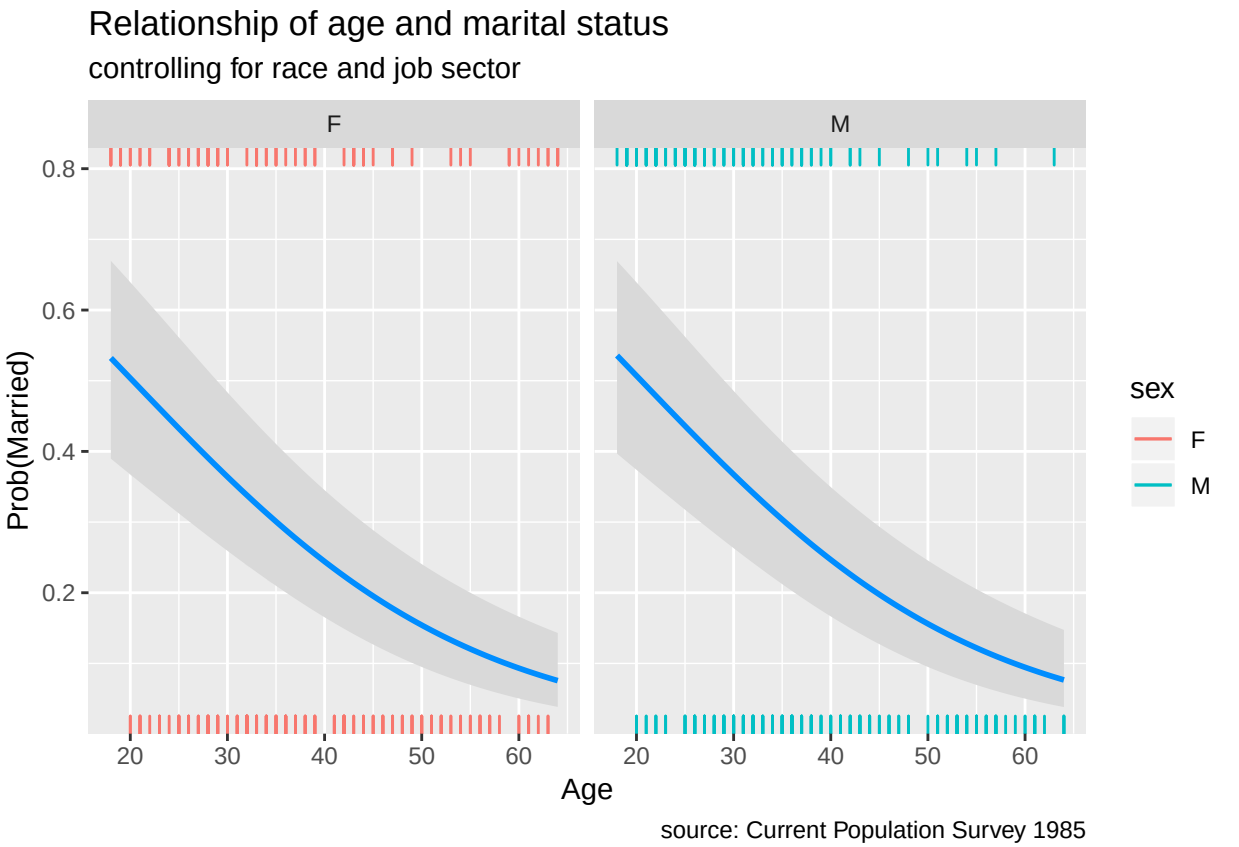
```
# plot results
library(ggplot2)
library(visreg)
visreg(cps85_glm, "age",
       gg = TRUE,
       scale="response") +
  labs(y = "Prob(Married)",
       x = "Age",
       title = "Relationship of age and marital status",
       subtitle = "controlling for sex, race, and job sector",
       caption = "source: Current Population Survey 1985")
```



The probability of being married is estimated to be roughly 0.5 at age 20 and decreases to 0.1 at age 60, controlling for the other variables.

We can create multiple conditional plots by adding a `by` option. For example, the following code will plot the probability of being married by age, seperately for men and women, controlling for race and job sector.

```
# plot results
library(ggplot2)
library(visreg)
visreg(cps85_glm, "age",
       by = "sex",
       gg = TRUE,
       scale="response") +
  labs(y = "Prob(Married)",
       x = "Age",
       title = "Relationship of age and marital status",
       subtitle = "controlling for race and job sector",
       caption = "source: Current Population Survey 1985")
```



In this data, the probability of marriage is very similar for men and women.

8.4 Survival plots

In many research settings, the response variable is the time to an event. This is frequently true in healthcare research, where we are interested in time to recovery, time to death, or time to relapse.

If the event has not occurred for an observation (either because the study ended or the patient dropped out) the observation is said to be *censored*.

The `NCCTG Lung Cancer` dataset in the `survival` package provides data on the survival times of patients with advanced lung cancer following treatment. The study followed patients for up 34 months.

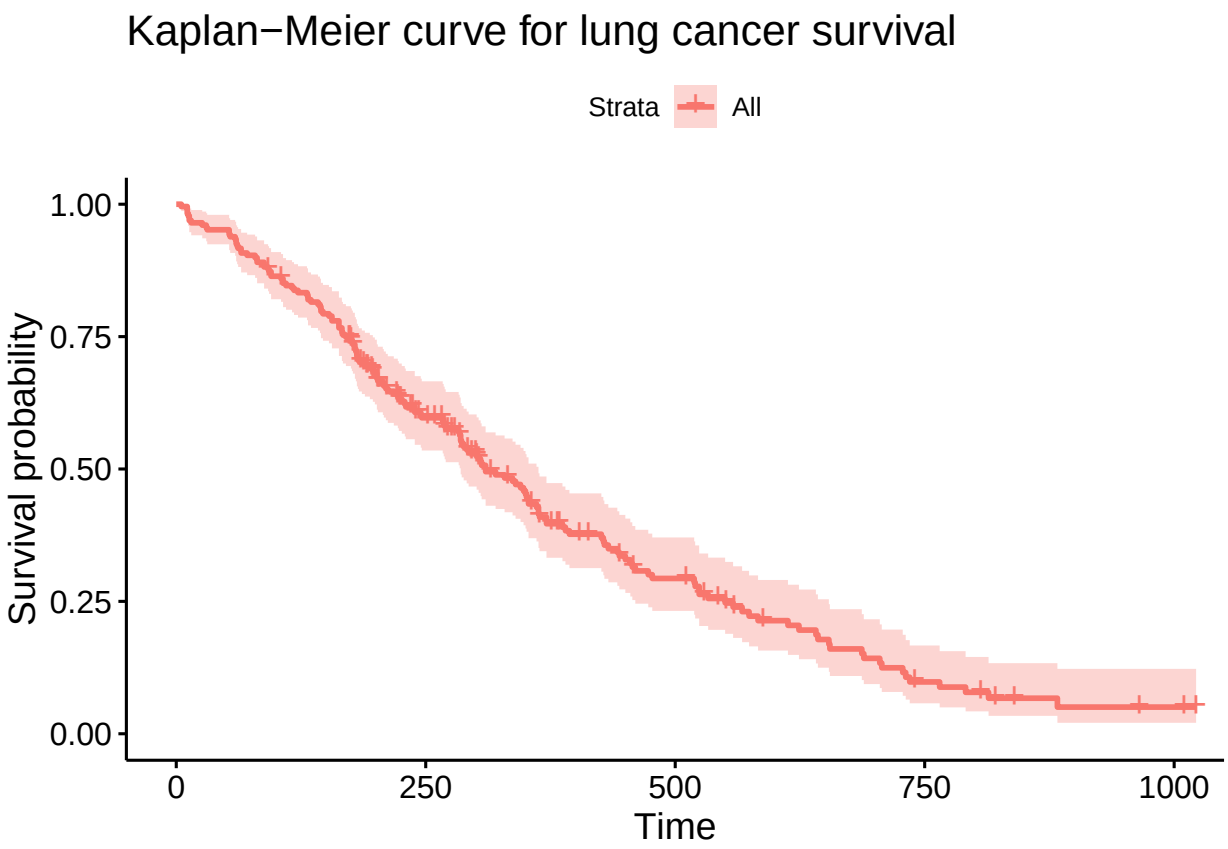
The outcome for each patient is measured by two variables

- *time* - survival time in days
- *status* - 1=censored, 2=dead

Thus a patient with *time=305 & status=2* lived 305 days following treatment. Another patient with *time=400 & status=1*, lived **at least** 400 days but was then lost to the study. A patient with *time=1022 & status=1*, survived to the end of the study (34 months).

A survival plot (also called a Kaplan-Meier Curve) can be used to illustrates the probability that an individual survives up to and including time t.

```
# plot survival curve
library(survival)
library(survminer)

data(lung)
sfit <- survfit(Surv(time, status) ~  1, data=lung)
ggsurvplot(sfit,
           title="Kaplan-Meier curve for lung cancer survival")
```



Roughly 50% of patients are still alive 300 days post treatment. Run `summary(sfit)` for more details.

It is frequently of great interest whether groups of patients have the same survival probabilities. In the next graph, the survival curve for men and women are compared.

```
# plot survival curve for men and women
sfit <- survfit(Surv(time, status) ~  sex, data=lung)
ggsurvplot(sfit,
           conf.int=TRUE,
           pval=TRUE,
           legend.labs=c("Male", "Female"),
           legend.title="Sex",
           palette=c("cornflowerblue", "indianred3"),
           title="Kaplan-Meier Curve for lung cancer survival",
           xlab = "Time (days)")
```

The `ggsurvplot` has many options. In particular, `conf.int` provides confidence intervals, while `pval` provides a log-rank test comparing the survival curves.

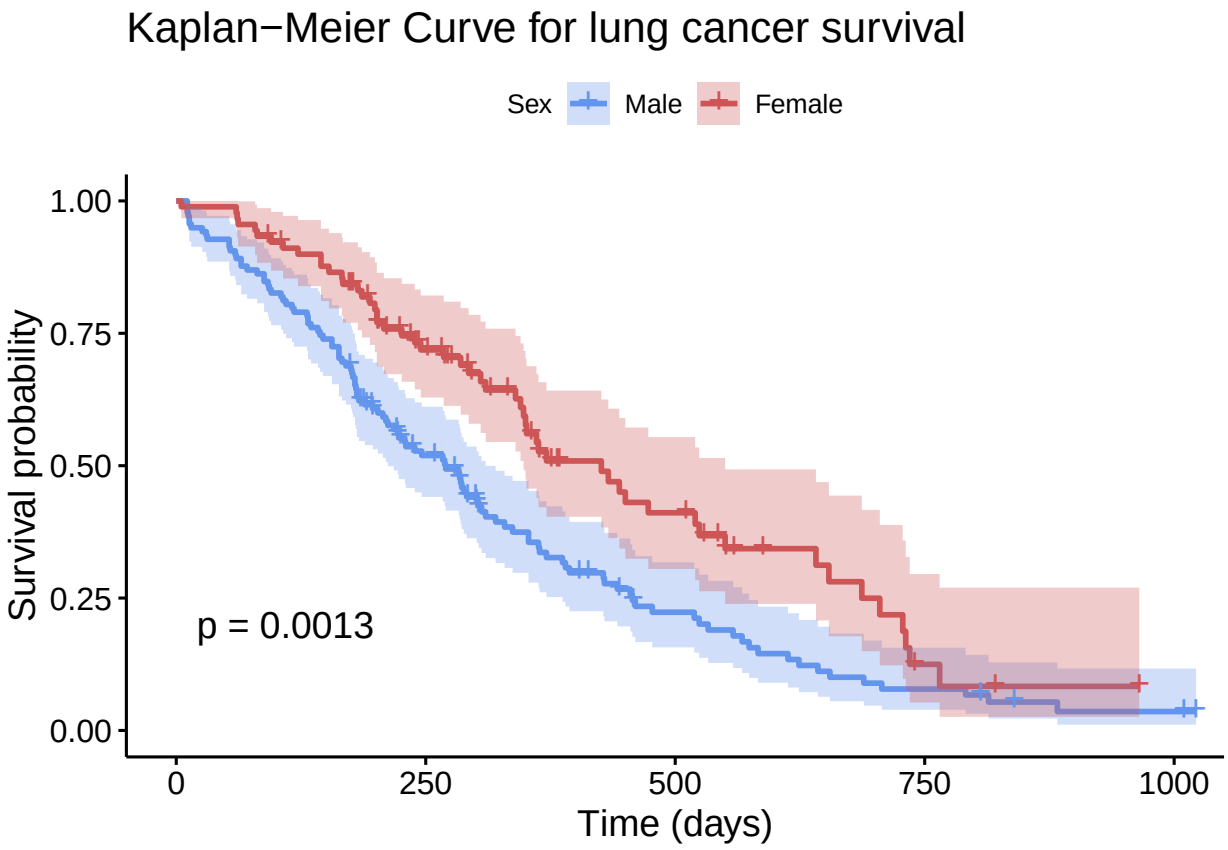


Figure 8.5: Comparison of survival curve

The p-value (0.0013) provides strong evidence that men and women have different survival probabilities following treatment.

8.5 Mosaic plots

Mosaic charts can display the relationship between categorical variables using rectangles whose areas represent the proportion of cases for any given combination of levels. The color of the tiles can also indicate the degree relationship among the variables.

Although mosaic charts can be created with `ggplot2` using the `ggmosaic` package, I recommend using the `vcd` package instead. Although it won't create `ggplot2` graphs, the package provides a more comprehensive approach to visualizing categorical data.

People are fascinated with the Titanic (or is it with Leo?). In the Titanic disaster, what role did sex and class play in survival? We can visualize the relationship between these three categorical variables using the code below.

```
# input data
library(readr)
titanic <- read_csv("titanic.csv")

# create a table
tbl <- xtabs(~Survived + Class + Sex, titanic)
ftable(tbl)
```

```
##                Sex Female Male
## Survived Class
## No       1st            4  118
##          2nd           13  154
##          3rd          106  422
##          Crew           3  670
## Yes      1st          141   62
##          2nd           93   25
##          3rd           90   88
##          Crew          20  192
```

```
# create a mosaic plot from the table
library(vcd)
mosaic(tbl, main = "Titanic data")
```

The size of the tile is proportional to the percentage of cases in that combination of levels. Clearly more passengers perished, than survived. Those that perished were primarily 3rd class male passengers and male crew (the largest group).

If we assume that these three variables are independent, we can examine the residuals from the model and shade the tiles to match. In the graph below, dark blue represents more cases than expected given independence. Dark red represents less cases than expected if independence holds.

```
mosaic(tbl,
       shade = TRUE,
       legend = TRUE,
       labeling_args = list(set_varnames = c(Sex = "Gender",
                                             Survived = "Survived",
```

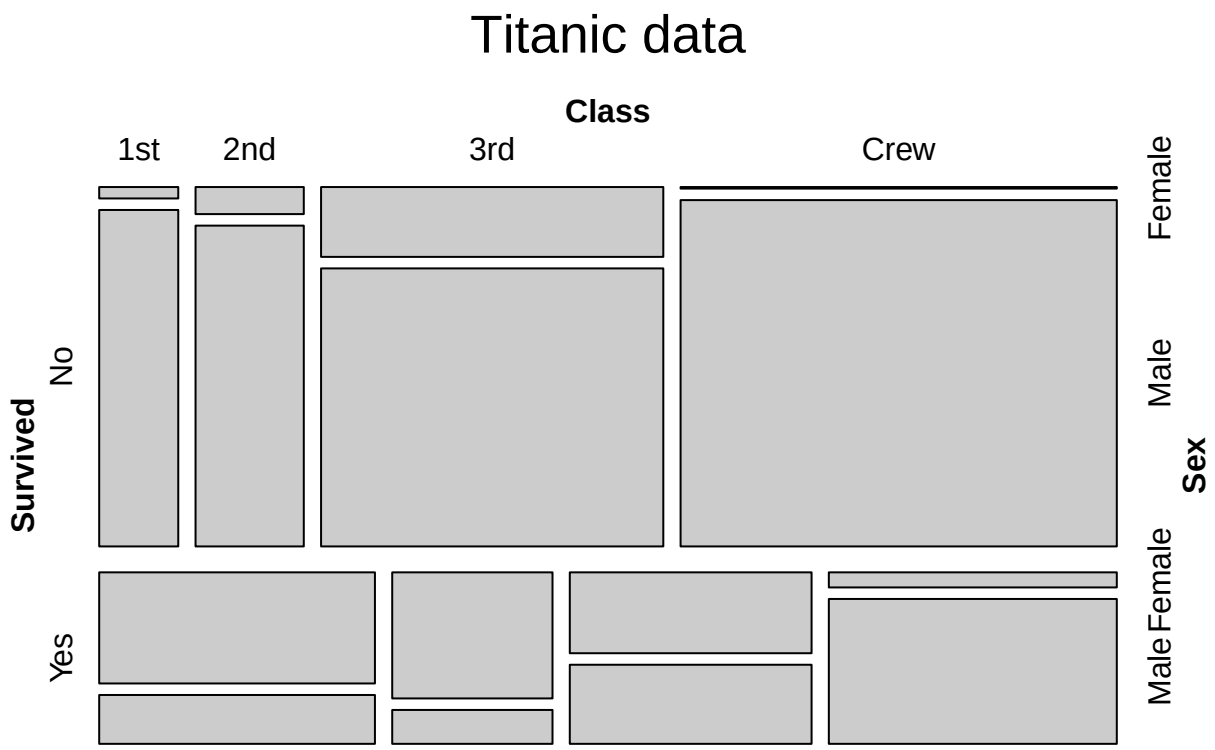


Figure 8.6: Basic mosaic plot

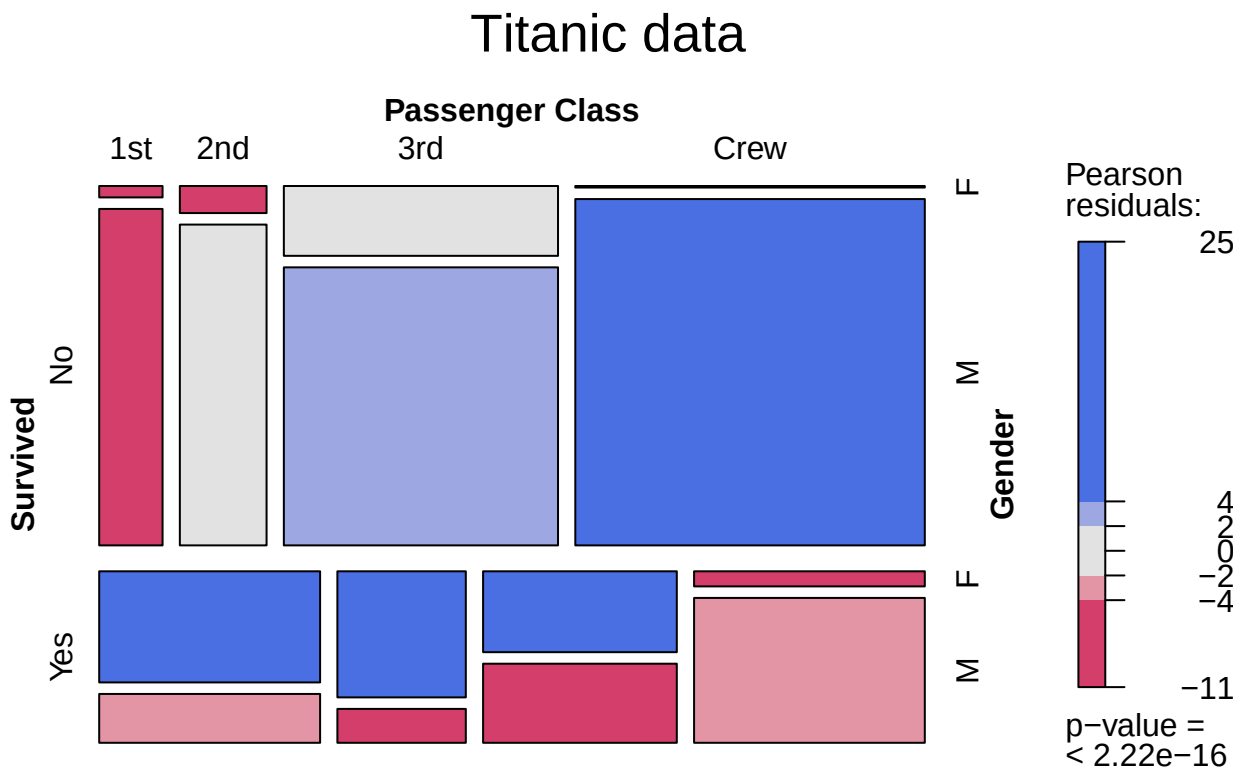


Figure 8.7: Mosaic plot with shading

```
                                          Class = "Passenger Class")),
      set_labels = list(Survived = c("No", "Yes"),
                        Class = c("1st", "2nd", "3rd", "Crew"),
                        Sex = c("F", "M")),
      main = "Titanic data")
```

We can see that if class, gender, and survival are independent, we are seeing many more male crew perishing, and 1st, 2nd and 3rd class females surviving than would be expected. Conversely, far fewer 1st class passengers (both male and female) died than would be expected by chance. Thus the assumption of independence is rejected. (Spoiler alert: Leo doesn't make it.)

For complicated tables, labels can easily overlap. See `labeling_border`, for plotting options.

Chapter 9

Other Graphs

Graphs in this chapter can be very useful, but don't fit in easily within the other chapters.

9.1 3-D Scatterplot

The `ggplot2` package and its extensions can't create a 3-D plot. However, you can create a 3-D scatterplot with the `scatterplot3d` function in the `scatterplot3d` package.

Let's say that we want to plot automobile mileage vs. engine displacement vs. car weight using the data in the `mtcars` dataframe.

```
# basic 3-D scatterplot
library(scatterplot3d)
with(mtcars, {
   scatterplot3d(x = disp,
                 y = wt,
                 z = mpg,
                 main="3-D Scatterplot Example 1")
})
```

Now lets, modify the graph by replacing the points with filled blue circles, add drop lines to the x-y plane, and create more meaningful labels.

```
library(scatterplot3d)
with(mtcars, {
  scatterplot3d(x = disp,
                y = wt,
                z = mpg,
                # filled blue circles
                color="blue",
                pch=19,
                # lines to the horizontal plane
                type = "h",
                main = "3-D Scatterplot Example 2",
                xlab = "Displacement (cu. in.)",
                ylab = "Weight (lb/1000)",
                zlab = "Miles/(US) Gallon")
})
```

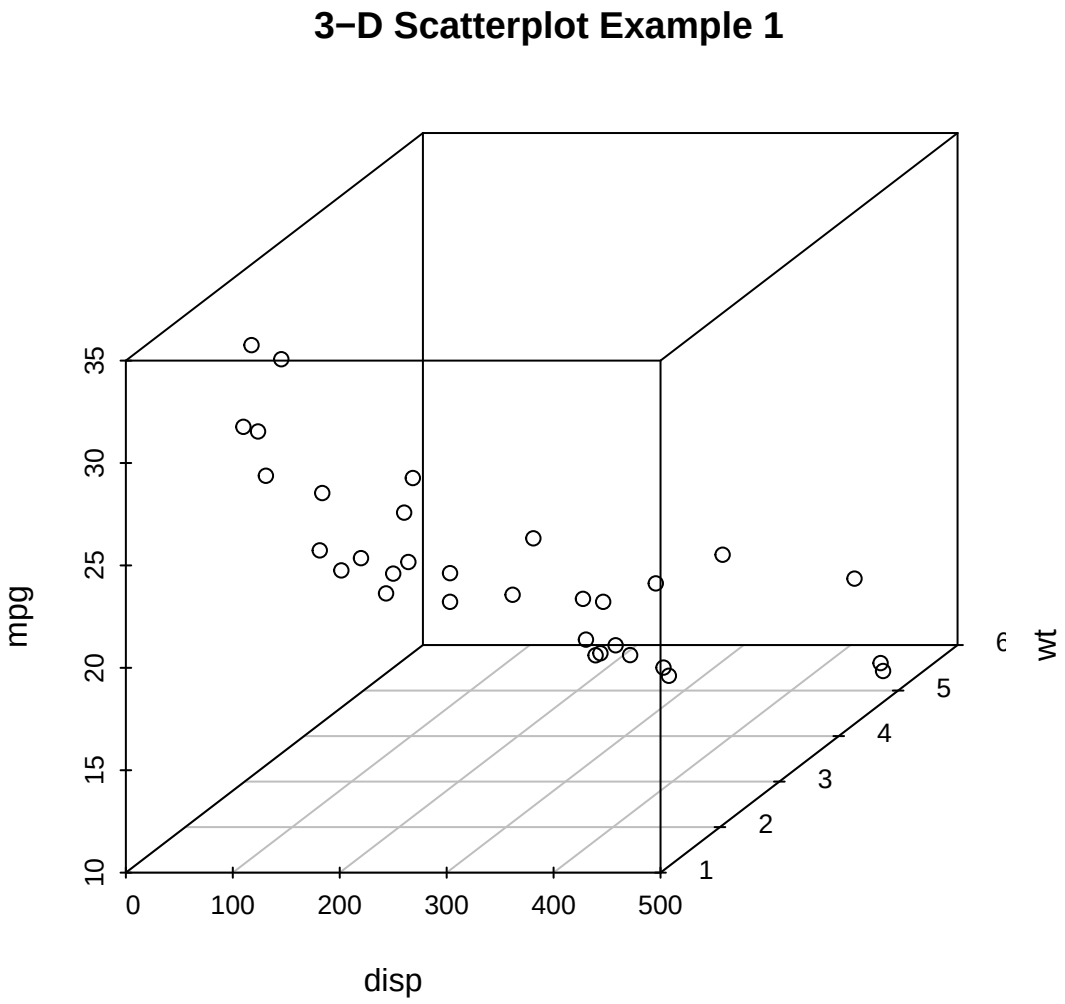


Figure 9.1: Basic 3-D scatterplot

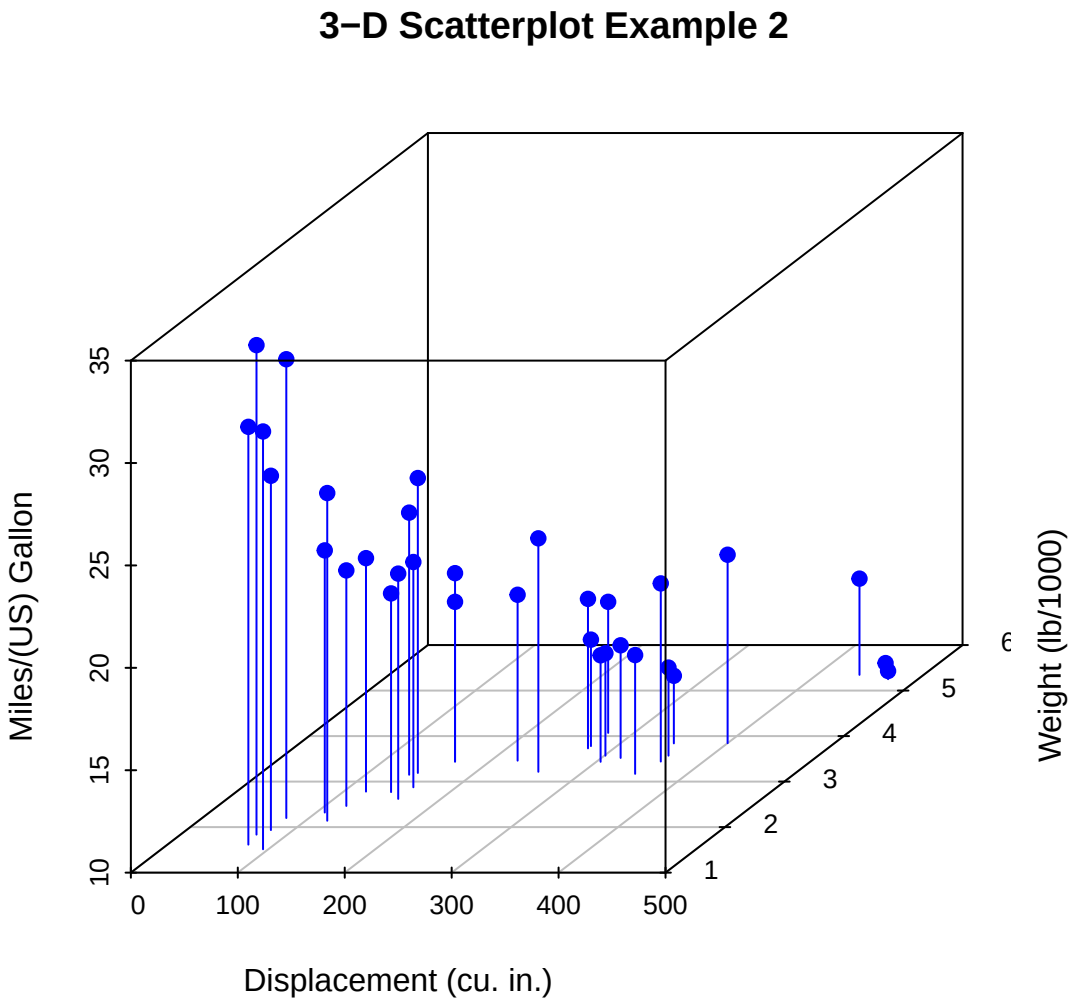


Figure 9.2: 3-D scatterplot with vertical lines

Next, let's label the points. We can do this by saving the results of the `scatterplot3d` function to an object, using the `xyz.convert` function to convert coordinates from 3-D (x, y, z) to 2D-projections (x, y), and apply the `text` function to add labels to the graph.

```
library(scatterplot3d)
with(mtcars, {
  s3d <- scatterplot3d(
    x = disp,
    y = wt,
    z = mpg,
    color = "blue",
    pch = 19,
    type = "h",
    main = "3-D Scatterplot Example 3",
    xlab = "Displacement (cu. in.)",
    ylab = "Weight (lb/1000)",
    zlab = "Miles/(US) Gallon")

  # convert 3-D coords to 2D projection
  s3d.coords <- s3d$xyz.convert(disp, wt, mpg)

  # plot text with 50% shrink and place to right of points
  text(s3d.coords$x,
       s3d.coords$y,
       labels = row.names(mtcars),
       cex = .5,
       pos = 4)
})
```

Almost there. As a final step, we will add information on the number of cylinders in each car. To do this, we'll add a column to the `mtcars` dataframe indicating the color for each point. For good measure, we will shorten the y-axis, change the drop lines to dashed lines, and add a legend.

```
library(scatterplot3d)

# create column indicating point color
mtcars$pcolor[mtcars$cyl == 4] <- "red"
mtcars$pcolor[mtcars$cyl == 6] <- "blue"
mtcars$pcolor[mtcars$cyl == 8] <- "darkgreen"

with(mtcars, {
    s3d <- scatterplot3d(
      x = disp,
      y = wt,
      z = mpg,
      color = pcolor,
      pch = 19,
      type = "h",
      lty.hplot = 2,
      scale.y = .75,
      main = "3-D Scatterplot Example 4",
      xlab = "Displacement (cu. in.)",
      ylab = "Weight (lb/1000)",
      zlab = "Miles/(US) Gallon")
```

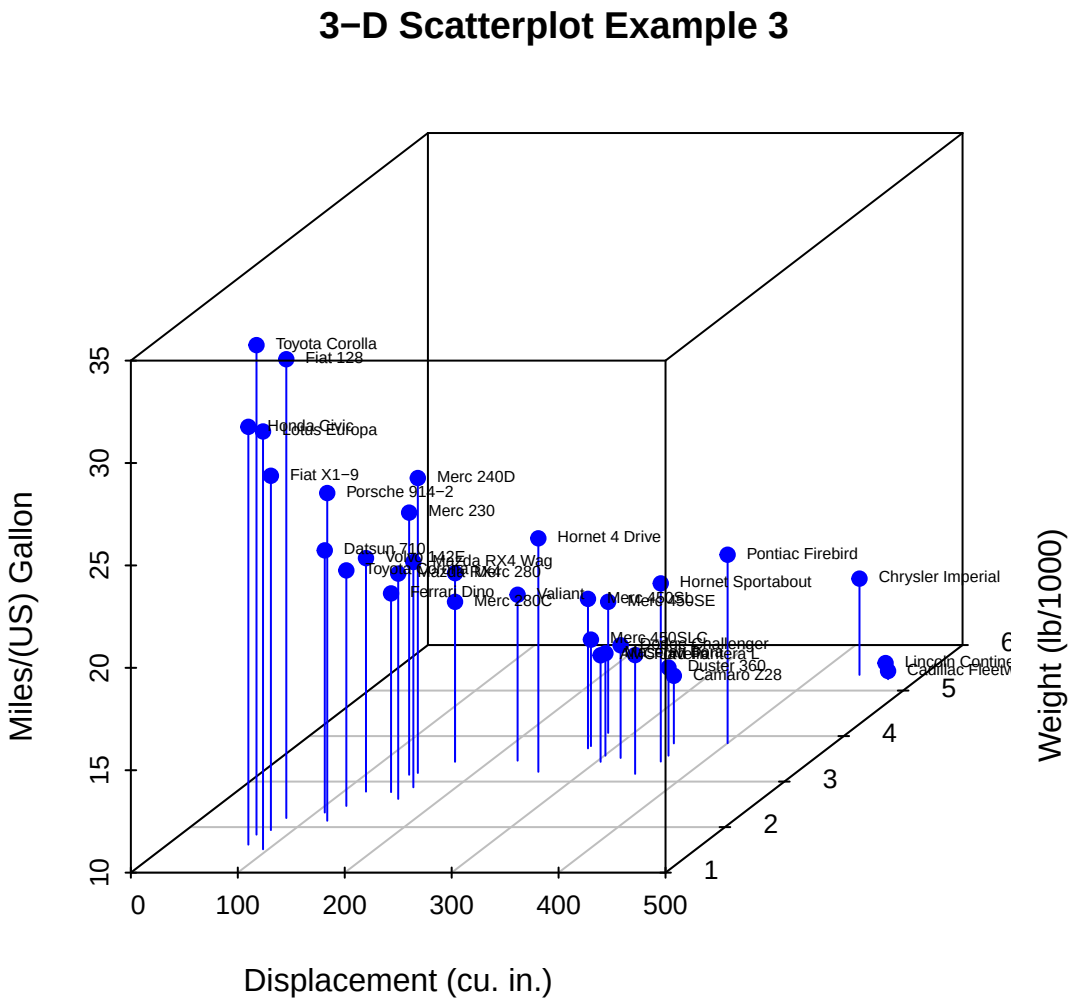
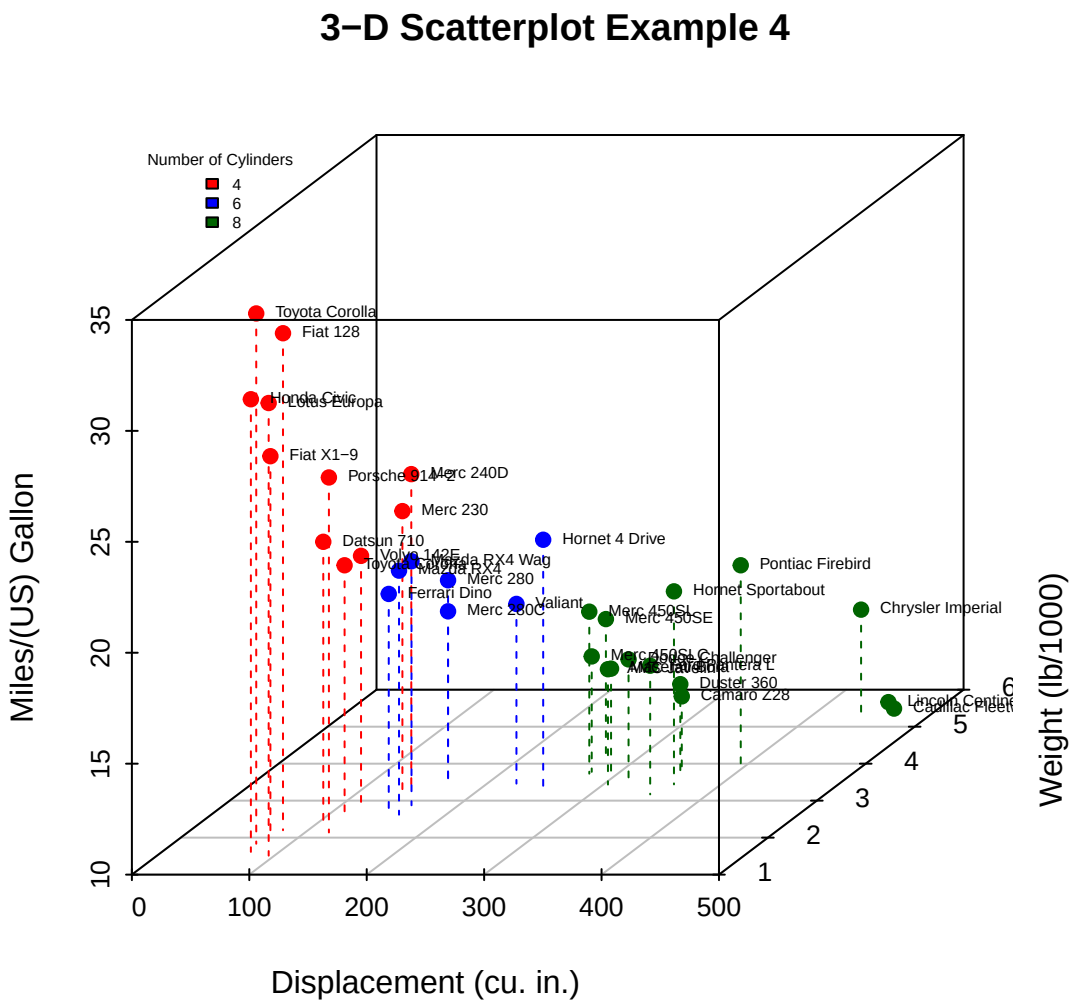


Figure 9.3: 3-D scatterplot with vertical lines and point labels

```
    s3d.coords <- s3d$xyz.convert(disp, wt, mpg)
    text(s3d.coords$x,
         s3d.coords$y,
         labels = row.names(mtcars),
         pos = 4,
         cex = .5)

# add the legend
legend(#location
       "topleft",
       inset=.05,
       # suppress legend box, shrink text 50%
       bty="n",
       cex=.5,
       title="Number of Cylinders",
       c("4", "6", "8"),
       fill=c("red", "blue", "darkgreen"))
})
```



We can easily see that the car with the highest mileage (Toyota Corolla) has low engine displacement, low weight, and 4 cylinders.

9.2 Biplots

A biplot is a specialized graph that attempts to represent the relationship between observations, between variables, and between observations and variables, in a low (usually two) dimensional space.

It's easiest to see how this works with an example. Let's create a biplot for the `mtcars` dataset, using the `fviz_pca` function from the `factoextra` package.

```
# create a biplot
# load data
data(mtcars)

# fit a principal components model
fit <- prcomp(x = mtcars,
```

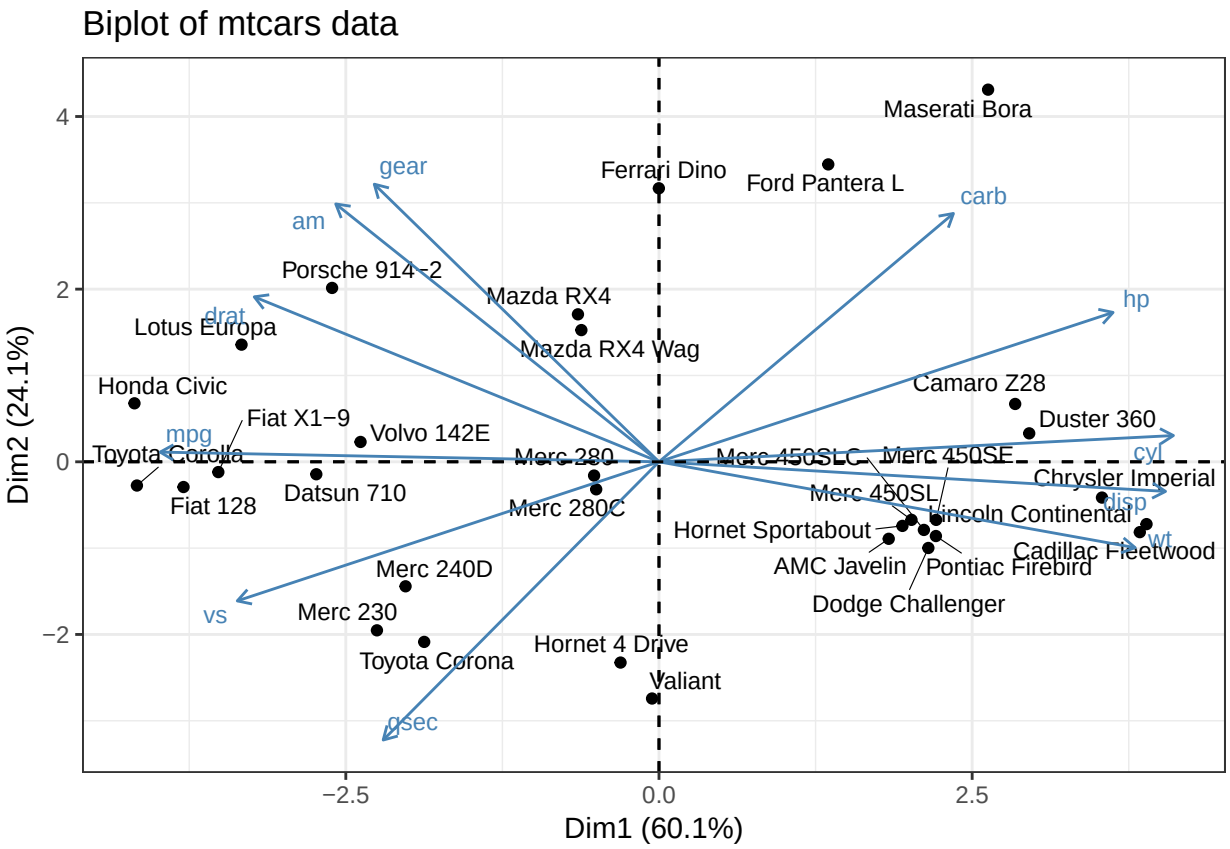


Figure 9.4: Basic biplot

```
              center = TRUE,
              scale = TRUE)

# plot the results
library(factoextra)
fviz_pca(fit,
         repel = TRUE,
         labelsize = 3) +
  theme_bw() +
  labs(title = "Biplot of mtcars data")
```

The `fviz_pca` function produces a `ggplot2` graph.

Dim1 and *Dim2* are the first two principal components - linear combinations of the original p variables.

$$PC_1 = \beta_{10} + \beta_{11}x_1 + \beta_{12}x_2 + \beta_{13}x_3 + \cdots + \beta_{1p}x_p$$

$$PC_2 = \beta_{20} + \beta_{21}x_1 + \beta_{22}x_2 + \beta_{23}x_3 + \cdots + \beta_{2p}x_p$$

The weights of these linear combinations ($\beta_{ij}s$) are chosen to maximize the variance accounted for in the original variables. Additionally, the principal components (PCs) are constrained to be uncorrelated with each other.

In this graph, the first PC accounts for 60% of the variability in the original data. The second PC accounts for 24%. Together, they account for 84% of the variability in the original $p = 11$ variables.

As you can see, both the observations (cars) and variables (car characteristics) are plotted in the same graph.

- Points represent observations. Smaller distances between points suggest similar values on the original set of variables. For example, the *Toyota Corolla* and *Honda Civic* are similar to each other, as are the *Chrysler Imperial* and *Liconln Continental*. However, the *Toyota Corolla* is very different from the *Lincoln Continental*.
- The vectors (arrows) represent variables. The angle between vectors are proportional to the correlation between the variables. Smaller angles indicate stronger correlations. For example, *gear* and *am* are positively correlated, *gear* and *qsec* are uncorrelated (90 degree angle), and *am* and *wt* are negatively correlated (angle greater then 90 degrees).
- The observations that are are farthest along the direction of a variable's vector, have the highest values on that variable. For example, the *Toyoto Corolla* and *Honda Civic* have higher values on *mpg*. The *Toyota Corona* has a higher *qsec*. The *Duster 360* has more *cylinders*.

Care must be taken in interpreting biplots. They are only accurate when the percentage of variance accounted for is high. Always check your conclusion with the original data.

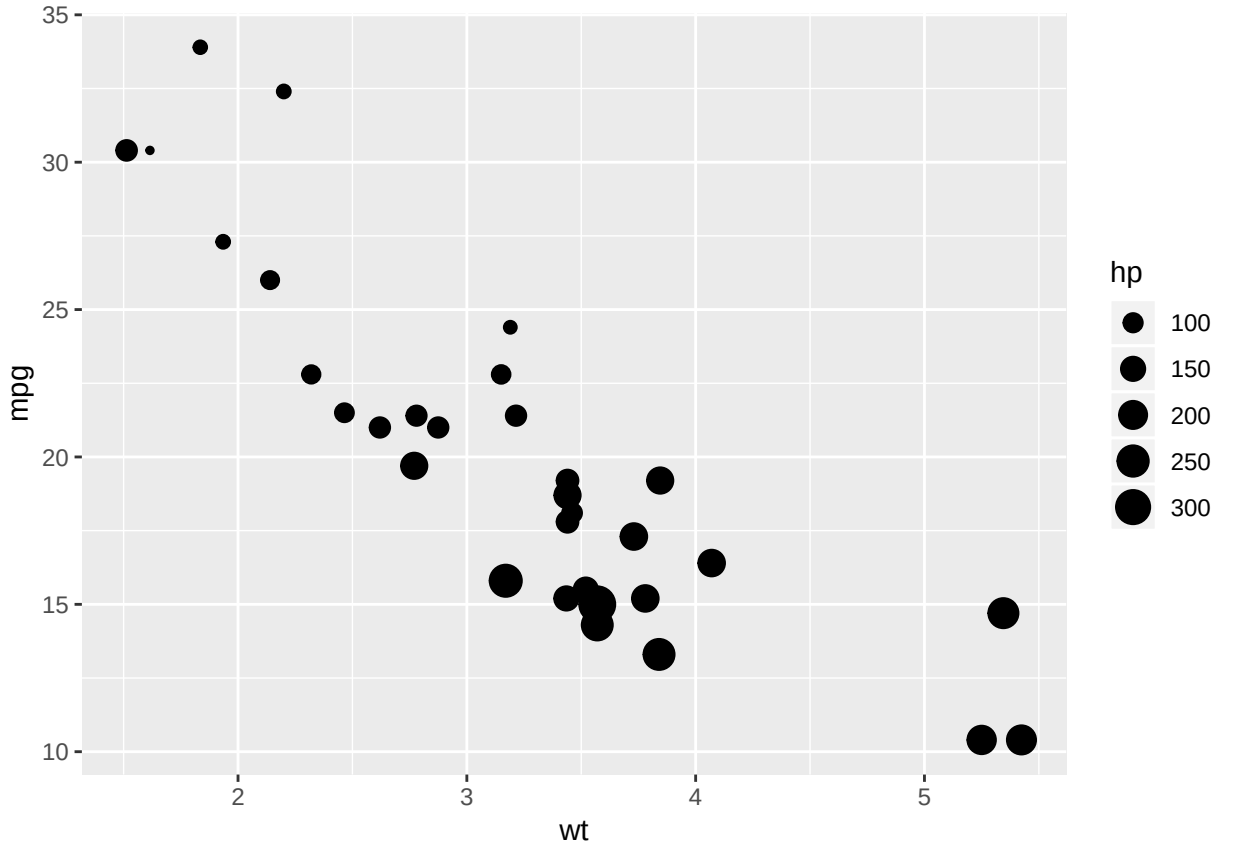
See the article by Forrest Young to learn more about interpreting biplots correctly.

9.3 Bubble charts

A bubble chart is basically just a scatterplot where the point size is proportional to the values of a third quantitative variable.

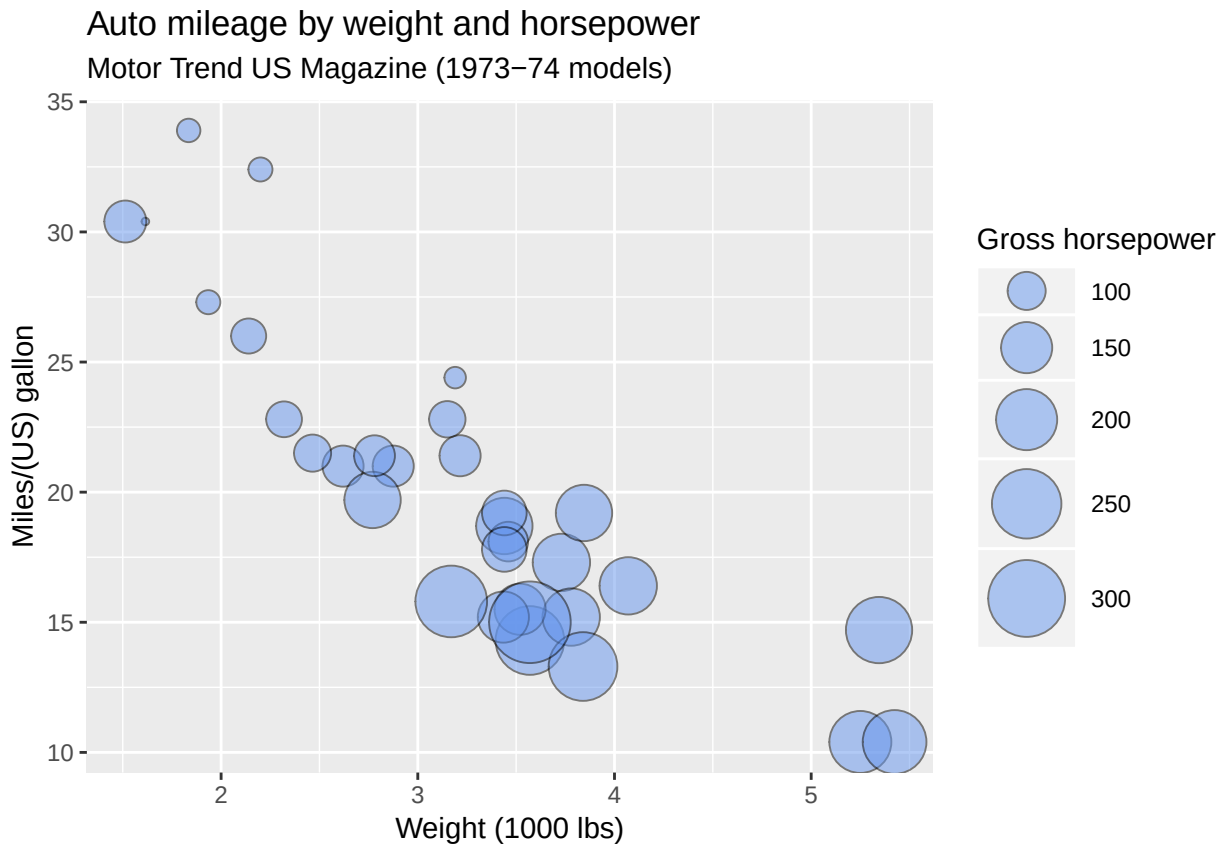
Using the mtcars dataset, let's plot car weight vs. mileage and use point size to represent horsepower.

```
# create a bubble plot
data(mtcars)
library(ggplot2)
ggplot(mtcars,
       aes(x = wt, y = mpg, size = hp)) +
  geom_point()
```



We can improve the default appearance by increasing the size of the bubbles, choosing a different point shape and color, and adding some transparency.

```
# create a bubble plot with modifications
ggplot(mtcars,
       aes(x = wt, y = mpg, size = hp)) +
  geom_point(alpha = .5,
             fill="cornflowerblue",
             color="black",
             shape=21) +
  scale_size_continuous(range = c(1, 14)) +
  labs(title = "Auto mileage by weight and horsepower",
       subtitle = "Motor Trend US Magazine (1973-74 models)",
       x = "Weight (1000 lbs)",
       y = "Miles/(US) gallon",
       size = "Gross horsepower")
```



The `range` parameter in the `scale_size_continuous` function specifies the minimum and maximum size of the plotting symbol. The default is `range = c(1, 6)`.

The `shape` option in the `geom_point` function specifies an circle with a border color and fill color.

Clearly, miles per gallon decreases with increased car weight and horsepower. However, there is one car with low weight, high horsepower, and high gas mileage. Going back to the data, it's the Lotus Europa.

Bubble charts are controversial for the same reason that pie charts are controversial. People are better at judging length than volume. However, they are quite popular.

9.4 Flow diagrams

A flow diagram represents a set of dynamic relationships. It usually captures the physical or metaphorical flow of people, materials, communications, or objects through a set of nodes in a network.

9.4.1 Sankey diagrams

In a Sankey diagram, the width of the line between two nodes is proportional to the flow amount. We'll demonstrate this with UK energy forecast data. The data contain energy production and consumption forecasts for the year 2050.

Building the graph requires two data frames, one containing node names and the second containing the links between the nodes and the amount of the flow between them.

```
# input data (data frames nodes and links)
load("Energy.RData")

# view nodes data frame
head(nodes)
```

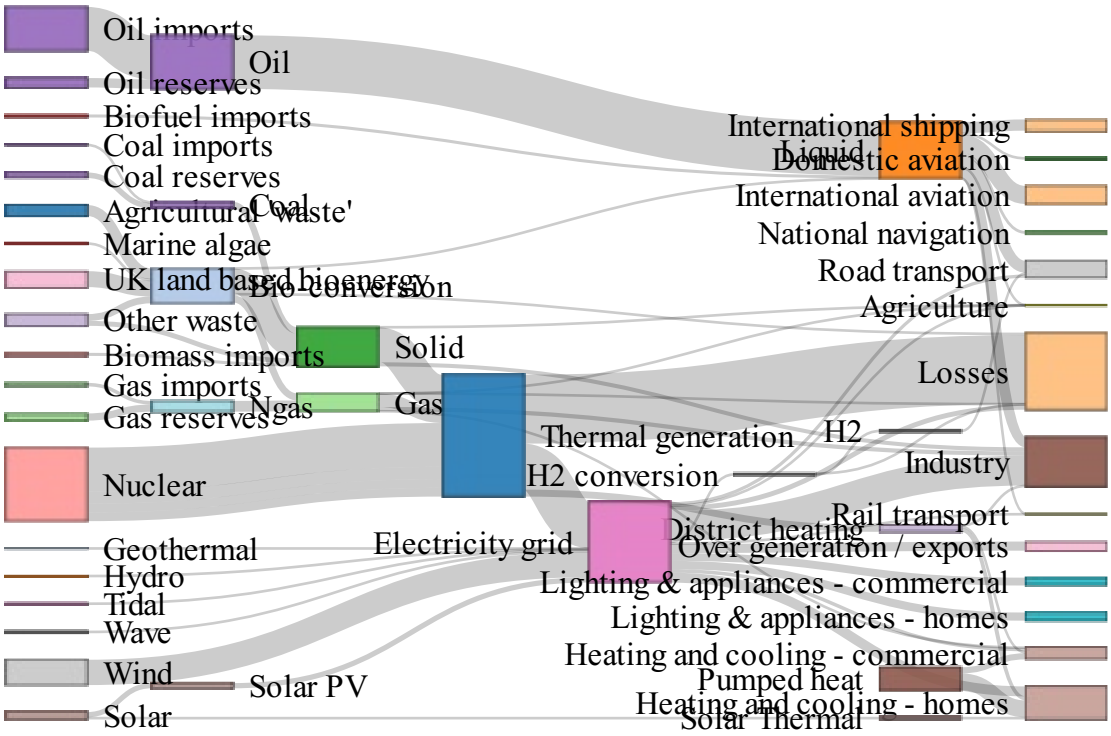
```
## # A tibble: 6 x 1
##   name
##   <chr>
## 1 Agricultural 'waste'
## 2 Bio-conversion
## 3 Liquid
## 4 Losses
## 5 Solid
## 6 Gas
```

```
# view links data frame
head(links)
```

```
## # A tibble: 6 x 3
##   source target   value
##    <int>  <int>   <dbl>
## 1      0      1 125.
## 2      1      2   0.597
## 3      1      3  26.9
## 4      1      4 280.
## 5      1      5  81.1
## 6      6      2  35.0
```

We'll build the diagram using the `sankeyNetwork` function in the `networkD3` package.

```
# create Sankey diagram
library(networkD3)
sankeyNetwork(Links = links,
              Nodes = nodes,
              Source = "source",
              Target = "target",
              Value = "value",
              NodeID = "name",
              units = "TWh", # optional units name for popups
              fontSize = 12,
              nodeWidth = 30)
```



Energy supplies are on the left and energy demands are on the right. Follow the flow from left to right. Notice that the graph is interactive (assuming you are viewing it on a web page). Try highlighting nodes and dragging them to new positions.

Sankey diagrams created with the `networkD3` package are *not* `ggplot2` graphs. Therefore, they can not be modified with `ggplot2` functions.

9.4.2 Alluvial diagrams

Alluvial diagrams are a subset of Sankey diagrams, and are more rigidly defined. A discussion of the differences can be found here.

When examining the relationship among categorical variables, alluvial diagrams can serve as alternatives to mosaic plots. In an alluvial diagram, blocks represent clusters of observations, and stream fields between the blocks represent changes to the composition of the clusters over time.

They can also be used when time is not a factor. As an example, let's diagram the survival of Titanic passengers, using the Titanic dataset.

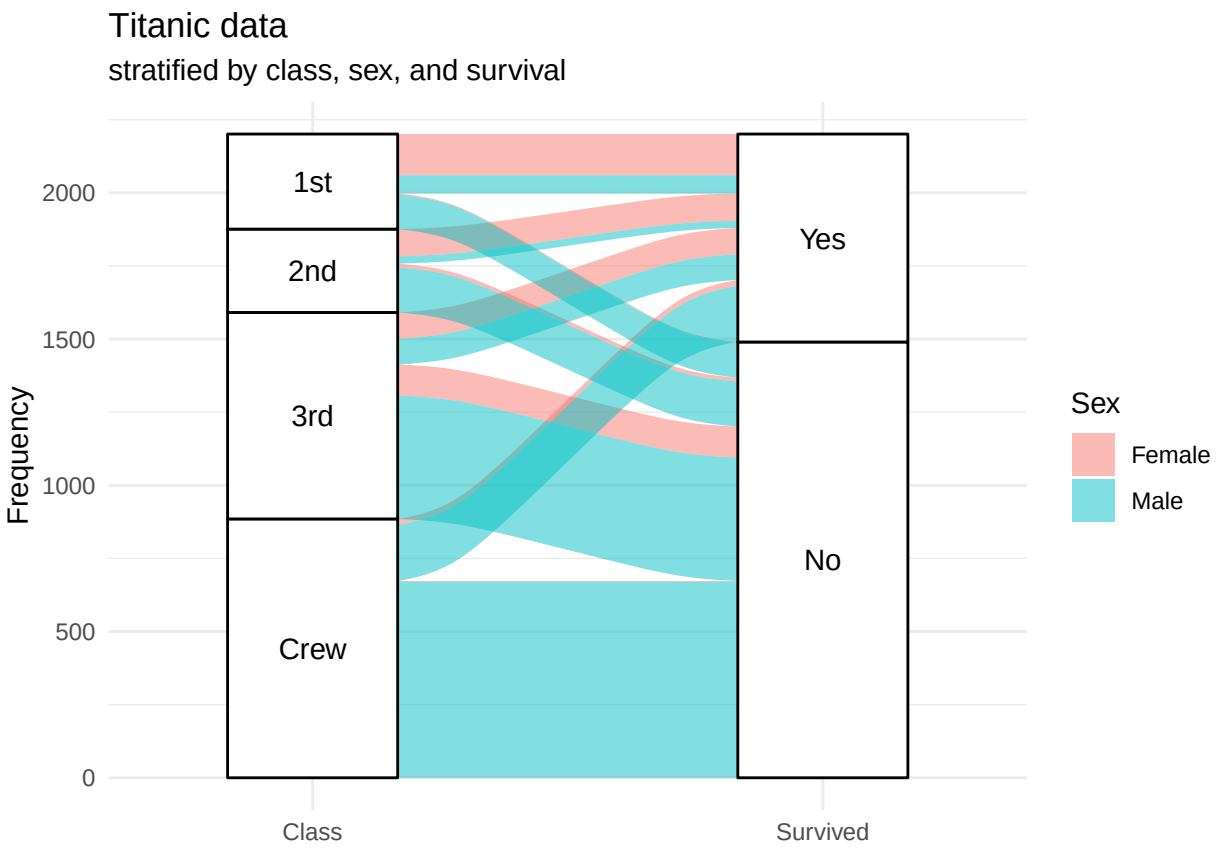
Alluvial diagrams are created with `ggalluvial` package, generating `ggplot2` graphs.

```
# input data
library(readr)
titanic <- read_csv("titanic.csv")

# summarize data
library(dplyr)
```

```
titanic_table <- titanic %>%
  group_by(Class, Sex, Survived) %>%
  count()

titanic_table$Survived <- factor(titanic_table$Survived,
                                 levels = c("Yes", "No"))

head(titanic_table)
```

```
## # A tibble: 6 x 4
## # Groups:   Class, Sex, Survived [6]
##   Class Sex    Survived     n
##   <chr> <chr>  <fct>    <int>
## 1 1st   Female No           4
## 2 1st   Female Yes        141
## 3 1st   Male   No         118
## 4 1st   Male   Yes         62
## 5 2nd   Female No          13
## 6 2nd   Female Yes         93
```

```
# create alluvial diagram
library(ggplot2)
library(ggalluvial)

ggplot(titanic_table,
       aes(axis1 = Class,
           axis2 = Survived,
           y = n)) +
  geom_alluvium(aes(fill = Sex)) +
  geom_stratum() +
  geom_text(stat = "stratum",
            label.strata = TRUE) +
  scale_x_discrete(limits = c("Class", "Survived"),
                   expand = c(.1, .1)) +
  labs(title = "Titanic data",
       subtitle = "stratified by class, sex, and survival",
       y = "Frequency") +
  theme_minimal()
```



Start at a node on the left and follow the stream field to the right. The height of the blocks represent the proportion of observations in that cluster and the height of the stream field represents the proportion of observations contained in both blocks they connect.

For example, most crew are male and do not survive. A much larger percent of 1st class females survive, than 1st class males.

Here is an alternative visualization. Survived becomes an axis and Class becomes the fill color. Colors are chosen from the viridis palette. Additionally, the legend is suppressed.

```
# create alternative alluvial diagram
library(ggplot2)
library(ggalluvial)
ggplot(titanic_table,
       aes(axis1 = Class,
           axis2 = Sex,
           axis3 = Survived,
           y = n)) +
  geom_alluvium(aes(fill = Class)) +
  geom_stratum() +
  geom_text(stat = "stratum",
            label.strata = TRUE) +
  scale_x_discrete(limits = c("Class", "Sex", "Survived"),
                   expand = c(.1, .1)) +
  scale_fill_viridis_d() +
  labs(title = "Titanic data",
       subtitle = "stratified by class, sex, and survival",
       y = "Frequency") +
```

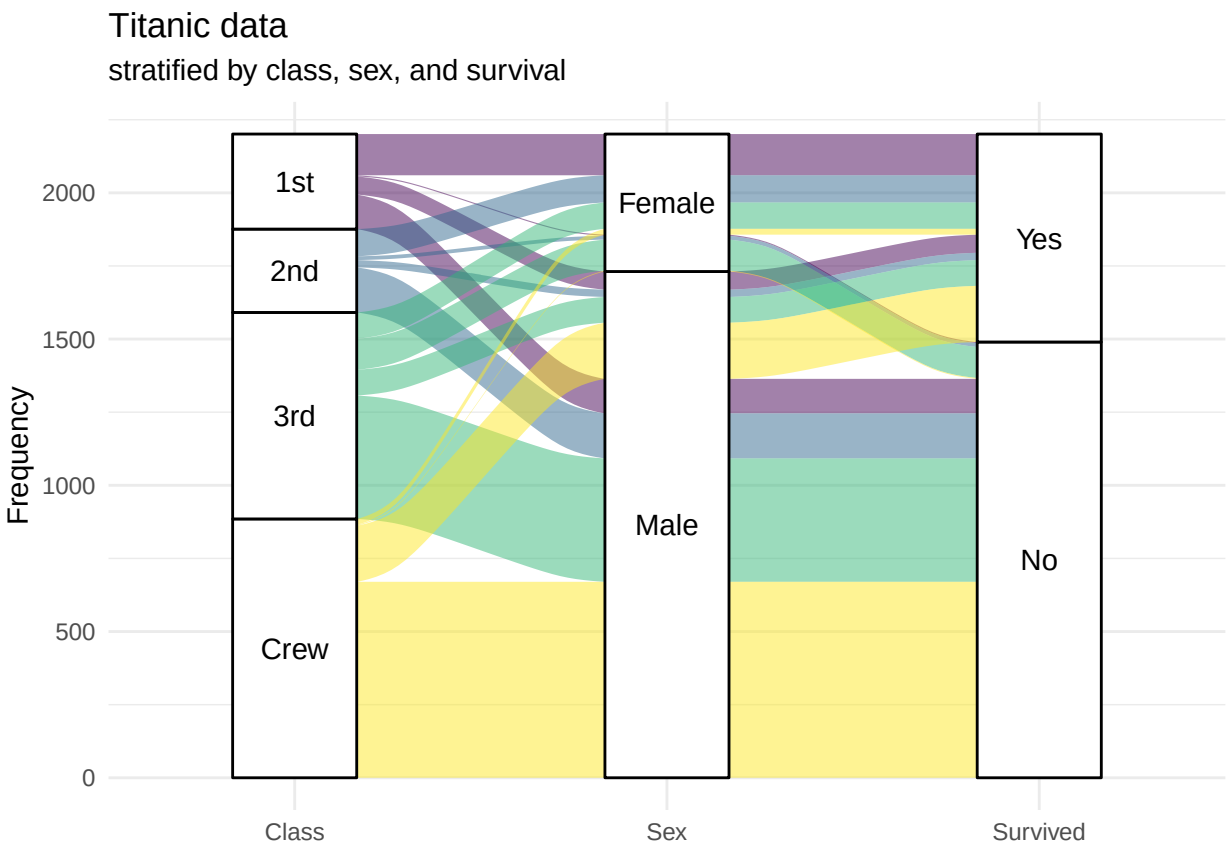


Figure 9.5: Alternative alluvial diagram

```
  theme_minimal() +
  theme(legend.position = "none")
```

I think that this version is a bit easier to follow.

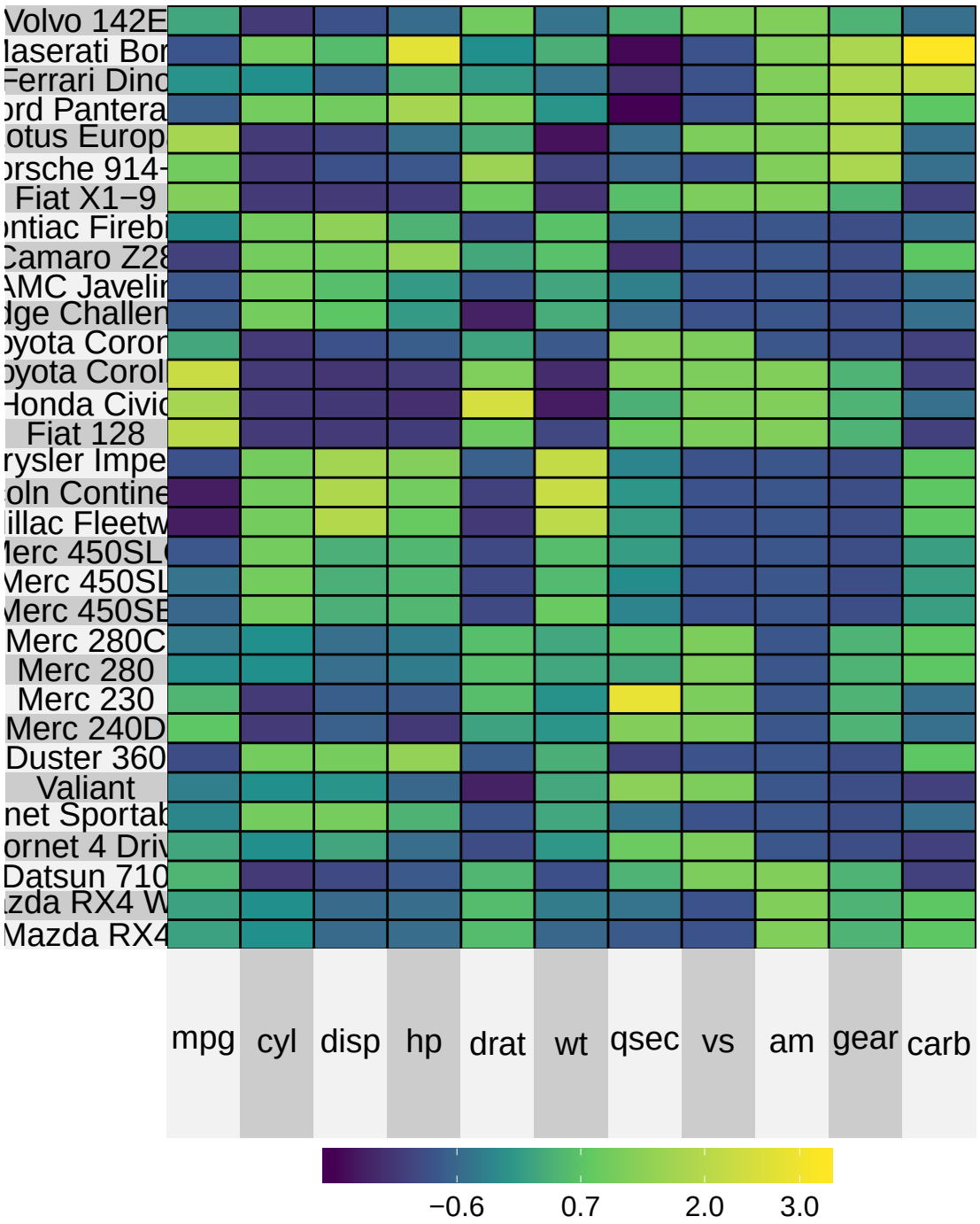
See the ggalluvial website for additional details.

9.5 Heatmaps

A heatmap displays a set of data using colored tiles for each variable value within each observation. There are many varieties of heatmaps. Although base R comes with a `heatmap` function, we'll use the more powerful `superheat` package (I love these names).

First, let's create a heatmap for the `mtcars` dataset that come with base R. The `mtcars` dataset contains information on 32 cars measured on 11 variables.

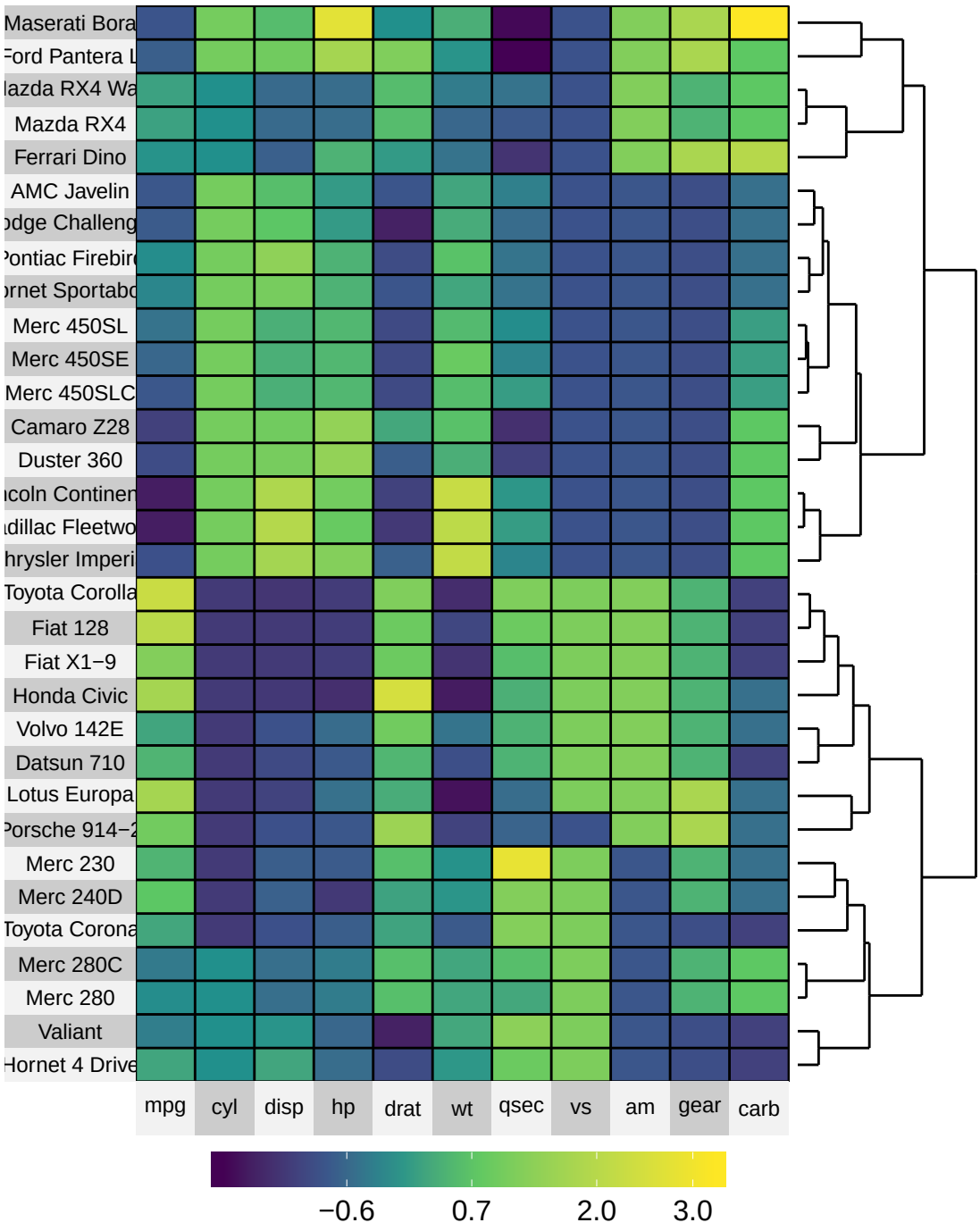
```
# create a heatmap
data(mtcars)
library(superheat)
superheat(mtcars, scale = TRUE)
```



The `scale = TRUE` options standardizes the columns to a mean of zero and standard deviation of one. Looking at the graph, we can see that the Merc 230 has a quarter mile time (*qsec*) the is well above average (bright yellow). The Lotus Europa has a weight is well below average (dark blue).

We can use clustering to sort the rows and/or columns. In the next example, we'll sort the rows so that cars that are similar appear near each other. We will also adjust the text and label sizes.

```
# sorted heat map
superheat(mtcars,
          scale = TRUE,
          left.label.text.size=3,
          bottom.label.text.size=3,
          bottom.label.size = .05,
          row.dendrogram = TRUE )
```



Here we can see that the Toyota Corolla and Fiat 128 have similar characteristics. The Lincoln Continental and Cadillac Fleetwood also have similar characteristics.

The `superheat` function requires that the data be in particular format. Specifically

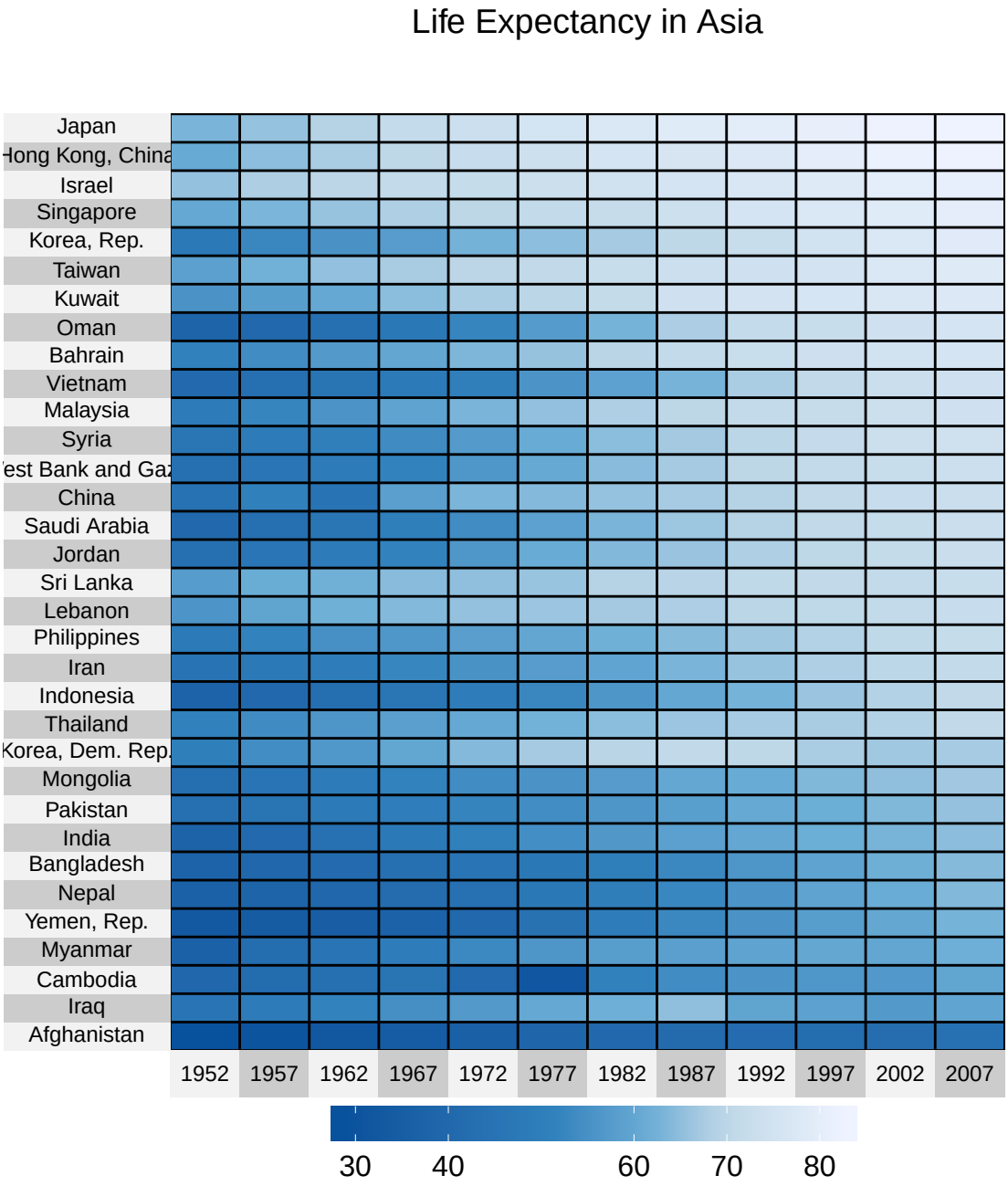
- the data most be all numeric

- the row names are used to label the left axis. If the desired labels are in a column variable, the variable must be converted to row names (more on this below)
- missing values are allowed

Let's use a heatmap to display changes in life expectancies over time for Asian countries. The data come from the `gapminder` dataset.

Since the data is in long format, we first have to convert to wide format. Then we need to ensure that it is a data frame and convert the variable *country* into row names. Finally, we'll sort the data by 2007 life expectancy. While we are at it, let's change the color scheme.

```
# create heatmap for gapminder data (Asia)
library(tidyr)
library(dplyr)

# load data
data(gapminder, package="gapminder")

# subset Asian countries
asia <- gapminder %>%
  filter(continent == "Asia") %>%
  select(year, country, lifeExp)

# convert to long to wide format
plotdata <- spread(asia, year, lifeExp)

# save country as row names
plotdata <- as.data.frame(plotdata)
row.names(plotdata) <- plotdata$country
plotdata$country <- NULL

# row order
sort.order <- order(plotdata$"2007")

# color scheme
library(RColorBrewer)
colors <- rev(brewer.pal(5, "Blues"))


# create the heat map
superheat(plotdata,
          scale = FALSE,
          left.label.text.size=3,
          bottom.label.text.size=3,
          bottom.label.size = .05,
          heat.pal = colors,
          order.rows = sort.order,
          title = "Life Expectancy in Asia")
```



Japan, Hong Kong, and Israel have the highest life expectancies. South Korea was doing well in the 80s but has lost some ground. Life expectancy in Cambodia took a sharp hit in 1977.

To see what you can do with heat maps, see the extensive `superheat` vignette.

9.6 Radar charts

A radar chart (also called a spider or star chart) displays one or more groups or observations on three or more quantitative variables.

In the example below, we'll compare dogs, pigs, and cows in terms of body size, brain size, and sleep characteristics (total sleep time, length of sleep cycle, and amount of REM sleep). The data come from the Mammal Sleep dataset.

Radar charts can be created with `ggradar` function in the `ggradar` package. Unfortunately, the package in not available on CRAN, so we have to install it from Github.

```
install.packages("devtools")
devtools::install_github("ricardo-bion/ggradar")
```

Next, we have to put the data in a specific format:

- The first variable should be called *group* and contain the identifier for each observation
- The numeric variables have to be rescaled so that their values range from 0 to 1

```
# create a radar chart

# prepare data
data(msleep, package = "ggplot2")
library(ggradar)
library(scales)
library(dplyr)

plotdata <- msleep %>%
  filter(name %in% c("Cow", "Dog", "Pig")) %>%
  select(name, sleep_total, sleep_rem,
         sleep_cycle, brainwt, bodywt) %>%
  rename(group = name) %>%
  mutate_at(vars(-group),
            funs(rescale))
plotdata

# generate radar chart
ggradar(plotdata,
        grid.label.size = 4,
        axis.label.size = 4,
        group.point.size = 5,
        group.line.width = 1.5,
        legend.text.size= 10) +
  labs(title = "Mammals, size, and sleep")
```

In the previous chart, the `mutate_at` function rescales all variables except *group*. The various `size` options control the font sizes for the percent labels, variable names, point size, line width, and legend labels respectively.

We can see from the chart that, relatively speaking, cows have large brain and body weights, long sleep cycles, short total sleep time and little time in REM sleep. Dogs in comparison, have small body and brain weights, short sleep cycles, and a large total sleep time and time in REM sleep (The obvious conclusion is that I want to be a dog - but with a bigger brain).

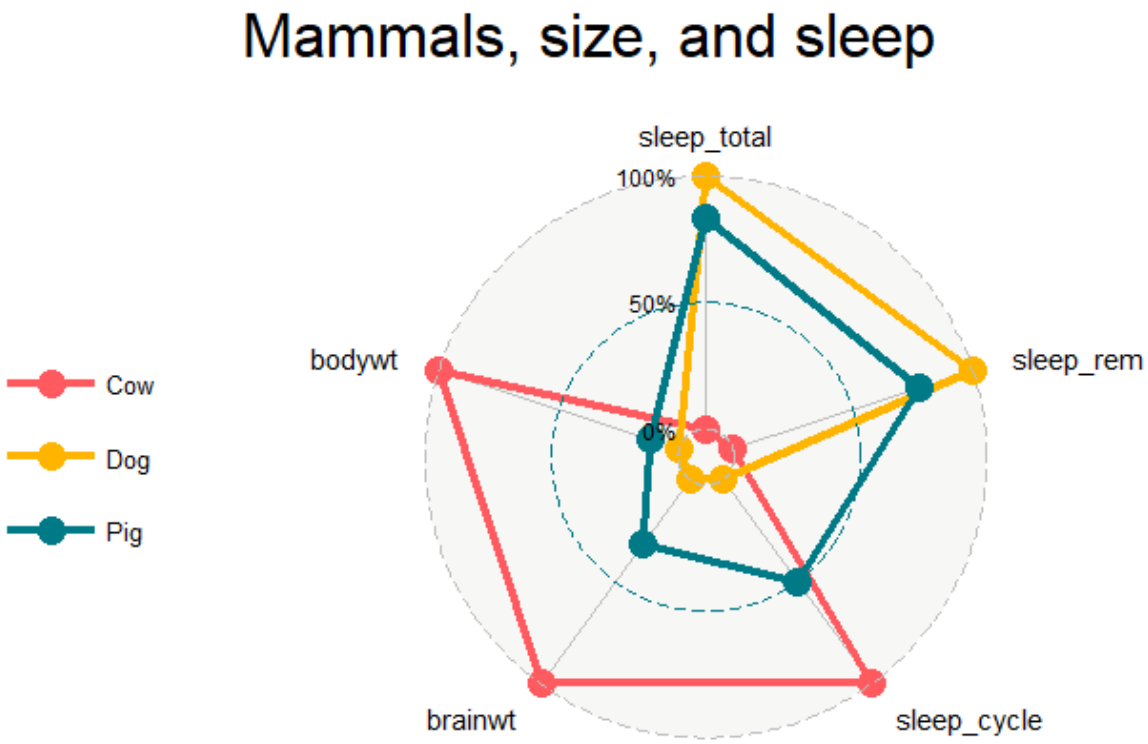


Figure 9.6: Basic radar chart

9.7 Scatterplot matrix

A scatterplot matrix is a collection of scatterplots organized as a grid. It is similar to a correlation plot but instead of displaying correlations, displays the underlying data.

You can create a scatterplot matrix using the `ggpairs` function in the `GGally` package.

We can illustrate its use by examining the relationships between mammal size and sleep characteristics. The data come from the `msleep` dataset that ships with `ggplot2`. Brain weight and body weight are highly skewed (think mouse and elephant) so we'll transform them to log brain weight and log body weight before creating the graph.

```
library(GGally)

# prepare data
data(msleep, package="ggplot2")
library(dplyr)
df <- msleep %>%
  mutate(log_brainwt = log(brainwt),
         log_bodywt = log(bodywt)) %>%
  select(log_brainwt, log_bodywt, sleep_total, sleep_rem)


# create a scatterplot matrix
ggpairs(df)
```

By default,

- the principal diagonal contains the kernel density charts for each variable.
- The cells below the principal diagonal contain the scatterplots represented by the intersection of the row and column variables. The variables across the top are the x-axes and the variables down the right side are the y-axes.
- The cells above the principal diagonal contain the correlation coefficients.

For example, as brain weight increases, total sleep time and time in REM sleep decrease.

The graph can be modified by creating custom functions.

```
# custom function for density plot
my_density <- function(data, mapping, ...){
  ggplot(data = data, mapping = mapping) +
    geom_density(alpha = 0.5,
                 fill = "cornflowerblue", ...)
}

# custom function for scatterplot
my_scatter <- function(data, mapping, ...){
  ggplot(data = data, mapping = mapping) +
    geom_point(alpha = 0.5,
               color = "cornflowerblue") +
    geom_smooth(method=lm,
                se=FALSE, ...)
```

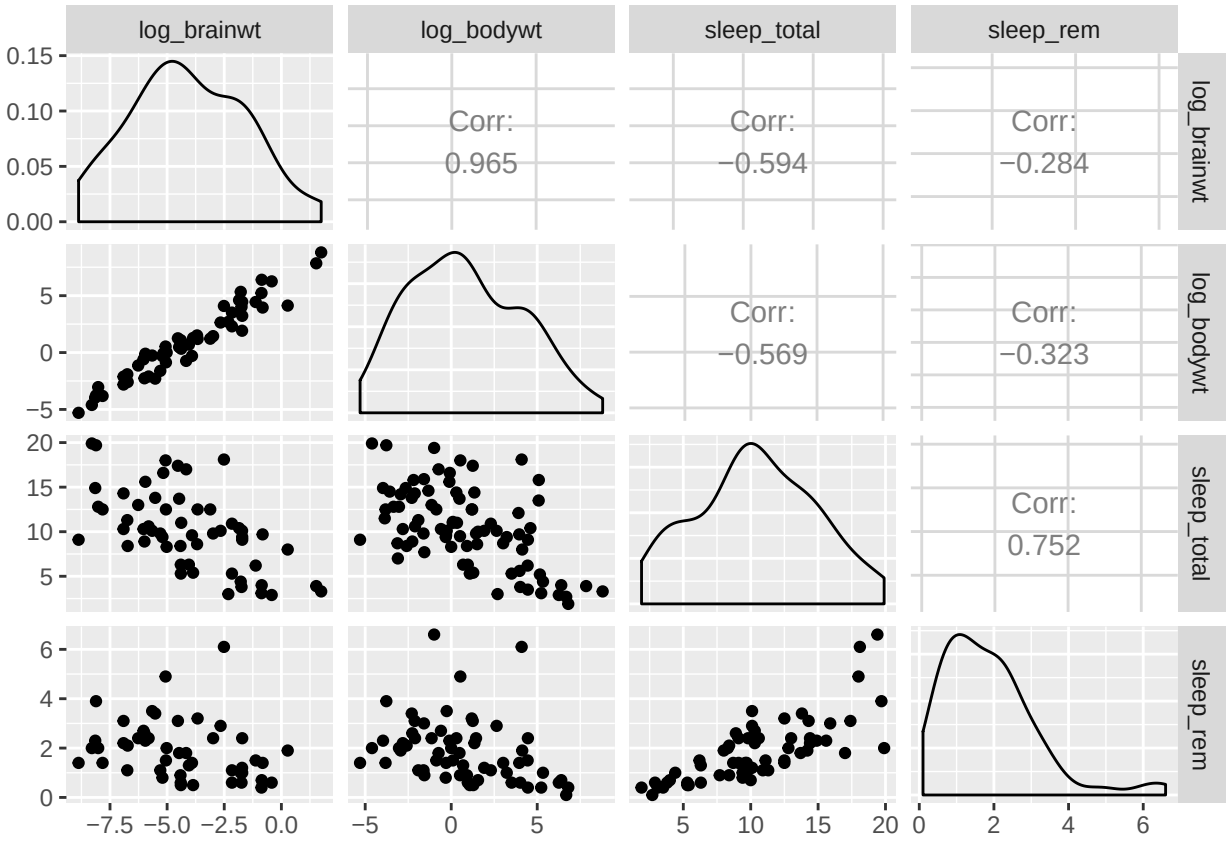


Figure 9.7: Scatterplot matrix

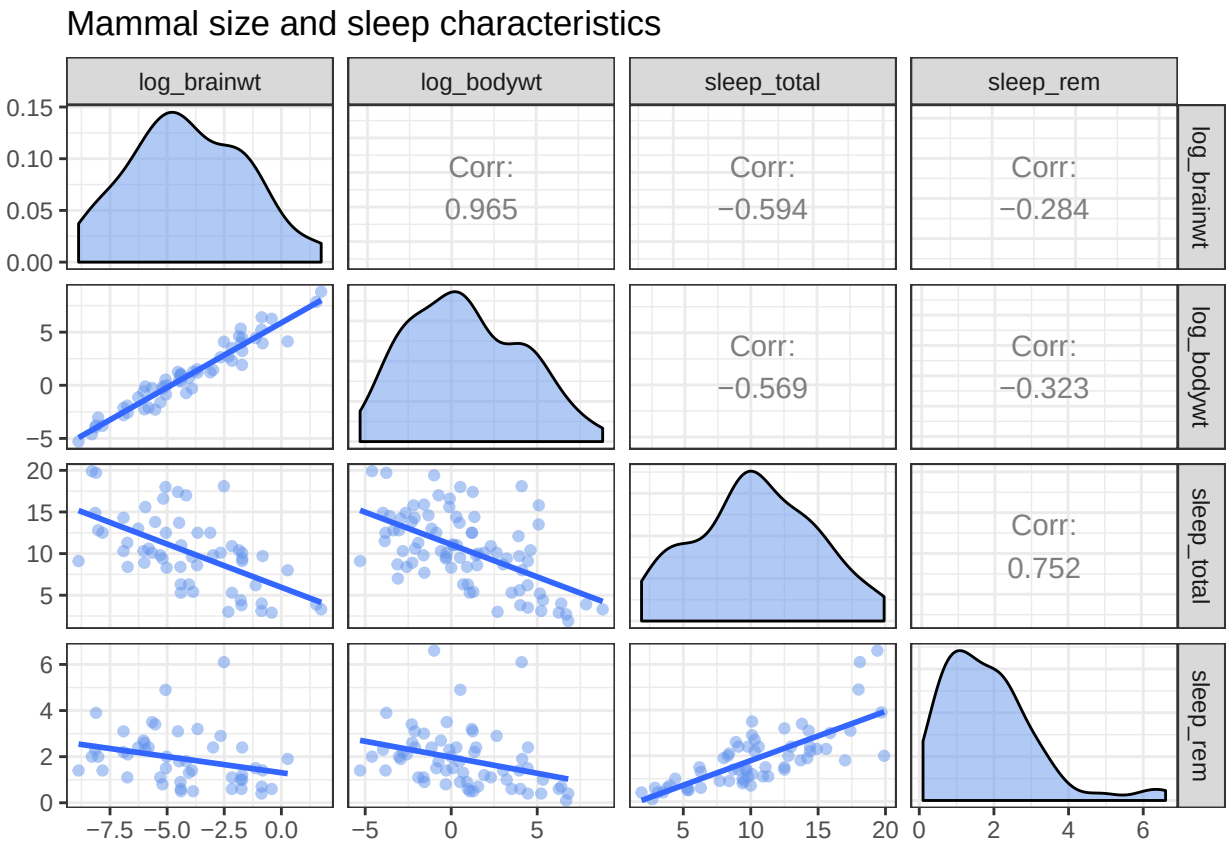


Figure 9.8: Customized scatterplot matrix

```
}


# create scatterplot matrix
ggpairs(df,
        lower=list(continuous = my_scatter),
        diag = list(continuous = my_density)) +
  labs(title = "Mammal size and sleep characteristics") +
  theme_bw()
```

Being able to write your own functions provides a great deal of flexibility. Additionally, since the resulting plot is a `ggplot2` graph, addition functions can be added to alter the theme, title, labels, etc. See the help for more details.

9.8 Waterfall charts

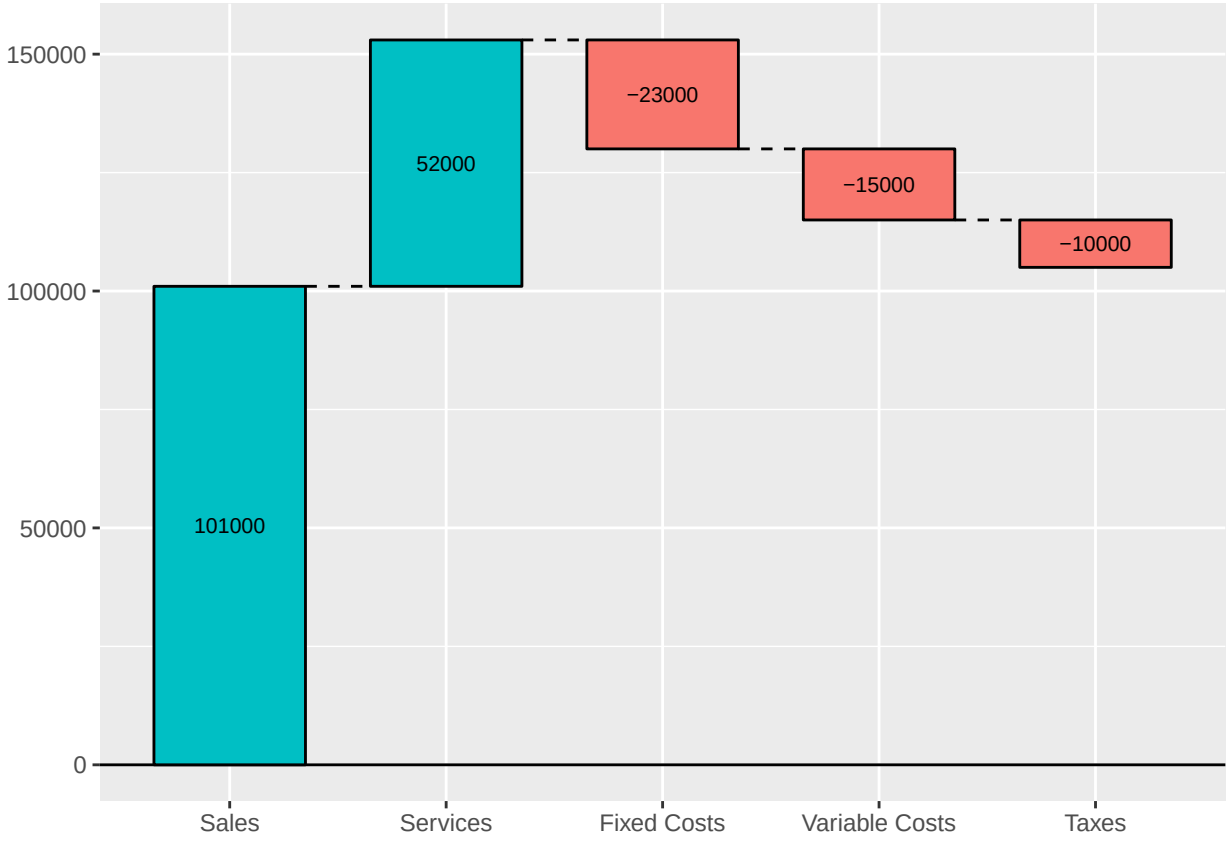
A waterfall chart illustrates the cumulative effect of a sequence of positive and negative values.

For example, we can plot the cumulative effect of revenue and expenses for a fictional company. First, let's create a dataset

```
# create company income statement
category <- c("Sales", "Services", "Fixed Costs",
              "Variable Costs", "Taxes")
amount <- c(101000, 52000, -23000, -15000, -10000)
income <- data.frame(category, amount)
```

Now we can visualize this with a waterfall chart, using the `waterfall` function in the `waterfalls` package.

```
# create waterfall chart
library(ggplot2)
library(waterfalls)
waterfall(income)
```



We can also add a total (net) column. Since the result is a `ggplot2` graph, we can use additional functions to customize the results.

```
# create waterfall chart with total column
waterfall(income,
          calc_total=TRUE,
          total_axis_text = "Net",
          total_rect_text_color="black",
          total_rect_color="goldenrod1") +
  scale_y_continuous(label=scales::dollar) +
  labs(title = "West Coast Profit and Loss",
       subtitle = "Year 2017",
       y="",
```

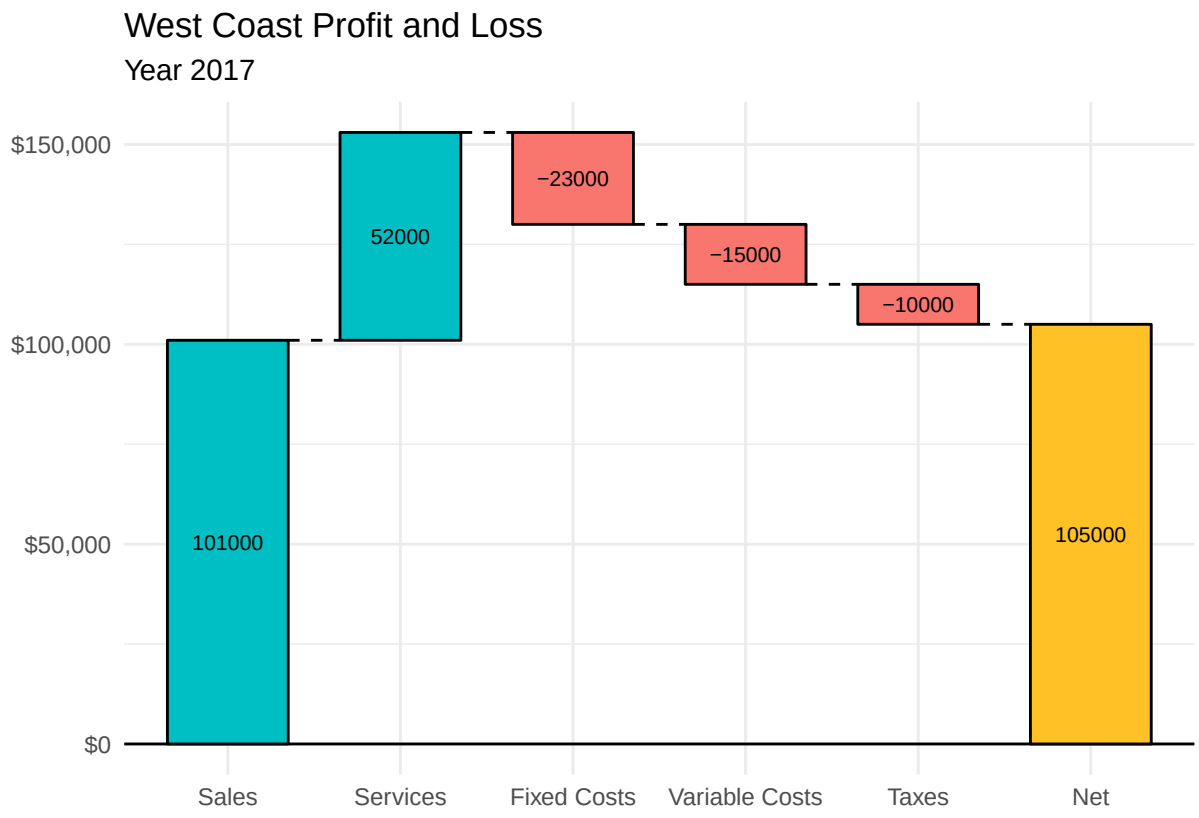


Figure 9.9: Waterfall chart with total column

```
       x="") +
  theme_minimal()
```

9.9 Word clouds

A word cloud (also called a tag cloud), is basically an infographic that indicates the frequency of words in a collection of text (e.g., tweets, a text document, a set of text documents). There is a very nice script produced by STHDA that will generate a word cloud directly from a text file.

To demonstrate, we'll use President Kennedy's Address during the Cuban Missile crisis.

To use the script, there are several packages you need to install first.

```
# install packages for text mining
install.packages(c("tm", "SnowballC",
                   "wordcloud", "RColorBrewer",
                   "RCurl", "XML")
```

Once the packages are installed, you can run the script on your text file.

```
# create a word cloud
script <- "http://www.sthda.com/upload/rquery_wordcloud.r"
source(script)
res<-rquery.wordcloud("JFKspeech.txt",
                      type ="file",
                      lang = "english")
```

As you can see, the most common words in the speech are *soviet*, *cuba*, *world*, *weapons*, etc. The terms *missle* and *ballistic* are used rarely. See the rquery.wordcloud page, for more details.

Chapter 10

Customizing Graphs

Graph defaults are fine for quick data exploration, but when you want to publish your results to a blog, paper, article or poster, you'll probably want to customize the results. Customization can improve the clarity and attractiveness of a graph.

This chapter describes how to customize a graph's axes, gridlines, colors, fonts, labels, and legend. It also describes how to add annotations (text and lines).

10.1 Axes

The x-axis and y-axis represent numeric, categorical, or date values. You can modify the default scales and labels with the functions below.

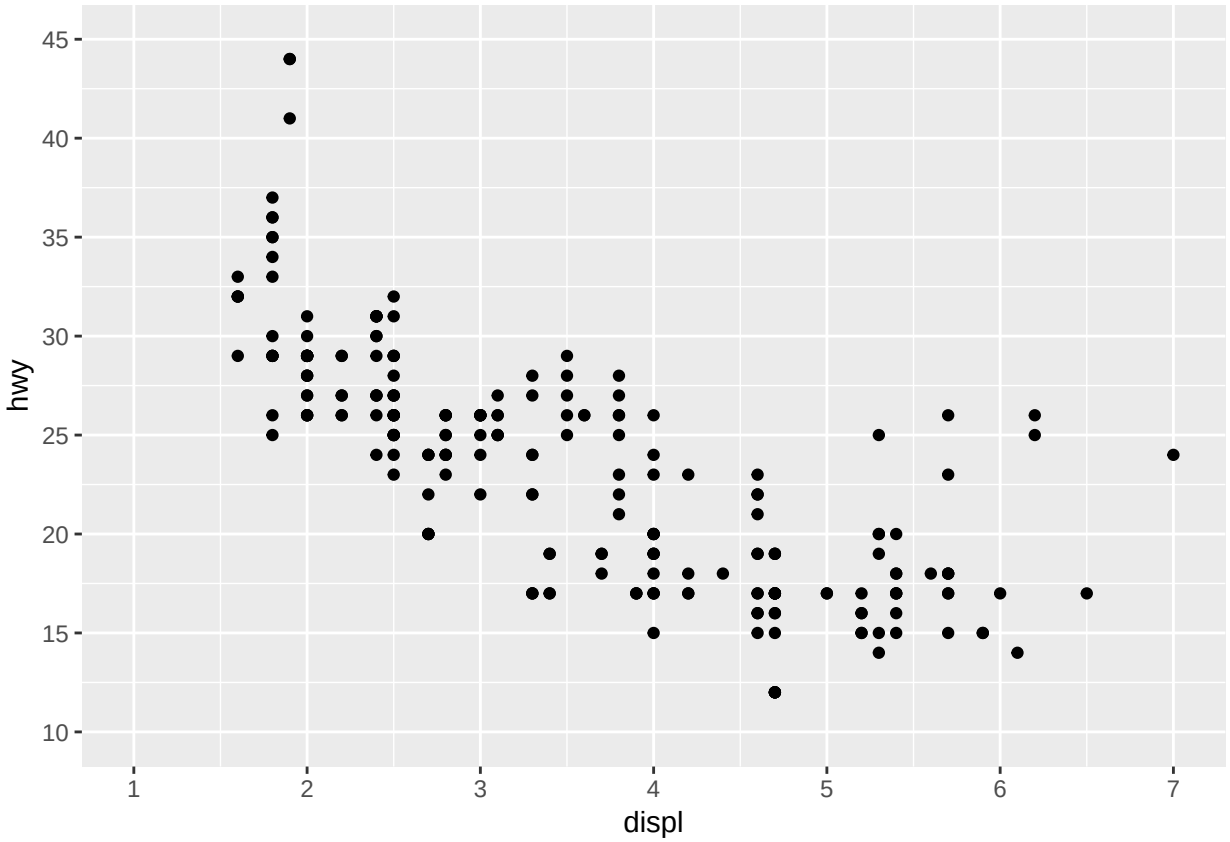
10.1.1 Quantitative axes

A quantitative axis is modified using the `scale_x_continuous` or `scale_y_continuous` function.

Options include

- `breaks` - a numeric vector of positions
- `limits` - a numeric vector with the min and max for the scale

```
# customize numerical x and y axes
library(ggplot2)
ggplot(mpg, aes(x=displ, y=hwy)) +
  geom_point() +
  scale_x_continuous(breaks = seq(1, 7, 1),
                     limits=c(1, 7)) +
  scale_y_continuous(breaks = seq(10, 45, 5),
                     limits=c(10, 45))
```



Numeric formats

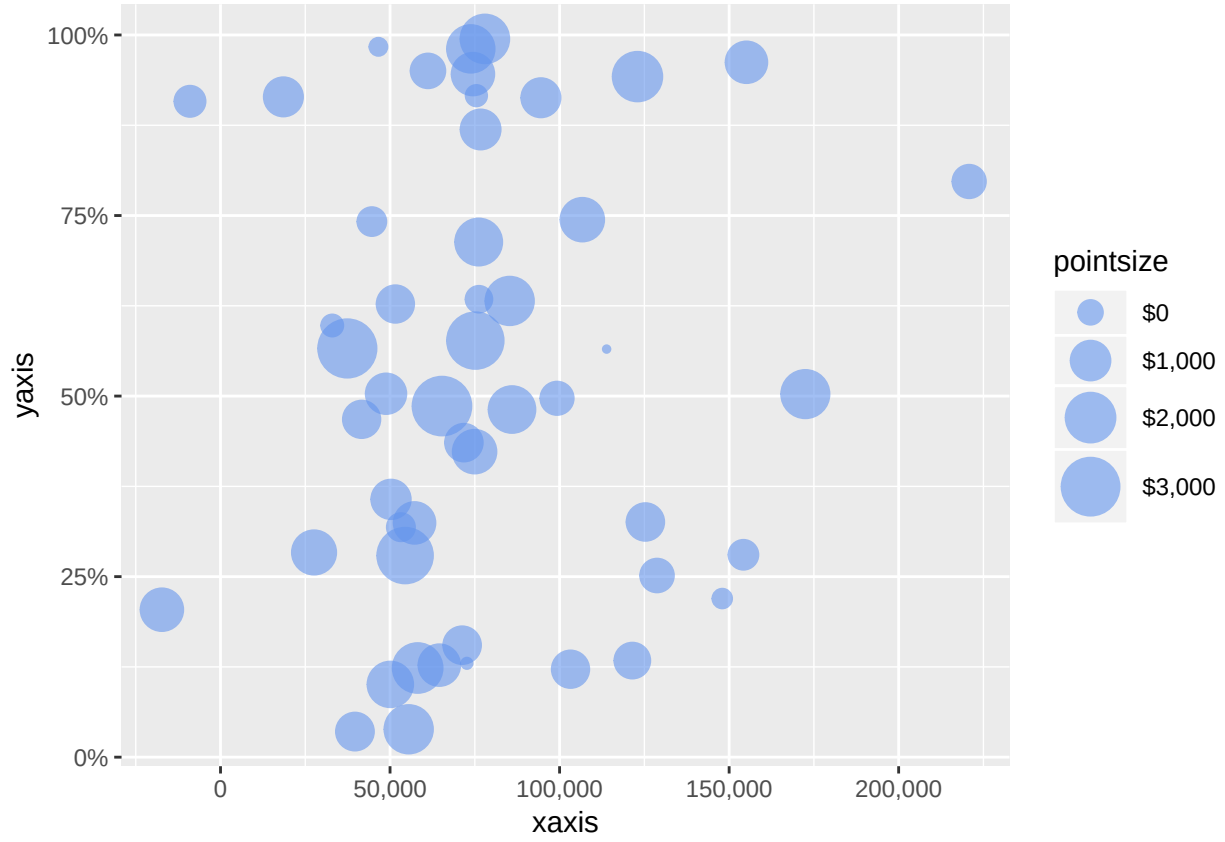
The `scales` package provides a number of functions for formatting numeric labels. Some of the most useful are

- `dollar`
- `comma`
- `percent`

Let's demonstrate these functions with some synthetic data.

```
# create some data
set.seed(1234)
df <- data.frame(xaxis = rnorm(50, 100000, 50000),
                 yaxis = runif(50, 0, 1),
                 pointsize = rnorm(50, 1000, 1000))
library(ggplot2)

# plot the axes and legend with formats
ggplot(df, aes(x = xaxis,
               y = yaxis,
               size=pointsize)) +
  geom_point(color = "cornflowerblue",
             alpha = .6) +
  scale_x_continuous(label = scales::comma) +
```

```
  scale_y_continuous(label = scales::percent) +
  scale_size(range = c(1,10), # point size range
             label = scales::dollar)
```



To format currency values as euros, you can use

`label = scales::dollar_format(prefix = "", suffix = "\u20ac")`.

10.1.2 Categorical axes

A categorical axis is modified using the `scale_x_discrete` or `scale_y_discrete` function.

Options include

- `limits` - a character vector (the levels of the quantitative variable in the desired order)
- `labels` - a character vector of labels (optional labels for these levels)

```
library(ggplot2)
# customize categorical x axis
ggplot(mpg, aes(x = class)) +
  geom_bar(fill = "steelblue") +
  scale_x_discrete(limits = c("pickup", "suv", "minivan",
                              "midsize", "compact", "subcompact",
                              "2seater"),
                   labels = c("Pickup\nTruck", "Sport Utility\nVehicle",
```

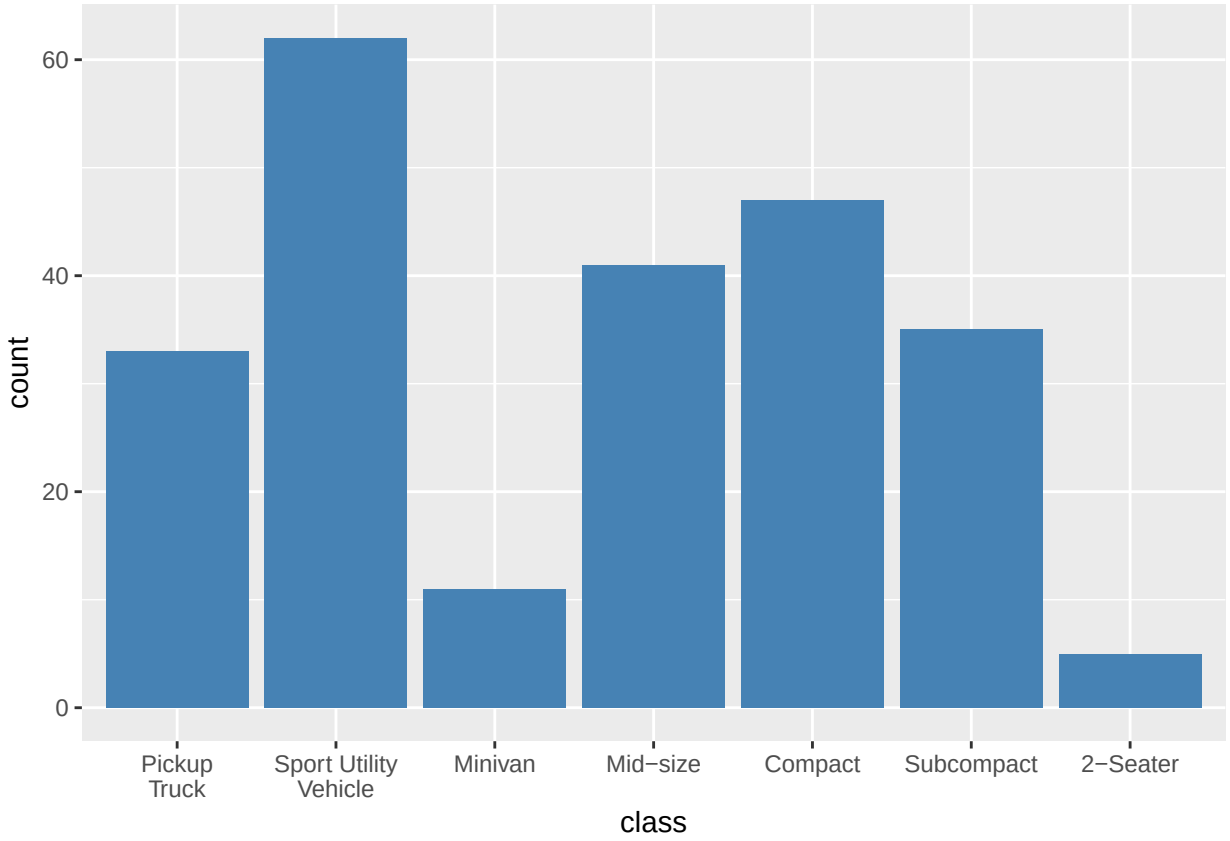


Figure 10.1: Customized categorical axis

```
                        "Minivan", "Mid-size", "Compact",
                        "Subcompact", "2-Seater"))
```

10.1.3 Date axes

A date axis is modified using the `scale_x_date` or `scale_y_date` function.

Options include

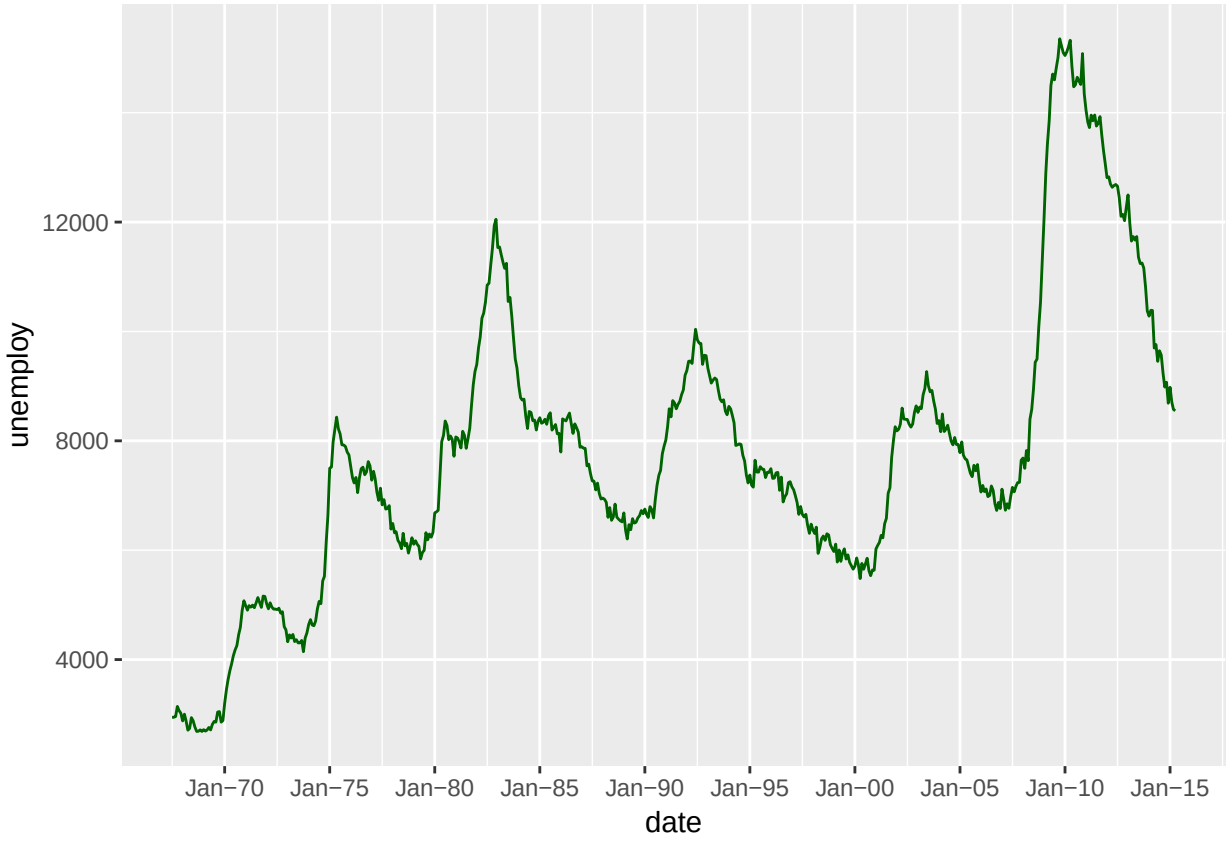
- `date_breaks` - a string giving the distance between breaks like “2 weeks” or “10 years”
- `date_labels` - A string giving the formatting specification for the labels

The table below gives the formatting specifications for date values.

Symbol	Meaning	Example
%d	day as a number (0-31)	01-31
%a	abbreviated weekday	Mon
%A	unabbreviated weekday	Monday
%m	month (00-12)	00-12
%b	abbreviated month	Jan
%B	unabbreviated month	January

Symbol	Meaning	Example
%y	2-digit year	07
%Y	4-digit year	2007

```
library(ggplot2)
# customize date scale on x axis
ggplot(economics, aes(x = date, y = unemploy)) +
  geom_line(color="darkgreen") +
  scale_x_date(date_breaks = "5 years",
               date_labels = "%b-%y")
```



Here is a help sheet for modifying scales developed from the online help.

10.2 Colors

The default colors in `ggplot2` graphs are functional, but often not as visually appealing as they can be. Happily this is easy to change.

Specific colors can be

- specified for points, lines, bars, areas, and text, or
- mapped to the levels of a variable in the dataset.

10.2.1 Specifying colors manually

To specify a color for points, lines, or text, use the `color = "colorname"` option in the appropriate geom. To specify a color for bars and areas, use the `fill = "colorname"` option.

Examples:

- `geom_point(color = "blue")`
- `geom_bar(fill = "steelblue")`

Colors can be specified by name or hex code.

To assign colors to the levels of a variable, use the `scale_color_manual` and `scale_fill_manual` functions. The former is used to specify the colors for points and lines, while the later is used for bars and areas.

Here is an example, using the `diamonds` dataset that ships with `ggplot2`. The dataset contains the prices and attributes of 54,000 round cut diamonds.

```
# specify fill color manually
library(ggplot2)
ggplot(diamonds, aes(x = cut, fill = clarity)) +
  geom_bar() +
  scale_fill_manual(values = c("darkred", "steelblue",
                               "darkgreen", "gold",
                               "brown", "purple",
                               "grey", "khaki4"))
```

If you are aesthetically challenged like me, an alternative is to use a predefined palette.

10.2.2 Color palettes

There are *many* predefined color palettes available in R.

10.2.2.1 RColorBrewer

The most popular alternative palettes are probably the ColorBrewer palettes.

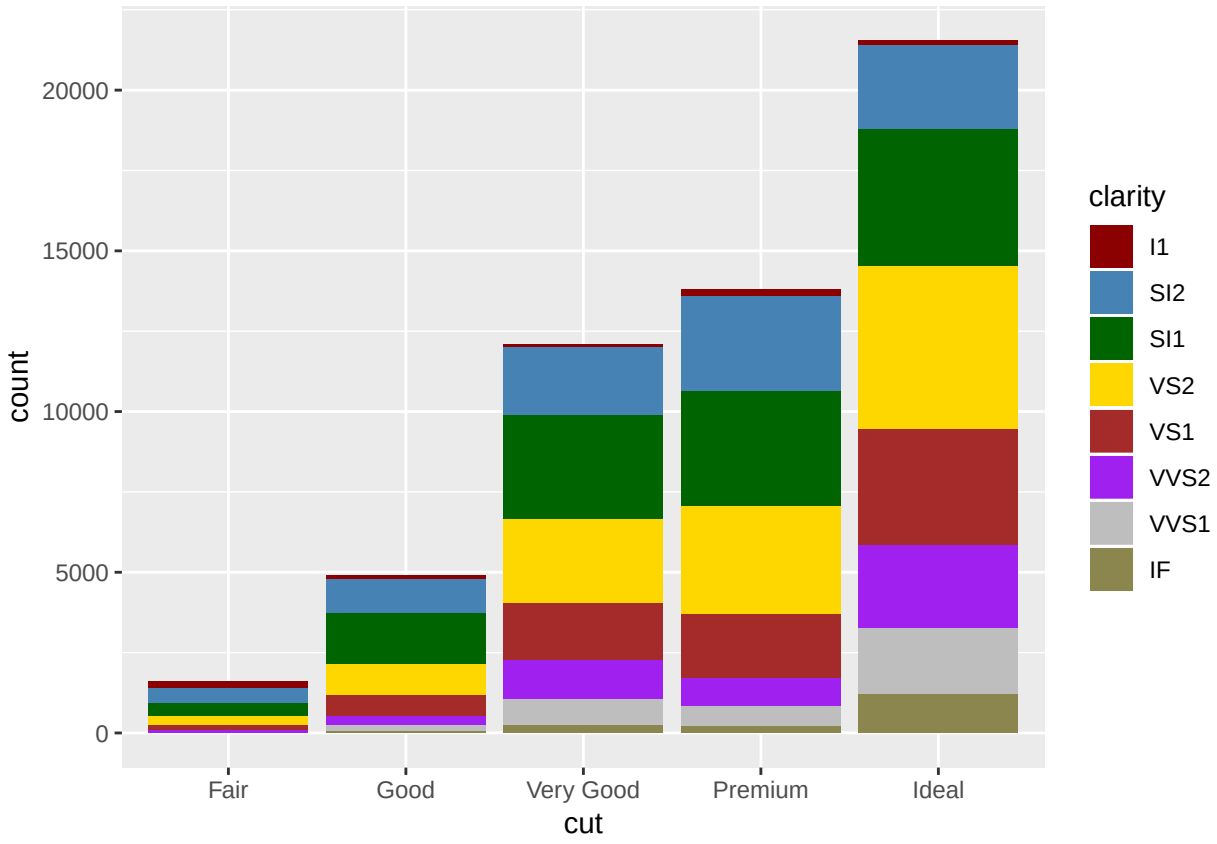
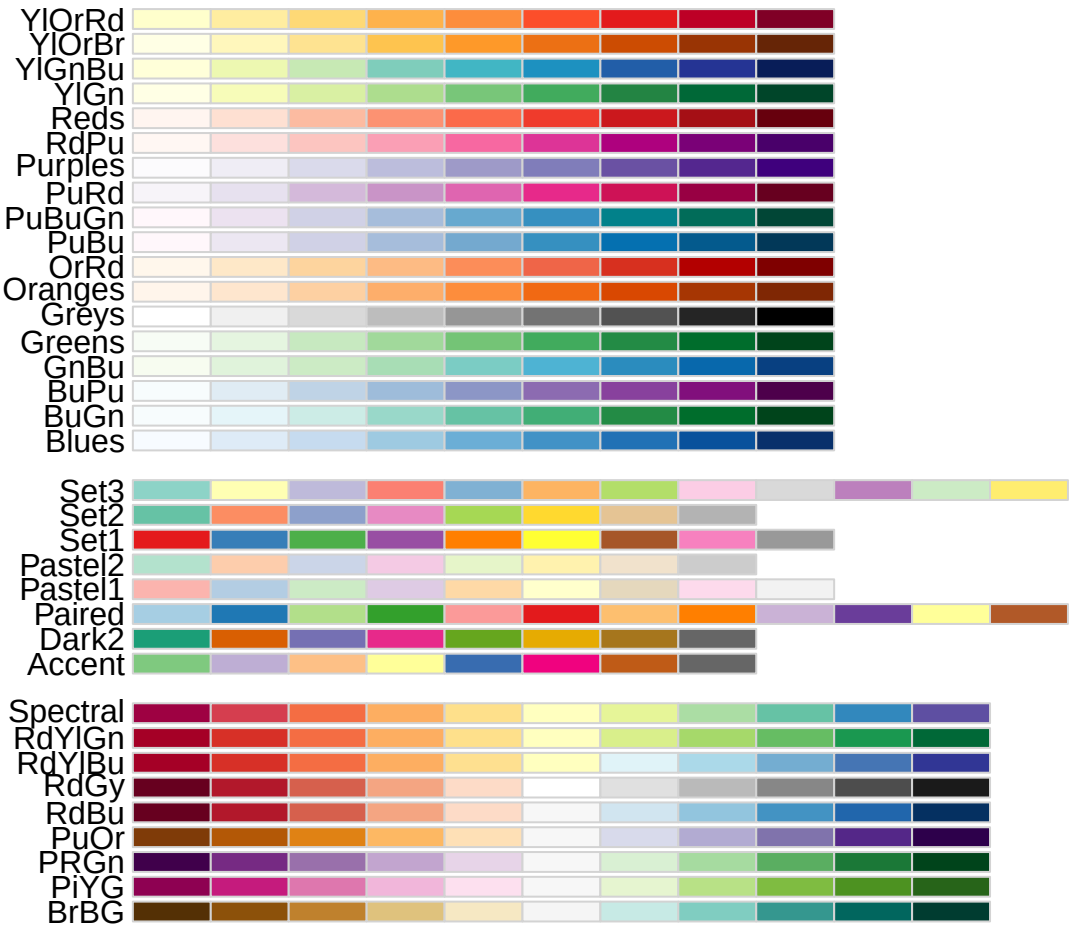


Figure 10.2: Manual color selection



You can specify these palettes with the `scale_color_brewer` and `scale_fill_brewer` functions.

```
# use an ColorBrewer fill palette
ggplot(diamonds, aes(x = cut, fill = clarity)) +
  geom_bar() +
  scale_fill_brewer(palette = "Dark2")
```

Adding `direction = -1` to these functions reverses the order of the colors in a palette.

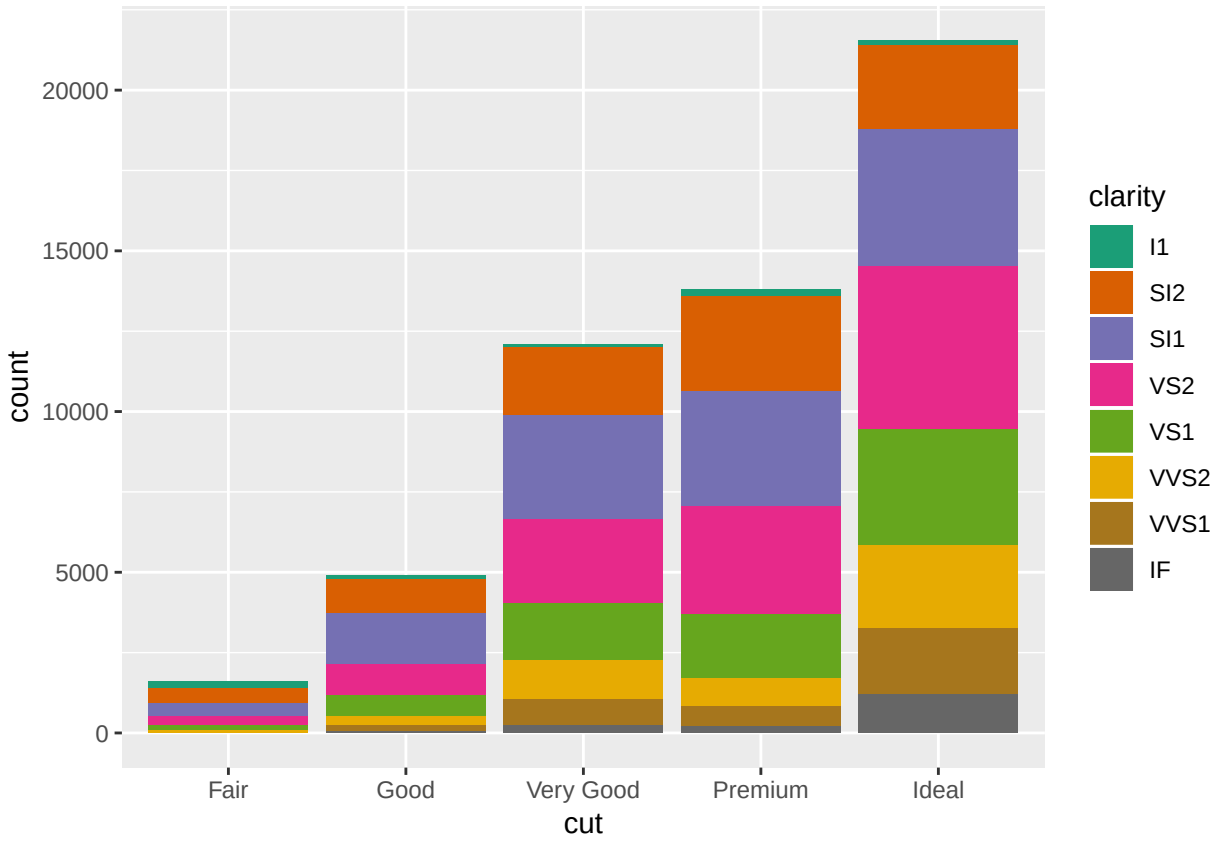


Figure 10.3: Using RColorBrewer

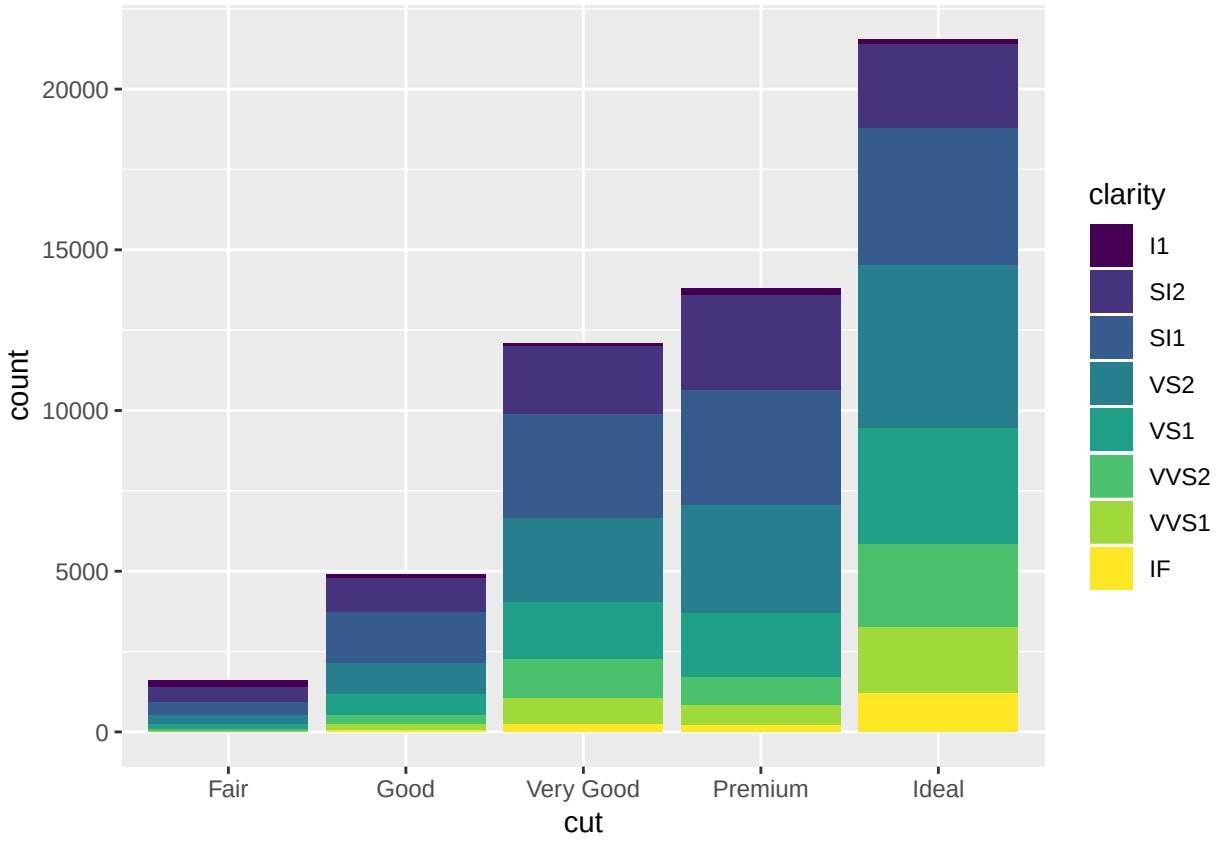


Figure 10.4: Using the viridis palette

10.2.2.2 Viridis

The viridis palette is another popular choice.

For continuous scales use

- `scale_fill_viridis_c`
- `scale_color_viridis_c`

For discrete (categorical scales) use

- `scale_fill_viridis_d`
- `scale_color_viridis_d`

```
# Use a viridis fill palette
ggplot(diamonds, aes(x = cut, fill = clarity)) +
  geom_bar() +
  scale_fill_viridis_d()
```

10.2.2.3 Other palettes

Other palettes to explore include dutchmasters, ggpomological, LaCroixColoR, nord, ochRe, palettetown, pals, rcartocolor, and wesanderson.

If you want to explore **all** the palette options (or nearly all), take a look at the paletter package.

To learn more about color specifications, see the *R Cookpage* page on ggplot2 colors. Also see the color choice advice in this book.

10.3 Points & Lines

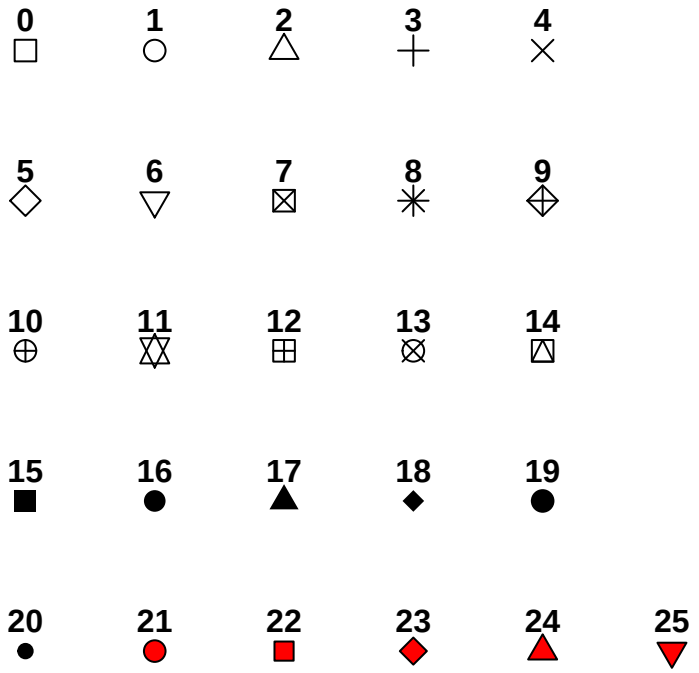
10.3.1 Points

For `ggplot2` graphs, the default point is a filled circle. To specify a different shape, use the `shape = #` option in the `geom_point` function. To map shapes to the levels of a categorical variable use the `shape = variablename` option in the `aes` function.

Examples:

- `geom_point(shape = 1)`
- geom_point(`aes(shape = sex)`)

Availabe shapes are given in the table below.



Shapes 21 through 26 provide for both a fill color and a border color.

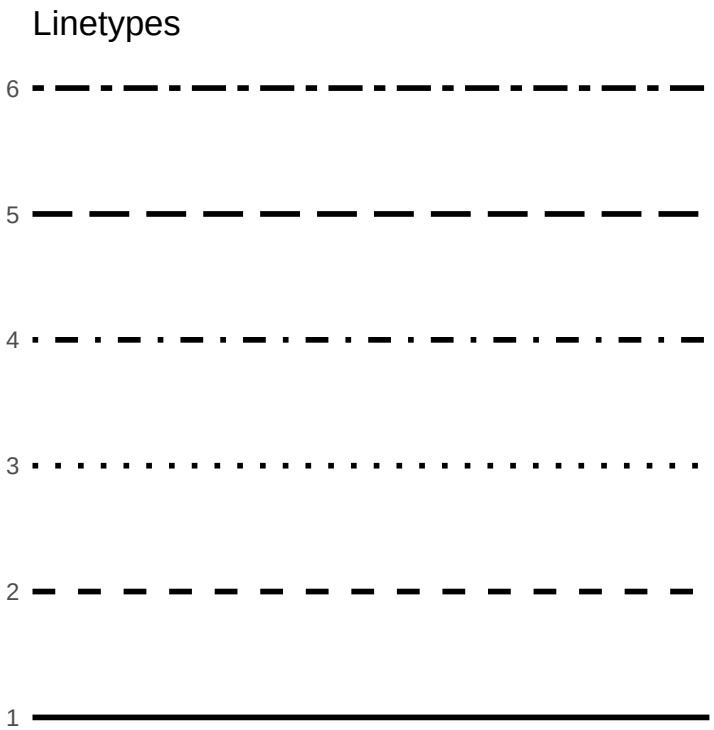
10.3.2 Lines

The default line type is a solid line. To change the linetype, use the `linetype = #` option in the `geom_line` function. To map linetypes to the levels of a categorical variable use the `linetype = variablename` option in the `aes` function.

Examples:

- `geom_line(linetype = 1)`
- geom_line(`aes(linetype = sex)`)

Availabe linetypes are given in the table below.



Fonts

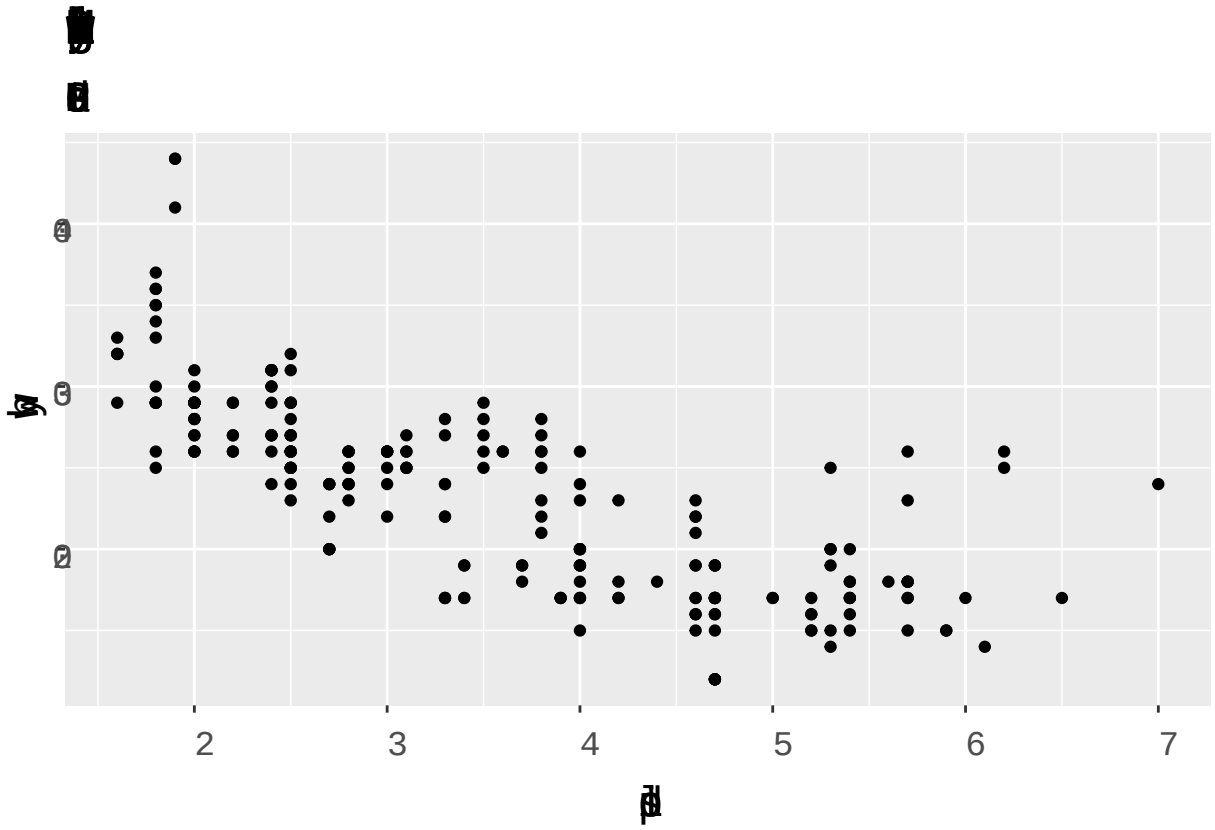
R does not have great support for fonts, but with a bit of work, you can change the fonts that appear in your graphs. First you need to install and set-up the `extrafont` package.

```
# one time install
install.packages("extrafont")
library(extrafont)
font_import()

# see what fonts are now available
fonts()
```

Apply the new font(s) using the `text` option in the `theme` function.

```
# specify new font
library(extrafont)
```

```
ggplot(mpg, aes(x = displ, y=hwy)) +
  geom_point() +
  labs(title = "Diplacement by Highway Mileage",
       subtitle = "MPG dataset") +
  theme(text = element_text(size = 16, family = "Comic Sans MS"))
```



To learn more about customizing fonts, see Working with R, Cairo graphics, custom fonts, and ggplot.

10.4 Legends

In `ggplot2`, legends are automatically created when variables are mapped to color, fill, linetype, shape, size, or alpha.

You have a great deal of control over the look and feel of these legends. Modifications are usually made through the `theme` function and/or the `labs` function. Here are some of the most sought after.

10.4.1 Legend location

The legend can appear anywhere in the graph. By default, it's placed on the right. You can change the default with

```
theme(legend.position = position)
```

where

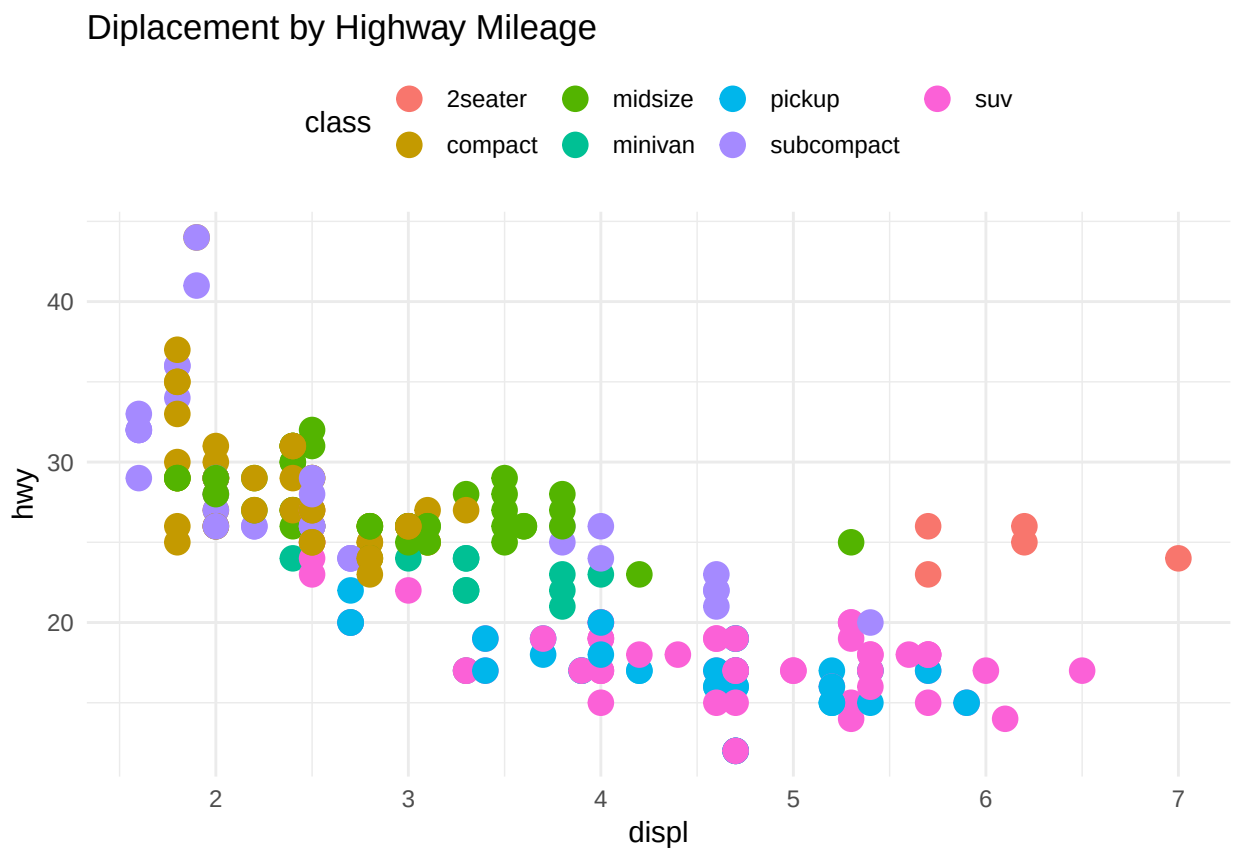


Figure 10.5: Moving the legend to the top

Position	Location
"top"	above the plot area
"right"	right of the plot area
"bottom"	below the plot area
"left"	left of the plot area
c(x, y)	within the plot area. The x and y values must range between 0 and 1. c(0,0) represents (left, bottom) and c(1,1) represents (right, top).
"none"	suppress the legend

For example, to place the legend at the top, use the following code.

```
# place legend on top
ggplot(mpg,
       aes(x = displ, y=hwy, color = class)) +
  geom_point(size = 4) +
  labs(title = "Diplacement by Highway Mileage") +
  theme_minimal() +
  theme(legend.position = "top")
```

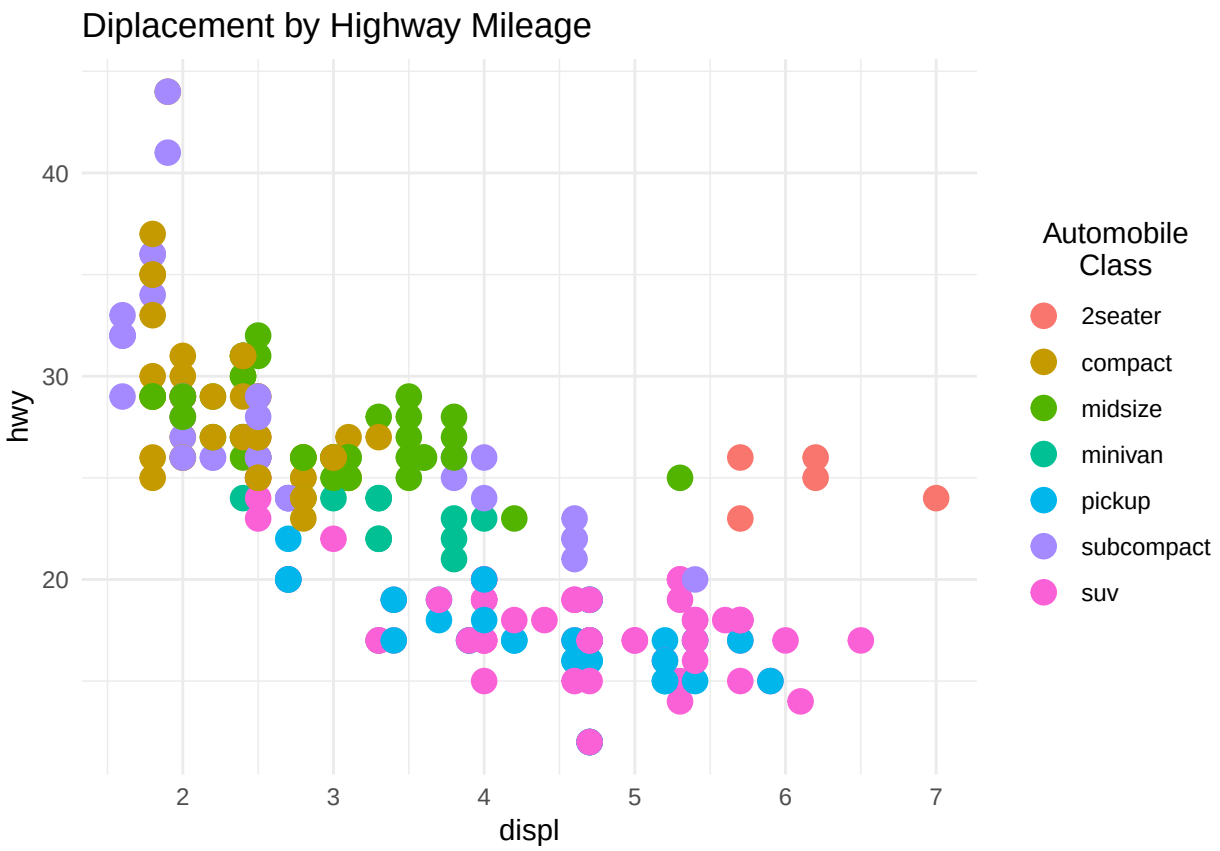


Figure 10.6: Changing the legend title

10.4.2 Legend title

You can change the legend title through the `labs` function. Use `color`, `fill`, `size`, `shape`, `linetype`, and `alpha` to give new titles to the corresponding legends.

The alignment of the legend title is controlled through the `legend.title.align` option in the `theme` function. (0=left, 0.5=center, 1=right)

```
# change the default legend title
ggplot(mpg,
       aes(x = displ, y=hwy, color = class)) +
  geom_point(size = 4) +
  labs(title = "Diplacement by Highway Mileage",
       color = "Automobile\nClass") +
  theme_minimal() +
  theme(legend.title.align=0.5)
```

See Hadley Wickam's legend attributes for more details.

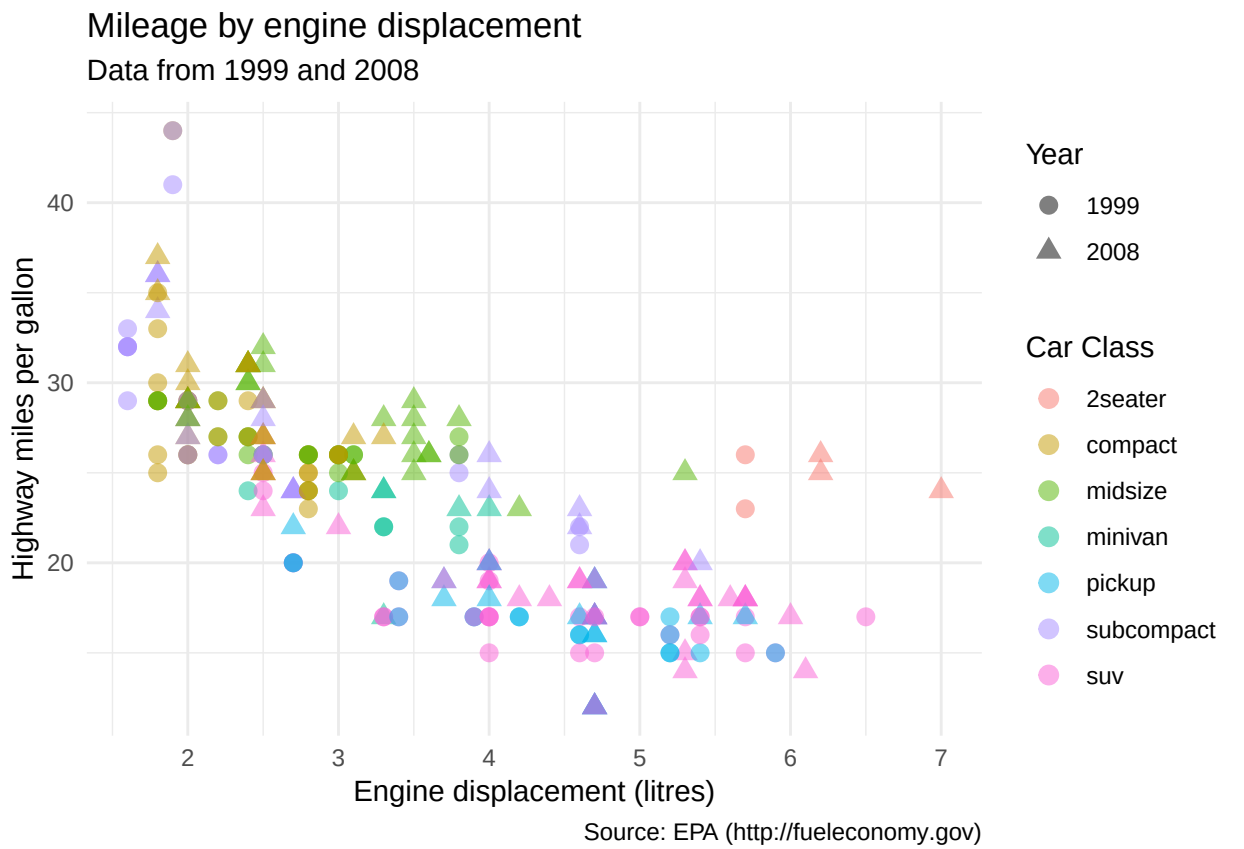
10.5 Labels

Labels are a key ingredient in rendering a graph understandable. They're are added with the `labs` function. Available options are given below.

option	Use
title	main title
subtitle	subtitle
caption	caption (bottom right by default)
x	horizontal axis
y	vertical axis
color	color legend title
fill	fill legend title
size	size legend title
linetype	linetype legend title
shape	shape legend title
alpha	transparency legend title
size	size legend title

For example

```
# add plot labels
ggplot(mpg,
       aes(x = displ, y=hwy,
           color = class,
           shape = factor(year))) +
  geom_point(size = 3,
             alpha = .5) +
  labs(title = "Mileage by engine displacement",
       subtitle = "Data from 1999 and 2008",
       caption = "Source: EPA (http://fueleconomy.gov)",
       x = "Engine displacement (litres)",
       y = "Highway miles per gallon",
       color = "Car Class",
       shape = "Year") +
  theme_minimal()
```



This is not a great graph - it is too busy, making the identification of patterns difficult. It would better to facet the year variable, the class variable or both. Trend lines would also be helpful.

10.6 Annotations

Annotations are addition information added to a graph to highlight important points.

10.6.1 Adding text

There are two primary reasons to add text to a graph.

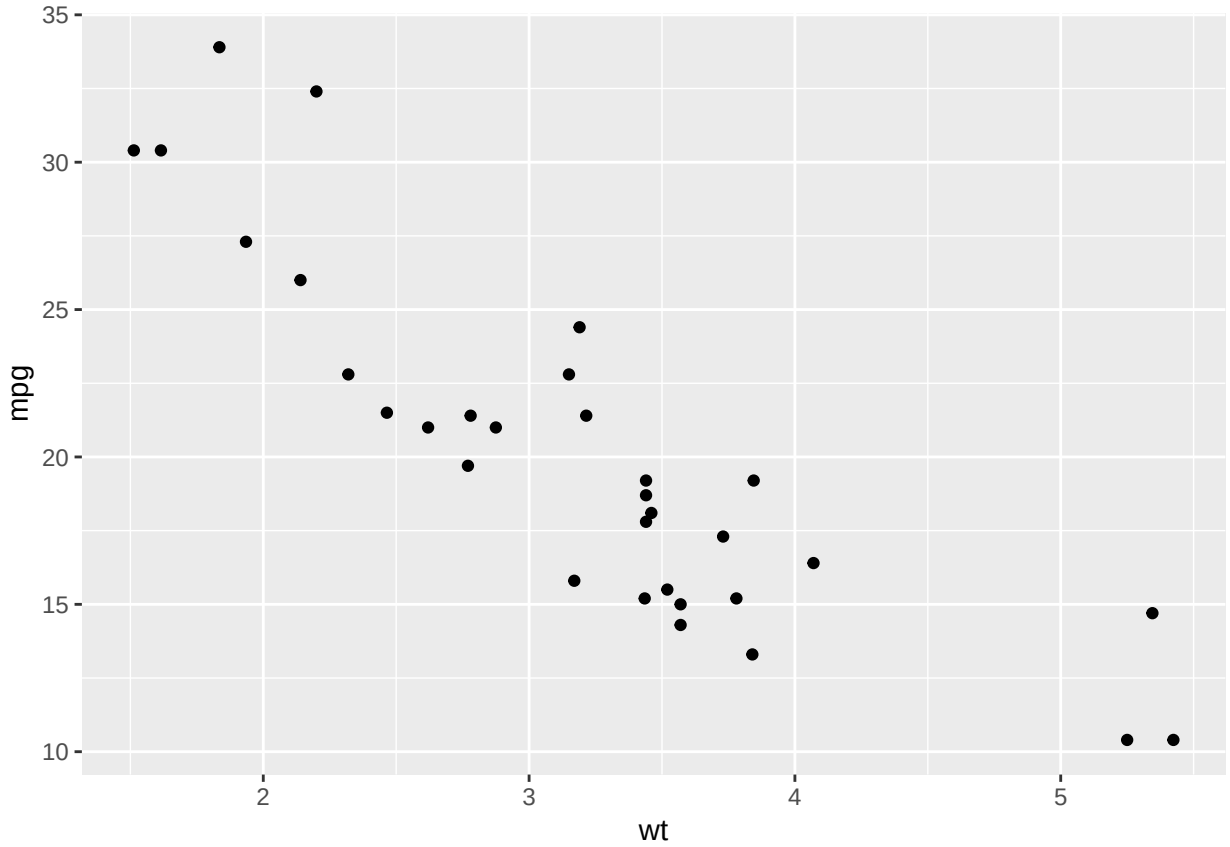
One is to identify the numeric qualities of a geom. For example, we may want to identify points with labels in a scatterplot, or label the heights of bars in a bar chart.

Another reason is to provide additional information. We may want to add notes about the data, point out outliers, etc.

10.6.1.1 Labeling values

Consider the following scatterplot, based on the car data in the mtcars dataset.

```
# basic scatterplot
data(mtcars)
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point()
```



Let's label each point with the name of the car it represents.

```
# scatterplot with labels
data(mtcars)
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_text(label = row.names(mtcars))
```

The overlapping labels make this chart difficult to read. There is a package called `ggrepel` that can help us here.

```
# scatterplot with non-overlapping labels
data(mtcars)
library(ggrepel)
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_text_repel(label = row.names(mtcars),
                  size=3)
```

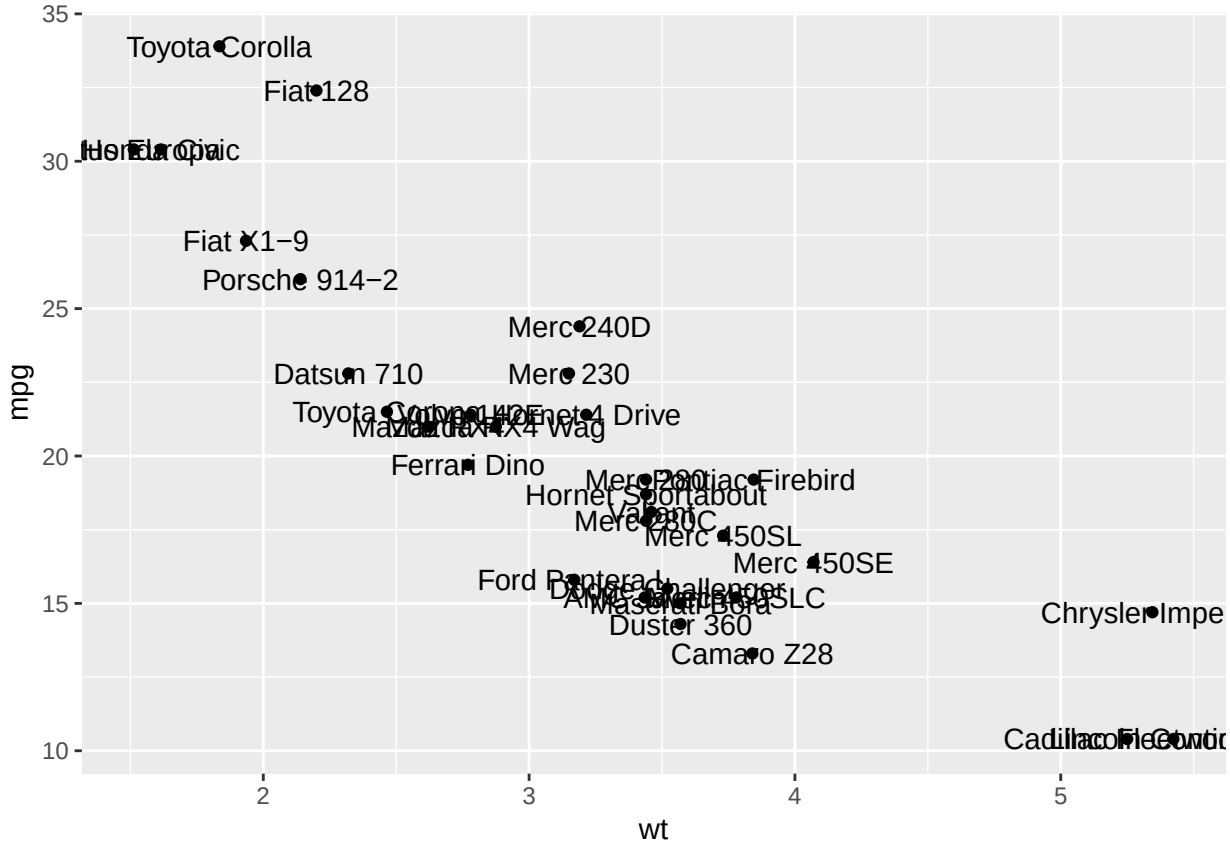
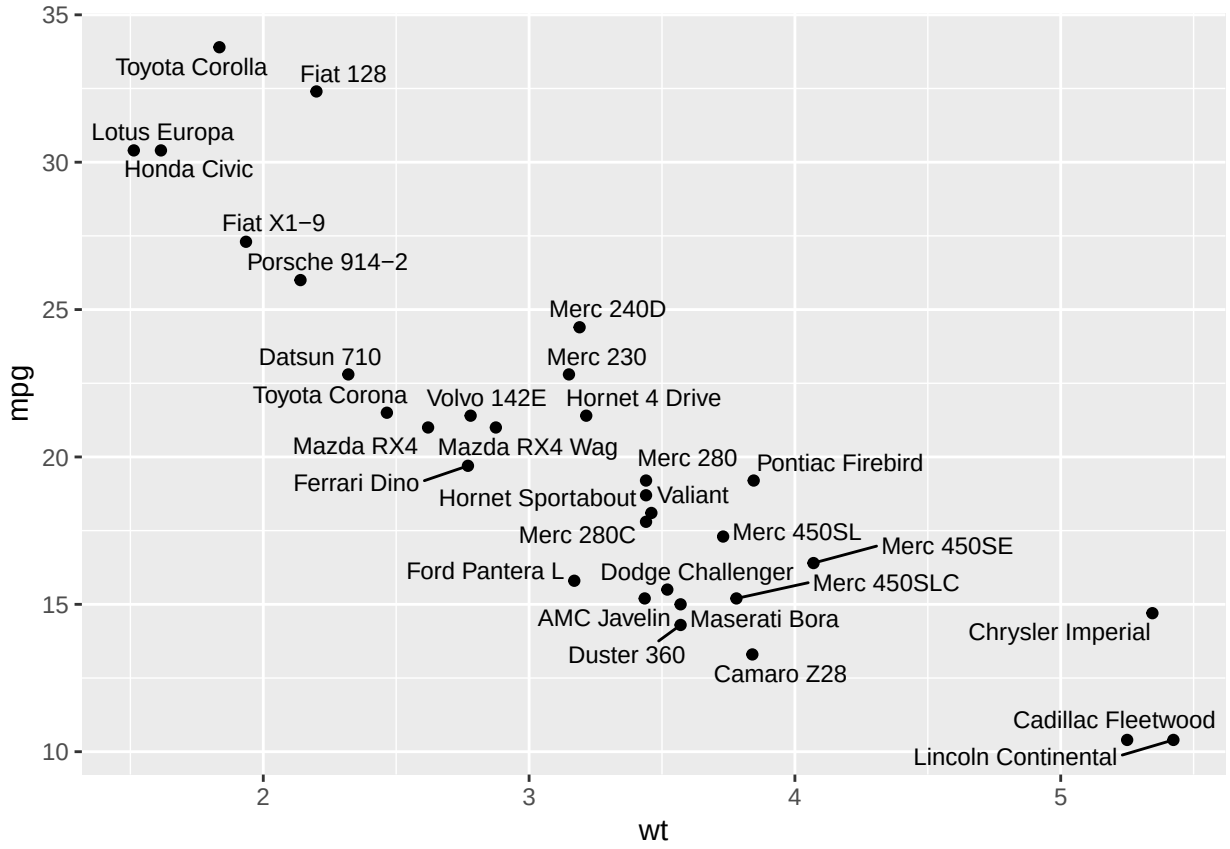


Figure 10.7: Scatterplot with labels



Much better.

Adding labels to bar charts is covered in the aptly named labeling bars section.

10.6.1.2 Adding additional information

We can place text anywhere on a graph using the `annotate` function. The format is

```
annotate("text",
         x, y,
         label = "Some text",
         color = "colorname",
         size=textsize)
```

where x and y are the coordinates on which to place the text. The `color` and `size` parameters are optional.

By default, the text will be centered. Use `hjust` and `vjust` to change the alignment.

- `hjust` 0 = left justified, 0.5 = centered, and 1 = right centered.
- `vjust` 0 = above, 0.5 = centered, and 1 = below.

Continuing the previous example.

```
# scatterplot with explanatory text
data(mtcars)
library(ggrepel)
```

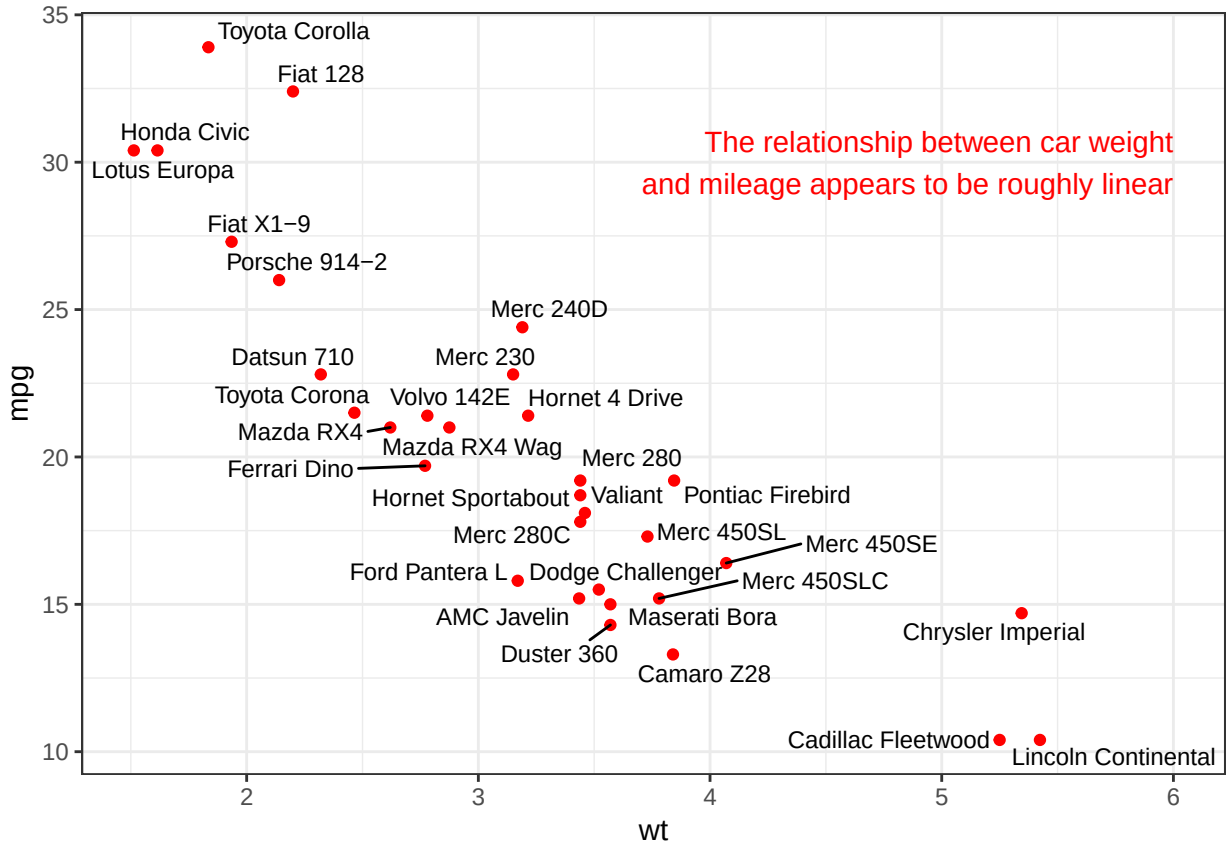


Figure 10.8: Scatterplot with arranged labels

```
txt <- paste("The relationship between car weight",
             "and mileage appears to be roughly linear",
             sep = "\n")
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point(color = "red") +
  geom_text_repel(label = row.names(mtcars),
                  size=3) +
  ggplot2::annotate("text",
                    6, 30,
                    label=txt,
                    color = "red",
                    hjust = 1) +
  theme_bw()
```

See this blog post for more details.

10.6.2 Adding lines

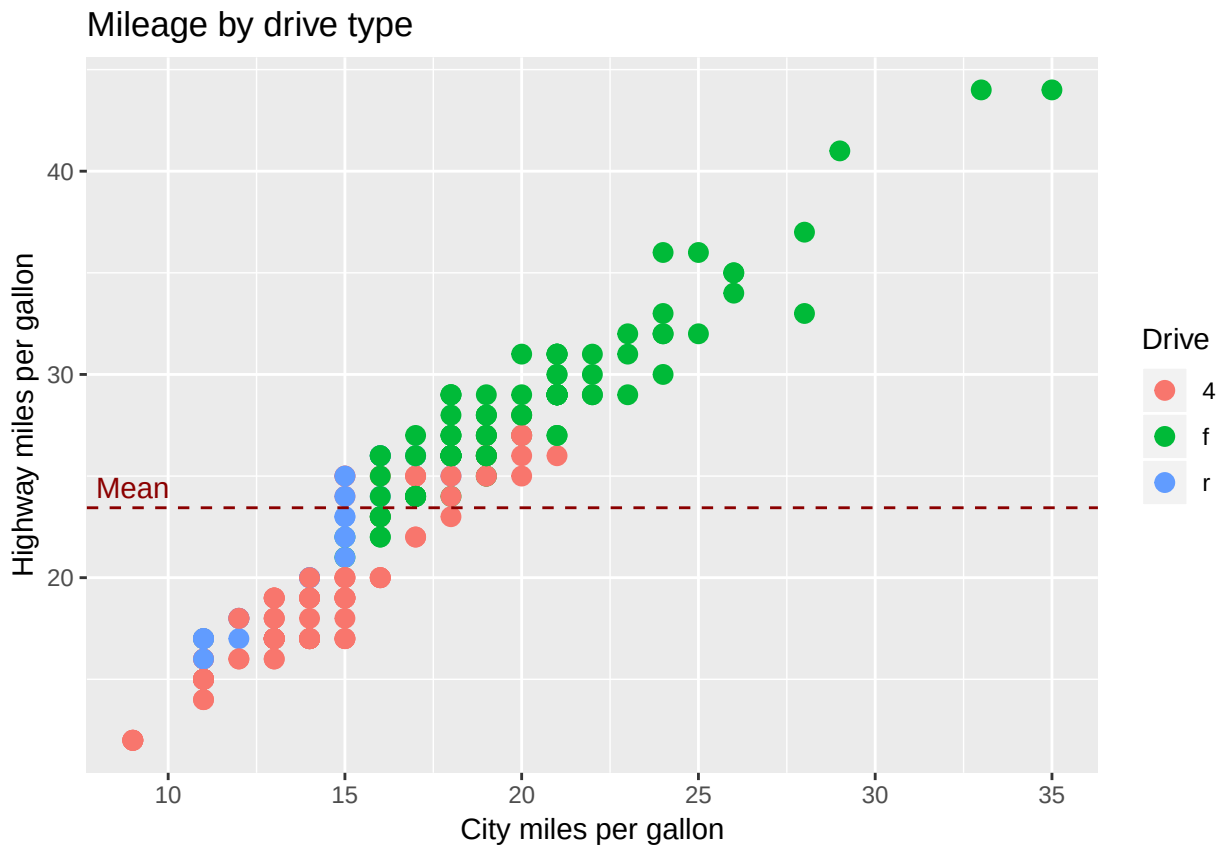
Horizontal and vertical lines can be added using:

- `geom_hline(yintercept = a)`

- `geom_vline(xintercept = b)`

where a is a number on the y-axis and b is a number on the x-axis respectively. Other option include `linetype` and `color`.

```
# add annotation line and text label
min_cty <- min(mpg$cty)
mean_hwy <- mean(mpg$hwy)
ggplot(mpg,
       aes(x = cty, y=hwy, color=drv)) +
  geom_point(size = 3) +
  geom_hline(yintercept = mean_hwy,
             color = "darkred",
             linetype = "dashed") +
  ggplot2::annotate("text",
           min_cty,
           mean_hwy + 1,
           label = "Mean",
           color = "darkred") +
  labs(title = "Mileage by drive type",
       x = "City miles per gallon",
       y = "Highway miles per gallon",
       color = "Drive")
```



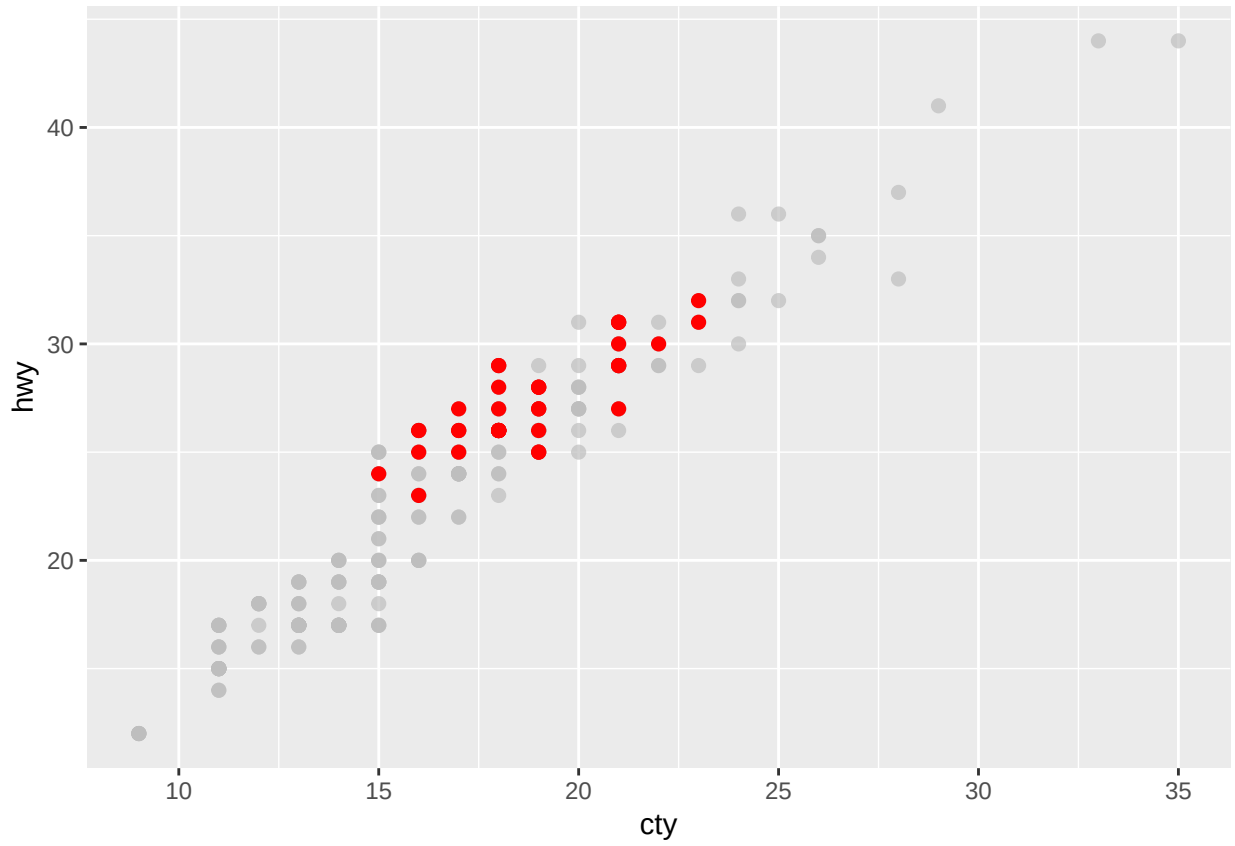
We could add a vertical line for the mean city miles per gallon as well. In any case, always label annotation lines in some way. Otherwise the reader will not know what they mean.

10.6.3 Highlighting a single group

Sometimes you want to highlight a single group in your graph. The `gghighlight` function in the `gghighlight` package is designed for this.

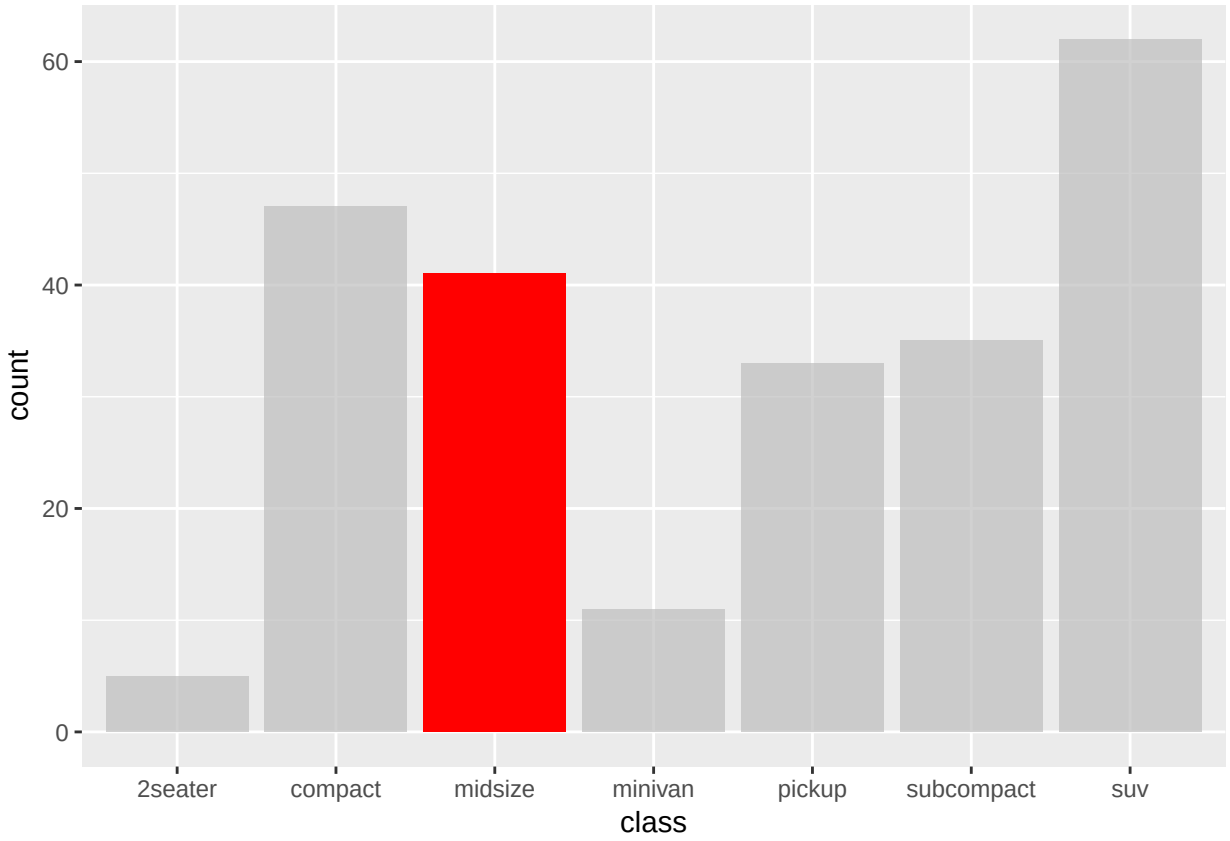
Here is an example with a scatterplot.

```
# highlight a set of points
library(ggplot2)
library(gghighlight)
ggplot(mpg, aes(x = cty, y = hwy)) +
  geom_point(color = "red",
             size=2) +
  gghighlight(class == "midsize")
```



Below is an example with a bar chart.

```
# highlight a single bar
library(gghighlight)
ggplot(mpg, aes(x = class)) +
  geom_bar(fill = "red") +
  gghighlight(class == "midsize")
```



There is nothing here that could not be done with base graphics, but it is more convenient.

10.7 Themes

`ggplot2` themes control the appearance of all non-data related components of a plot. You can change the look and feel of a graph by altering the elements of its theme.

10.7.1 Altering theme elements

The `theme` function is used to modify individual components of a theme.

The parameters of the `theme` function are described in a cheatsheet developed from the online help.

Consider the following graph. It shows the number of male and female faculty by rank and discipline at a particular university in 2008-2009. The data come from the Salaries for Professors dataset.

```
# create graph
data(Salaries, package = "carData")
p <- ggplot(Salaries, aes(x = rank, fill = sex)) +
  geom_bar() +
  facet_wrap(~discipline) +
  labs(title = "Academic Rank by Gender and Discipline",
       x = "Rank",
       y = "Frequency",
       fill = "Gender")
p
```

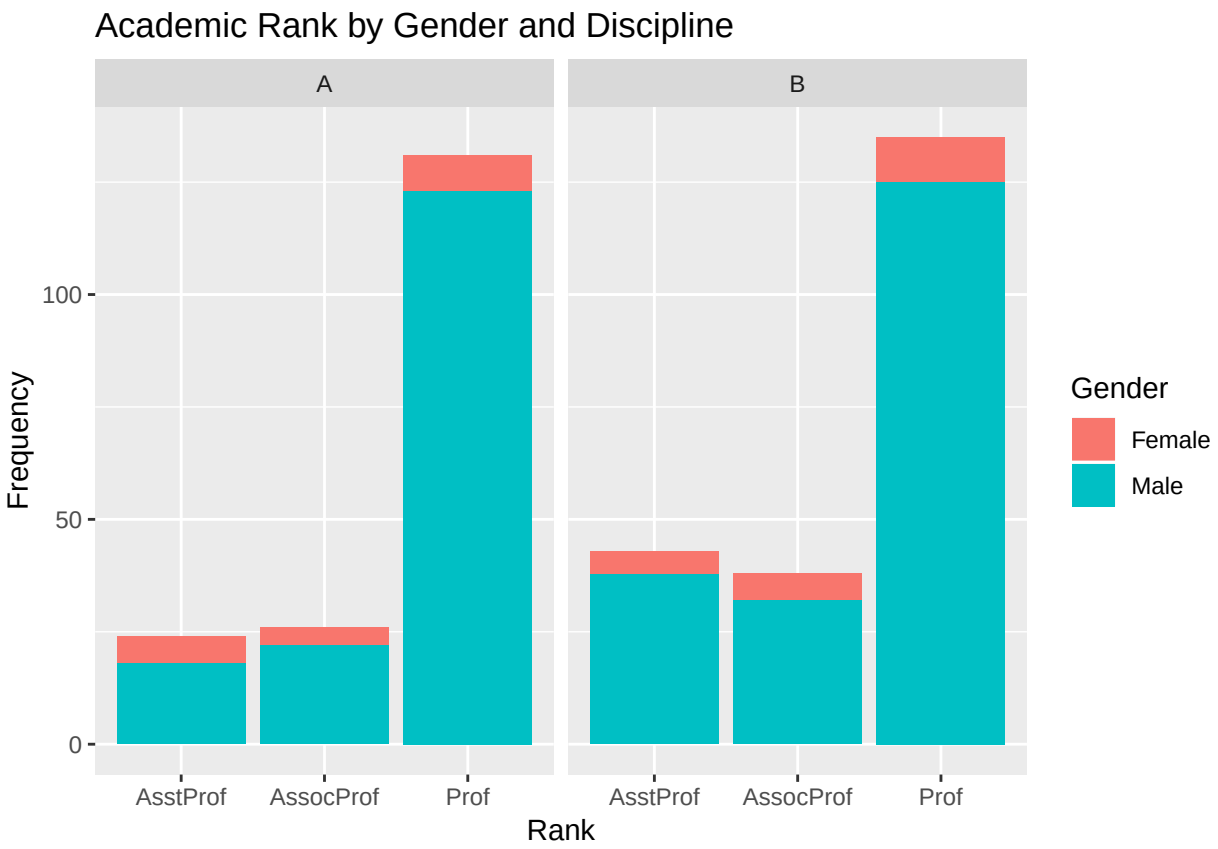


Figure 10.9: Graph with default theme

Let's make some changes to the theme.

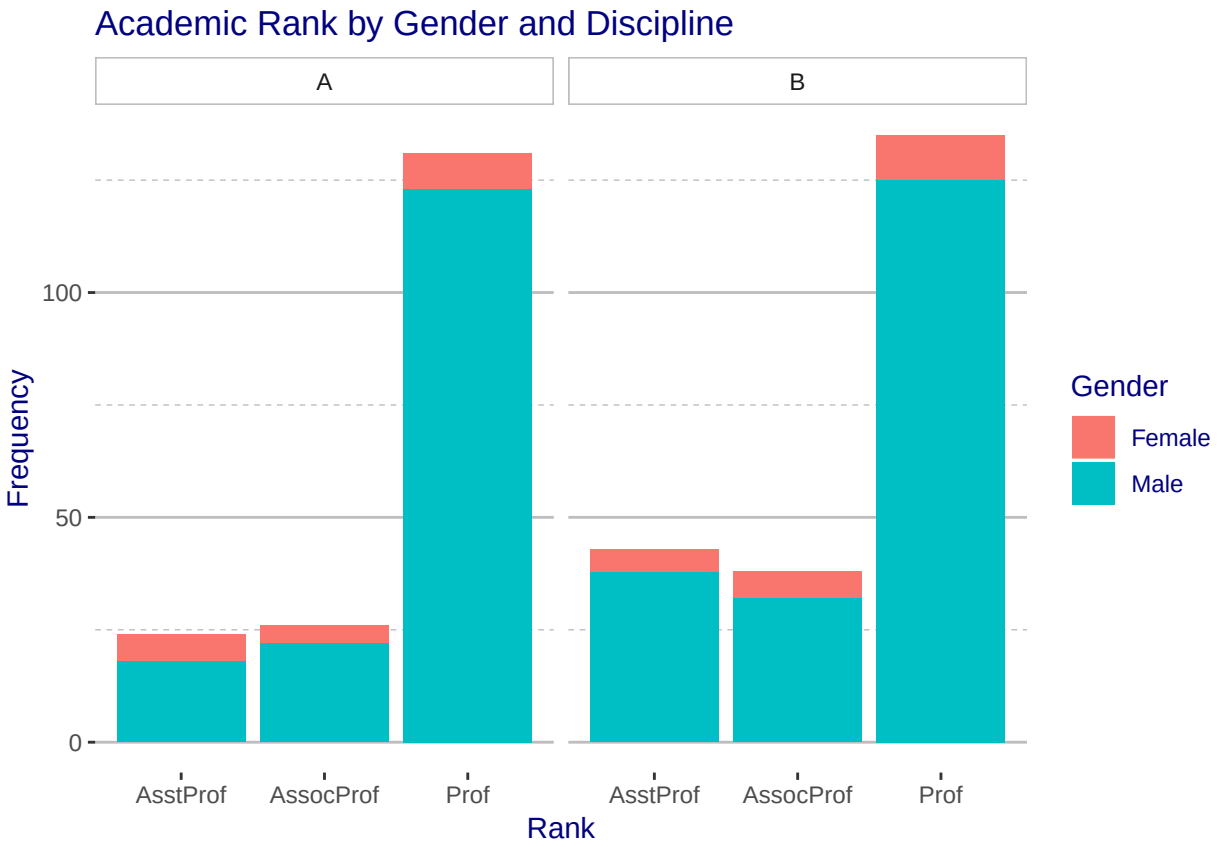
- Change label text from black to navy blue

- Change the panel background color from grey to white
- Add solid grey lines for major y-axis grid lines
- Add dashed grey lines for minor y-axis grid lines
- Eliminate x-axis grid lines

- Change the strip background color to white with a grey border

Using the cheat sheet gives us

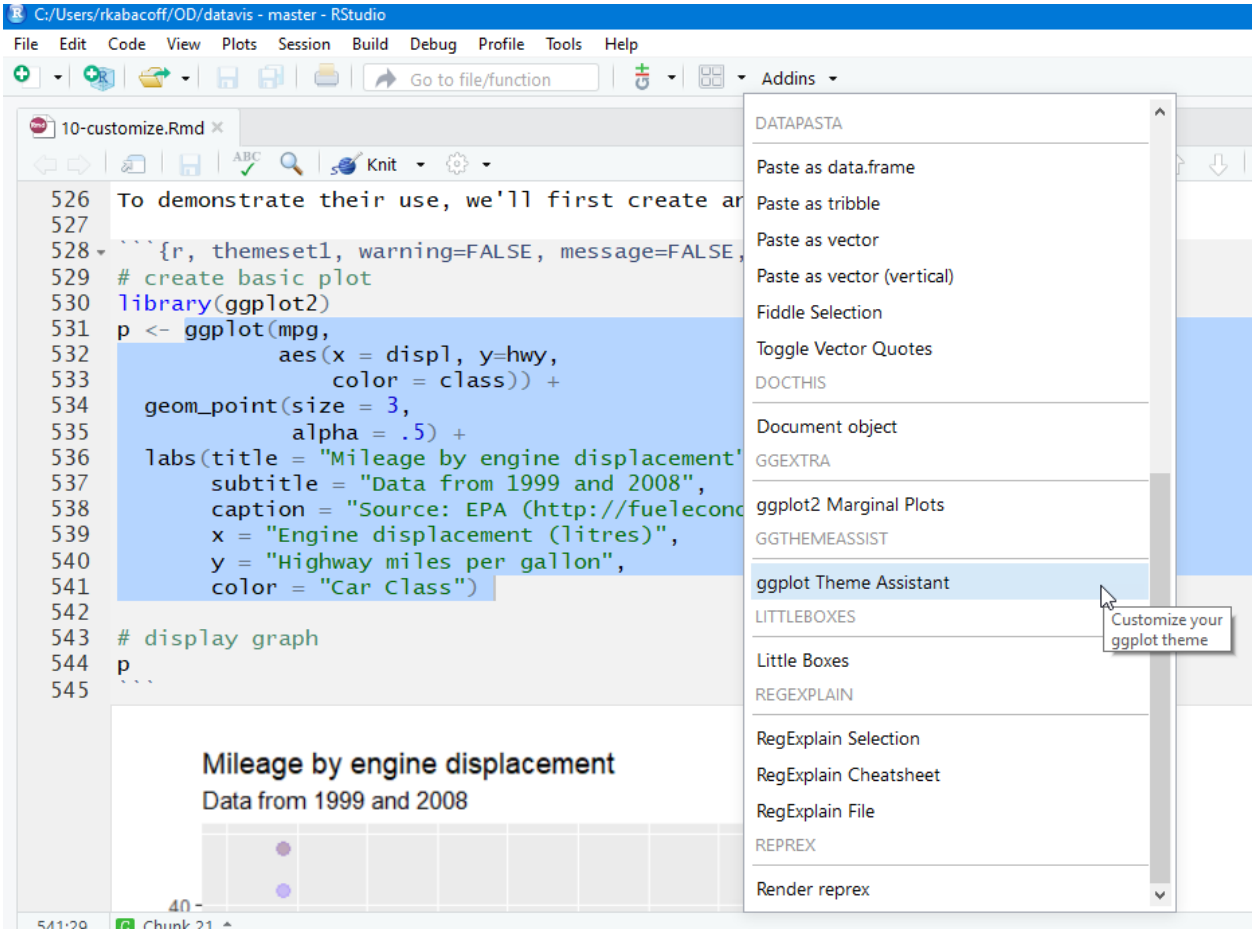
```
p +
  theme(text = element_text(color = "navy"),
        panel.background = element_rect(fill = "white"),
        panel.grid.major.y = element_line(color = "grey"),
        panel.grid.minor.y = element_line(color = "grey",
                                          linetype = "dashed"),
        panel.grid.major.x = element_blank(),
        panel.grid.minor.x = element_blank(),
        strip.background = element_rect(fill = "white", color="grey"))
```



Wow, this looks pretty awful, but you get the idea.

10.7.1.1 ggThemeAssist

If you would like to create your own theme using a GUI, take a look at `ggThemeAssist`. After you install the package, a new menu item will appear under Addins in RStudio.



Highlight the code that creates your graph, then choose the `ggThemeAssist` option from the **Addins** drop-down menu. You can change many of the features of your theme using point-and-click. When you're done, the `theme` code will be appended to your graph code.

10.7.2 Pre-packaged themes

I'm not a very good artist (just look at the last example), so I often look for pre-packaged themes that can be applied to my graphs. There are many available.

Some come with `ggplot2`. These include *theme_classic*, *theme_dark*, *theme_gray*, *theme_grey*, *theme_light* *theme_linedraw*, *theme_minimal*, and *theme_void*. We've used *theme_minimal* often in this book. Others are available through add-on packages.

10.7.2.1 ggthemes

The `ggthemes` package come with 19 themes.

Theme	Description
theme_base	Theme Base
theme_calc	Theme Calc
theme_economist	ggplot color theme based on the Economist
theme_economist_white	ggplot color theme based on the Economist
theme_excel	ggplot color theme based on old Excel plots

Theme	Description
theme_few	Theme based on Few's "Practical Rules for Using Color in Charts"
theme_fivethirtyeight	Theme inspired by fivethirtyeight.com plots
theme_foundation	Foundation Theme
theme_gdocs	Theme with Google Docs Chart defaults
theme_hc	Highcharts JS theme
theme_igray	Inverse gray theme
theme_map	Clean theme for maps
theme_pander	A ggplot theme originated from the pander package
theme_par	Theme which takes its values from the current 'base' graphics parameter values in 'par'.
theme_solarized	ggplot color themes based on the Solarized palette
theme_solarized_2	ggplot color themes based on the Solarized palette
theme_solid	Theme with nothing other than a background color
theme_stata	Themes based on Stata graph schemes
theme_tufte	Tufte Maximal Data, Minimal Ink Theme
theme_wsj	Wall Street Journal theme

To demonstrate their use, we'll first create and save a graph.

```
# create basic plot
library(ggplot2)
p <- ggplot(mpg,
            aes(x = displ, y=hwy,
                color = class)) +
  geom_point(size = 3,
             alpha = .5) +
  labs(title = "Mileage by engine displacement",
       subtitle = "Data from 1999 and 2008",
       caption = "Source: EPA (http://fueleconomy.gov)",
       x = "Engine displacement (litres)",
       y = "Highway miles per gallon",
       color = "Car Class")

# display graph
p
```

Now let's apply some themes.

```
# add economist theme
library(ggthemes)
p + theme_economist()
```

```
# add fivethirtyeight theme
p + theme_fivethirtyeight()
```

```
# add wsj theme
p + theme_wsj(base_size=8)
```

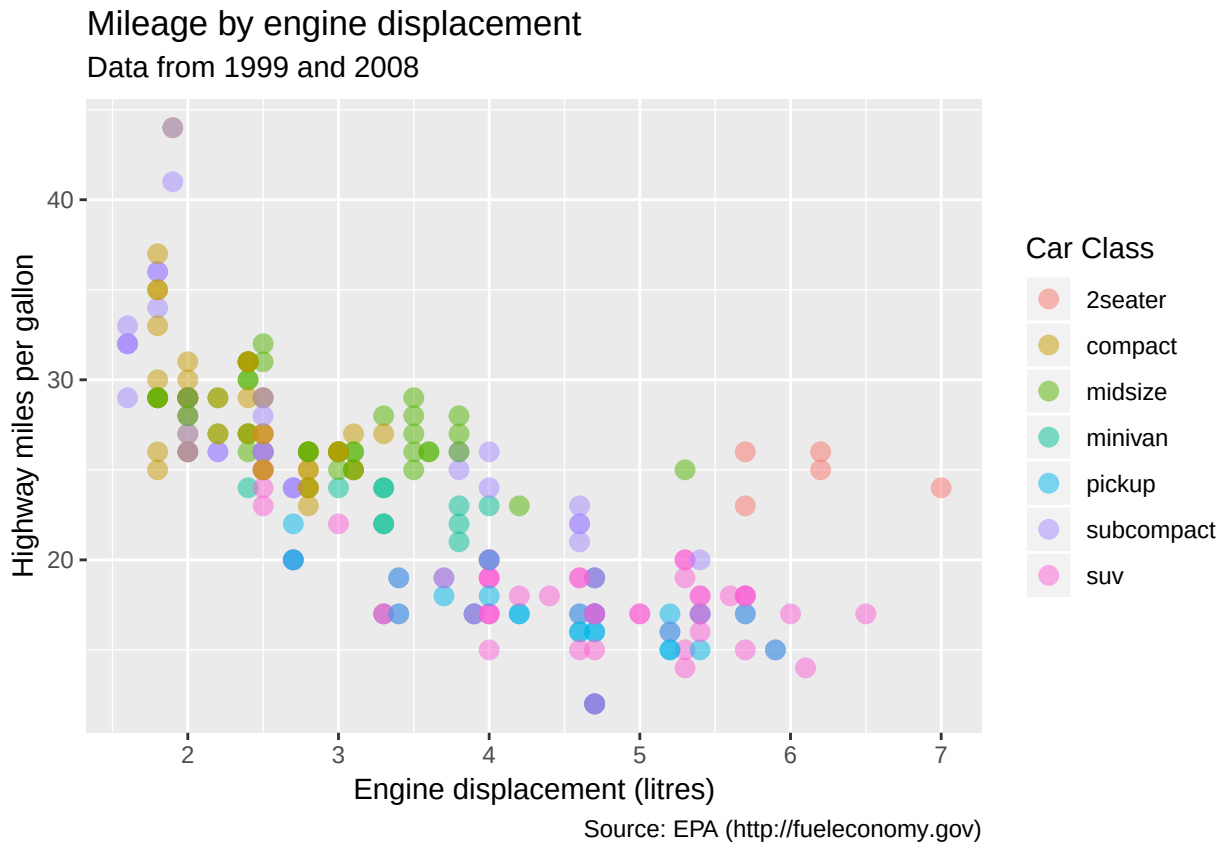


Figure 10.10: Default theme

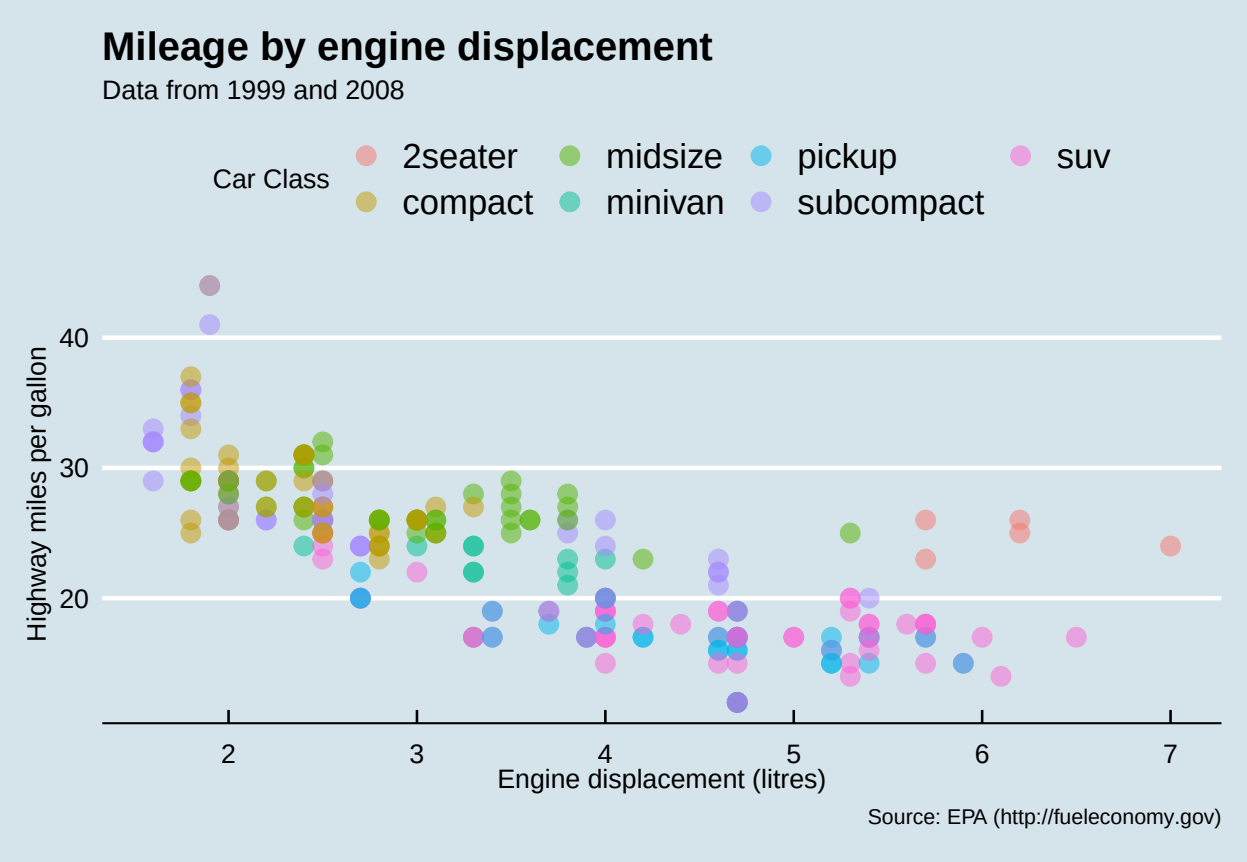


Figure 10.11: Economist theme

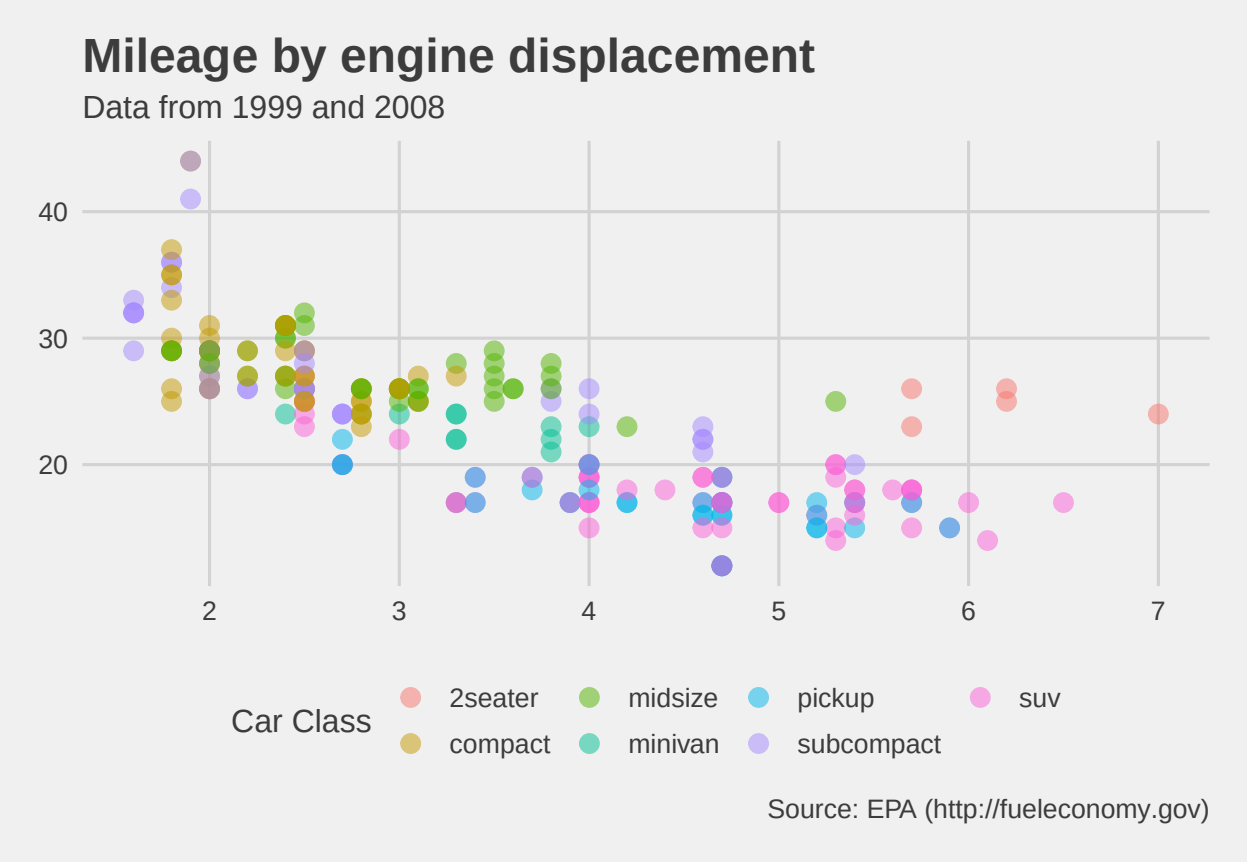
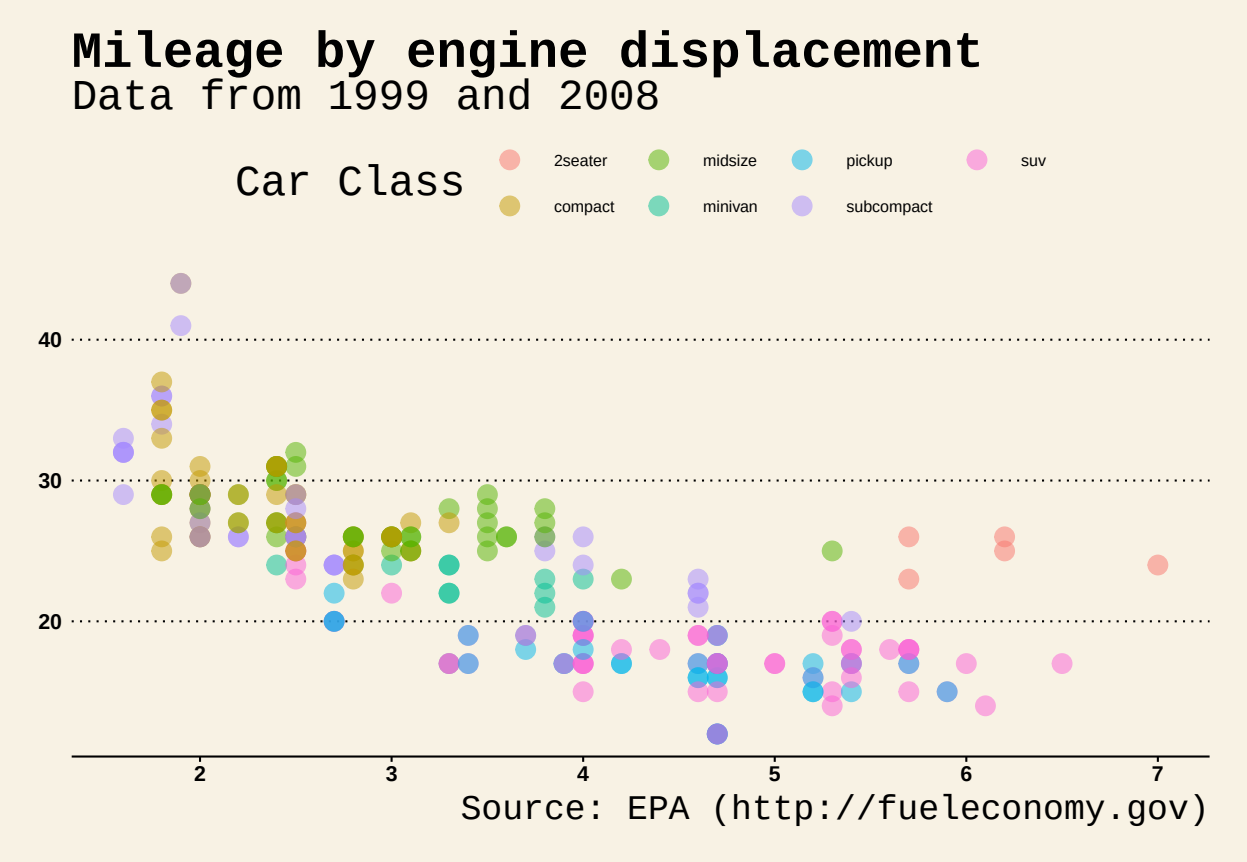


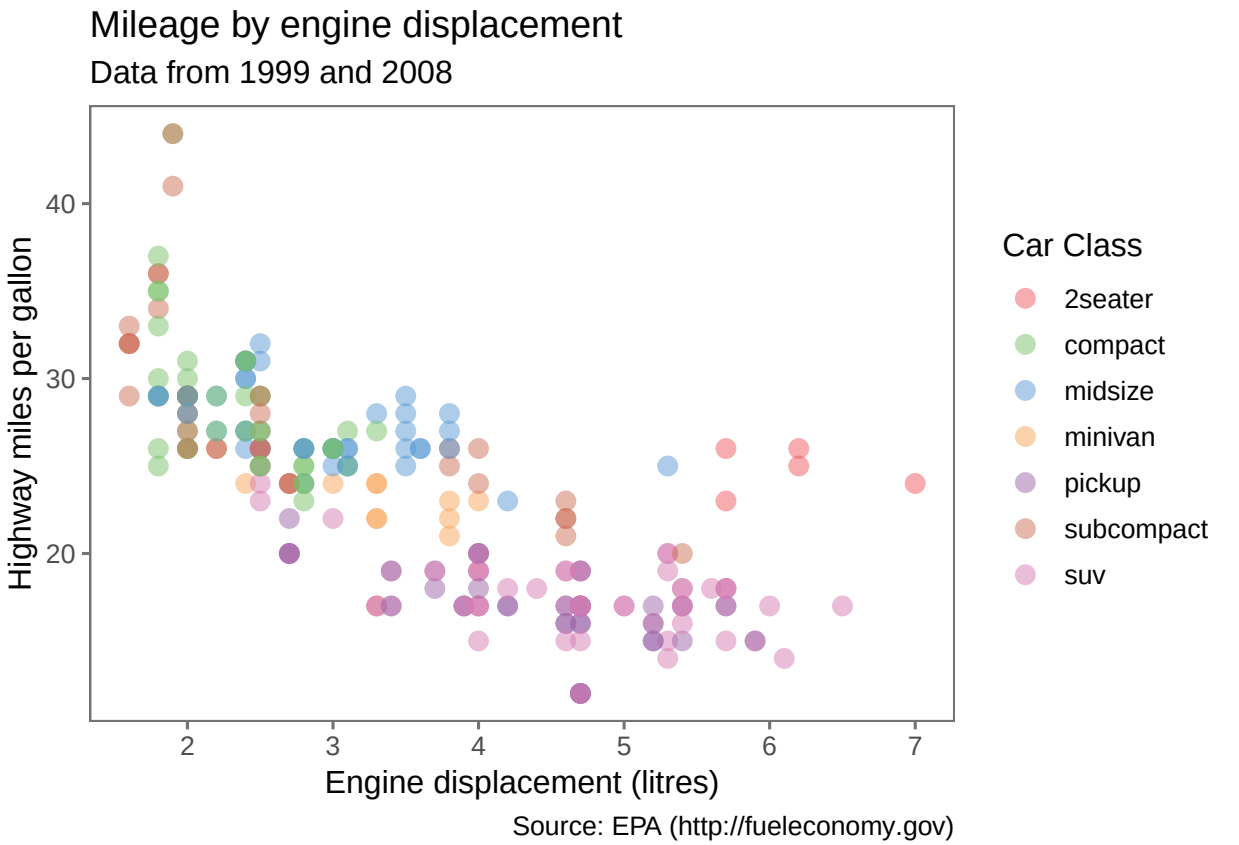
Figure 10.12: Five Thirty Eight theme



By default, the font size for the wsj theme is usually too large. Changing the `base_size` option can help.

Each theme also comes with scales for colors and fills. In the next example, both the `few` theme and colors are used.

```
# add few theme
p + theme_few() + scale_color_few()
```



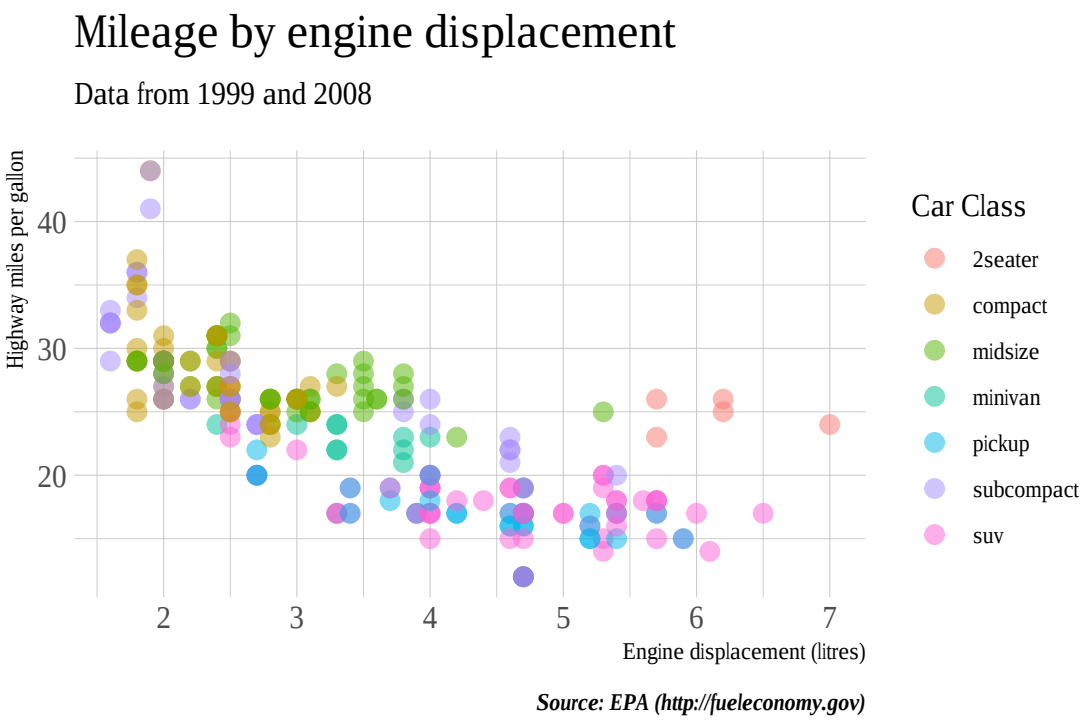
Try out different themes and scales to find one that you like.

10.7.2.2 hrbrthemes

The `hrbrthemes` package is focused on typography-centric themes. The results are charts that tend to have a clean look.

Continuing the example plot from above

```
# add few theme
library(hrbrthemes)
p + theme_ipsum()
```



See the hrbrthemes homepage for additional examples.

10.7.2.3 ggthemer

The `ggthemer` package offers a wide range of themes (17 as of this printing).

The package is not available on CRAN and must be installed from GitHub.

```
# one time install
install.packages("devtools")
devtools::install_github('cttobin/ggthemr')
```

The functions work a bit differently. Use the `ggthemr("themename")` function to set future graphs to a given theme. Use `ggthemr_reset()` to return future graphs to the `ggplot2` default theme.

Current themes include *flat*, *flat dark*, *camoflauge*, *chalk*, *copper*, *dust*, *earth*, *fresh*, *grape*, *grass*, *greyscale*, *light*, *lilac*, *pale*, *sea*, *sky*, and *solarized*.

```
# set graphs to the flat dark theme
library(ggthemr)
ggthemr("flat dark")
p
```

```
ggthemr_reset()
```

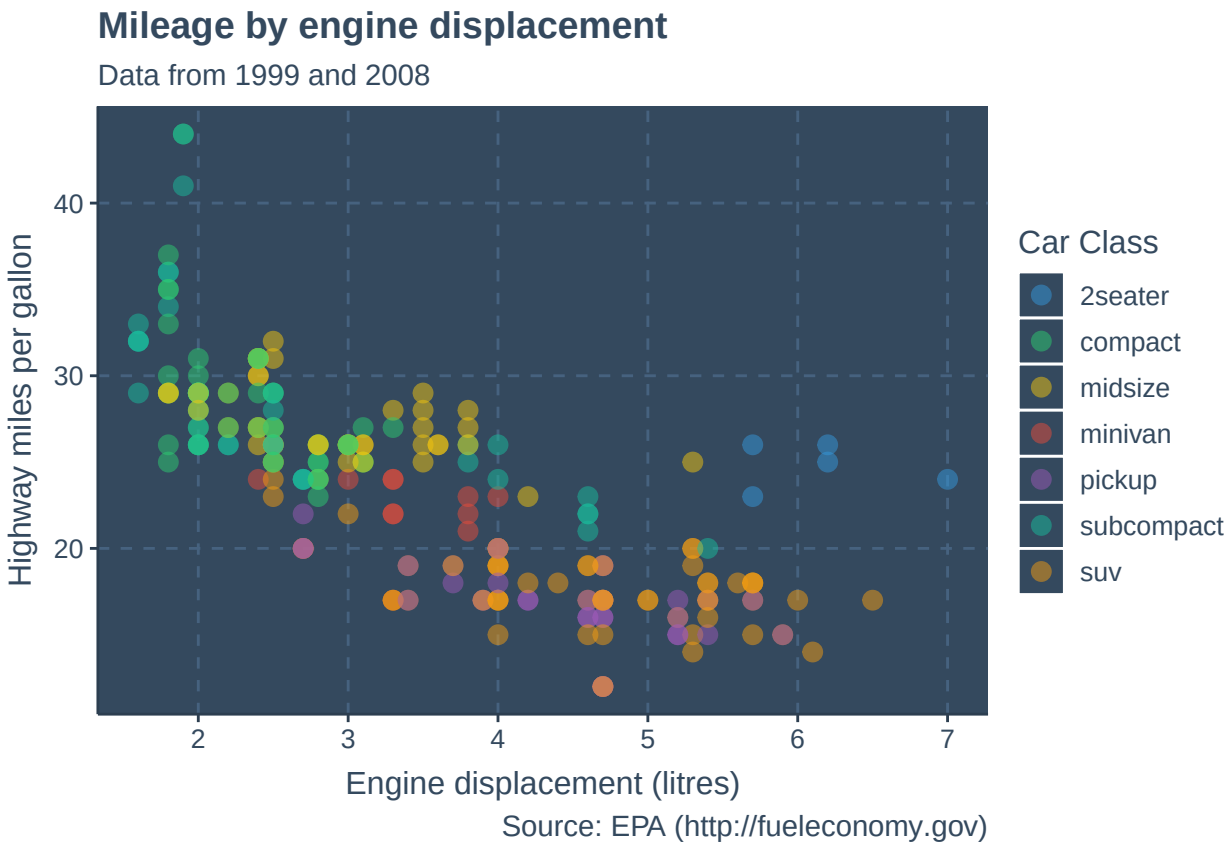


Figure 10.13: Ipsum theme

I would not actually use this theme for this particular graph. It is difficult to distinguish colors. Which green represents compact cars and which represents subcompact cars?

Select a theme that best conveys the graph's information to your audience.

Chapter 11

Saving Graphs

Graphs can be saved via the RStudio interface or through code.

11.1 Via menus

To save a graph using the RStudio menus, go to the **Plots** tab and choose **Export**.

11.2 Via code

Any `ggplot2` graph can be saved as an object. Then you can use the `ggsave` function to save the graph to disk.

```
# save a graph
library(ggplot2)
p <- ggplot(mtcars,
            aes(x = wt , y = mpg)) +
  geom_point()
ggsave(p, filename = "mygraph.png")
```

The graph will be saved in the format defined by the file extension (*png* in the example above). Common formats are *pdf*, *jpeg*, *tiff*, *png*, *svg*, and *wmf* (windows only).

11.3 File formats

Graphs can be saved in several formats. The most popular choices are given below.

Format	Extension
Portable Document Format	pdf
JPEG	jpeg
Tagged Image File Format	tiff
Portable Network Graphics	png
Scaleable Vector Graphics	svg
Windows Metafile	wmf

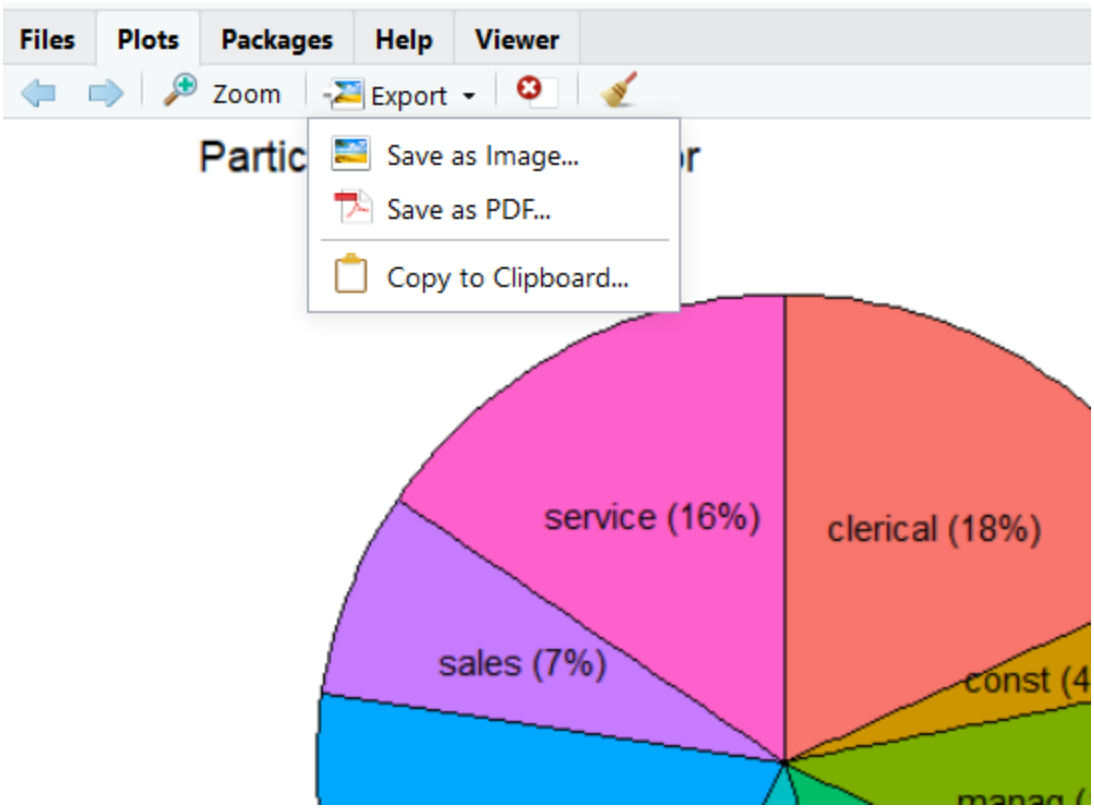


Figure 11.1: Export menu

The *pdf*, *svg*, and *wmf* formats are lossless - they resize without fuzziness or pixelation. The other formats are lossy - they will pixelate when resized. This is especially noticeable when small images are enlarged.

If you are creating graphs for webpages, the *png* format is recommended.

The *jpeg* and *tif* formats are usually reserved for photographs.

The *wmf* format is usually recommended for graphs that will appear in Microsoft Word or PowerPoint documents. MS Office does not support *pdf* or *svg* files, and the *wmf* format will rescale well. However, note that *wmf* files will lose any transparency settings that have been set.

If you want to continue editing the graph after saving it, use the *pdf* or *svg* format.

11.4 External editing

Sometimes it's difficult to get a graph just right programmatically. Most magazines and newspapers (print and electronic) fine-tune graphs after they have been created. They change the fonts, move labels around, add callouts, change colors, add additional images or logos, and the like.

If you save the graph in *svg* or *pdf* format, you can use a vector graphics editing program to modify it using point and click tools. Two popular vector graphics editors are *Illustrator* and *Inkscape*.

Inkscape is an opensource application that can be freely downloaded for Mac OS X, Windows, and Linux. Open the graph file in *Inkscape*, edit it to suite your needs, and save it in the format desired.

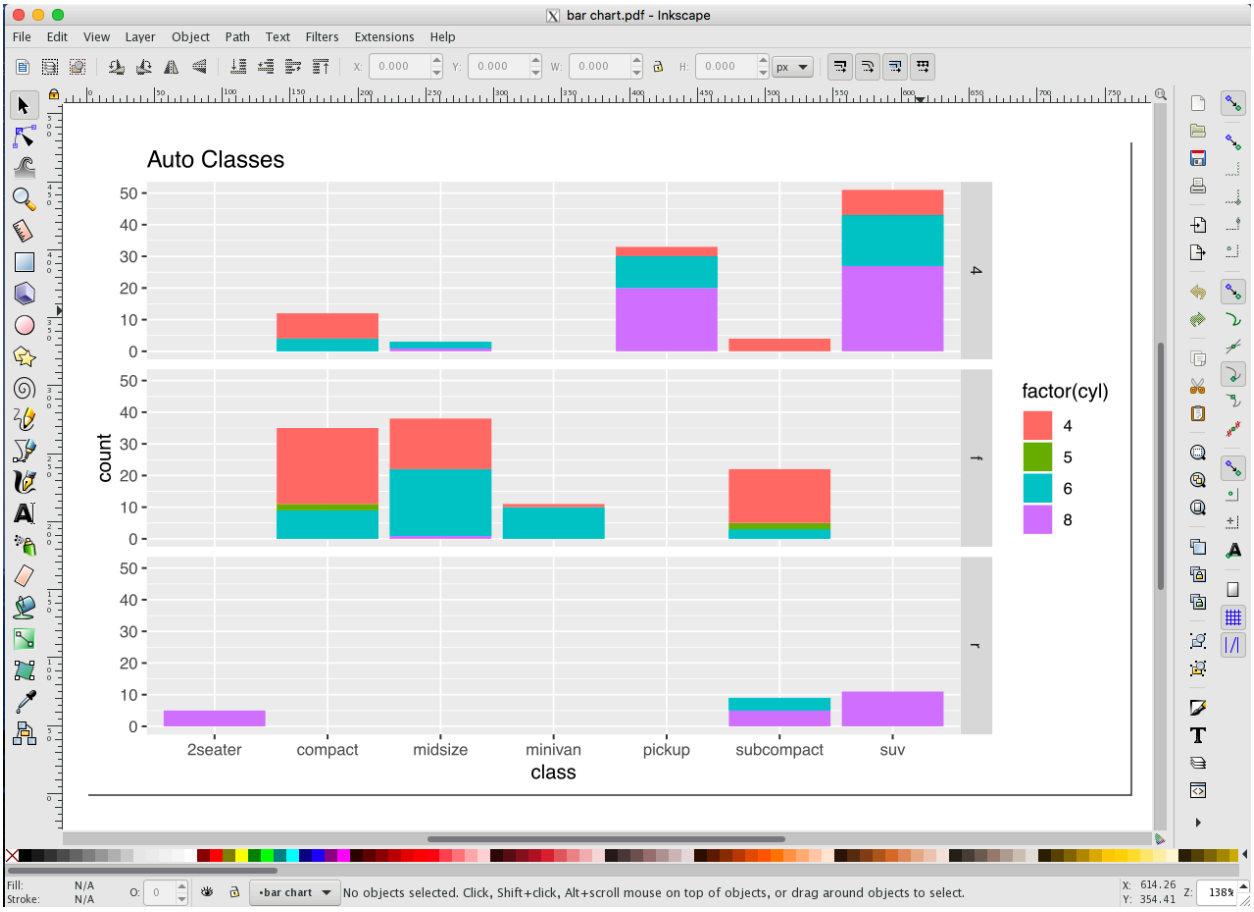


Figure 11.2: Inkscape

Chapter 12

Interactive Graphs

This book has focused on static graphs - images that can be placed in papers, posters, slides, and journal articles. Through connections with JavaScript libraries, R can also generate interactive graphs that can be placed on web pages.

Interactive graphics are beyond the scope of this book. This chapter will point out some of the best options, so you can explore them further. Most use htmlwidgets for R.

> Note that if your are reading this on an iPad, some features will not be available (such as mouseover).

12.1 leaflet

Leaflet is a javascript library for interactive maps. The leaflet package can be used to generate leaflet graphs R.

The following is a simple example. Click on the pin, zoom in and out with the +/- buttons or mouse wheel, and drag the map around with the hand cursor.

```
# create leaflet graph
library(leaflet)
leaflet() %>%
  addTiles() %>%
  addMarkers(lng=-72.6560002,
             lat=41.5541829,
             popup="The birthplace of quantitative wisdom.</br>
             No, Waldo is not here.")
```

You can create both dot density and choropleth maps. The package website offers a detailed tutorial and numerous examples.

12.2 plotly

Plotly is both a commercial service and open source product for creating high end interactive visualizations. The plotly package allows you to create `plotly` interactive graphs from within R. In addition, any `ggplot2` graph can be turned into a plotly graph.

Figure 12.1: Leaflet graph

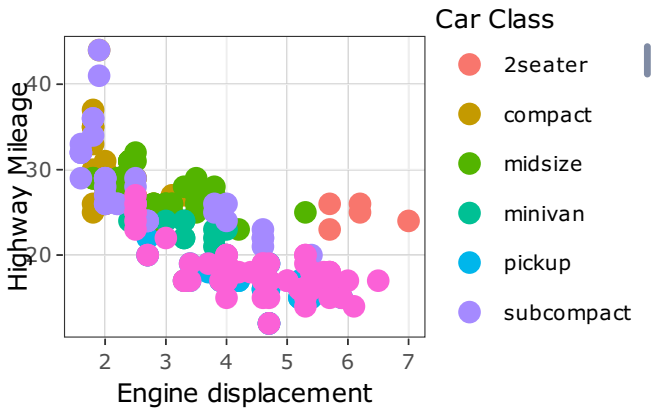


Figure 12.2: Plotly graph

Using the Fuel Economy data, we'll create an interactive graph displaying highway mileage vs. engine displace by car class.

Mousing over a point displays information about that point. Clicking on a legend point, removes that class from the plot. Clicking on it again, returns it.

```
# create plotly graph.
library(ggplot2)
library(plotly)

p <- ggplot(mpg, aes(x=displ,
                     y=hwy,
                     color=class)) +
  geom_point(size=3) +
  labs(x = "Engine displacement",
       y = "Highway Mileage",
       color = "Car Class") +
  theme_bw()

ggplotly(p)
```

There are several sources of good information on plotly. See the plotly R pages and the online plotly for R

book. Additionally, DataCamp offers a free interactive tutorial.

12.3 rbokeh

rbokeh is an interface to the Bokeh graphics library.

We'll create another graph using the `mtcars` dataset, showing engine displace vs. miles per gallon by number of engine cylinders. Mouse over, and try the various control to the right of the image.

```
# create rbokeh graph

# prepare data
data(mtcars)
mtcars$name <- row.names(mtcars)
mtcars$cyl <- factor(mtcars$cyl)

# graph it
library(rbokeh)
figure() %>%
  ly_points(disp, mpg, data=mtcars,
            color = cyl, glyph = cyl,
            hover = list(name, mpg, wt))
```

You can create some remarkable graphs with Bokeh. See the homepage for examples.

12.4 rCharts

rCharts can create a wide range of interactive graphics. In the example below, a bar chart of hair vs. eye color is created. Try mousing over the bars. You can interactively choose between grouped vs. stacked plots and include or exclude cases by eye color.

```
# create interactive bar chart
library(rCharts)
hair_eye_male = subset(as.data.frame(HairEyeColor),
                       Sex == "Male")
n1 <- nPlot(Freq ~ Hair,
            group = 'Eye',
            data = hair_eye_male,
            type = 'multiBarChart'
)
n1$set(width = 600)
n1$show('iframesrc', cdn=TRUE)
```

To learn more, visit the project homepage.

12.5 highcharter

The `highcharter` package provides access to the Highcharts JavaScript graphics library. The library is free for non-commercial use.

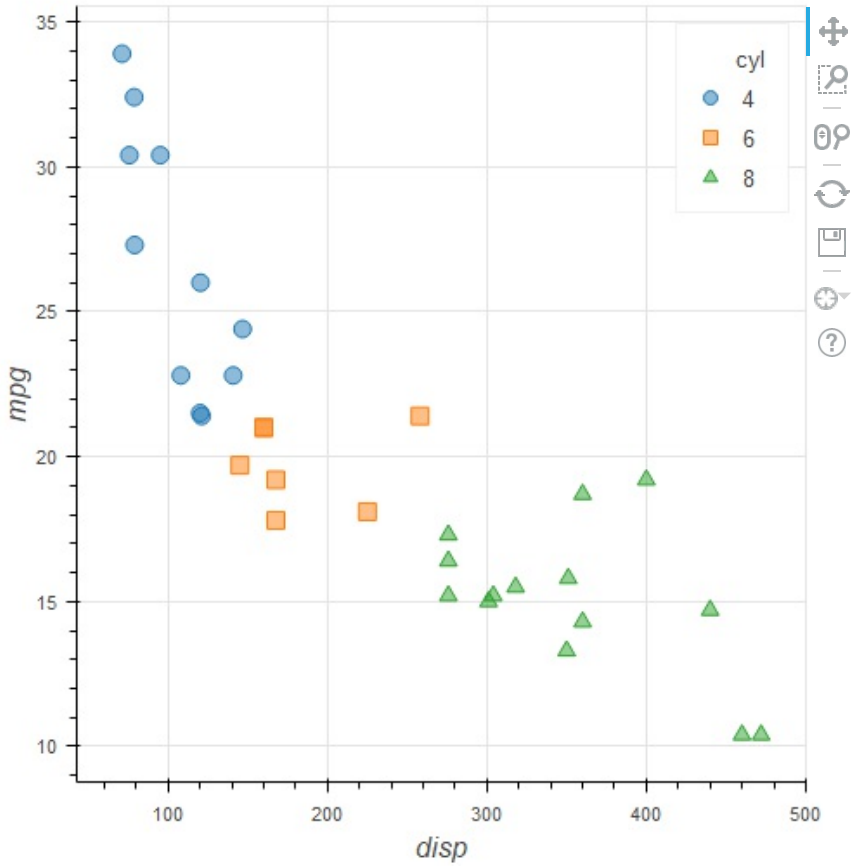


Figure 12.3: Bokeh graph

Let's use `highcharter` to create an interactive line chart displaying life expectancy over time for several Asian countries. The data come from the Gapminder dataset. Again, mouse over the lines and try clicking on the legend names.

```
# create interactive line chart
library(highcharter)

# prepare data
data(gapminder, package = "gapminder")
library(dplyr)
asia <- gapminder %>%
  filter(continent == "Asia") %>%
  select(year, country, lifeExp)

# convert to long to wide format
library(tidyr)
plotdata <- spread(asia, country, lifeExp)

# generate graph
h <- highchart() %>%
  hc_xAxis(categories = plotdata$year) %>%
  hc_add_series(name = "Afghanistan",
                data = plotdata$Afghanistan) %>%
  hc_add_series(name = "Bahrain",
                data = plotdata$Bahrain) %>%
  hc_add_series(name = "Cambodia",
                data = plotdata$Cambodia) %>%
  hc_add_series(name = "China",
                data = plotdata$China) %>%
  hc_add_series(name = "India",
                data = plotdata$India) %>%
  hc_add_series(name = "Iran",
                data = plotdata$Iran)

h
```

Like all of the interactive graphs in this chapter, there are options that allow the graph to be customized.

```
# customize interactive line chart
h <- h %>%
  hc_title(text = "Life Expectancy by Country",
           margin = 20,
           align = "left",
           style = list(color = "steelblue")) %>%
  hc_subtitle(text = "1952 to 2007",
              align = "left",
              style = list(color = "#2b908f",
                           fontWeight = "bold")) %>%
  hc_credits(enabled = TRUE, # add credits
             text = "Gapminder Data",
             href = "http://gapminder.com") %>%
  hc_legend(align = "left",
            verticalAlign = "top",
            layout = "vertical",
```

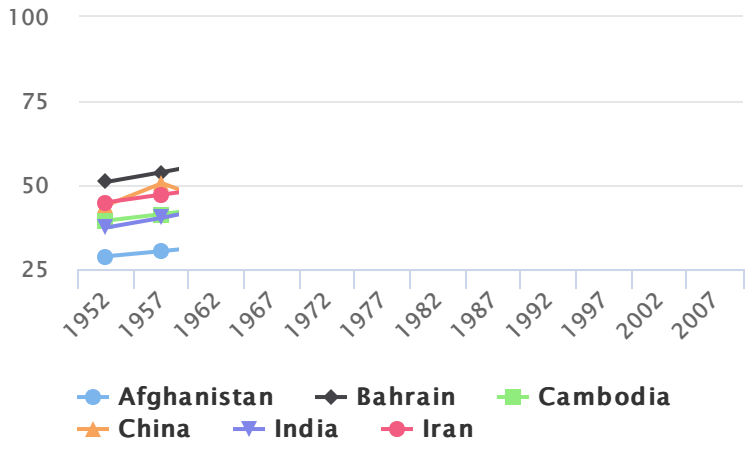


Figure 12.4: HighCharts graph

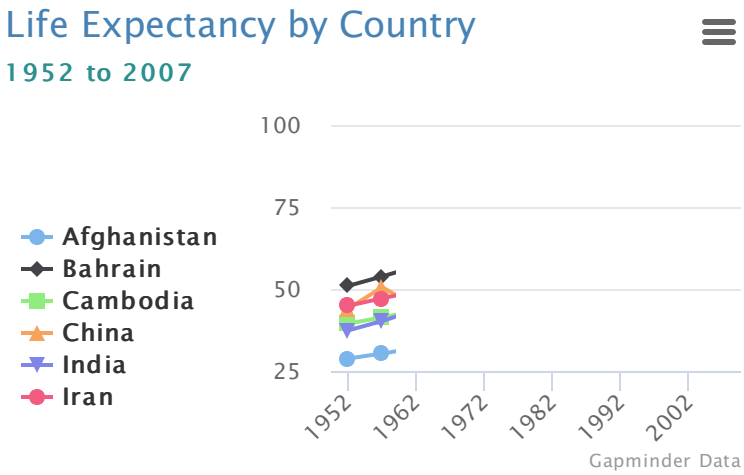


Figure 12.5: HighCharts graph with customization

```
            x = 0,
            y = 100) %>%
  hc_tooltip(crosshairs = TRUE,
             backgroundColor = "#FCFFC5",
             shared = TRUE,
             borderWidth = 4) %>%
  hc_exporting(enabled = TRUE)

h
```

There is a wealth of interactive plots available through the marriage of R and JavaScript. Choose the approach that works for you.

Chapter 13

Advice / Best Practices

This section contains some thoughts on what makes a good data visualization. Most come from books and posts that others have written, but I'll take responsibility for putting them here.

13.1 Labeling

Everything on your graph should be labeled including the

- *title* - a clear short title letting the reader know what they're looking at
  - *Relationship between experience and wages by gender*
- *subtitle* - an optional second (smaller font) title giving additional information
  - *Years 2016-2018*
- *caption* - source attribution for the data
  - *source: US Department of Labor - www.bls.gov/bls/blswage.htm*
- *axis label*s - clear labels for the x and y axes
  - short but descriptive
  - include units of measurement
    - *Engine displacement (cu. in.)*
    - *Survival time (days)*
    - *Patient age (years)*
- *legend* - short informative title and labels
  - *Male* and *Female* - not 0 and 1 !!
- *lines* and *bars* - label any trend lines, annotation lines, and error bars

Basically, the reader should be able to understand your graph without having to wade through paragraphs of text. When in doubt, show your data visualization to someone who has not read your article or poster and ask them if anything is unclear.

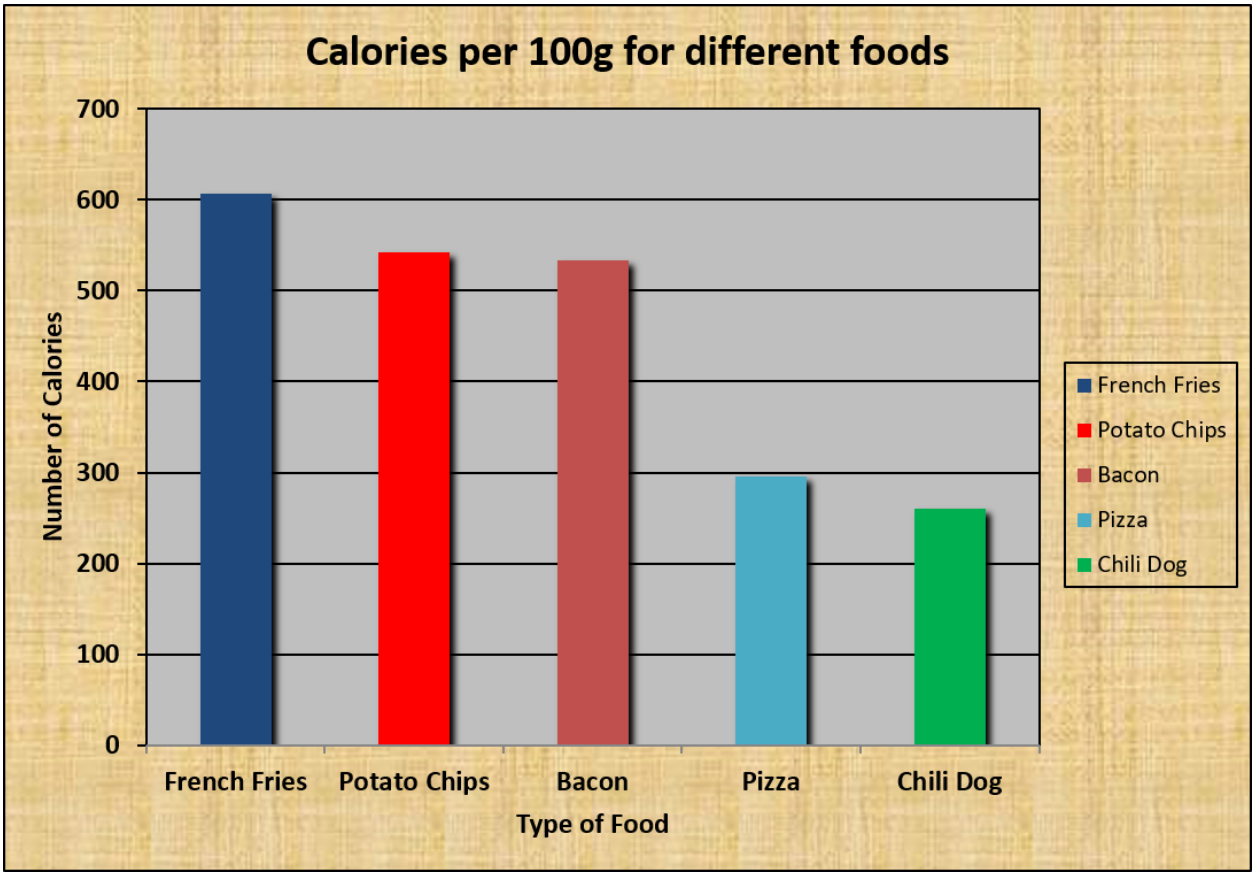


Figure 13.1: Graph with chart junk

13.2 Signal to noise ratio

In data science, the goal of data visualization is to communicate information. Anything that doesn't support this goals should be reduced or eliminated.

> **Chart Junk** - visual elements of charts that aren't necessary to comprehend the information represented by the chart or that distract from this information. (Wikipedia)

Consider the following graph. The goal is to compare the calories in bacon to the other four foods.

(Disclaimer: I got this graph from somewhere, but I can't remember where. If you know, please tell me, so that I can make a proper attribution. Also bacon always wins.)

If the goal is to compare the calories in bacon to other foods, much of this visualization is unnecessary and distracts from the task.

Think of all the things that are superfluous:

- the tan background border
- the grey background color
- the 3-D effect on the bars
- the legend (it doesn't add anything, the bars are already labeled)
- the colors of bars (they don't signify anything)

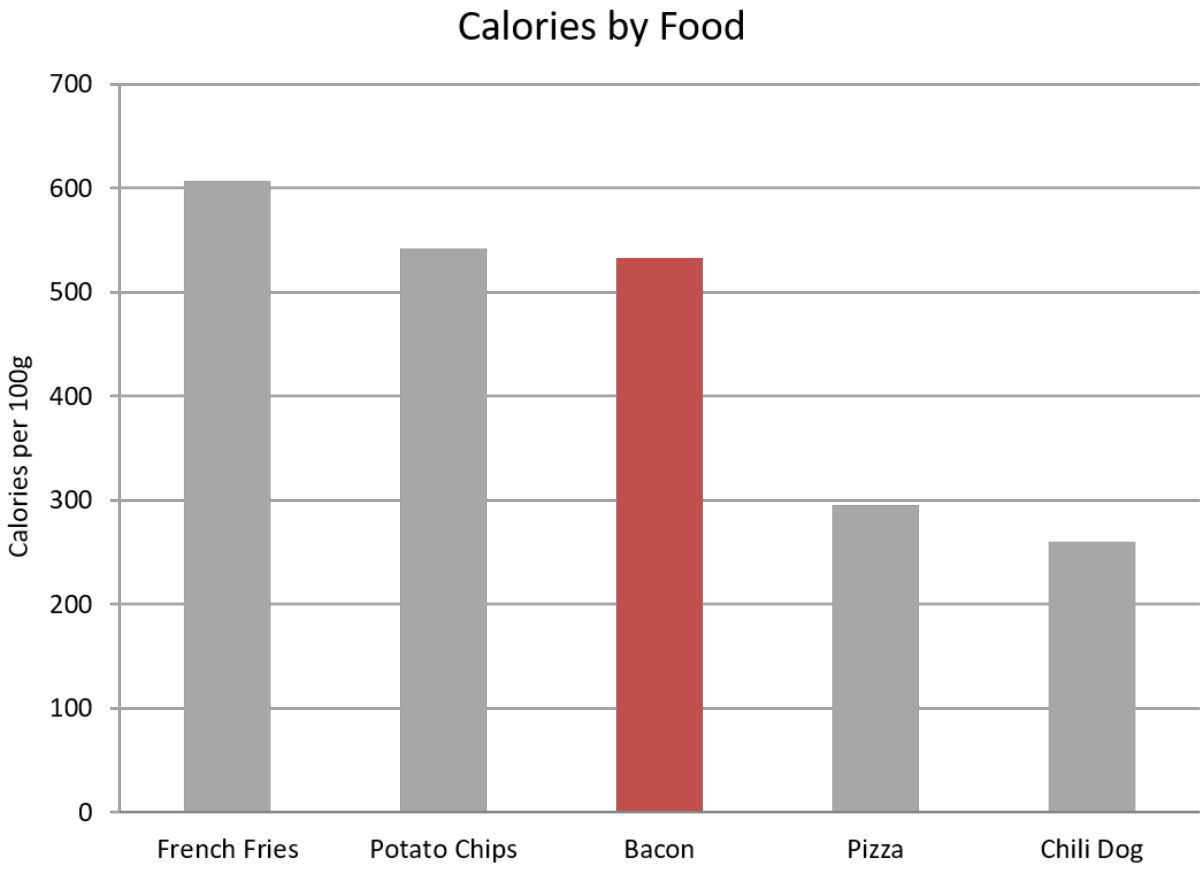


Figure 13.2: Graph with chart junk removed

Here is an alternative.

The chart junk has been removed. In addition

- the x-axis label isn't needed - these are obviously foods
- the y-axis is given a better label
- the title has been simplified (the word *different* in redundant)
- the bacon bar is the only colored bar - it makes comparisons easier
- the grid lines have been made lighter (gray rather than black) so they don't distract

I may have gone a bit far leaving out the x-axis label. It's a fine line, knowing when to stop simplifying.

In general, you want to reduce chart junk to a minimum. In other words, **more signal, less noise**.

13.3 Color choice

Color choice is about more than aesthetics. Choose colors that help convey the information contained in the plot.

The article How to Pick the Perfect Color Combination for Your Data Visualization is a great place to start.

Basically, think about selecting among sequential, diverging, and qualitative color schemes:

- sequential - for plotting a quantitative variable that goes from low to high
- diverging - for contrasting the extremes (low, medium, and high) of a quantitative variable
- qualitative - for distinguishing among the levels of a categorical variable

The article above can help you to choose among these schemes. Additionally, the `RColorBrewer` package provides palettes categorized in this way.

Other things to keep in mind:

- Make sure that text is legible - avoid dark text on dark backgrounds, light text on light backgrounds, and colors that clash in a discordant fashion (i.e. they hurt to look at!)
- Avoid combinations of red and green - it can be difficult for a colorblind audience to distinguish these colors

Other helpful resources are Practical Rules for Using Color in Charts and Expert Color Choices for Presenting Data.

13.4 y-Axis scaling

OK, this is a big one. You can make an effect seem massive or insignificant depending on how you scale a numeric y-axis.

Consider the following the example comparing the 9-month salaries of male and female assistant professors. The data come from the Academic Salaries dataset.

```
# load data
data(Salaries, package="carData")

# get means, standard deviations, and
```

```
# 95% confidence intervals for
# assistant professor salary by sex
library(dplyr)
df <- Salaries %>%
  filter(rank == "AsstProf") %>%
  group_by(sex) %>%
  summarize(n = n(),
            mean = mean(salary),
            sd = sd(salary),
            se = sd / sqrt(n),
            ci = qt(0.975, df = n - 1) * se)

df
```

```
## # A tibble: 2 x 6
##   sex        n   mean    sd    se    ci
##   <fct>  <int>  <dbl> <dbl> <dbl> <dbl>
## 1 Female    11 78050. 9372. 2826. 6296.
## 2 Male      56 81311. 7901. 1056. 2116.
```

```
# create and save the plot
library(ggplot2)
p <- ggplot(df,
            aes(x = sex, y = mean, group=1)) +
  geom_point(size = 4) +
  geom_line() +
  scale_y_continuous(limits = c(77000, 82000),
                     label = scales::dollar) +
  labs(title = "Mean salary differences by gender",
       subtitle = "9-mo academic salary in 2007-2008",
       caption = paste("source: Fox J. and Weisberg, S. (2011)",
                       "An R Companion to Applied Regression,",
                       "Second Edition Sage"),
       x = "Gender",
       y = "Salary") +
  scale_y_continuous(labels = scales::dollar)
```

First, let's plot this with a y-axis going from 77,000 to 82,000.

```
# plot in a narrow range of y
p + scale_y_continuous(limits=c(77000, 82000))
```

There appears to be a very large gender difference.

Next, let's plot the same data with the y-axis going from 0 to 125,000.

```
# plot in a wide range of y
p + scale_y_continuous(limits = c(0, 125000))
```

There doesn't appear to be any gender difference!

The goal of ethical data visualization is to represent findings with as little distortion as possible. This means choosing an appropriate range for the y-axis. Bar charts should almost always start at y = 0. For other charts, the limits really depends on a subject matter knowledge of the expected range of values.

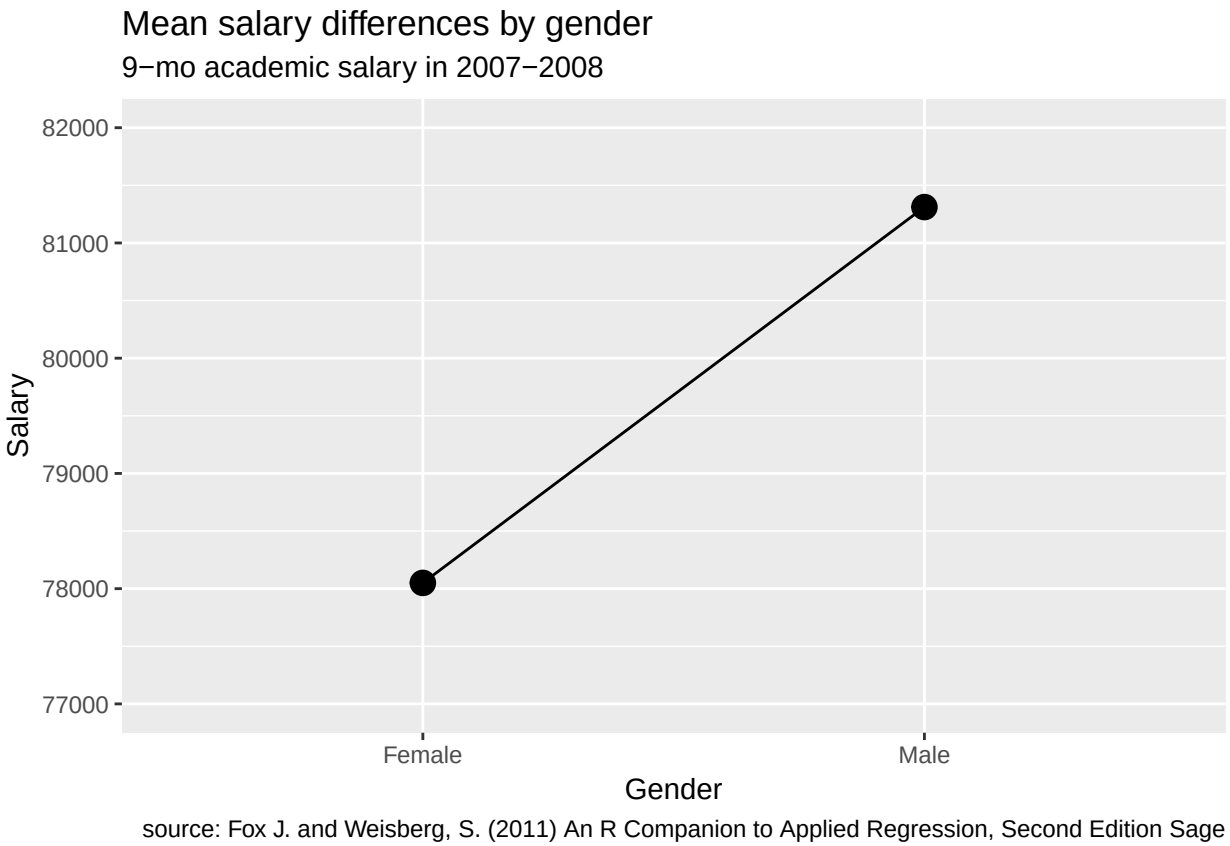


Figure 13.3: Plot with limited range of Y

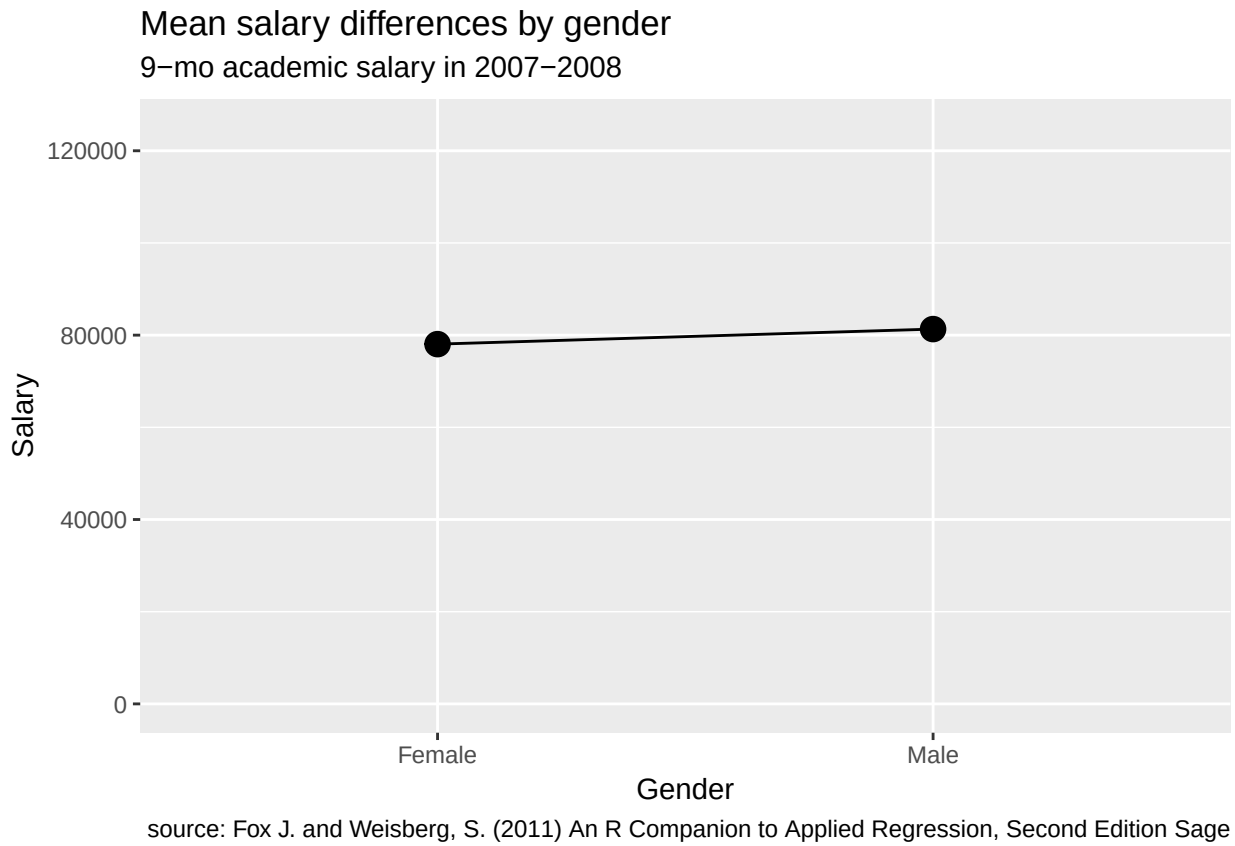
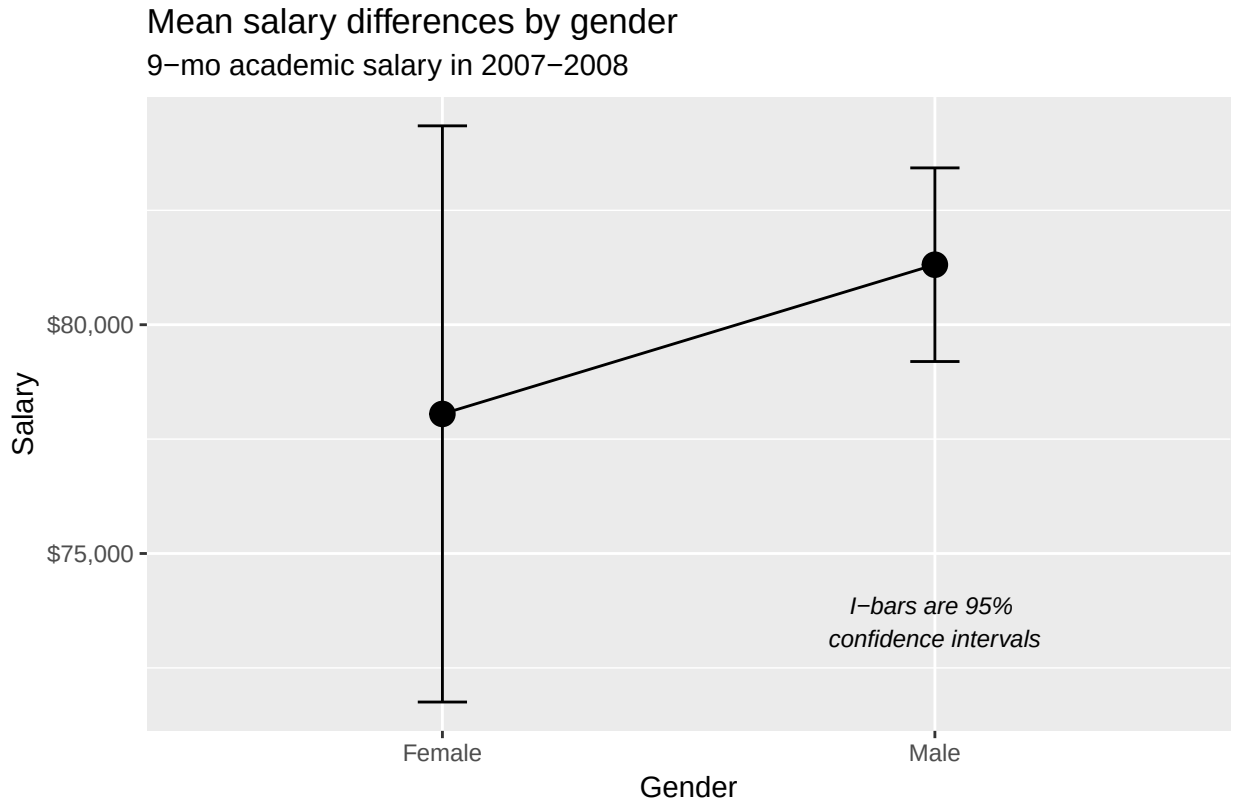


Figure 13.4: Plot with limited range of Y

We can also improve the graph by adding in an indicator of the uncertainty (see the section on Mean/SE plots).

```
# plot with confidence limits
p +  geom_errorbar(aes(ymin = mean - ci,
                       ymax = mean + ci),
                       width = .1) +
  ggplot2::annotate("text",
           label = "I-bars are 95% \nconfidence intervals",
           x=2,
           y=73500,
           fontface = "italic",
           size = 3)
```



source: Fox J. and Weisberg, S. (2011) An R Companion to Applied Regression, Second Edition Sage

The difference doesn't appear to exceeds chance variation.

13.5 Attribution

Unless it's your data, each graphic should come with an attribution - a note directing the reader to the source of the data. This will usually appear in the caption for the graph.

13.6 Going further

If you would like to learn more about `ggplot2` there are several good sources, including

- the `ggplot2` homepage
- the book ggplot2: Elegenat Graphics for Data Anaysis (be sure to get the second edition)
- the eBook R for Data Science - the data visualization chapter
- the `ggplot2` cheatsheet

If you would like to learn more about data visualization in general, here are some useful resources.

- Harvard Business Reviews - Visualizations that really work

- the website Information is Beautiful
- the book Beautiful Data: The Stories Behind Elegant Data Solutions
- the Wall Street Journal's - Guide to Information Graphics
- the book The Truthful Art

13.7 Final Note

With the growth (or should I say deluge?) of readily available data, the field of data visualization is exploding. This explosion is supported by the availability of exciting new graphical tools. It's a great time to learn and explore. Enjoy!

Appendix A

Datasets

The appendix describes the datasets used in this book.

A.1 Academic salaries

The Salaries for Professors dataset comes from the `carData` package. It describes the 9 month academic salaries of 397 college professors at a single institution in 2008-2009. The data were collected as part of the administration's monitoring of gender differences in salary.

The dataset can be accessed using

```
data(Salaries, package="carData")
```

It is also provided in other formats, so that you can practice importing data.

Format	File
Comma delimited text	Salaries.csv
Tab delimited text	Salaries.txt
Excel spreadsheet	Salaries.xlsx
SAS file	Salaries.sas7bdat
Stata file	Salaries.dta
SPSS file	Salaries.sav

A.2 Starwars

The starwars dataset comes from the `dplyr` package. It describes 13 characteristics of 87 characters from the Starwars universe. The data are extracted from the Star Wars API.

A.3 Mammal sleep

The msleep dataset comes from the `ggplot2` package. It is an updated and expanded version of a dataset by Save and West, describing the sleeping characteristics of 83 mammals.

The dataset can be accessed using

```
data(msleep, package="ggplot2")
```

A.4 Marriage records

The Marriage dataset comes from the `mosiacData` package. It is contains the marriage records of 98 individuals collected from a probate court in Mobile County, Alabama.

The dataset can be accessed using

```
data(Marriage, package="mosaicData")
```

A.5 Fuel economy data

The mpg dataset from the `ggplot2` package, contains fuel economy data for 38 popular models of car, for the years 1999 and 2008.

The dataset can be accessed using

```
data(mpg, package="ggplot2")
```

A.6 Gapminder data

The gapminder dataset from the `gapminder` package, contains longitudinal data (1952-2007) on life expectancy, GDP per capita, and population for 142 countries.

The dataset can be accessed using

```
data(gapminder, package="gapminder")
```

A.7 Current Population Survey (1985)

The CPS85 dataset from the `mosaicData` package, contains 1985 data on wages and other characteristics of workers.

The dataset can be accessed using

```
data(CPS85, package="mosaicData")
```

A.8 Houston crime data

The crime dataset from the `ggmap` package, contains the time, date, and location of six types of crimes in Houston, Texas between January 2010 and August 2010.

The dataset can be accessed using

```
data(crime, package="ggmap")
```

A.9 US economic timeseries

The economics dataset from the `ggplot2` package, contains the monthly economic data gathered from Jan 1967 to Jan 2015.

The dataset can be accessed using

```
data(economics, package="ggplot2")
```

A.10 Saratoga housing data

The Saratoga housing dataset contains information on 1,728 houses in Saratoga Country, NY, USA in 2006. Variables include price (in thousands of US dollars) and 15 property characteristics (lotsize, living area, age, number of bathrooms, etc.)

The dataset can be accessed using

```
data(SaratogaHouses, package="mosaicData")
```

A.11 US population by age and year

The uspopage dataset describes the age distribution of the US population from 1900 to 2002.

The dataset can be accessed using

```
data(uspopage, package="gcookbook")
```

A.12 NCCTG lung cancer data

The lung dataset describes the survival time of 228 patients with advanced lung cancer from the North Central Cancer Treatment Group.

The dataset can be accessed using

```
data(lung, package="survival")
```

A.13 Titanic data

The Titanic dataset provides information on the fate of Titanic passengers, based on class, sex, and age. The dataset comes in table form with base R. It is provided here as data frame.

The dataset can be accessed using

```
library(readr)
titanic <- read_csv("titanic.csv")
```

A.14 JFK Cuban Missle speech

The John F. Kennedy Address is a raw text file containing the president's October 22, 1962 speech on the Cuban Missle Crisis. The text was obtained from the JFK Presidential Library and Museum.

The text can be accessed using

```
library(readr)
text <- read_csv("JFKspeech.txt")
```

A.15 UK Energy forecast data

The UK energy forecast dataset contains data forecasts for energy production and consumption in 2050. The data are in an RData file that contains two data frames.

- The `node` data frame contains the names of the nodes (production and consumption types).
- The `links` data fame contains the *source* (originating node), *target* (target node), and *value* (flow amount between the nodes).

The data come from Mike Bostock's Sankey Diagrams page and the `network3D` homepage and can be accessed with the statement

```
load("Energy.RData")
```

A.16 US Mexican American Population

The Mexcian American Population data is a raw tab delimited text file containing the percentage of Mexican Americans by US state from the 2010 Census. The actual dataset was obtained from Wikipedia.

The data can be accessed using

```
library(readr)
text <- read_csv("mexican_american.csv")
```

Appendix B

About the Author

Robert Kabacoff is a data scientist with 30 years of experience in research methodology, data visualization, predictive analytics, and statistical programming.

As a Professor of the Practice in the Quantiative Analysis Center at Wesleyan University, he teaches courses in applied data analysis, machine learning, data journalism, and advance R programming.

Rob is the author of R in Action: Data analysis and graphics with R (2nd ed.), and maintains a popular website on R programming called Quick-R.

Appendix C

About the QAC

The Quantitative Analysis Center (QAC) is a collaborative effort of academic and administrative departments at Wesleyan University. It coordinates support for quantitative analysis across the curriculum, and provides an institutional framework for collaboration across departments and disciplines in the area of data analysis. Through its programs and courses, it seeks to facilitate data science education and the integration of quantitative teaching and research activities.