Introduction to Data Analysis with R & Reproducible Data Science

Institute for Research in Statistics and its Applications at the University of Minnesota

Contents

Home

With the increasing availability of data with broad applications (and the sheer size of some of these data), it is more important than ever to be able to elucidate trends, decisions, and stories from data. Our team will offer a hands on introduction to Data Science and Statistics using the free and publicly available software R. Assuming no background knowledge of software or Statistics, we will bring you up to speed on some of the most useful, modern, and popular data analysis techniques.

This short course is divided into multiple modules. On day one we will explore the basic features of R and the power of R for constructing visualizations, summaries, hypothesis tests, and statistical models from data. The modules on day two will cover a gentle introduction to logistic regression and conclude with an in-depth discussion on best practices for reproducible Data Science research and practice using R Markdown and github.

Chapter 1

Introduction to R & RStudio

Author: Alicia Johnson

This workshop is motivated by the increasing need for tools that can be used to elucidate trends, decisions, and stories from data. This practice is broadly referred to as "data science", a workflow for which is presented by Garrett Grolemund & Hadley Wickham:

source: http://r4ds.had.co.nz/explore-intro.html

Workshop Outline & Goals

Our goal is to provide a hands on introduction to navigating the data science pipeline with R and github. You will walk away with a solid foundation upon which you can build for your own research.

Day 1:

- Introduction to R & RStudio
- Data Visualization
- Simple Statistics
- Linear Regression

Day 2:

- Logistic regression
- Principles of Reproducible Research
- R Markdown
- Github in R

1.1 Getting Started

Whether your data are generated via simulation, collected via a survey, observed in a scientific experiment, scraped from the web, etc, you need *software* to explore and construct inferences from these data. In this workshop, we'll use the **R** statistical software. Why R?

- it's free
- it's open source
- it's flexible / useful for a wide variety of applications
- it has a huge online community
- it can be used to create reproducible documents, apps, books, etc. (In fact, *this* document was constructed within RStudio.)

Before the workshop, you were asked to download/update both R & RStudio:

1. **Download & install R**: https://mirror.las.iastate.edu/CRAN/
2. **Download & install RStudio**: https://www.rstudio.com/products/rstudio/download/
   Be sure to download the *free* version. Note that RStudio is an *integrated development environment (IDE)* for R, combining the power of R with extra automation tools.

Once you open RStudio, you'll see four panes, each serving a different function:

1.2 Basic features

Using R as a calculator

We can use R as a simple calculator. Try the following:

```
2 + 3
2 * 3
2^3
(2 + 3)^2
2 + 3^2
```

Comments

Once you start saving your work, it's helpful to *comment your code*. To this end, R ignores anything after `#`.

```
# Calculate 3 squared
3^2
```

Built-in functions

R also includes built-in *functions* to which we supply *arguments*: `function(arguments)`.

```
sqrt(9)

# The sum() function calculates the sum of the listed numbers
```

```
# Does the order of arguments matter?
sum(2, 3)
sum(3, 2)

# What does the rep() function do?
# Does the order of arguments matter?
rep(2, 3)
rep(3, 2)

# Arguments have names
rep(x = 3, times = 2)
rep(times = 2, x = 3)

# Is R case sensitive?  eg: Can we spell rep() as Rep()?
Rep(2, 3)
```

Assignment

We can assign & store R output.

```
# Store the result of rep(3, 2) as two_threes
two_threes <- rep(3, 2)

# Check out the results
two_threes

# Do something to the results
two_threes + 5

# Names cannot include spaces!
two threes <- rep(3,2)
```

1.3 Getting Help

- Curious about what happens if you change R code in some way? Try it! Playing around with a function is the best way to learn about its functionality.
- Can't remember the code you used in a past analysis? Search for it under the "History" tab in the upper right hand panel in RStudio.
- Did you make a mistake and don't want to retype all of your work? Use the up arrow! You can access & subsequently edit any previous line of code by using the up arrow.
- Don't know what a certain function does or how it works? You have a couple of options:
  - Use `?` within RStudio to access help files.

    ```
    ?rep
    ```

  - Google!
  - There's a massive RStudio community at http://stackoverflow.com/. If you have a question, somebody's probably already written about it.

1.4 Working with data

The following data were utilized in the fivethirtyeight.com article "The Dollar-And-Cents Case Against Hollywood's Exclusion of Women" which analyzes movies that do/don't pass the **Bechdel test**. Image from Wikipedia:

A movie passes the test if it meets the following criteria:

- there are ≥ 2 female characters
- the female characters talk to each other
- at least 1 time, they talk about something other than a male character

The fivethirtyeight analysis utilizes the following data:

Data Structure

Tidy data tables have two key features:

1. Each row represents a single **observational unit** of the sample.
2. Each column represents a **variable**, ie. an attribute of the cases.
3. The data are not treated like code. There are no extras - no row summaries, column summaries, data entry notes, comments, graphs, etc. All data manipulation should be done within R! All comments about the data collection, variables, etc should be provided in a separate **code book**.

Question: What are the units of observation in the above data? What are the variables?

Accessing / Importing Data

Data are stored in countless different locations (eg: your computer, Google drive, Wiki, etc) and in countless formats (eg: xls, csv, tables, etc). Luckily for us, the Bechdel data are already stored within R in the `fivethirtyeight` **package**. In general, packages developed & shared by R users provide specialized functions and data. **IF AND ONLY IF** you didn't install the `fivethirtyeight` package before the workshop, you'll need to do so now:

```
install.packages("fivethirtyeight", dependencies = TRUE)
```

Once you install the package, you can load the package in your R session and access the `bechdel` data:

```
library(fivethirtyeight)
data(bechdel)
```

You can also access the **codebook** for these data:

```
?bechdel
```

Examining Data Structure in R

Before we do any data analysis we have to understand the structure of our data. Try the following.

```
# View the data table in a separate panel
View(bechdel)

# Check out the first rows in the console
head(bechdel)

# Obtain the data dimensions: rows x columns
dim(bechdel)

# Get the variable names
names(bechdel)
```

Examining Specific Variables

```
# Access a single variable using "$"
bechdel$budget_2013
bechdel$clean_test

# Determine levels/categories of categorical variables
levels(factor(bechdel$clean_test))
levels(factor(bechdel$binary))
```

Subsetting Specific Units of Observation

We can obtain a subset of observations that satisfy a criterion defined by a variable within the data set:

```
# Subset of movies with a 2013 budget under 1 million dollars
cheap <- subset(bechdel, budget_2013 < 1000000)
dim(cheap)
head(cheap)

# Subset of movies that fail the test
failures <- subset(bechdel, binary == "FAIL")
dim(failures)
head(failures)

# Subset of movies that EITHER have a budget under 1 million dollars OR fail the test
cheap_or_fail <- subset(bechdel, budget_2013 < 1000000 | binary == "FAIL")
dim(cheap_or_fail)
```

```
head(cheap_or_fail)

# Subset of movies that BOTH have a budget under 1 million dollars AND fail the test
cheap_and_fail <- subset(bechdel, budget_2013 < 1000000 & binary == "FAIL")
dim(cheap_and_fail)
head(cheap_and_fail)
```

Some useful syntax for subsetting:

- `<` (less than), `<=` (less than or equal to), `>` (greater than), `>=` (greater than or equal to), `==` (equal to)
- `&` (and), `|` (or)

1.5 Exercises

Let's apply the above tools to the `US_births_2000_2014` data within the `fivethirtyeight package`. These data were used in the fivethirtyeight.com article "Some People Are Too Superstitious To Have A Baby On Friday The 13th".

1. Load the data into your console and examine the codebook.
2. View the data set in a separate panel.
3. Check out the first 6 cases in your console.
4. What are the units of observation (rows)? What are the variables?
5. How much data do we have?
6. What are the names of the variables?
7. Access the `day_of_week` variable alone. What are the levels/category labels for this variable?
8. Create a subset that contains only the births that occur on Fridays. Store this as `OnlyFridays`. Find the dimensions of this subset.
9. Create a subset that contains only births in 2014. Store this as `Only2014`. Find the dimensions of this subset.

Solutions:

```
#1
library(fivethirtyeight)
data(US_births_2000_2014)

#2
View(US_births_2000_2014)

#3
head(US_births_2000_2014)

#4
# Each row = a single day
# Variables include number of births, day of week, etc on that date

#5
dim(US_births_2000_2014)

#6
```

```r
names(US_births_2000_2014)

#7
US_births_2000_2014$day_of_week
levels(factor(US_births_2000_2014$day_of_week))

#8
OnlyFridays <- subset(US_births_2000_2014, day_of_week == "Fri")
dim(OnlyFridays)

#9
Only2014 <- subset(US_births_2000_2014, year == 2014)
dim(Only2014)
```

Chapter 2

Data Visualization

Author: Alicia Johnson

The following data set on the 2016 election is stored as a *csv* file at https://www.macalester.edu/~ajohns24/Data/electionDemographics16.csv:

This data set combines the county level election results provided by Tony McGovern (shared on github), county level demographic data from the `df_county_demographics` data set within the `choroplethr` R package, and historical information about red/blue/purple states. Let's take a quick look:

```
# Use read.csv() to import the csv file
election <- read.csv("https://www.macalester.edu/~ajohns24/Data/electionDemographics16.csv")
```

```
# Check it out!
dim(election)       # dimensions
head(election, 2)   # first 2 rows
names(election)     # variable names
```

Now that we understand the structure of this data set, we can start to ask some questions:

- To what degree did Trump support vary from county to county?
- In what number of counties did Trump win?
- What's the relationship between Trump's 2016 support and Romney's 2012 support?
- What's the relationship between Trump's support and the "color" of the state in which the county exists?

Visualizing the data is the first natural step in answering these questions. Why?

- Visualizations help us understand what we're working with: What are the scales of our variables? Are there any outliers, i.e. unusual cases? What are the patterns among our variables?
- This understanding will inform our next steps: What statistical tool / model is appropriate?
- Once our analysis is complete, visualizations are a powerful way to communicate our findings and tell a story.

2.1 ggplot

We'll construct visualizations using the `ggplot` function in RStudio. Though the `ggplot` learning curve can be steep, its "grammar" is intuitive and generalizable once mastered. The `ggplot` plotting function is stored in the `ggplot2` package:

```
library(ggplot2)
```

The best way to learn about `ggplot` is to just play around. Don't worry about memorizing the syntax. Rather, focus on the *patterns* and *potential* of their application. There's a helpful cheat sheet for future reference:

GGPLOT CHEAT SHEET

2.2 Univariate visualizations

We'll start with **univariate** visualizations.

Categorical Variables

Consider the categorical `winrep_2016` variable which indicates whether Trump won the county:

```
levels(factor(election$winrep_2016))
## [1] "FALSE" "TRUE"
```

A table provides a simple summary of the number of counties that fall into these 2 categories:

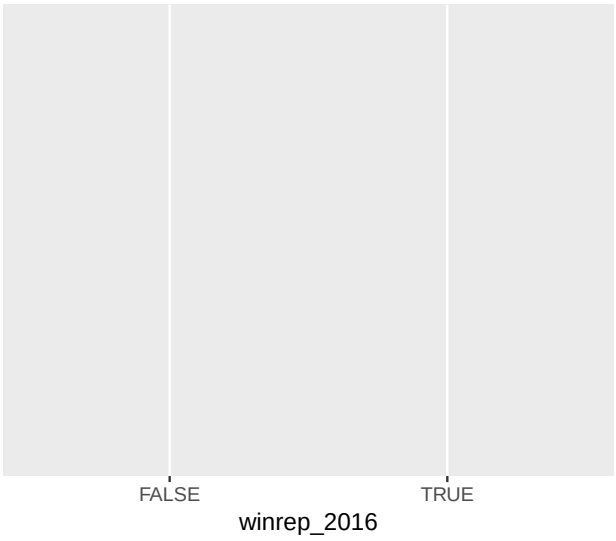
```
table(election$winrep_2016)
```

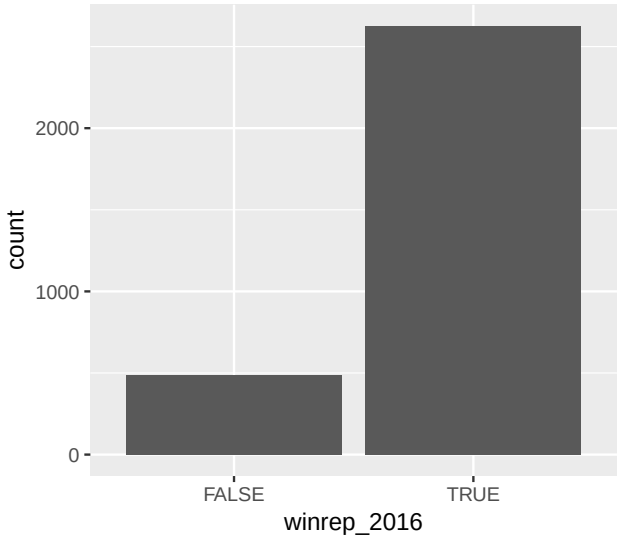
```
##
## FALSE  TRUE
##   487  2625
```

A bar chart provides a visualization of this table. Try out the code below that builds up from a simple to a customized bar chart. At each step determine how each piece of code contributes to the plot.

```
# Set up a plotting frame
ggplot(election, aes(x = winrep_2016))

# Add a layer with bars
ggplot(election, aes(x = winrep_2016)) +
    geom_bar()
```

In summary:





Quantitative Variables

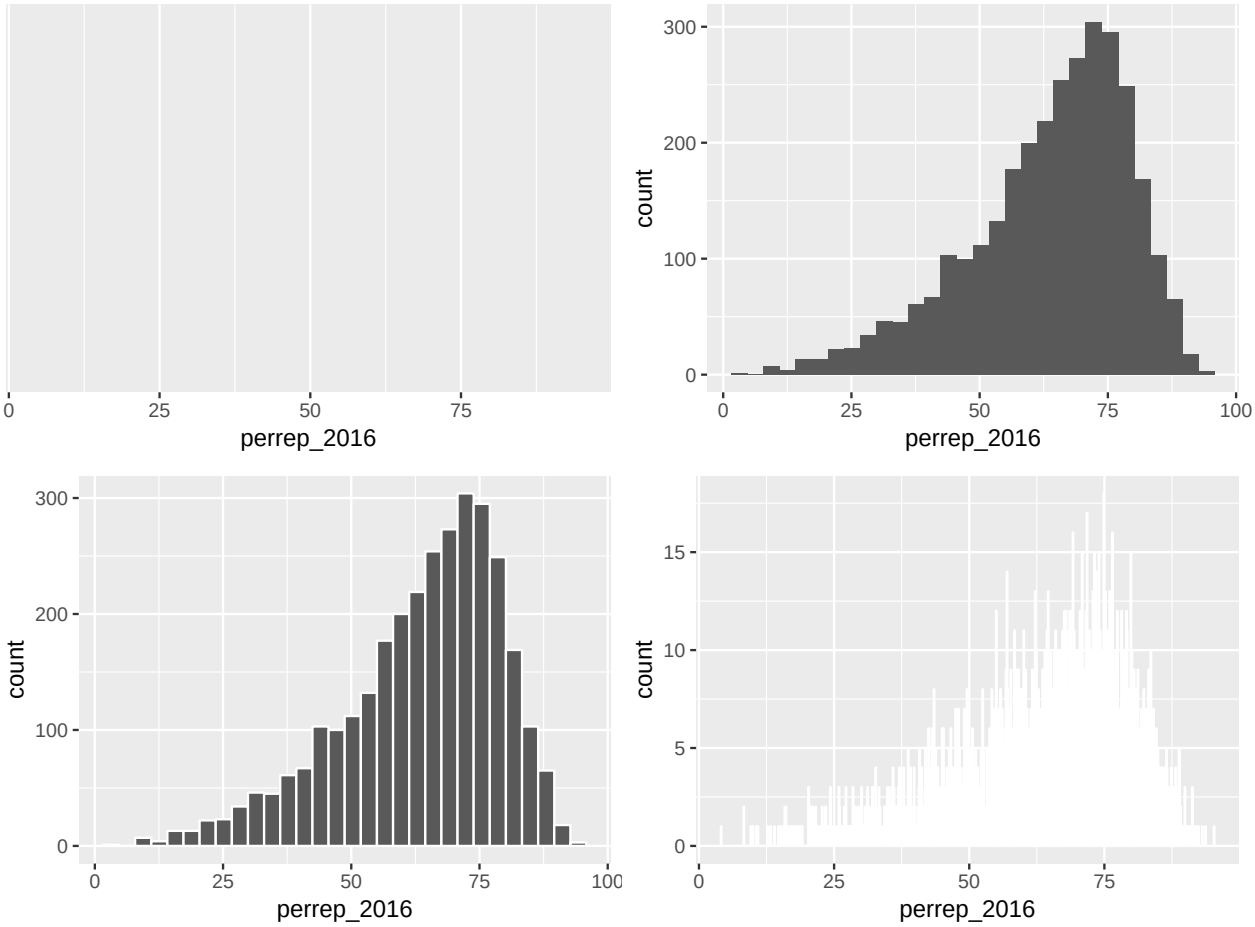
The quantitative `perrep_2016` variable summarizes Trump's percent of the vote in each county. Quantitative variables require different summary tools than categorical variables. We'll explore 2 methods for graphing quantitative variables: histograms & density plots.

Histograms are constructed by (1) dividing up the observed range of the variable into 'bins' of equal width; and (2) counting up the number of cases that fall into each bin. Try out the code below.

```
# Set up a plotting frame
ggplot(election, aes(x = perrep_2016))

# Add a histogram layer
ggplot(election, aes(x = perrep_2016)) +
    geom_histogram()

# Change the border colors
ggplot(election, aes(x = perrep_2016)) +
```

```
    geom_histogram(color = "white")

# Change the bin width
ggplot(election, aes(x = perrep_2016)) +
    geom_histogram(color = "white", binwidth = 0.10)
```

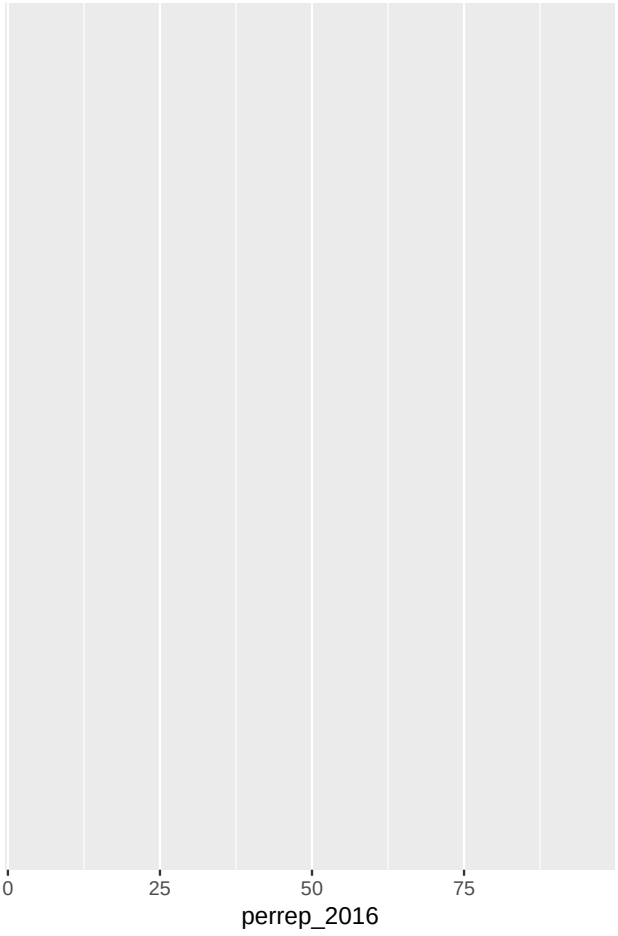
In summary:

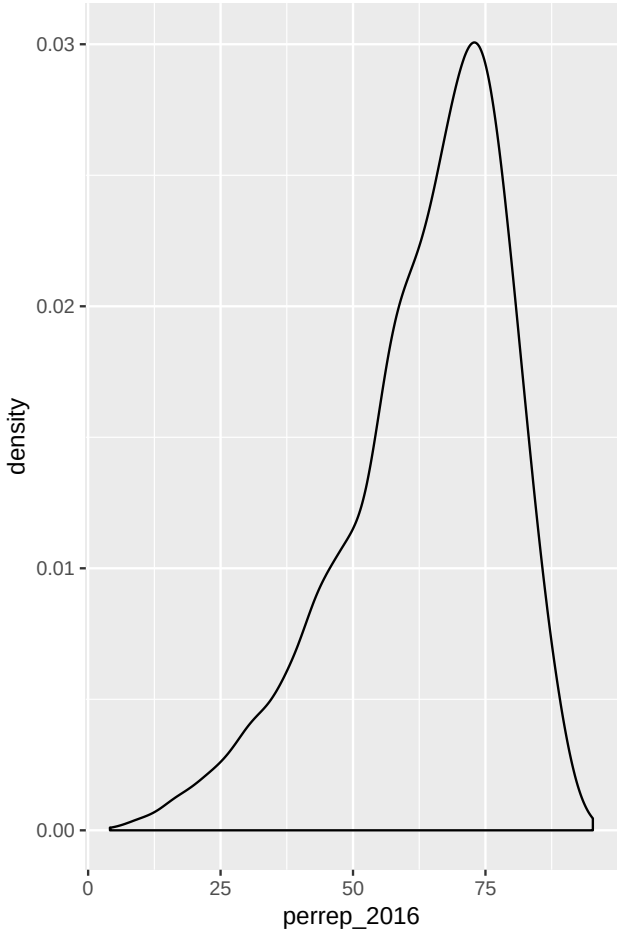


Density plots are essentially smooth versions of the histogram. Instead of sorting cases into discrete bins, the "density" of cases is calculated across the entire range of values. The greater the number of cases, the greater the density! The density is then scaled so that the area under the density curve **always equals 1** and the area under any fraction of the curve represents the fraction of cases that lie in that range.

```
# Set up the plotting frame
ggplot(election, aes(x = perrep_2016))

# Add a density curve
ggplot(election, aes(x = perrep_2016)) +
    geom_density()
```

In summary:





2.3 Visualizing Relationships

Consider the data on just 6 of the counties:

Before constructing graphics of the relationships among these variables, we need to understand what features these graphics should have. Without peaking at the exercises, challenge yourself to think about how we might graph the relationships among the following sets of variables:

- `perrep_2016` vs `perrep_2012`
- `perrep_2016` vs `StateColor`
- `perrep_2016` vs `perrep_2012` *and* `StateColor` (in 1 plot)
- `perrep_2016` vs `perrep_2012` *and* `median_rent` (in 1 plot)

Run through the following exercises which introduce different approaches to visualizing *relationships*.

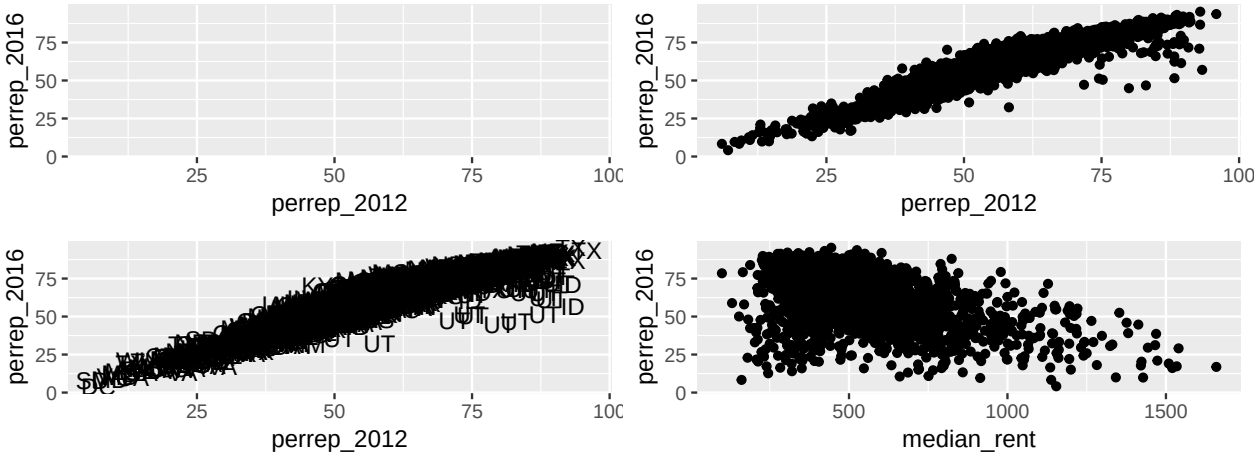
Scatterplots of 2 quantitative variables

Each quantitative variable has an axis. Each case is represented by a dot.

```
# Just a graphics frame
ggplot(election, aes(y = perrep_2016, x = perrep_2012))

# Add a scatterplot layer
ggplot(election, aes(y = perrep_2016, x = perrep_2012)) +
    geom_point()

# Add a scatterplot layer: label by state
ggplot(election, aes(y = perrep_2016, x = perrep_2012)) +
    geom_text(aes(label = abb))

# Another predictor
```

```
ggplot(election, aes(y = perrep_2016, x = median_rent)) +
    geom_point()
```

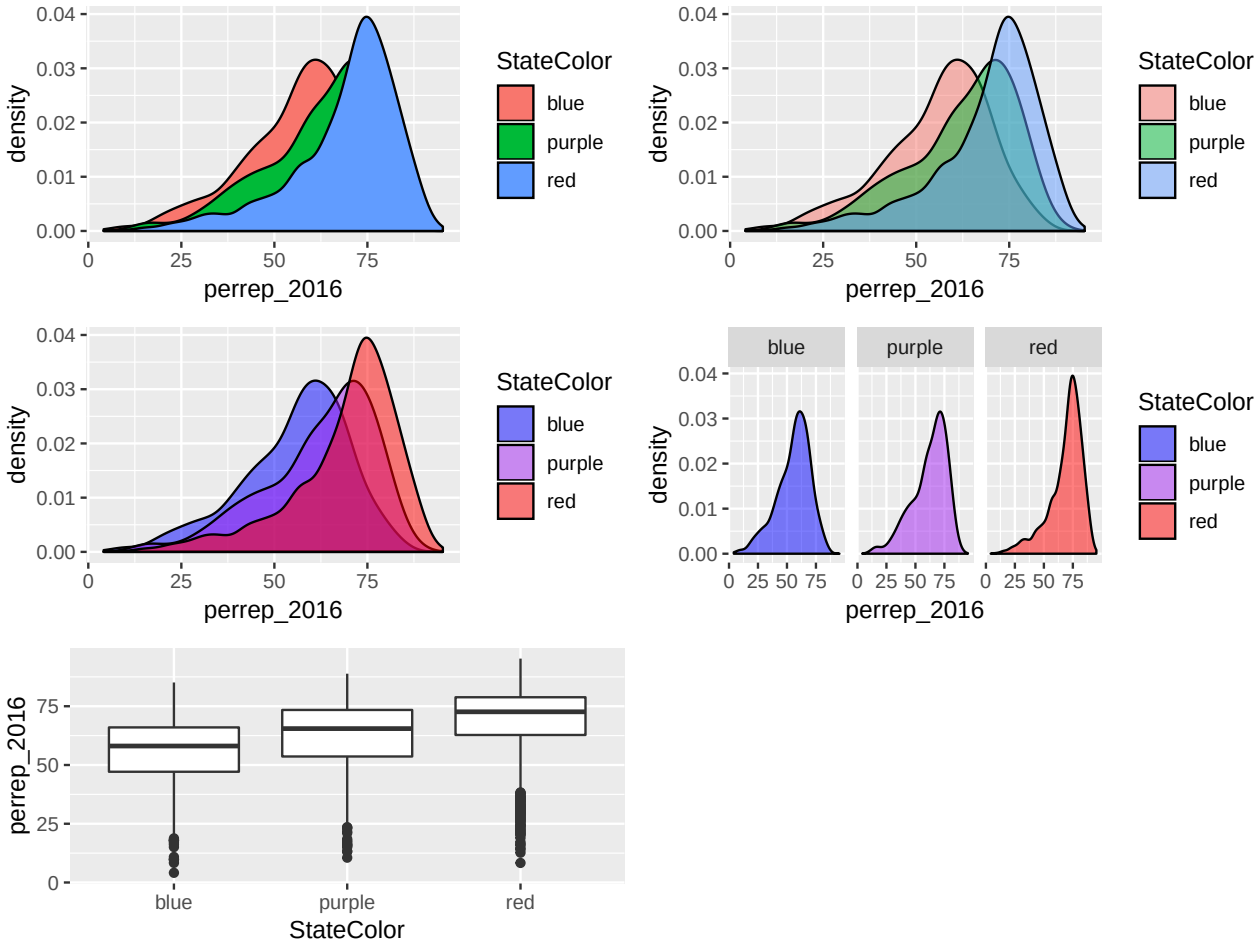
In summary:



Side-by-side plots of 1 quantitative variable vs 1 categorical variable

```
# Density plots by group
ggplot(election, aes(x = perrep_2016, fill = StateColor)) +
    geom_density()

# To see better: add transparency
ggplot(election, aes(x = perrep_2016, fill = StateColor)) +
    geom_density(alpha = 0.5)

# Fix the color scale!
ggplot(election, aes(x = perrep_2016, fill = StateColor)) +
    geom_density(alpha = 0.5) +
    scale_fill_manual(values = c("blue","purple","red"))

# To see better: split groups into separate plots
ggplot(election, aes(x = perrep_2016, fill = StateColor)) +
    geom_density(alpha = 0.5) +
    facet_wrap( ~ StateColor) +
    scale_fill_manual(values=c("blue","purple","red"))


# Or use boxplots!
ggplot(election, aes(x = StateColor, y = perrep_2016)) +
  geom_boxplot()
```

In summary:

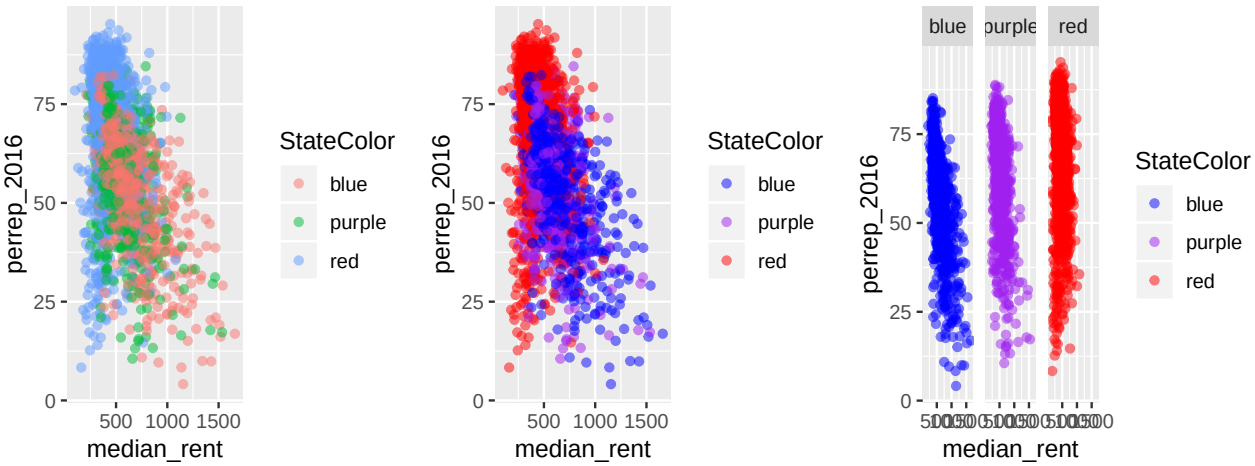


Scatterplots of 1 quantitative variable vs 1 categorical & 1 quantitative variable

If `median_rent` and `StateColor` both explain some of the variability in `perrep_2016`, why not include both in our analysis?! Let's.

```
# Scatterplot: id groups using color
ggplot(election, aes(y = perrep_2016, x = median_rent, color = StateColor)) +
    geom_point(alpha = 0.5)

# Fix the color scale!
ggplot(election, aes(y = perrep_2016, x = median_rent, color = StateColor)) +
    geom_point(alpha = 0.5) +
    scale_color_manual(values = c("blue","purple","red"))

# Scatterplot: split/facet by group
ggplot(election, aes(y=perrep_2016, x=median_rent, color=StateColor)) +
    geom_point(alpha=0.5) +
    facet_wrap( ~ StateColor) +
    scale_color_manual(values=c("blue","purple","red"))
```

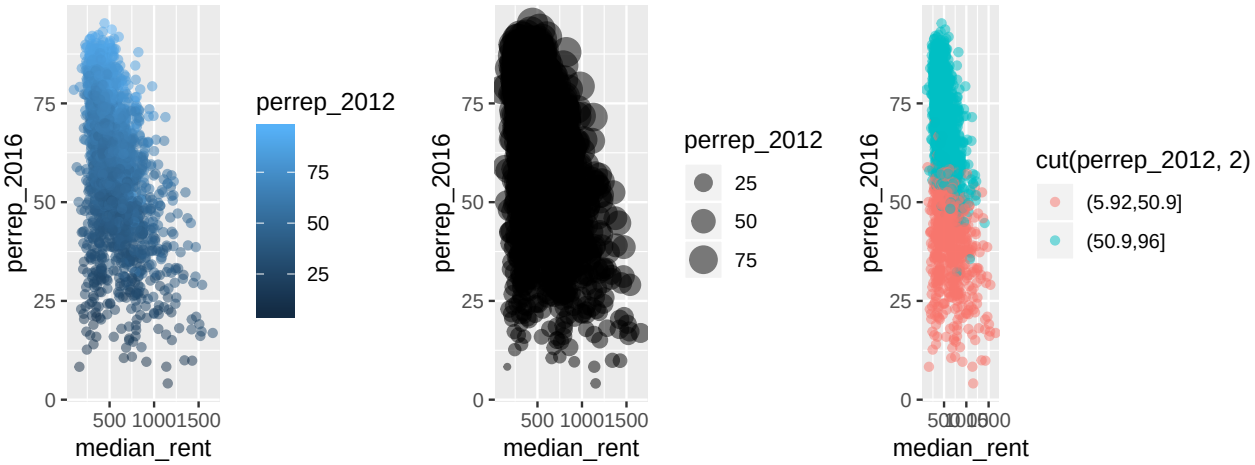
In summary:



Plots of 3 quantitative variables

```
# Scatterplot: represent third variable using color
ggplot(election, aes(y = perrep_2016, x = median_rent, color = perrep_2012)) +
    geom_point(alpha = 0.5)

# Scatterplot: represent third variable using size
ggplot(election, aes(y = perrep_2016, x = median_rent, size = perrep_2012)) +
    geom_point(alpha = 0.5)

# Scatterplot: discretize the third variable
ggplot(election, aes(y = perrep_2016, x = median_rent, color = cut(perrep_2012,3))) +
    geom_point(alpha = 0.5)
```

In summary:



2.4 Exercises

Recall the `US_births_2000_2014` data in the `fivethirtyeight package`:

```
library(fivethirtyeight)
data("US_births_2000_2014")
```

In the previous activity, we investigated the basic features of this data set:

```
dim(US_births_2000_2014)
## [1] 5479    6
head(US_births_2000_2014, 2)
## # A tibble: 2 x 6
##    year month date_of_month date       day_of_week births
##   <int> <int>         <int> <date>     <ord>        <int>
## 1  2000     1             1 2000-01-01 Sat           9083
## 2  2000     1             2 2000-01-02 Sun           8006
names(US_births_2000_2014)
## [1] "year"          "month"         "date_of_month" "date"
## [5] "day_of_week"   "births"
levels(factor(US_births_2000_2014$day_of_week))
## [1] "Sun"   "Mon"   "Tues"  "Wed"   "Thurs" "Fri"   "Sat"
```

Let's graphically explore these variables and the relationships among them! **Solutions are below.** NOTE: This set of exercises is inspired by the work of Randy Pruim for the MAA statPREP program.

1. First, let's focus on 2014:

   ```
   only_2014 <- subset(US_births_2000_2014, year == 2014)
   ```

   Construct a *univariate* visualization of `births`. Describe the variability in births from day to day in 2014.

2. The time of year might explain some of this variability. Construct a plot that illustrates the relationship between `births` and `date` in 2014. NOTE: Make sure that births, our variable of interest, is on the y-axis and treat `date` as quantitative.

3. One goofy thing that stands out are the 2-3 distinct groups of points. Add a layer to this plot that explains the distinction between these groups.

4. There are some exceptions to the rule in exercise 3, ie. some cases that should belong to group 1 but behave like the cases in group 2. Explain why these cases are exceptions - what explains the anomalies / why these are special cases?

5. Next, consider all births from 2000-2014. Construct 1 graphic that illustrates births trends across all of these years.

6. Finally, consider only those births that occur on Fridays:

   ```
   only_fri <- subset(US_births_2000_2014, day_of_week=="Fri")
   ```

   Define a new variable `fri13` that indicates whether the case falls on a Friday in the 13th date of the month:

   ```
   only_fri$fri13 <- (only_fri$date_of_month == 13)
   ```

   Construct and comment on a plot of that illustrates the distribution of births among Fridays that fall on & off the 13th. Do you see any evidence of superstition?

SOLUTIONS:

```
# 1
only_2014 <- subset(US_births_2000_2014, year == 2014)
ggplot(only_2014, aes(x = births)) +
  geom_histogram(color = "white")
ggplot(only_2014, aes(x = births)) +
  geom_density()

#2
```

```
ggplot(only_2014, aes(y = births, x = date)) +
  geom_point()

#3
ggplot(only_2014, aes(y = births, x = date, color = day_of_week)) +
  geom_point()
ggplot(only_2014, aes(y = births, x = date)) +
    geom_text(aes(label = day_of_week))

#4
#HOLIDAYS!  For example, Thanksgiving is a Thursday in late November

#5
ggplot(US_births_2000_2014, aes(y = births, x = date, color = day_of_week)) +
  geom_point()

#6
only_fri <- subset(US_births_2000_2014, day_of_week=="Fri")
only_fri$fri13 <- (only_fri$date_of_month == 13)
ggplot(only_fri, aes(x = births, fill = fri13)) +
  geom_density(alpha = 0.5)
ggplot(only_fri, aes(y = births, x = fri13)) +
  geom_boxplot()
```

2.5 Extra

We've covered some basic graphics. However, different types of relationships require different visualization strategies. For example, there's a geographical component to the `election` data. If you have time, try to construct some maps of the election related variables. To this end, you'll need to install the `choroplethr` and `choroplethrMaps` packages:

```
install.packages("choroplethr", dependencies = TRUE)
install.packages("choroplethrMaps", dependencies = TRUE)
```

```
library(choroplethr)
library(choroplethrMaps)

# Make a map of Trump support
election$value <- election$perrep_2016
county_choropleth(election)

# A map of Trump wins
election$value <- election$winrep_2016
county_choropleth(election)

# Try another variable of interest to you!!
```

Chapter 3

Simple Statistics in R

Author: Nathaniel E. Helwig

3.1 Chapter Outline and Goals

In this chapter, we will cover how to...

- Load, explore, and summarize data
- Calculate descriptive statistics
- Create reproducible plots and tables
- Perform one and two sample t-tests
- Fit one-way analysis of variance models
- Conduct simple correlation tests

R has many helpful functions for simple descriptive and inferential statistics, which make reproducible research easy!

3.2 Minnesota Beer Data

3.2.1 Overview

The Minnesota beer data has 44 beers measured on 7 variables:

1) *Brewery*: Name of the brewery (**factor** with 8 levels)
2) *Beer*: Name of the beer (**factor** with 44 levels)
3) *Description*: Description of the beer (**factor** with 37 levels)
4) *Style*: Style of the beer (**factor** with 3 levels)
5) ABV: Alcohol by volume (**numeric**)
6) IBU: International bitterness units (**integer**)
7) *Rating*: Beer Advocate rating (**integer**)

Data obtained by NEH from Beer Advocate and the websites of the eight breweries.

3.2.2 Load the Data

Use the **read.csv** function to load the *beer* data into R

```
beer <- read.csv("http://users.stat.umn.edu/~helwig/notes/MNbeer.csv")
```

The **dim** function returns the number of rows and columns of the data frame

```
dim(beer)
```

```
## [1] 44  7
```

The *beer* data frame has 44 beers (rows) measured on 7 variables (columns).

The **names** function returns the names of the variables in a data frame

```
names(beer)
```

```
## [1] "Brewery"     "Beer"        "Description" "Style"       "ABV"
## [6] "IBU"         "Rating"
```

3.2.3 Look at the Data

The **head** function returns the first six lines of a data frame

```
head(beer)
```

```
##       Brewery         Beer              Description Style ABV IBU Rating
## 1     Bauhaus  Wonderstuff     New Bohemian Pilsner Lager 5.4  48     88
## 2     Bauhaus    Stargazer German Style Schwarzbier Lager 5.0  28     87
## 3     Bauhaus  Wagon Party    West Cost Style Lager Lager 5.4  55     86
## 4     Bauhaus    Sky-Five!        Midwest Coast IPA   IPA 6.7  70     86
## 5 Bent Paddle         Kanu         Session Pale Ale   Ale 4.8  48     85
## 6 Bent Paddle Venture Pils            Pilsner Lager Lager 5.0  38     87
```

The **summary** function provides a summary of each variable in a data frame

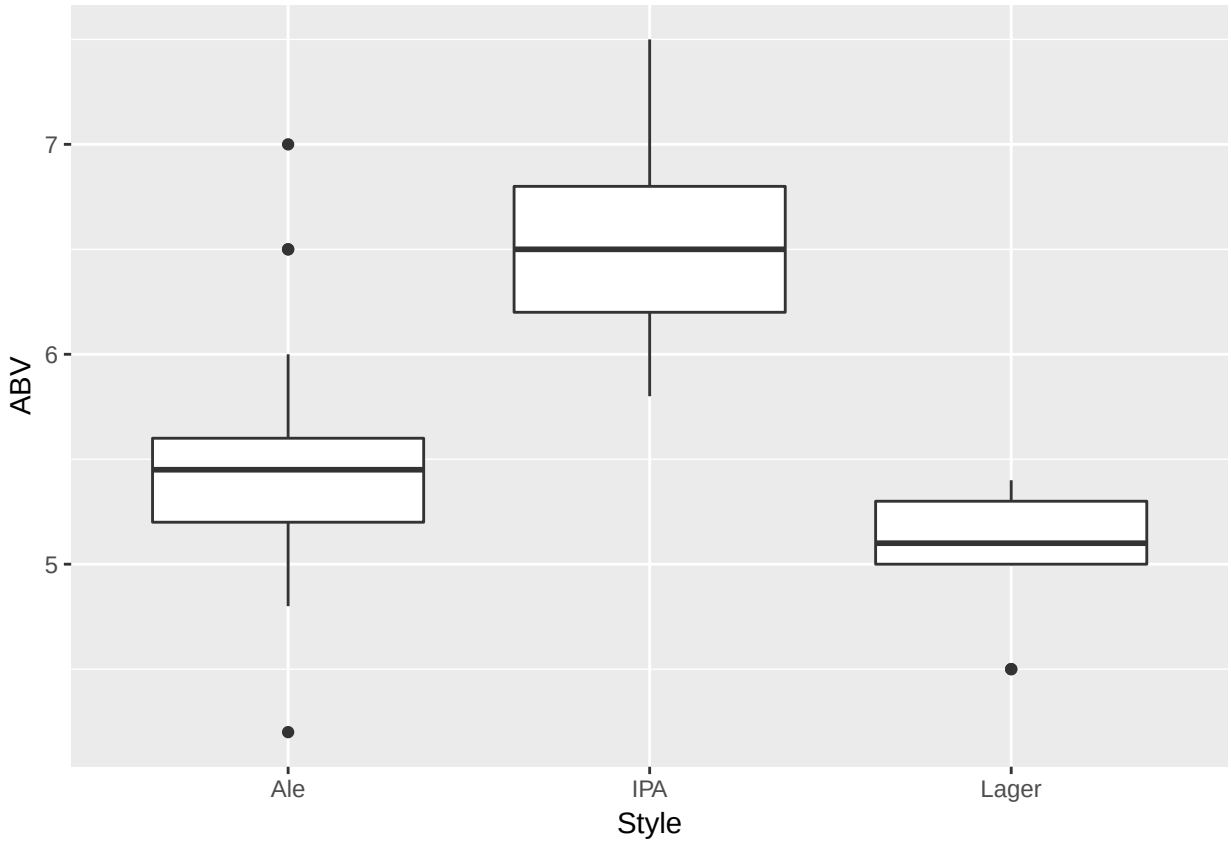
```
summary(beer)
```

```
##           Brewery                Beer                    Description
##  Indeed       :7   14* ESB         : 1   India Pale Ale        : 5
##  Summit       :7   B-Side Pils     : 1   English IPA           : 2
##  Surly        :7   Batch 300       : 1   Pilsner Lager         : 2
##  Bent Paddle  :5   Bender          : 1   Porter                : 2
##  Fulton       :5   Bent Hop        : 1   American Blonde Ale   : 1
##  Urban Growler:5   Big Boot Rye IPA: 1   Belgian Style Pale Ale: 1
##  (Other)      :8   (Other)         :38   (Other)               :31
##    Style         ABV             IBU            Rating
##  Ale  :18   Min.   :4.200   Min.   :15.00   Min.   :79.00
##  IPA  :17   1st Qu.:5.200   1st Qu.:33.00   1st Qu.:85.00
##  Lager: 9   Median :5.600   Median :48.50   Median :87.00
##             Mean   :5.818   Mean   :51.07   Mean   :87.18
##             3rd Qu.:6.500   3rd Qu.:68.25   3rd Qu.:90.00
##             Max.   :7.500   Max.   :99.00   Max.   :98.00
##
```

3.2.4 Plot the Data

We can check out boxplots of the relationship between ABV and style of beer:

```
ggplot(beer, aes(x = Style, y = ABV)) +
  geom_boxplot()
```

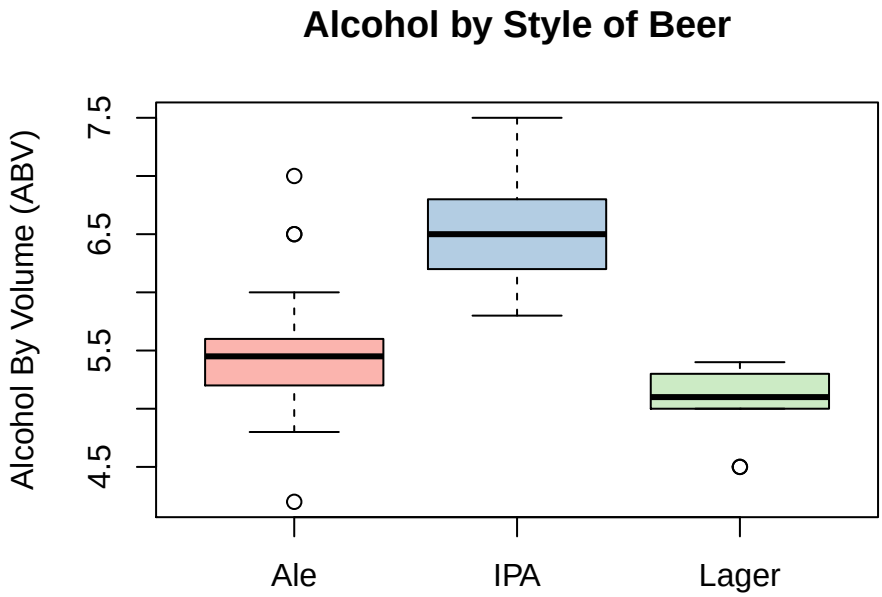


There is, of course, more than one way to achieve this task! For example, we can also make boxplots using the `boxplot()` function in "base R". The **brewer.pal** function in the *RColorBrewer* package creates ColorBrewer palettes for plotting

```
library(RColorBrewer)
MyColors <- brewer.pal(nlevels(beer$Style), "Pastel1")
```

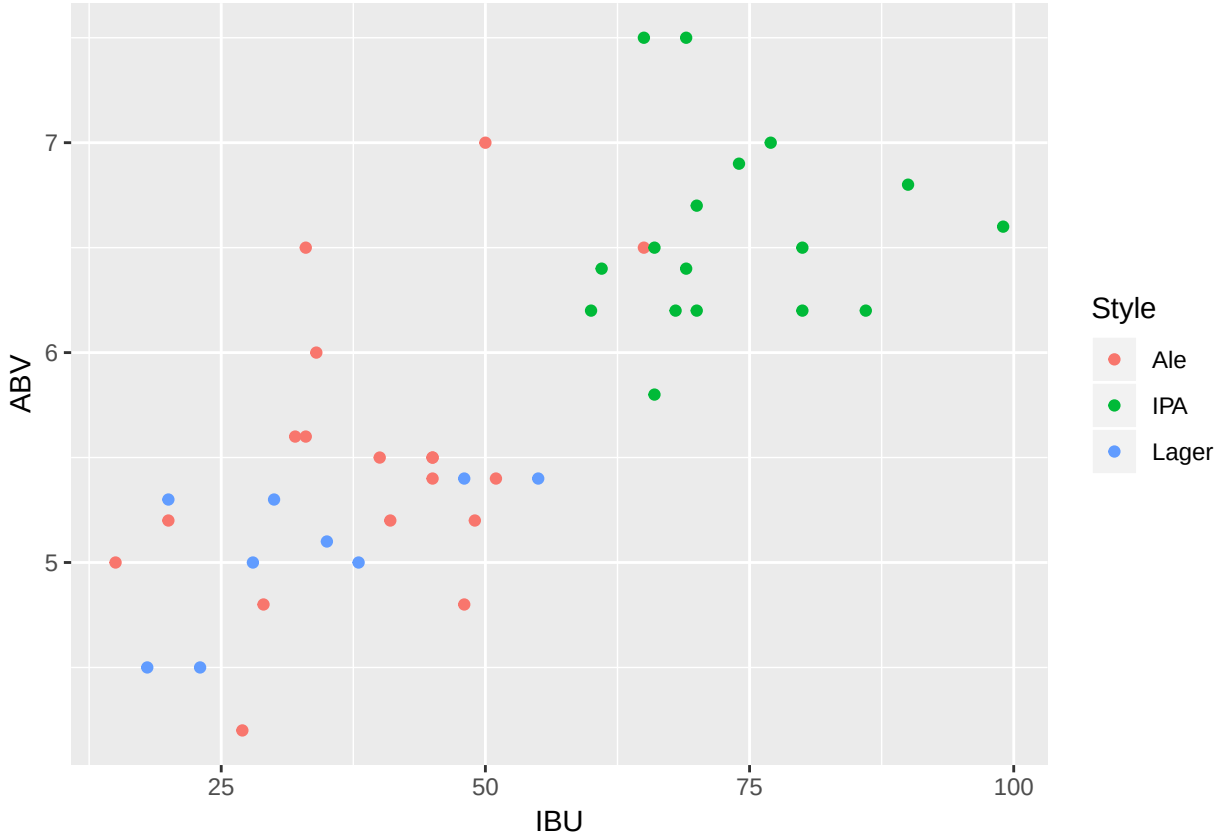
The **boxplot** function creates simple boxplots

```
boxplot(ABV ~ Style, data = beer, ylab = "Alcohol By Volume (ABV)",
        main = "Alcohol by Style of Beer", col = MyColors)
```



Further, we can plot the relationship between `ABV` and `IBU` by `Style`:

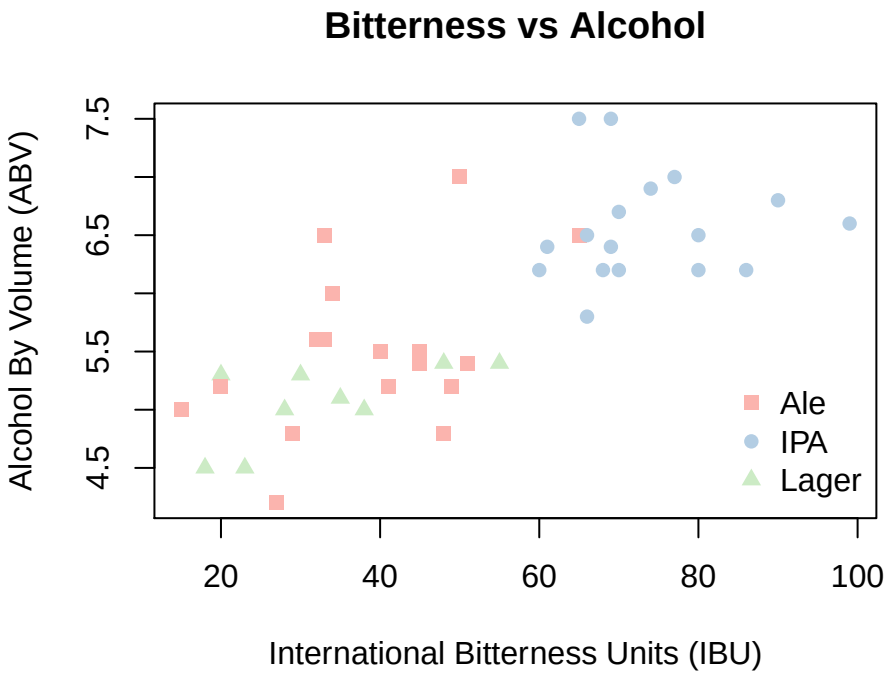
```
ggplot(beer, aes(x = IBU, y = ABV, color = Style)) +
  geom_point()
```



Alternatively, we can use the **plot()** function:

```
StyleInt <- as.integer(beer$Style)
plot(beer$IBU, beer$ABV, xlab = "International Bitterness Units (IBU)",
```

```
     ylab = "Alcohol By Volume (ABV)", pch = StyleInt + 14,
     main = "Bitterness vs Alcohol", col = MyColors[StyleInt])
legend("bottomright", legend = levels(beer$Style), pch = 15:17,
       col = MyColors, bty = "n")
```



R has functions for saving plots to various figure formats:

- **bmp** function for bitmap graphics
- **jpeg** function for JPEG graphics
- **png** function for Portable Network Graphics
- **tiff** function for Tag Image File Format graphics
- **pdf** or **dev.copy2pdf** functions for Portable Document Format graphics
- **postscript** or **dev.copy2eps** functions for PostScript graphics

3.3 Descriptive Statistics in R

3.3.1 Overview

We often need to calculate simple descriptive statistics of variables in a data frame, e.g., to make summary tables. As we have already seen, R is a function based and object oriented programming language. To obtain descriptive statistics, we input an object (e.g., column of a data frame) into the corresponding function. Thankfully, functions in R often have intuitive names—you can typically guess the name of the function you need!

3.3.2 Minimum and Maximum

To calculate the minimum or maximum of a variable, we could use the **min** or **max** functions

```
min(beer$ABV)
```

```
## [1] 4.2
```

```
max(beer$ABV)
```

```
## [1] 7.5
```

or the **range** function to return both the minimum and maximum

```
range(beer$ABV)
```

```
## [1] 4.2 7.5
```

The minimum ABV in the sample is 4.2% and the maximum is 7.5%. To determine which beers have the min/max ABV values, we can use the **which.min** and **which.max** functions

```
minmaxID <- c(which.min(beer$ABV), which.max(beer$ABV))
beer[minmaxID,]
```

```
##    Brewery              Beer      Description Style ABV IBU Rating
## 12  Indeed Lucy Session Sour Session Sour Ale   Ale 4.2  27     86
## 38   Surly         Overrated   West Coast IPA   IPA 7.5  69     91
```

3.3.3 Mean, Standard Deviation, and Variance

The **mean** function calculates the sample mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$

```
mean(beer$ABV)
```

```
## [1] 5.818182
```

The **sd** function calculates the sample standard deviation $s = \{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2\}^{1/2}$

```
sd(beer$ABV)
```

```
## [1] 0.8176178
```

The **var** function calculates the sample variance $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$

```
var(beer$ABV)
```

```
## [1] 0.6684989
```

The mean ABV is about 5.82% with a standard deviation of about 0.82% (variance of about 0.67%).

3.3.4 Medians and Quantiles

The **median** function calculates the sample median of a vector

```
median(beer$ABV)
```

```
## [1] 5.6
```

and the **quantile** function can be used for other quantiles

```
quantile(beer$ABV)
```

```
##   0%  25%  50%  75% 100%
##  4.2  5.2  5.6  6.5  7.5
```

```
quantile(beer$ABV, probs = seq(0, 1, length=11))
```

```
##   0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
## 4.20 4.86 5.16 5.30 5.42 5.60 6.20 6.40 6.50 6.87 7.50
```

The median is 5.6% ABV, which implies that half of the beers have at least 5.6% ABV.

3.3.5 Factor Level Information

The **levels** function extracts the levels (i.e., unique values) of a factor variable

```
levels(beer$Style)
```

```
## [1] "Ale"   "IPA"   "Lager"
```

and the **nlevels** function returns the number of levels of a factor

```
nlevels(beer$Style)
```

```
## [1] 3
```

The 44 beers are classified into one of three Styles: Ale, IPA, or Lager.

3.3.6 Covariances and Correlations

The **cov** function calculates the covariance $c = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$ between two variables

```
cov(beer$ABV, beer$IBU)
```

```
## [1] 13.22664
```

or a covariance matrix between the columns of an input data frame

```
cov(beer[, c("ABV","IBU","Rating")])
```

```
##               ABV       IBU    Rating
## ABV     0.6684989  13.22664  1.510571
## IBU    13.2266385 461.18129 30.871036
## Rating  1.5105708  30.87104 13.919662
```

The **cor** function calculates the correlation $r = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\{\sum_{i=1}^{n}(x_i-\bar{x})^2\}^{1/2}\{\sum_{i=1}^{n}(y_i-\bar{y})^2\}^{1/2}}$ between two variables

```
cor(beer$ABV, beer$IBU)
```

```
## [1] 0.7532919
```

or a correlation matrix between the columns of an input data frame

```
cor(beer[, c("ABV","IBU","Rating")])
```

```
##              ABV       IBU    Rating
## ABV    1.0000000 0.7532919 0.4951952
## IBU    0.7532919 1.0000000 0.3853018
## Rating 0.4951952 0.3853018 1.0000000
```

3.3.7 Applying Functions to Multiple Variables

To apply a function to several columns, we can use the **apply** function

```
apply(beer[, c("ABV","IBU","Rating")], 2, range)
```

```
##      ABV IBU Rating
## [1,] 4.2  15     79
## [2,] 7.5  99     98
```

```
apply(beer[, c("ABV","IBU","Rating")], 2, mean)
```

```
##       ABV       IBU    Rating
##  5.818182 51.068182 87.181818
```

3.3.8 Applying Functions at Levels of Factors

Use the **tapply** (ragged apply) function to apply some function to a numeric variable separately at each level of a factor variable. For example, we could apply the **range** function to the *ABV* variable separately for each *Style* of beer

```
tapply(beer$ABV, beer$Style, range)
```

```
## $Ale
## [1] 4.2 7.0
##
## $IPA
## [1] 5.8 7.5
##
## $Lager
## [1] 4.5 5.4
```

In the given sample, Ales range from 4.2% to 7% ABV, India Pale Ales range from 5.8% to 7.5% ABV, and Lagers range from 4.5% to 5.4% ABV.

3.3.9 Making a Table

Use the **cbind** (column combine) function in combination with the **tapply** function to create tables

```
tab1 <- cbind(tapply(beer$Rating, beer$Style, length),
              tapply(beer$ABV, beer$Style, mean),
              tapply(beer$ABV, beer$Style, sd),
              tapply(beer$IBU, beer$Style, mean),
              tapply(beer$IBU, beer$Style, sd),
              tapply(beer$Rating, beer$Style, mean),
              tapply(beer$Rating, beer$Style, sd))
colnames(tab1) <- c("n", "ABV.Mean", "ABV.SD", "IBU.Mean",
                    "IBU.SD", "Rating.Mean", "Rating.SD")
rtab1 <- round(tab1, 2)
rtab1
```

```
##        n ABV.Mean ABV.SD IBU.Mean IBU.SD Rating.Mean Rating.SD
## Ale   18     5.49   0.67    39.00  12.28       86.83      3.50
## IPA   17     6.56   0.46    73.53  10.59       88.18      4.54
## Lager  9     5.06   0.35    32.78  12.58       86.00      1.87
```

The **write.csv** function can be used to save the table

```
write.csv(rtab1, file = "~/Desktop/table1.csv", row.names = TRUE)
```

After some minor stylistic edits, the table is ready for publication—without having to manually type or copy-paste numbers!

The **kable** function (in the *knitr* package) includes nicely formatted tables in R Markdown documents

Table 3.1: Table 1: Sample size (n) and variable means and standard deviations (SD) for each style of beer.

	n	ABV.Mean	ABV.SD	IBU.Mean	IBU.SD	Rating.Mean	Rating.SD
Ale	18	5.49	0.67	39.00	12.28	86.83	3.50
IPA	17	6.56	0.46	73.53	10.59	88.18	4.54
Lager	9	5.06	0.35	32.78	12.58	86.00	1.87

```
library(knitr)
kable(rtab1, caption = "Table 1: Sample size (n) and variable means and standard deviations (SD) for ea
```

3.4 Student's t-Test in R

3.4.1 One Sample t-Test

Mass produced beers (e.g., Bud Light, Miller Lite, etc.) have 4.2% ABV. Suppose we want to test if Minnesota beers have the same mean ABV as mass produced beers

$$H_0 : \mu = 4.2 \quad \text{vs.} \quad H_1 : \mu \neq 4.2$$

where μ is the mean ABV, and H_0 and H_1 denote the null and alternative hypotheses. Assuming that the ABV scores are normally distributed, the **t.test** function can be used to test the null hypothesis

```
t.test(beer$ABV, mu = 4.2)
```

```
##
##  One Sample t-test
##
## data:  beer$ABV
## t = 13.128, df = 43, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 4.2
## 95 percent confidence interval:
##  5.569603 6.066760
## sample estimates:
## mean of x
##  5.818182
```

The observed t statistic is $t = 13.13$ with 43 degrees of freedom, resulting in a p-value of essentially zero—we reject H_0 using any standard α level. The sample mean is $\bar{x} = 5.8\%$ ABV and the 95% confidence interval for the μ (population mean ABV of Minnesota beers) is 5.6% to 6.1% ABV.

If we expect that the Minnesota beers have higher ABV than mass produced beers, i.e.,

$$H_0 : \mu = 4.2 \quad \text{vs.} \quad H_1 : \mu > 4.2$$

we need to adjust the *alternative* input

```
t.test(beer$ABV, mu = 4.2, alternative = "greater")
```

```
##
##  One Sample t-test
##
## data:  beer$ABV
## t = 13.128, df = 43, p-value < 2.2e-16
## alternative hypothesis: true mean is greater than 4.2
```

```
## 95 percent confidence interval:
##  5.610972      Inf
## sample estimates:
## mean of x
##  5.818182
```

The only noteworthy difference is that the confidence interval is now a 95% lower bound for the mean ABV of Minnesota beers, which we expect to be at least 5.6% ABV. Note that changing the alternative also changes the p-value, but for this example we do not notice (because the p-value is so small).

3.4.2 Two Sample t-Test

Suppose that we want to test if IPA beers have higher ABV than non-IPAs (Ales and Lagers)

$$H_0 : \mu_1 = \mu_2 \quad \text{vs.} \quad H_1 : \mu_1 > \mu_2$$

where μ_1 and μ_2 denote the mean ABV of IPA and non-IPA beers, respectively.

To use the **t.test** function for a two sample t-test, we need to input two vectors

```
beer$IPA <- (beer$Style == "IPA")
t.test(beer$ABV[beer$IPA], beer$ABV[!beer$IPA], alternative = "greater")
```

```
##
##  Welch Two Sample t-test
##
## data:  beer$ABV[beer$IPA] and beer$ABV[!beer$IPA]
## t = 7.4454, df = 40.527, p-value = 2.093e-09
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  0.9415048       Inf
## sample estimates:
## mean of x mean of y
##  6.564706  5.348148
```

The observed t statistic is $t = 7.45$ with 40.53 degrees of freedom, resulting in a p-value of essentially zero—we reject H_0 using any standard α level. The sample mean difference is $\bar{x}_1 - \bar{x}_2 = 1.22\%$ ABV and the 95% lower-bound confidence interval reveals that we expect IPAs to have at least 0.94% more ABV than non-IPAs.

The default uses the Welch version, which does not assume equal variance for the two groups. The *var.equal* input can be used to change this assumption, which produces the classic two sample t-test

```
t.test(beer$ABV[beer$IPA], beer$ABV[!beer$IPA], alternative = "greater",
       var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  beer$ABV[beer$IPA] and beer$ABV[!beer$IPA]
## t = 6.9809, df = 42, p-value = 7.74e-09
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  0.9234437       Inf
## sample estimates:
## mean of x mean of y
##  6.564706  5.348148
```

Note that the observed t-test statistic, p-value, and 95% lower-bound are slightly different, but our conclusion does not change: we expect IPAs to have at least 0.9% more ABV than non-IPAs.

3.5 One-Way ANOVA in R

3.5.1 Omnibus F-Test

Extending the previous example, suppose that we want to test if the mean ABV differs for the three Styles of beer

$$H_0 : \mu_j = \mu \text{ for all } j \quad \text{vs.} \quad H_1 : \mu_j \neq \mu \text{ for some } j$$

where μ_j denotes the mean ABV of the three Styles of beer: Ales, IPAs, and Lagers. Assuming that the ABV scores a normally distributed, we can use the **aov** (analysis of variance) function

```
amod <- aov(ABV ~ Style, data = beer)
summary(amod)
```

```
##              Df Sum Sq Mean Sq F value   Pr(>F)
## Style         2  16.59   8.297      28 2.16e-08 ***
## Residuals    41  12.15   0.296
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The observed F statistic is $F = 28$ with 2 numerator and 41 denominator degrees of freedom, resulting in a p-value of essentially zero—we reject H_0 using any standard α level. We conclude that the mean ABV of Minnesota beers depends on the Style of beer, but the results do not directly reveal which Styles significantly differ from one another.

3.5.2 Pairwise Comparisons (Tukey's HSD)

To determine which Styles significantly differ in their mean ABV, we can use Tukey's Honest Significant Differences (HSD) procedure via the **TukeyHSD** function

```
TukeyHSD(amod)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = ABV ~ Style, data = beer)
##
## $Style
##                 diff        lwr        upr     p adj
## IPA-Ale    1.0702614  0.6225693  1.5179536 0.0000024
## Lager-Ale -0.4388889 -0.9793078  0.1015301 0.1313124
## Lager-IPA -1.5091503 -2.0548418 -0.9634589 0.0000001
```

The pairwise comparisons reveal that

- IPAs have significantly higher mean ABV than Ales. The estimated mean difference is $\hat{\delta}_1 = \hat{\mu}_2 - \hat{\mu}_1 = 1.07$ with 95% confidence interval $\delta_1 \in [0.62, 1.52]$.
- Lagers and Ales do not significantly differ in mean ABV. The estimated mean difference is $\hat{\delta}_2 = \hat{\mu}_3 - \hat{\mu}_1 = -0.44$ with 95% confidence interval $\delta_2 \in [-0.98, 0.10]$.
- Lagers have significantly lower mean ABV than IPAs. The estimated mean difference is $\hat{\delta}_3 = \hat{\mu}_3 - \hat{\mu}_2 = -1.51$ with 95% confidence interval $\delta_3 \in [-2.05, -0.96]$.

3.6 Correlation Tests in R

Suppose that we want to test if a beer's ABV and Rating are postively correlated

$$H_0 : \rho = 0 \quad \text{vs.} \quad H_1 : \rho > 0$$

where ρ is the population correlation between the ABV and Rating. Assuming that the ABV and Rating variables follow a bivariate normal distribution, we can use the **cor.test** function

```
cor.test(beer$ABV, beer$Rating, alternative = "greater")
```

```
##
##  Pearson's product-moment correlation
##
## data:  beer$ABV and beer$Rating
## t = 3.6939, df = 42, p-value = 0.000316
## alternative hypothesis: true correlation is greater than 0
## 95 percent confidence interval:
##  0.2784833 1.0000000
## sample estimates:
##       cor
## 0.4951952
```

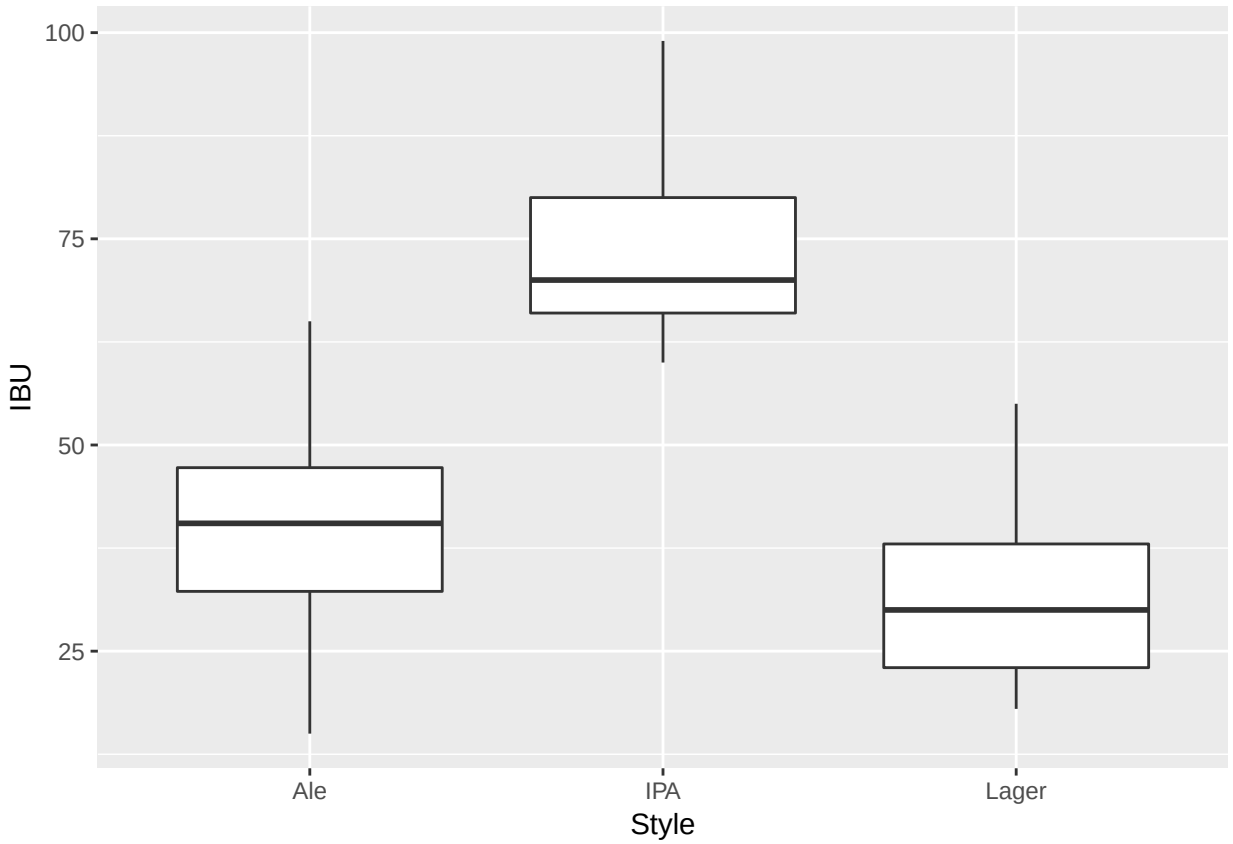
The estimated sample correlation is $r = 0.495$ and the 95% lower-bound confidence interval reveals that we expect the ABV and Ratings to have a positive correlation of at least $\rho = 0.28$.

3.7 Exercises

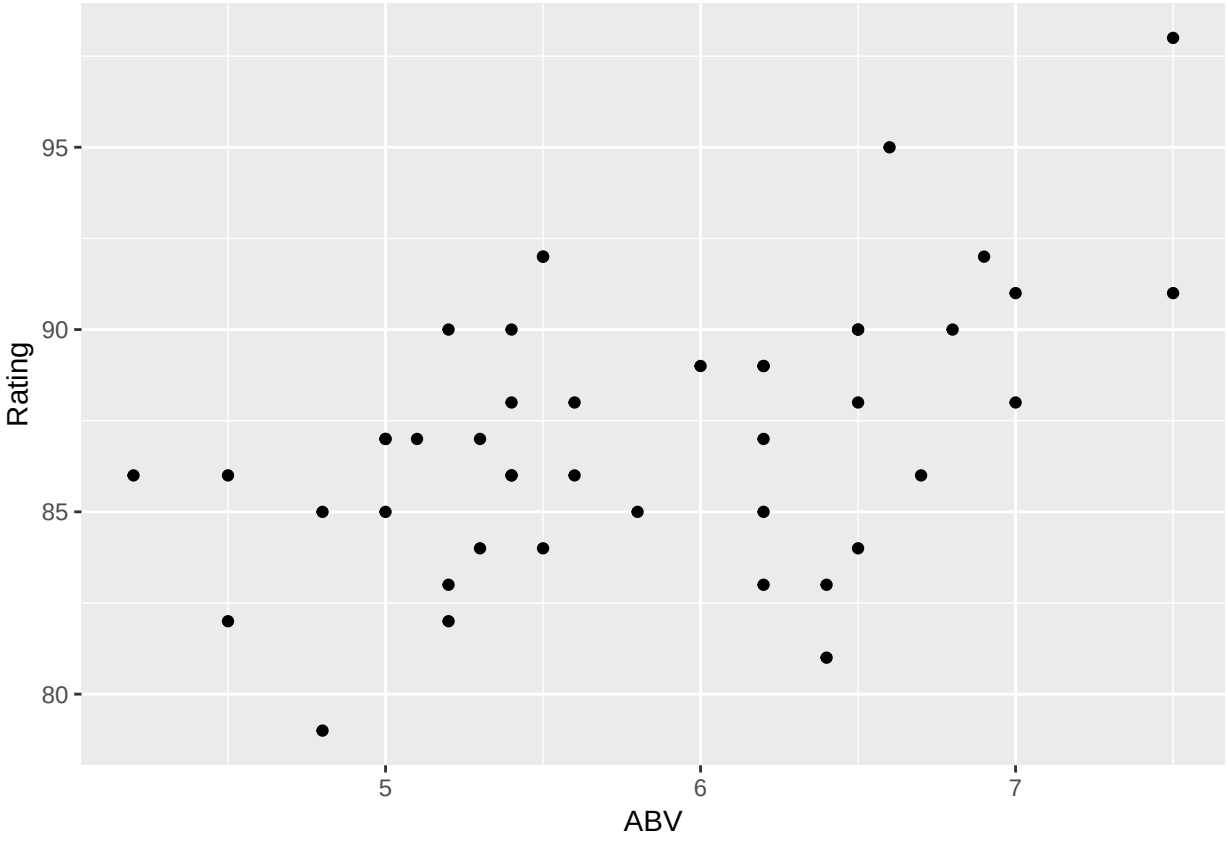
1) Load the Minnesota Beer Data into R.
2) Make a boxplot of the IBUs by Style of beer.
3) Make a scatterplot of the ABV (x-axis) by Rating (y-axis).
4) Calculate some descriptive statistics for the IBU variable.
5) Create a table showing the sample size and variable means and standard deviations for each Brewery.
6) Repeat the t-tests using the IBU variable as the response.
7) Repeat the one-way ANOVA using the IBU variable as the response.
8) Repeat the correlation test using the IBU and Rating variables.

SOLUTIONS

```
#1 Load the Minnesota Beer Data into R.
beer <- read.csv("http://users.stat.umn.edu/~helwig/notes/MNbeer.csv")

#2) Make a boxplot of the IBUs by Style of beer.
# Notice that IPAs tend to have higher IBU (bitterness)
ggplot(beer, aes(x = Style, y = IBU)) +
  geom_boxplot()
```




```
#3) Make a scatterplot of the ABV (x-axis) by Rating (y-axis).
# Notice that Rating tends to increase with ABV (alcohol level)
ggplot(beer, aes(x = ABV, y = Rating)) +
  geom_point()
```




```
#4) Calculate some descriptive statistics for the IBU variable.
summary(beer$IBU)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   15.00   33.00   48.50   51.07   68.25   99.00
```

```
#5) Create a table showing the sample size and variable means and standard deviations for each Brewery.
tab1 <- cbind(tapply(beer$Rating, beer$Brewery, length),
              tapply(beer$Rating, beer$Brewery, mean),
              tapply(beer$Rating, beer$Brewery, sd))
colnames(tab1) <- c("n", "Rating.Mean", "Rating.SD")
rtab1 <- round(tab1, 2)
rtab1
```

```
##                n Rating.Mean Rating.SD
## Bauhaus        4       86.75      0.96
## Bent Paddle    5       87.60      1.67
## Fulton         5       84.40      5.03
## Indeed         7       87.86      2.12
## Steel Toe      4       88.00      2.45
## Summit         7       85.43      3.10
## Surly          7       92.14      3.53
## Urban Growler 5       83.80      1.48
```

```
#6) Repeat the *t*-tests using the IBU variable as the response.
t.test(beer$IBU)
```

```
##
##  One Sample t-test
```

```
##
## data:  beer$IBU
## t = 15.774, df = 43, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  44.53914 57.59722
## sample estimates:
## mean of x
##  51.06818
```

```
#7) Repeat the one-way ANOVA using the IBU variable as the response.
# The p-value here indicates that IBU significantly differs by Style
amod <- aov(IBU ~ Style, data = beer)
summary(amod)
```

```
##              Df Sum Sq Mean Sq F value   Pr(>F)
## Style         2  14209    7105   51.81 5.98e-12 ***
## Residuals    41   5622     137
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#8) Repeat the correlation test using the IBU and Rating variables.
# The p-value here indicates that there's a significant, positive
# correlation between beers' IBU & Rating.
cor.test(beer$IBU, beer$Rating, alternative = "greater")
```

```
##
##  Pearson's product-moment correlation
##
## data:  beer$IBU and beer$Rating
## t = 2.706, df = 42, p-value = 0.0049
## alternative hypothesis: true correlation is greater than 0
## 95 percent confidence interval:
##  0.1482865 1.0000000
## sample estimates:
##       cor
## 0.3853018
```

Chapter 4

Linear Regression in R

Author: Nathaniel E. Helwig

4.1 Chapter Outline and Goals

In this chapter, we will cover how to...

- Fit simple and multiple linear regression models
- Test the significance of regression coefficients
- Plot and interpret the regression results
- Make predictions from fit regression models

R's **lm** (linear model) function will be the primary tool used in the chapter.

4.2 Minnesota Beer Data (Reminder)

4.2.1 Overview

The Minnesota beer data has 44 beers measured on 7 variables:

1) *Brewery*: Name of the brewery (**factor** with 8 levels)
2) *Beer*: Name of the beer (**factor** with 44 levels)
3) *Description*: Description of the beer (**factor** with 37 levels)
4) *Style*: Style of the beer (**factor** with 3 levels)
5) *ABV*: Alcohol by volume (**numeric**)
6) *IBU*: International bitterness units (**integer**)
7) *Rating*: Beer Advocate rating (**integer**)

Data obtained by NEH from Beer Advocate and the websites of the eight breweries.

4.2.2 Load and Look at the Data

Use the **read.csv** function to load the *beer* data into R

```
beer <- read.csv("http://users.stat.umn.edu/~helwig/notes/MNbeer.csv")
```

The **head** function returns the first six lines of a data frame

```
head(beer)
```

```
##       Brewery         Beer              Description Style ABV IBU Rating
## 1     Bauhaus  Wonderstuff     New Bohemian Pilsner Lager 5.4  48     88
## 2     Bauhaus    Stargazer German Style Schwarzbier Lager 5.0  28     87
## 3     Bauhaus  Wagon Party    West Cost Style Lager Lager 5.4  55     86
## 4     Bauhaus    Sky-Five!        Midwest Coast IPA   IPA 6.7  70     86
## 5 Bent Paddle         Kanu         Session Pale Ale   Ale 4.8  48     85
## 6 Bent Paddle Venture Pils            Pilsner Lager Lager 5.0  38     87
```

4.3 Simple Linear Regression

4.3.1 Fit the Model

Consider a simple linear regression model of the form

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where y_i is the Rating of the i-th beer (response), x_i is the ABV of the i-th beer (predictor), β_0 is the unknown regression intercept, β_1 is the unknown regression slope, and $\epsilon_i \sim \mathrm{N}(0, \sigma^2)$ is a latent Gaussian error term. To fit the model, we can use the **lm** function

```
mod <- lm(Rating ~ ABV, data = beer)
```

The first input is the regression formula (Response ~ Predictor), and the second input is the data frame containing the variables in the regression formula. Note that *mod* is an object of class *lm*, which is a list containing information about the fit model.

```
class(mod)
```

```
## [1] "lm"
```

```
names(mod)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

For example, the *$coefficients* element contains the estimated regression coefficients

```
mod$coefficients
```

```
## (Intercept)         ABV
##   74.034788    2.259646
```

which reveal that the expected Rating increases by about 2.26 points for every 1 unit (i.e., 1%) increaese in ABV.

4.3.2 Inference Information

To obtain a more detailed summary of the fit model, use the **summary** function

```
modsum <- summary(mod)
names(modsum)
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

```
modsum
```

```
##
## Call:
## lm(formula = Rating ~ ABV, data = beer)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.4965 -2.1563  0.3593  1.6670  7.0179
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  74.0348     3.5933  20.604  < 2e-16 ***
## ABV           2.2596     0.6117   3.694 0.000632 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.28 on 42 degrees of freedom
## Multiple R-squared:  0.2452, Adjusted R-squared:  0.2272
## F-statistic: 13.65 on 1 and 42 DF,  p-value: 0.000632
```

Note that summarizing an *lm* object returns the estimated error standard deviation *sigma* ($\hat{\sigma} = 3.28$), the coefficient of determination *r.squared* ($R^2 = 0.2452$), and a *coefficient* inference table for testing $H_0 : \beta_j = 0$ versus $H_1 : \beta_j \neq 0$. The observed t statistic for testing the slope parameter is $t = 3.69$ with 42 degrees of freedom, resulting in a p-value less than 0.001—we reject H_0 using any standard α level.

Use the **confint** function to obtain confidence intervals for regression coefficients

```
confint(mod, "ABV")
```

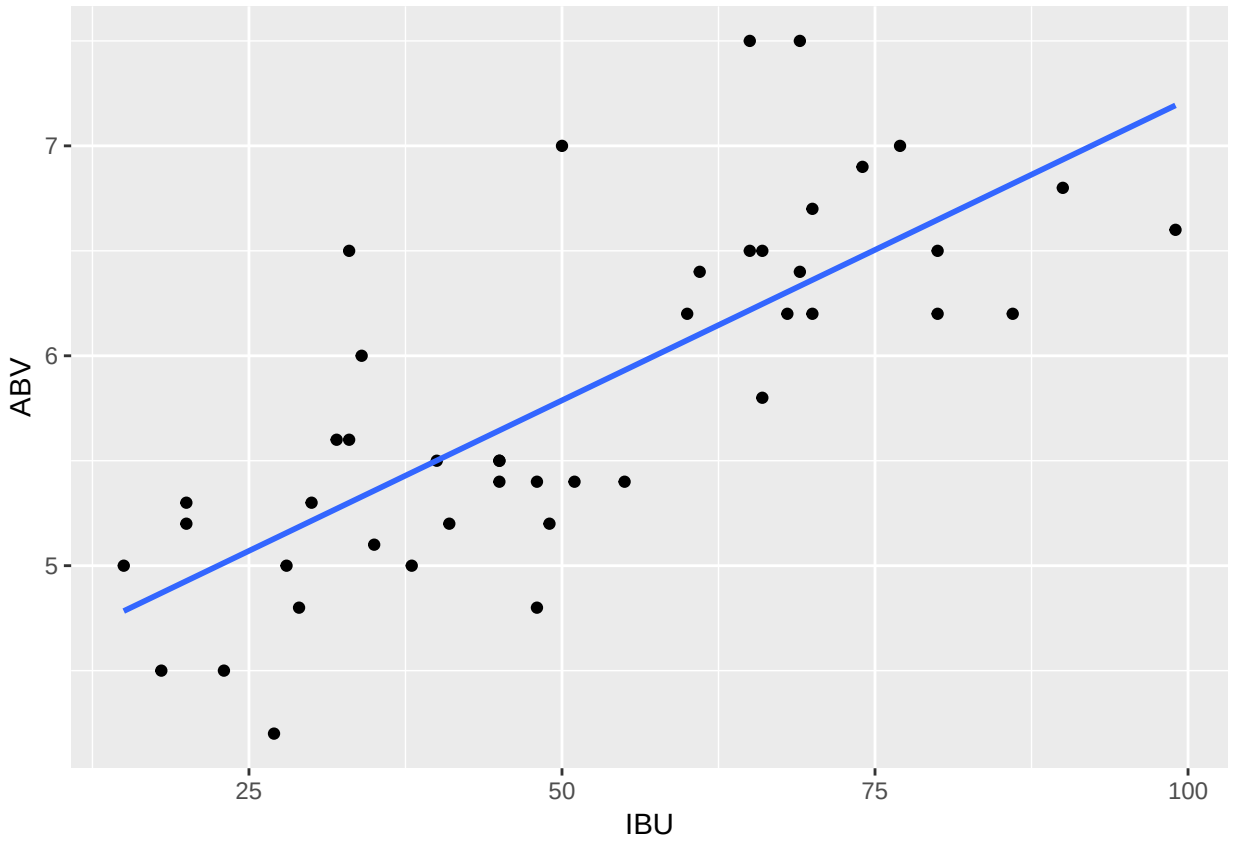
```
##        2.5 %   97.5 %
## ABV 1.025152 3.494139
```

The 95% confidence interval for β_1 reveals that we expect the average Rating to increase by 1.03 to 3.49 points for each additional 1% ABV.

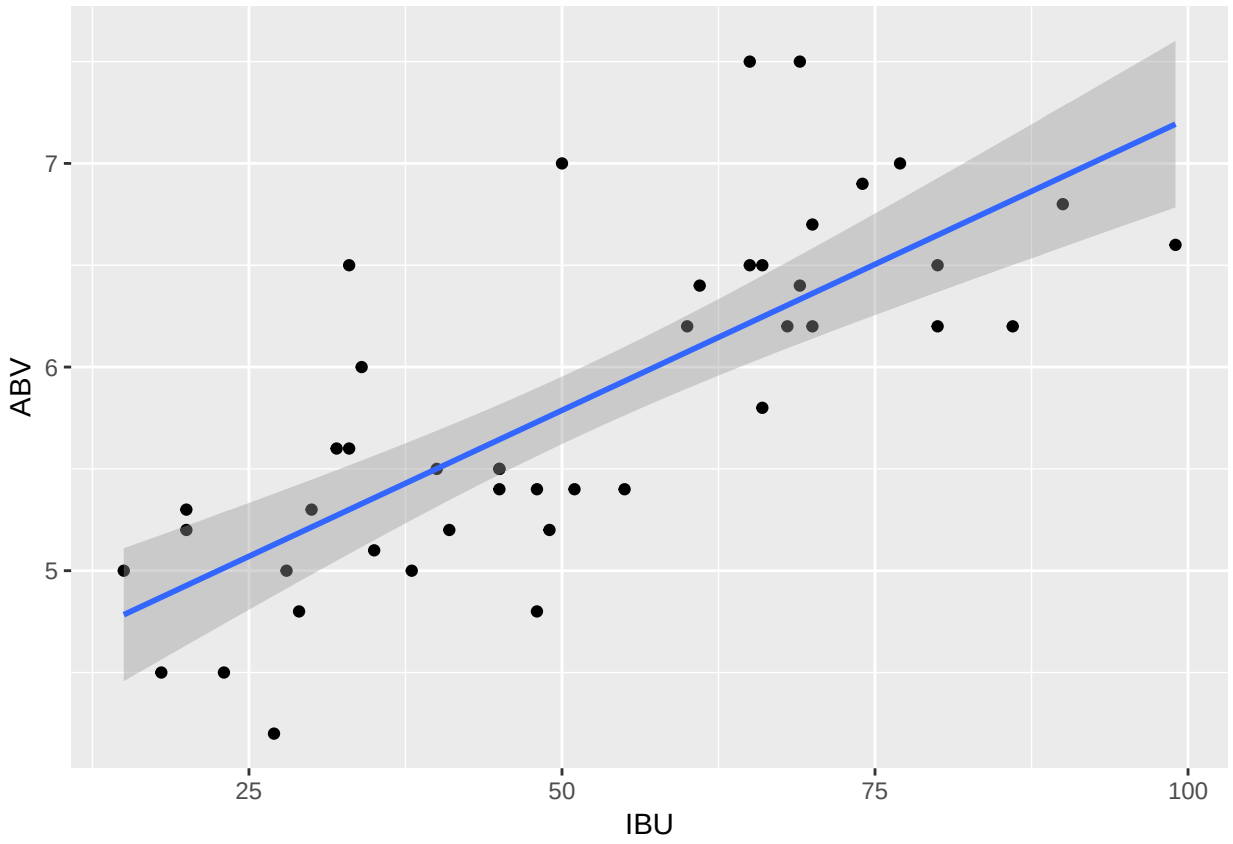
4.3.3 Plot the Regression Line

It's easy to include the least-squares regression line on a scatterplot by adding a `geom_smooth()` to the `ggplot()`:

```
# Without "standard error bars"
ggplot(beer, aes(x = IBU, y = ABV)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```




```
# With "standard error bars"
ggplot(beer, aes(x = IBU, y = ABV)) +
  geom_point() +
  geom_smooth(method = "lm", se = TRUE)
```



“‘

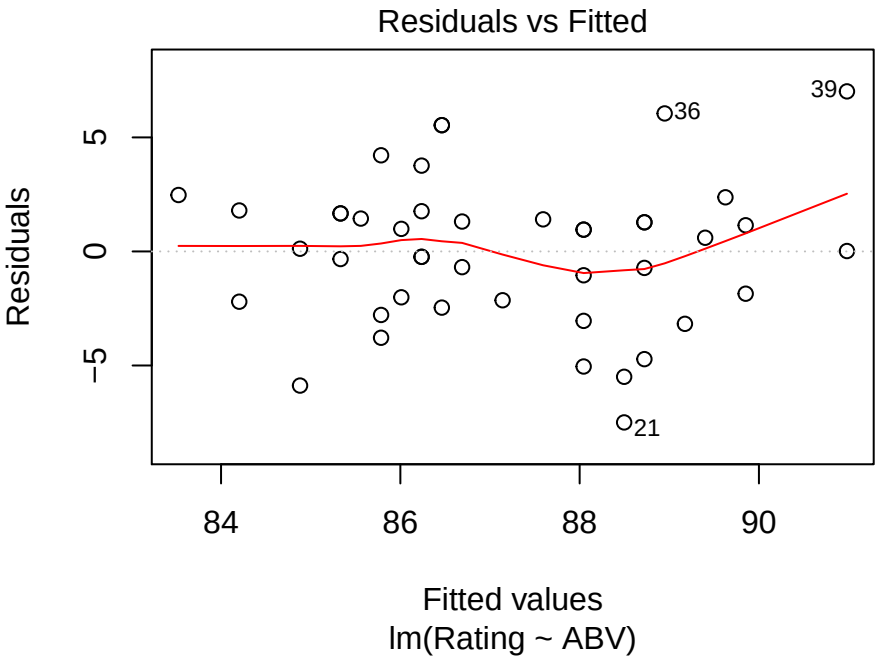
Alternatively, we can use the **abline()** function:

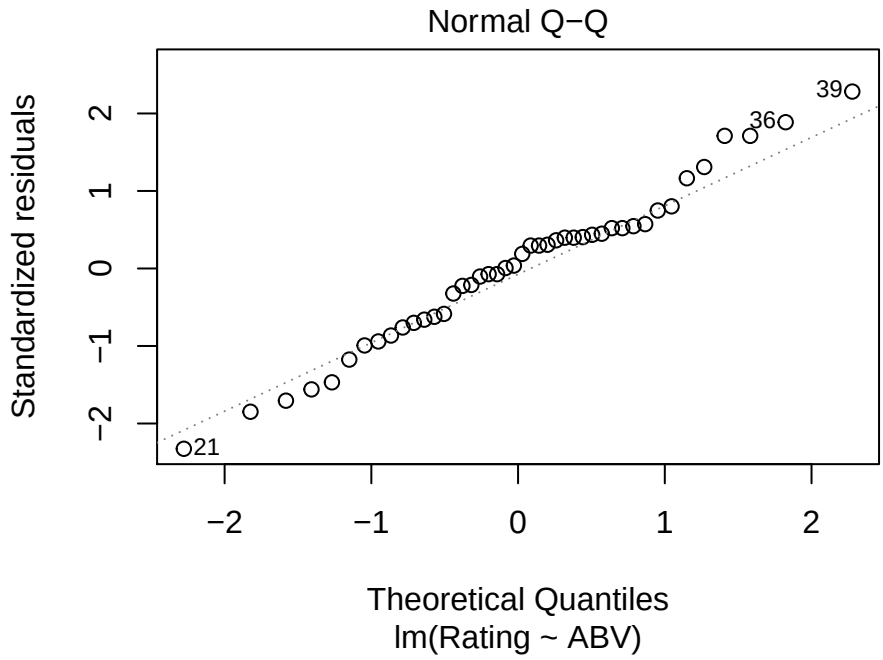
```
plot(beer$ABV, beer$Rating, xlab = "Alcohol By Volume",
     ylab = "Beer Advocate Rating", main = "Alcohol by Rating")
abline(mod)
```

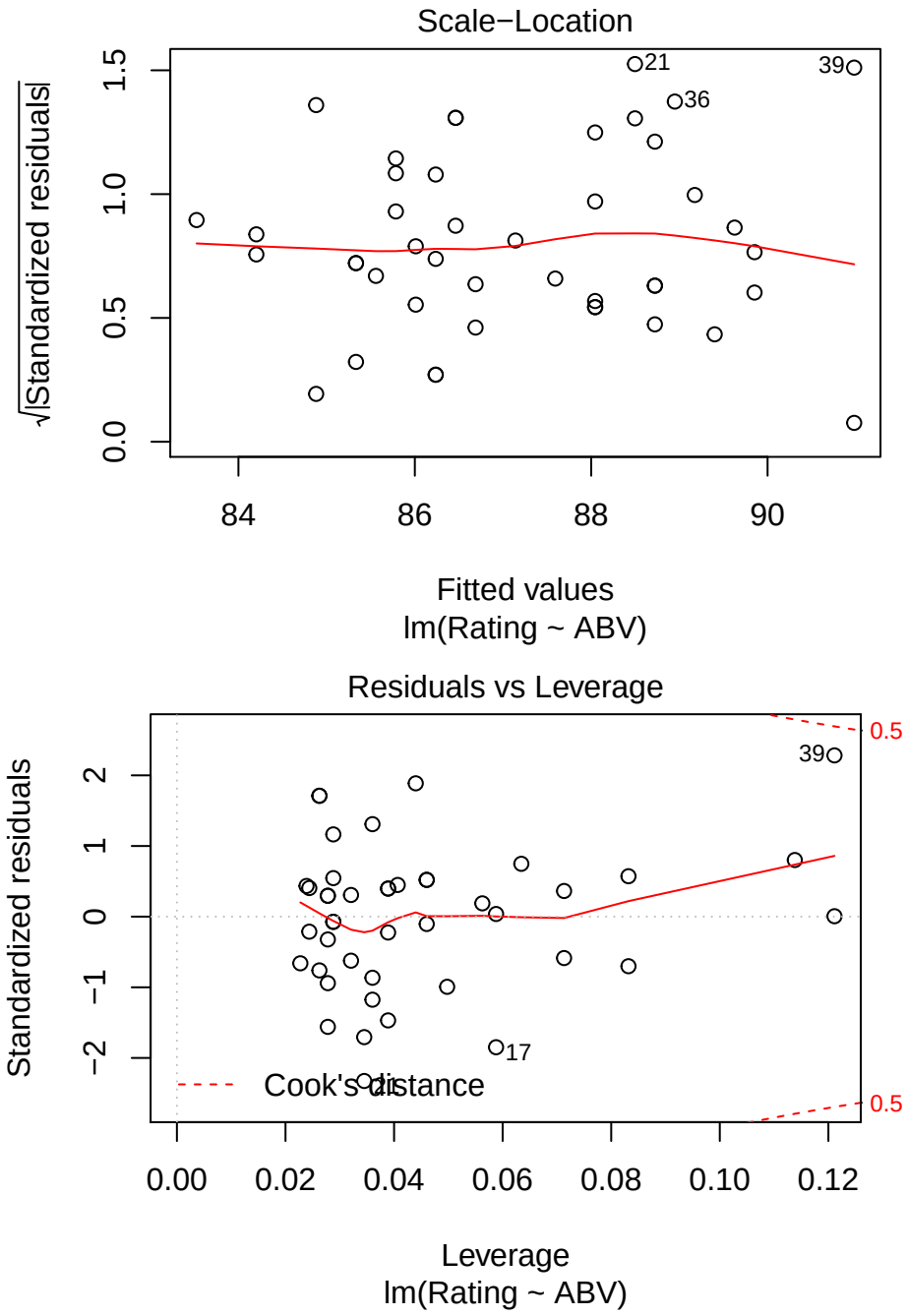
4.3.4 Diagnostic and Influence Plots

R makes it really easy to create simple diagnostic and influence plots for a fit regression model:

```
plot(mod)
```







4.3.5 Prediction for New Data

We often want to use a fit regression model to create predictions for new data. In R, this involves first creating the data frame of new predictor scores

```
newdata <- data.frame(ABV = seq(4.2, 7.5, by = 0.1))
```

which we input to the **predict** function along with the fit model

```
newfit <- predict(mod, newdata)
newfit
```

```
##        1        2        3        4        5        6        7        8
## 83.52530 83.75127 83.97723 84.20319 84.42916 84.65512 84.88109 85.10705
##        9       10       11       12       13       14       15       16
## 85.33302 85.55898 85.78495 86.01091 86.23688 86.46284 86.68880 86.91477
##       17       18       19       20       21       22       23       24
## 87.14073 87.36670 87.59266 87.81863 88.04459 88.27056 88.49652 88.72249
##       25       26       27       28       29       30       31       32
## 88.94845 89.17441 89.40038 89.62634 89.85231 90.07827 90.30424 90.53020
##       33       34
## 90.75617 90.98213
```

By default, the **predict** function returns a vector of predictions $\hat{y}_{i(\text{new})} = \hat{\beta}_0 + \hat{\beta}_1 x_{i(\text{new})}$. To obtain the corresponding standard errors of the predictions, we can use the *se.fit* input

```
newfitse <- predict(mod, newdata, se.fit = TRUE)
newfitse
```

```
## $fit
##        1        2        3        4        5        6        7        8
## 83.52530 83.75127 83.97723 84.20319 84.42916 84.65512 84.88109 85.10705
##        9       10       11       12       13       14       15       16
## 85.33302 85.55898 85.78495 86.01091 86.23688 86.46284 86.68880 86.91477
##       17       18       19       20       21       22       23       24
## 87.14073 87.36670 87.59266 87.81863 88.04459 88.27056 88.49652 88.72249
##       25       26       27       28       29       30       31       32
## 88.94845 89.17441 89.40038 89.62634 89.85231 90.07827 90.30424 90.53020
##       33       34
## 90.75617 90.98213
##
## $se.fit
##         1         2         3         4         5         6         7
## 1.1064827 1.0521132 0.9985311 0.9458702 0.8942934 0.8439992 0.7952311
##         8         9        10        11        12        13        14
## 0.7482877 0.7035342 0.6614152 0.6224659 0.5873171 0.5566893 0.5313647
##        15        16        17        18        19        20        21
## 0.5121308 0.4996912 0.4945590 0.4969606 0.5067890 0.5236260 0.5468247
##        22        23        24        25        26        27        28
## 0.5756164 0.6092087 0.6468540 0.6878872 0.7317387 0.7779320 0.8260744
##        29        30        31        32        33        34
## 0.8758444 0.9269800 0.9792672 1.0325312 1.0866283 1.1414400
##
## $df
## [1] 42
##
## $residual.scale
## [1] 3.279704
```

The *interval* input can be used to create confidence and prediction intervals

```
newfitCI <- predict(mod, newdata, interval = "confidence")
newfitPI <- predict(mod, newdata, interval = "prediction")
head(newfitCI)
```

```
##        fit      lwr      upr
## 1 83.52530 81.29233 85.75827
## 2 83.75127 81.62801 85.87452
## 3 83.97723 81.96211 85.99235
```

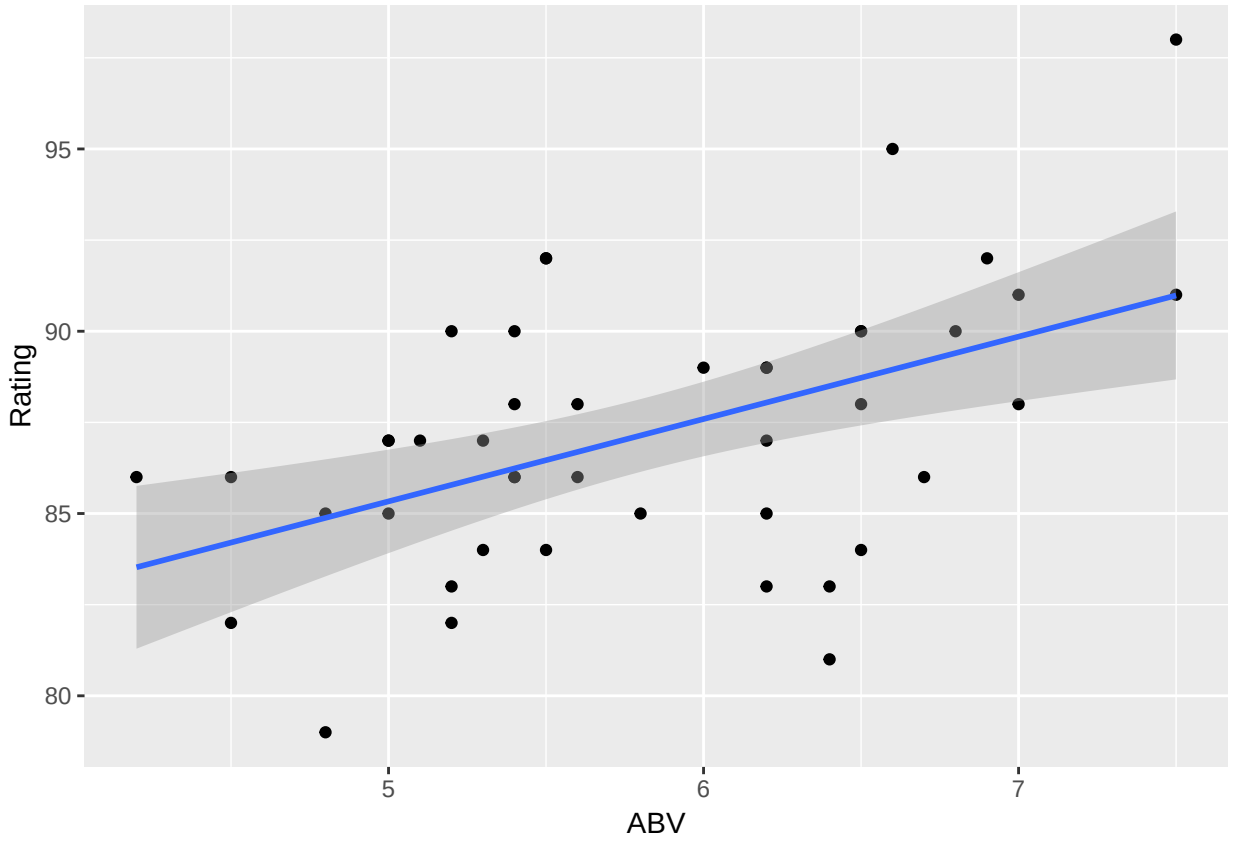
```
## 4 84.20319 82.29435 86.11204
## 5 84.42916 82.62440 86.23392
## 6 84.65512 82.95186 86.35838
```

```
head(newfitPI)
```

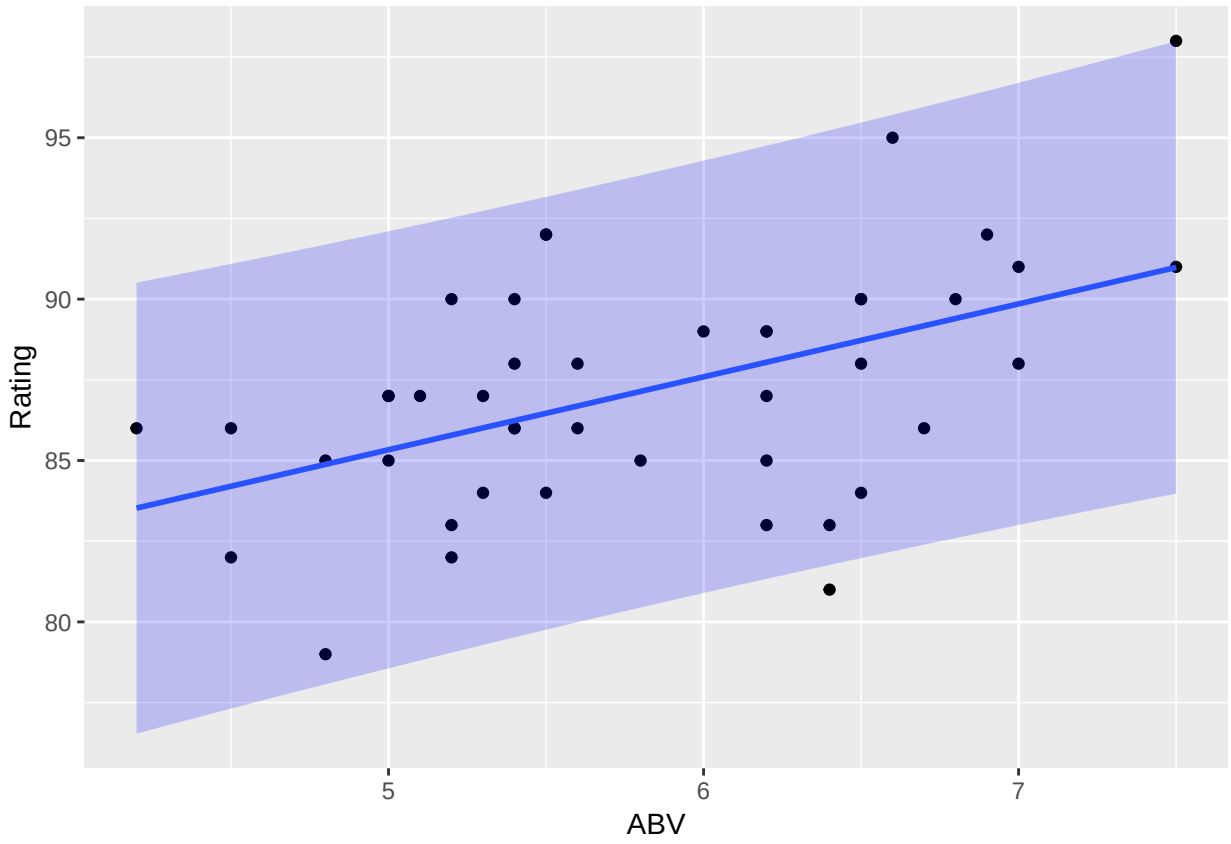
```
##        fit      lwr      upr
## 1 83.52530 76.54007 90.51053
## 2 83.75127 76.80033 90.70220
## 3 83.97723 77.05856 90.89590
## 4 84.20319 77.31472 91.09166
## 5 84.42916 77.56880 91.28951
## 6 84.65512 77.82077 91.48948
```

The confidence and prediction intervals can be plotted using:

```
# Confidence intervals
ggplot(beer, aes(x = ABV, y = Rating)) +
  geom_point() +
  geom_smooth(method = "lm", se = TRUE)
```

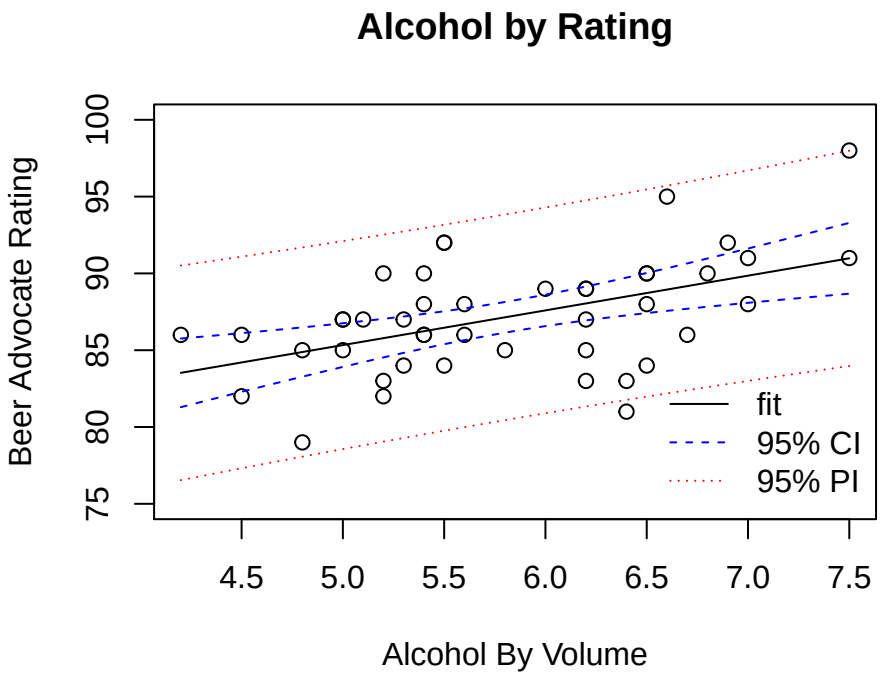



```
# Prediction intervals
newfitPI <- data.frame(newfitPI)
newfitPI$ABV <- newdata$ABV
ggplot(beer, aes(x = ABV, y = Rating)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  geom_ribbon(data = newfitPI, aes(x = ABV, y = fit, ymin = lwr, ymax = upr), fill = "blue", alpha = 0.
```



Or:

```
plot(beer$ABV, beer$Rating, xlab = "Alcohol By Volume",
     ylab = "Beer Advocate Rating", main = "Alcohol by Rating",
     ylim = c(75, 100))
lines(newdata$ABV, newfitCI[,1])
lines(newdata$ABV, newfitCI[,2], lty = 2, col = "blue")
lines(newdata$ABV, newfitCI[,3], lty = 2, col = "blue")
lines(newdata$ABV, newfitPI[,2], lty = 3, col = "red")
lines(newdata$ABV, newfitPI[,3], lty = 3, col = "red")
legend("bottomright", lty = 1:3, legend = c("fit", "95% CI", "95% PI"),
       col = c("black", "blue", "red"), bty = "n")
```



4.4 Multiple Linear Regression

4.4.1 Overview

A multiple linear regression model has the form

$$y_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} + \epsilon_i$$

where y_i is the response for the i-th observation, x_{ij} is the j-th predictor for the i-th observation, β_0 is the unknown regression intercept, β_j is the unknown regression slope for the j-th predictor, and $\epsilon_i \sim \mathrm{N}(0, \sigma^2)$ is a latent Gaussian error term. Note that β_j gives the expected change in the response variable for a 1-unit change in the j-th predictor variable conditioned on the other predictors, i.e., holding all other predictors constant.

4.4.2 Additive Effects

We will start by considering a model predicting the Rating from the additive effects of ABV and Brewery

```
amod <- lm(Rating ~ ABV + Brewery, data = beer)
```

Note that this model allows each Brewery to have a unique regression intercept (Bauhaus is the baseline), but assumes that the slope between ABV and Rating is the same for each Brewery. We can summarize the model using the same approach as before:

```
amodsum <- summary(amod)
amodsum
```

```
##
## Call:
## lm(formula = Rating ~ ABV + Brewery, data = beer)
```

```
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -4.702 -1.477  0.217  1.207  5.505
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)          77.82164    3.27842  23.738  < 2e-16 ***
## ABV                   1.58726    0.53419   2.971  0.00533 **
## BreweryBent Paddle    1.01666    1.75996   0.578  0.56719
## BreweryFulton        -2.27857    1.75923  -1.295  0.20373
## BreweryIndeed         0.87472    1.64545   0.532  0.59836
## BrewerySteel Toe      0.05955    1.89701   0.031  0.97513
## BrewerySummit        -1.16837    1.64440  -0.711  0.48209
## BrewerySurly          4.25343    1.68773   2.520  0.01644 *
## BreweryUrban Growler -3.22777    1.76155  -1.832  0.07542 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.622 on 35 degrees of freedom
## Multiple R-squared:  0.5979, Adjusted R-squared:  0.506
## F-statistic: 6.506 on 8 and 35 DF,  p-value: 3.502e-05
```

Compared to the simple linear regression model containing only the ABV predictor, we have noticeably reduced the residual standard deviation estimate *sigma* ($\hat{\sigma} = 2.622$) and increased the coefficient of (multiple) determination *r.squared* ($R^2 = 0.5979$).

The **anova** and **Anova** functions can be used to test the significance of terms

```
library(car)
```

```
## Loading required package: carData
```

```
anova(amod)  # Type I (sequential) SS test
```

```
## Analysis of Variance Table
##
## Response: Rating
##           Df Sum Sq Mean Sq F value    Pr(>F)
## ABV        1 146.77 146.774 21.3451 5.026e-05 ***
## Brewery    7 211.10  30.157  4.3857  0.001393 **
## Residuals 35 240.67   6.876
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Anova(amod)  # Type II SS test
```

```
## Anova Table (Type II tests)
##
## Response: Rating
##            Sum Sq Df F value   Pr(>F)
## ABV        60.709  1  8.8288 0.005331 **
## Brewery   211.102  7  4.3857 0.001393 **
## Residuals 240.669 35
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that **anova** tests the effects sequentially (ABV alone, then Brewery given ABV), whereas the **Anova**

function (in the *car* package) tests the effects conditioned on the other effect (ABV given Brewery, Brewery given ABV). Using the Type II tests from the **Anova** function, we see that both ABV ($F_{1,35} = 8.83, p = 0.005$) and Brewery ($F_{7,35} = 4.39, p = 0.001$) significantly add to the prediction of the beer's Rating.

4.4.3 Interaction Effects

Next we consider a model predicting the Rating from the interaction effects of ABV and Brewery

```
imod <- lm(Rating ~ ABV * Brewery, data = beer)
```

Note that formula notation is shorthand for *Rating ~ ABV + Brewery + ABV:Brewery*, so this model allows each Brewery to have a unique regression intercept and slope relating ABV and Rating. We can summarize the model using the same approach as before:

```
imodsum <- summary(imod)
imodsum
```

```
##
## Call:
## lm(formula = Rating ~ ABV * Brewery, data = beer)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.2905 -1.3040 -0.0471  1.4774  4.9434
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)                90.4203    12.4097   7.286 6.22e-08 ***
## ABV                        -0.6525     2.1920  -0.298    0.768
## BreweryBent Paddle        -17.0655    17.8235  -0.957    0.347
## BreweryFulton             -18.8852    14.6059  -1.293    0.207
## BreweryIndeed              -8.1702    14.4123  -0.567    0.575
## BrewerySteel Toe          -16.6529    16.6037  -1.003    0.324
## BrewerySummit             -11.2505    15.7834  -0.713    0.482
## BrewerySurly              -12.0484    14.6792  -0.821    0.419
## BreweryUrban Growler       -8.6554    19.7273  -0.439    0.664
## ABV:BreweryBent Paddle      3.2331     3.1819   1.016    0.318
## ABV:BreweryFulton           2.9580     2.5806   1.146    0.261
## ABV:BreweryIndeed           1.6240     2.5265   0.643    0.526
## ABV:BrewerySteel Toe        2.8851     2.7839   1.036    0.309
## ABV:BrewerySummit           1.7846     2.8071   0.636    0.530
## ABV:BrewerySurly            2.8236     2.5109   1.125    0.270
## ABV:BreweryUrban Growler    1.0034     3.4276   0.293    0.772
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.813 on 28 degrees of freedom
## Multiple R-squared:  0.6297, Adjusted R-squared:  0.4313
## F-statistic: 3.174 on 15 and 28 DF,  p-value: 0.004029
```

Compared to the additive model, we have slightly increased the residual standard deviation estimate *sigma* ($\hat{\sigma} = 2.813$) and increased the coefficient of (multiple) determination *r.squared* ($R^2 = 0.6297$).

Use the **Anova** function to test the signifiance of the effects

```
library(car)
Anova(imod)  # Type II SS test
```

```
## Anova Table (Type II tests)
##
## Response: Rating
##              Sum Sq Df F value   Pr(>F)
## ABV          60.709  1  7.6695 0.009854 **
## Brewery     211.102  7  3.8098 0.005009 **
## ABV:Brewery  19.030  7  0.3434 0.926653
## Residuals   221.639 28
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The results reveal that the interaction effect is not significant ($F_{7,28} = 0.34, p = 0.927$), but the main effects of ABV ($F_{1,28} = 7.67, p = 0.01$) and Brewery ($F_{7,28} = 3.81, p = 0.005$) are significant at the classic $\alpha = 0.05$ significance level.

4.4.4 Comparing Fit Models

To compare the fit models, we can use the **anova** function for F-tests

```
anova(mod, amod, imod)
```

```
## Analysis of Variance Table
##
## Model 1: Rating ~ ABV
## Model 2: Rating ~ ABV + Brewery
## Model 3: Rating ~ ABV * Brewery
##   Res.Df    RSS Df Sum of Sq      F   Pr(>F)
## 1     42 451.77
## 2     35 240.67  7    211.10 3.8098 0.005009 **
## 3     28 221.64  7     19.03 0.3434 0.926653
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

or the **AIC** function to extract Akaike's information criterion

```
AIC(mod, amod, imod)
```

```
##      df      AIC
## mod   3 233.3420
## amod 10 219.6329
## imod 17 230.0085
```

In this case, the F-tests and AIC values suggest that the additive model should be preferred. We conclude that each Brewery has a unique baseline Rating, and increasing the ABV by 1% corresponds to an expected 1.59 point increase in the Rating.

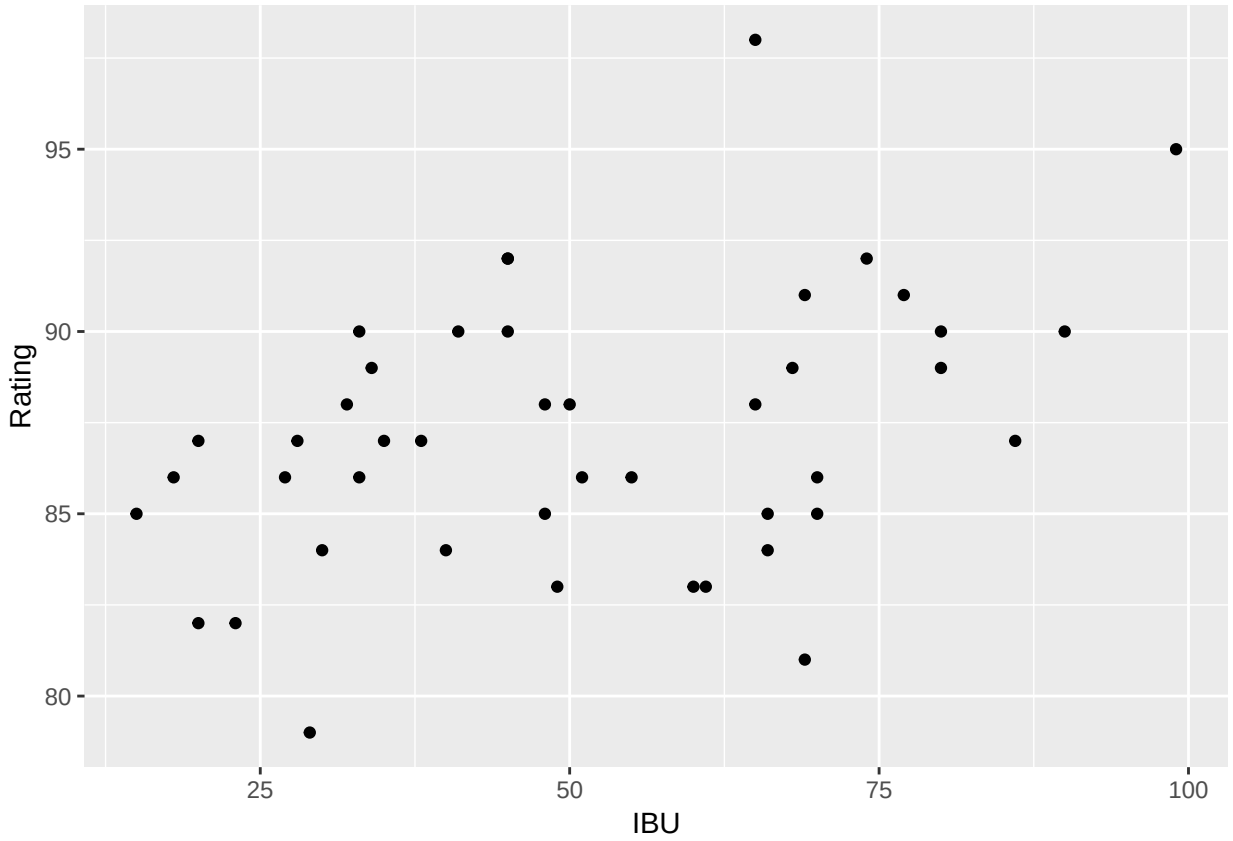
4.5 Exercises

1) Load the Minnesota Beer Data into R.
2) Make a scatterplot of the IBU (x-axis) by Rating (y-axis)
3) Fit a simple linear regression model predicting Rating from IBU.

4) Is there a significant linear relationship between IBU and Rating?
5) Plot the linear relationship, along with 95% confidence and prediction intervals.
6) Fit a multiple linear regression model predicting Rating from the additive effects of IBU and Brewery.
7) Fit a multiple linear regression model predicting Rating from the additive and interaction effects of IBU and Brewery.
8) Considering the models you fit in Ex 3, 7, 8, which do you prefer and why?

Solutions:

```
## 1: Load the Minnesota Beer Data into R
beer <- read.csv("http://users.stat.umn.edu/~helwig/notes/MNbeer.csv")

# 2: Make a scatterplot of the IBU (x-axis) by Rating (y-axis)
ggplot(beer, aes(x = IBU, y = Rating)) +
  geom_point()
```



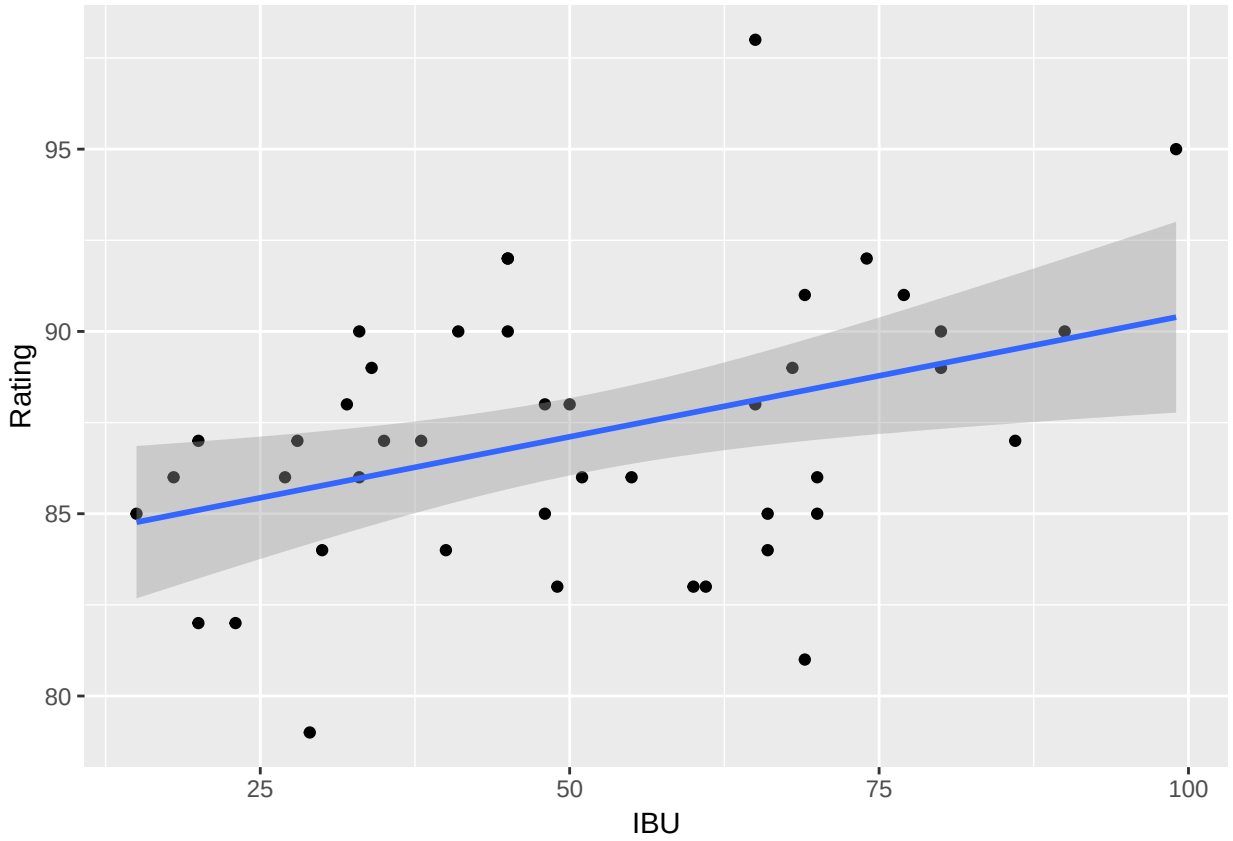

```
# 3: Fit a simple linear regression model predicting Rating from IBU
# For every 1 IBU increase, Rating tends to increase by 0.07 points.
my_model <- lm(Rating ~ IBU, data = beer)
coef(my_model)
```

```
## (Intercept)         IBU
## 83.76336277  0.06693905
```

```
#4: Is there a significant linear relationship between IBU and Rating?
# Answer: Yes!  Notice that the p-value in the IBU row
# below is 0.  Also, notice that the confidence bands below
# don't include the flat, 0 slope line.
summary(my_model)
```

```
##
## Call:
## lm(formula = Rating ~ IBU, data = beer)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.3822 -2.4669  0.3309  2.0854  9.8856
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 83.76336    1.36811  61.225   <2e-16 ***
## IBU          0.06694    0.02474   2.706   0.0098 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.484 on 42 degrees of freedom
## Multiple R-squared:  0.1485, Adjusted R-squared:  0.1282
## F-statistic: 7.322 on 1 and 42 DF,  p-value: 0.0098
```

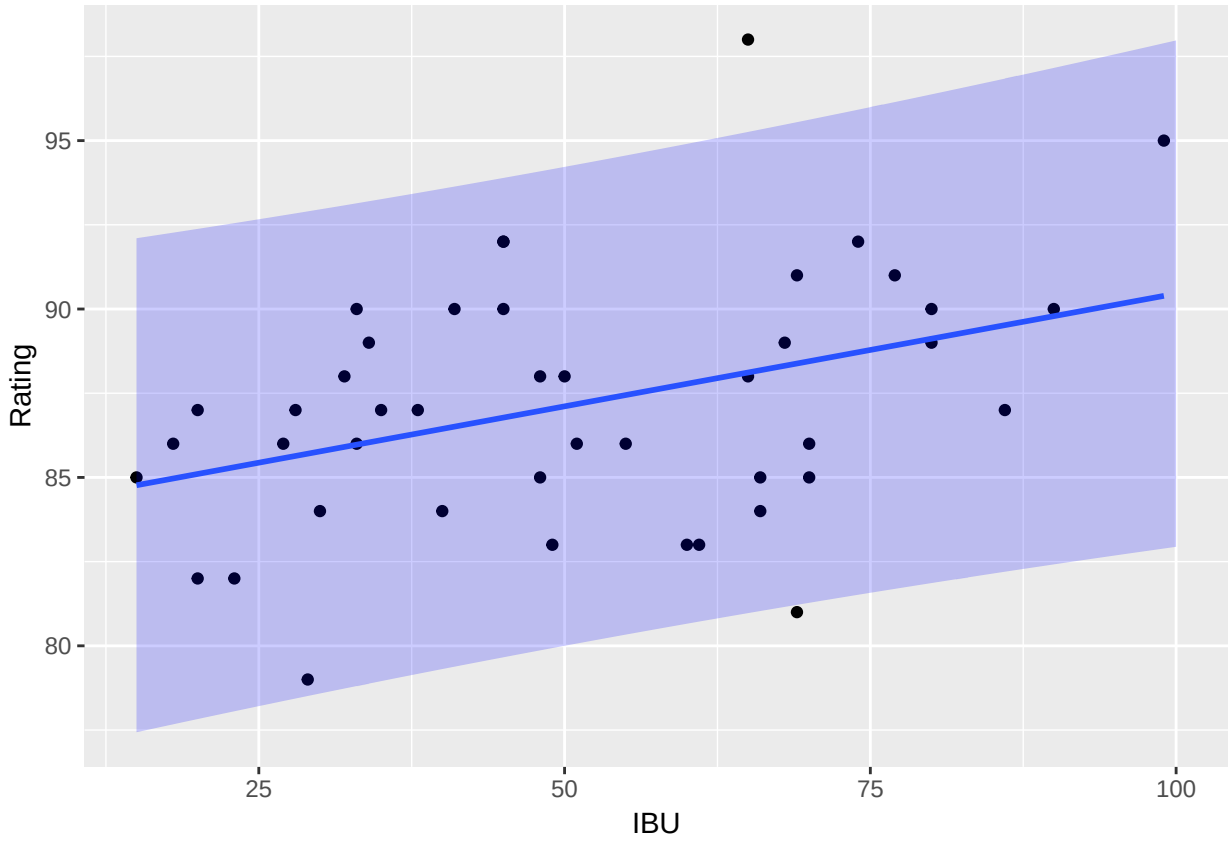
```
#5: Plot the linear relationship, along with 95% confidence and prediction intervals
# The confidence intervals reflect the plausible TREND in this relationship.
# The prediction intervals reflect the plausible DEVIATIONS from this trend.
ggplot(beer, aes(x = IBU, y = Rating)) +
  geom_point() +
  geom_smooth(method = "lm", se = TRUE)
```




```
# prediction bands (this code = yuck)
newdata  <- data.frame(IBU = seq(15, 100, by = 1))
```

```
newfit   <- predict(my_model, newdata)
newfitPI <- predict(my_model, newdata, interval = "prediction")
newfitPI <- data.frame(newfitPI)
newfitPI$IBU <- newdata$IBU
ggplot(beer, aes(x = IBU, y = Rating)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  geom_ribbon(data = newfitPI, aes(x = IBU, y = fit, ymin = lwr, ymax = upr), fill = "blue", alpha = 0.
```

```
## Warning: Ignoring unknown aesthetics: y
```




```
#6: Fit a multiple linear regression model predicting Rating from the additive effects of IBU and Brewe
model_2 <- lm(Rating ~ IBU + Brewery, data = beer)
summary(model_2)
```

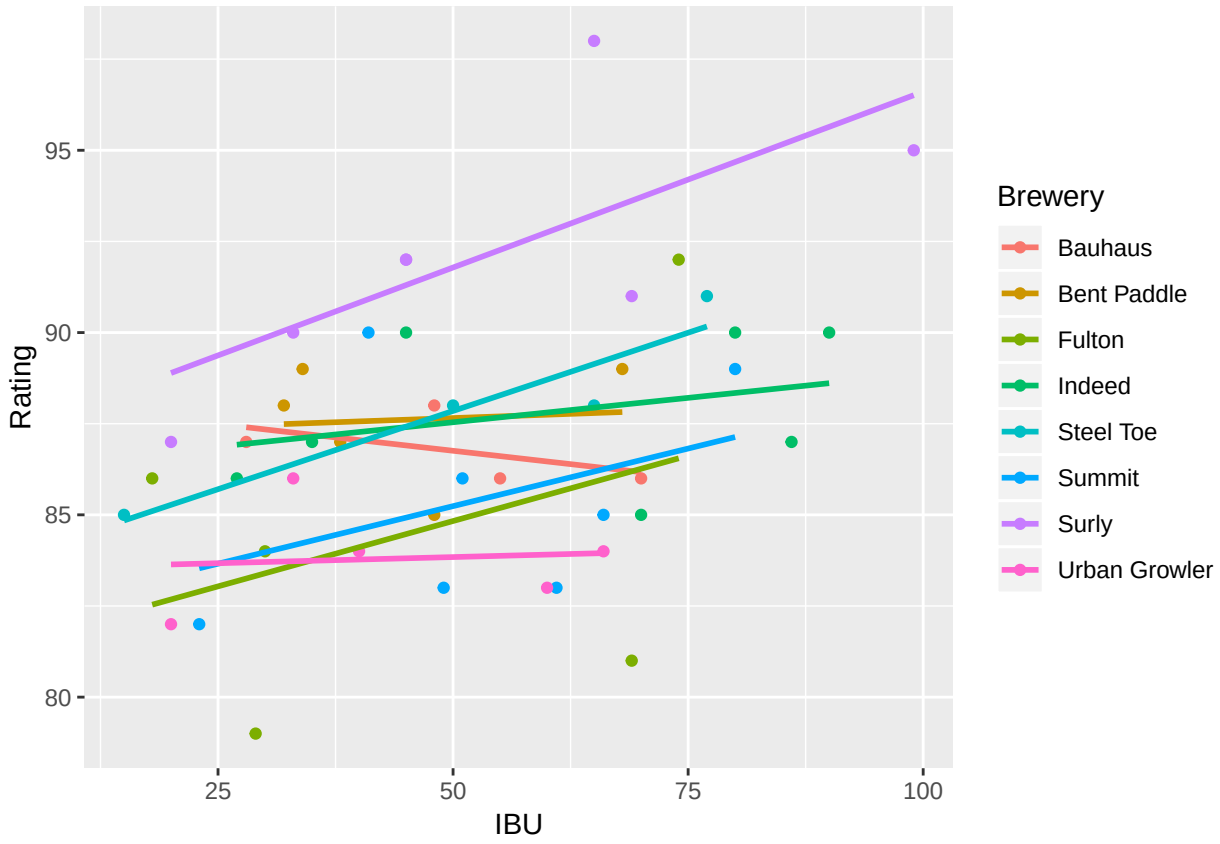
```
##
## Call:
## lm(formula = Rating ~ IBU + Brewery, data = beer)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7732 -1.7126  0.0889  1.2031  5.9521
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          83.98984    1.65525  50.741  < 2e-16 ***
## IBU                   0.05493    0.01969   2.790  0.00848 **
## BreweryBent Paddle    1.19330    1.78467   0.669  0.50811
```

```
## BreweryFulton        -2.00670    1.78467  -1.124  0.26849
## BreweryIndeed         0.46958    1.67917   0.280  0.78139
## BrewerySteel Toe      1.16761    1.87697   0.622  0.53793
## BrewerySummit        -1.47248    1.66443  -0.885  0.38237
## BrewerySurly          5.20257    1.66495   3.125  0.00357 **
## BreweryUrban Growler -2.59571    1.78495  -1.454  0.15479
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.654 on 35 degrees of freedom
## Multiple R-squared:  0.5881, Adjusted R-squared:  0.4939
## F-statistic: 6.246 on 8 and 35 DF,  p-value: 5.108e-05
```

```
#7: Fit a multiple linear regression model predicting Rating from the additive and interaction effects
model_3 <- lm(Rating ~ IBU * Brewery, data = beer)
summary(model_3)
```

```
##
## Call:
## lm(formula = Rating ~ IBU * Brewery, data = beer)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.1925 -1.5054 -0.0439  1.1809  5.4490
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)               88.22268    4.88650  18.054   <2e-16 ***
## IBU                       -0.02931    0.09313  -0.315    0.755
## BreweryBent Paddle        -1.02635    6.56017  -0.156    0.877
## BreweryFulton             -6.97752    5.59517  -1.247    0.223
## BreweryIndeed             -2.02179    5.70948  -0.354    0.726
## BrewerySteel Toe          -4.66505    5.96999  -0.781    0.441
## BrewerySummit             -6.14258    5.99866  -1.024    0.315
## BrewerySurly              -1.26059    5.52626  -0.228    0.821
## BreweryUrban Growler      -4.71814    5.99284  -0.787    0.438
## IBU:BreweryBent Paddle     0.03848    0.13323   0.289    0.775
## IBU:BreweryFulton          0.10101    0.10813   0.934    0.358
## IBU:BreweryIndeed          0.05608    0.10323   0.543    0.591
## IBU:BrewerySteel Toe       0.11515    0.11102   1.037    0.309
## IBU:BrewerySummit          0.09249    0.11216   0.825    0.417
## IBU:BrewerySurly           0.12576    0.10291   1.222    0.232
## IBU:BreweryUrban Growler   0.03605    0.11883   0.303    0.764
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.814 on 28 degrees of freedom
## Multiple R-squared:  0.6297, Adjusted R-squared:  0.4313
## F-statistic: 3.174 on 15 and 28 DF,  p-value: 0.00403
```

```
ggplot(beer, aes(x = IBU, y = Rating, color = Brewery)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```




```
#8 Considering the models you fit in Ex 3, 7, 8, which do you prefer and why?
# I prefer the middle model.  Including Brewery in the model appears
# to improve predictions.  The "interactive" effects seem to be overkill.
```

Chapter 5

Logistic Regression in R

Author: Christina Knudson

5.1 Introduction

The previous chapter covered linear regression, which models a Gaussian response variable. This chapter covers logistic regression, which is used to model a binary response variable. Here are some examples of binary responses:

- Whether or not a person wears a helmet while biking
- Whether or not a dog is adopted
- Whether or not a beer is given an award
- Whether or not a tree survives a storm

5.1.1 Goals

In this chapter, we will cover how to...

- Fit a logistic regression model in R with quantitative and/or categorical predictors.
- Interpret logistic regression model.
- Calculate probabilities.
- Test the significance of regression coefficients.

R's **glm** (generalized linear model) function will be the primary tool used in the chapter.

5.1.2 Horseshoe Crab Data

Throughout this module, we will refer to the horseshoe crab data. Some female crabs attract many males while others do not attract any. The males that cluster around a female are called "satellites." In order to understand what influences the presence of satellite crabs, researchers selected female crabs and collected data on the following characteristics:

- the color of her shell
- the condition of her spine
- the width of her carapace shell (in centimeters)
- the number of male satellites
- the weight of the female (in grams)

In today's example, we will use the width of a female's shell to predict the probability of her having one or more satellites. Let's start by loading the data.

```
crabs <- read.csv("http://www.cknudson.com/data/crabs.csv")
head(crabs)
```

```
##    color spine width satell weight y
## 1 medium   bad  28.3      8   3050 1
## 2   dark   bad  22.5      0   1550 0
## 3  light  good  26.0      9   2300 1
## 4   dark   bad  24.8      0   2100 0
## 5   dark   bad  26.0      4   2600 1
## 6 medium   bad  23.8      0   2100 0
```

You can learn more about this data set here.

5.2 Model Basics

5.2.1 Notation and Setup

Recall that a simple linear regression model has the following form:

$$\hat{y}_i = \beta_0 + \beta_1 x_i$$

where x_i is the predictor, β_0 is the unknown regression intercept, β_1 is the unknown regression slope, and $\hat{y}_i$ is the predicted response given x_i.

In order to model a binary response variable, we need to introduce p_i, the probability of something happening. For example, this might be the probability of a person wearing a helmet, the probability of a dog being adopted, or the probability of a beer winning an award. Then our logistic regression model has the following form:

$$\log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_i.$$

Recall that we estimated β_0 and β_1 to characterize the linear relationship between x_i and y_i in the simple linear regression setting. In the logistic regression setting, we will estimate β_0 and β_1 in order to understand the relationship between x_i and p_i.

As a final introductory note, we define the odds as $\dfrac{p_i}{1-p_i}$.

5.2.2 Fit a Logistic Regression Model

To fit a logistic regression model, we can use the **glm** function:

```
mod <- glm(y ~ width, family = binomial, data = crabs)
```

The first input is the regression formula (Response ~ Predictor), the second input indicates we have a binary response, and the third input is the data frame. To find the regression coefficients (i.e. the estimates of β_0 and β_1), we can use the **coef** command

```
coef(mod)
```

```
## (Intercept)       width
## -12.3508177   0.4972306
```

We can now enter these estimates into our logistic regression equation, just as we did in the simple linear regression setting. Our logistic regression equation is

$$\log\left(\frac{p_i}{1-p_i}\right) = 12.3508 + 0.4972 \text{ width}_i,$$

where width_i is the width of a female crab's carapace shell and p_i is her probability of having one or more satellites.

5.2.3 Interpret the Model

To do some basic interpretation, let's focus on the predictor's coefficient: 0.4972. First, notice this is a **positive** number. This tells us that wider crabs have **higher** chances of having one or more satellites. If the predictor's coefficient were **zero**, there would be **no** linear relationship between the width of a female's shell and her log odds of having one or more satellites. If the predictor's coefficient were **negative**, then wider crabs would have **lower** chances of having one or more satellites.

5.2.4 Calculate Probabilities

Let's use our model for a female crab with a carapace shell that is 25 centimeters in width. (Note: this crab's shell width is within the range of our data set.) We start by simply substituting this crab's width into our regression equation:

$$\begin{aligned}\log\left(\frac{p_i}{1-p_i}\right) &= -12.3508 + 0.4972 \text{ width}_i \\ &= -12.3508 + 0.4972 * 25 \\ &= 0.0792.\end{aligned}$$

We could say that the log odds of a 25 cm female having satellites is about 0.0792, but let's make this more interpretable to everyday humans by translating this to a probability. We use a little algebra to solve for p_i:

$$\begin{aligned}\log\left(\frac{p_i}{1-p_i}\right) &= 0.0792 \\ \Rightarrow \left(\frac{p_i}{1-p_i}\right) &= \exp(0.0792) \\ \Rightarrow p_i &= \frac{\exp(0.0792)}{1+\exp(0.0792)} \\ &= 0.5198\end{aligned}$$

Here is our interpretation: the probability of a 25 cm wide female crab having one of more satellites is about 0.5198.

5.2.5 Test a Regression Coefficient

An essential question in regression is "Does this predictor actually help us predict the response?" In other words, we want to know whether there really is a relationship between our predictor and our response.

In the context of the crabs, we want to know whether the carapace width really does have a linear relationship with the log odds of satellites.

- The null hypothesis is that her carapace width has no linear relationship to her log odds of having satellites.
- The alternative hypothesis is that her carapace width has a linear relationship to her log odds of having satellites.

Remember: we assume there is *no* relationship until we find evidence that there *is* a relationship. That is, we assume that carapace width is unrelated to the probability of satellites; in order to say the carapace width is related to the probability of satellites, we need evidence. This evidence comes in the form of a statistical test.

The results of this test are reported in the **summary** command, which was introduced in the linear regression setting. Below is the summary for our crab model.

```
summary(mod)
```

```
##
## Call:
## glm(formula = y ~ width, family = binomial, data = crabs)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0281  -1.0458   0.5480   0.9066   1.6942
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -12.3508     2.6287  -4.698 2.62e-06 ***
## width         0.4972     0.1017   4.887 1.02e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 225.76  on 172  degrees of freedom
## Residual deviance: 194.45  on 171  degrees of freedom
## AIC: 198.45
##
## Number of Fisher Scoring iterations: 4
```

Although the summary provides many details about our model, we focus for now on the coefficients table in the center of the summary output. Recall that our model has two regression coefficients: the intercept β_0 and the slope β_1. The first row of the coefficients table displays information for β_0 and the second row displays information for β_1. The columns of the coefficients table provide the estimates of our regression coefficients, their standard errors (smaller standard errors indicate higher certainty), test statistics (for testing whether the regression coefficients differ from zero), and the p-values associated with the test statistics.

In order to answer our question ("Does this predictor actually help us predict the response?"), we focus on β_1, the regression coefficient on the predictor. (Recall that if $\beta_1 = 0$, then the log odds have no linear relationship with the predictor.) We find the test results in te two right-most columns and the second row in the the coefficients table of the summary. These entries show us that our test statistic is 4.887 and our p-value is 1.02e-06 (or 1.02×10^{-6}). We compare our p-value to a "significance level" (such as 0.05); **because our p-value is smaller than our significance level, we have evidence that the log odds of satellites have a significant linear relationship with the carapace width.**

5.3 Half-Time Exercises

5.3.1 Female Horseshoe Crab Weight

Continue using the horseshoe crab data to investigate the relationship between a female's weight and the log odds of her having satellites.

- Create a logistic regression using the female's weight as the predictor and whether she has satellites as the response variable.
- Write down the regression equation.
- Do heavier females having higher or lower chances of having satellites?
- Consider a female weighing 2000 grams. What is the probability that she has one or more satellites?
- Is there a linear relationship between a female crab's weight and her log odds of satellites? Find the p-value and use a significance level of 0.05.

5.3.2 Boundary Water Blowdown

The Boundary Water Canoe Area experienced wind speeds over 90 miles per hour on July 4, 1999. As a result, many trees were blown down. The data set below contains information on the diameter of each tree (**D**) and whether the tree died (**y**). **y** is 1 if the tree died and 0 if the tree survived. You can learn more about this data set here.

```
blowdown <- read.csv("http://www.cknudson.com/data/blowdown.csv")
```

Use this data set to understand the relationship between a tree's diameter and its log odds of death.

- Create a logistic regression using the tree's diameter the predictor and whether it died as the response variable.
- Write down the regression equation.
- Did thicker trees having higher or lower chances of dying?
- Consider a tree 20 cm in diameter. What is the probability that it died?
- Is there a linear relationship between the tree's diameter and its log odds of dying? Find the p-value and use a significance level of 0.05.

5.3.3 Beer

Recall the beer data set introduced by Alicia Johnson and Nathaniel Helwig. Imagine that you have a friend with a rather black-and-white outlook. If a beer's rating is 90 or higher, your friend considers this beer good. Scores lower than 90 indicate the beer is not good, according to your friend's rule. We have added a binary variable (**Good**) to to represent your friend's classification strategy: **Good** is 1 if the beer has a score of at least 90 and 0 otherwise.

- Use logistic regression to decide whether beers with higher **ABV** (alcohol by volume) are more likely to be "Good."
- Choose your favorite beer off the list and calculate its probability of having a score of at least 90.

```
beer <- read.csv("http://www.cknudson.com/data/MNbeer.csv")
```

5.3.4 State Colors

Recall the election data set introduced by Alicia Johnson. The variable **Red** has been added to the data set to indicate whether the state's color is red (1 if red, 0 otherwise).

- Is per capita income related to whether a state is red?

- If a state has a high per capita income, is it more or less likely to be red?

```
election <- read.csv("https://www.macalester.edu/~ajohns24/data/IMAdata1.csv")
election$Red <- as.numeric(election$StateColor == "red")
```

5.4 Beyond the Basics

5.4.1 Interpret the Model (Again!)

We can look beyond just whether the predictor's coefficient (β_1) is positive or negative. Exponentiating both sides of the regression equation produces

$$\frac{p_i}{1-p_i} = \exp(\beta_0 + \beta_1 x_i) = \exp(\beta_0)\exp(\beta_1 x_i).$$

To understand the relationship between our predictor and the log odds, let's see what happens when we increase x_i by one unit. That is, let's replace x_i with $x_i + 1$.

$$\frac{p_i}{1-p_i} = \exp(\beta_0)\exp(\beta_1(x_i+1)) = \exp(\beta_0)\exp(\beta_1 x_i)\exp(\beta_1).$$

That is, our odds for predictor value x_i are $\exp(\beta_0)\exp(\beta_1 x_i)$ while the odds for predictor value $x_i + 1$ are $\exp(\beta_0)\exp(\beta_1 x_i)\exp(\beta_1)$. These two odds differ by a factor of $\exp(\beta_1)$. That is, the ratio of the two odds is $\exp(\beta_1)$:

$$\frac{\exp(\beta_0)\exp(\beta_1 x_i)\exp(\beta_1)}{\exp(\beta_0)\exp(\beta_1 x_i)} = \exp(\beta_1).$$

Therefore, we can say that a one unit increase in the predictor is associated with a $\exp(\beta_1)$ multiplicative change in the odds. Let's try this with the horseshoe crab carapace width model.

```
exp(coef(mod))
```

```
##  (Intercept)        width
## 4.326214e-06 1.644162e+00
```

A one centimeter increase in carapace width is associated with a 1.64 multiplicative change in the odds of having satellites. Alternatively, imagine two female crabs that have carapace widths that differ by exactly 1 cm. The odds of the larger crab having satellites is approximately 1.64 times the odds of the smaller crab having satellites.

5.4.2 Incorporate a Categorical Predictor

Rather than using a quantitative predictor, we can use a categorical predictor. Let's try using the spine condition in our horseshoe crab data. Crabs were categorized according to whether they had two good spines (*good*), two worn or broken spines (*bad*), or one worn/broken spine and one good spine (*middle*).

We will fit the model using the following **glm** setup.

```
spinemod <- glm(y ~ 0 + spine, data = crabs, family = binomial)
coef(spinemod)
```

```
##    spinebad   spinegood spinemiddle
##   0.5955087   0.8602013  -0.1335314
```

This produces three log odds, one for each group (bad, good, and middle). If we eliminate the **0+**, we create a model with a reference group; this will compare the bad spine group to the other two groups.

```
spinemod2 <- glm(y ~ spine, data = crabs, family = binomial)
coef(spinemod2)
```

```
## (Intercept)   spinegood spinemiddle
##   0.5955087   0.2646926  -0.7290401
```

```
exp(coef(spinemod2))
```

```
## (Intercept)   spinegood spinemiddle
##   1.8139535   1.3030303   0.4823718
```

This model tells us that

- the odds of satellites for a female with two bad spines is 1.8.
- the odds of satellites for a female with two good spines is 1.3 times the odds of satellites for a female with two bad spines.
- the odds of satellites for a female with one good spine and one bad spine is .48 times the odds of satellites for a female with two bad spines.

Sometimes people have a hard time interpreting odds ratios below 1. In this case, it is useful to flip the ratio.

```
1/exp(coef(spinemod2))
```

```
## (Intercept)   spinegood spinemiddle
##   0.5512821   0.7674419   2.0730897
```

Therefore, we can reword the third bullet to the following: the odds of satellites for a female with two bad spines is 2 times the odds of satellites for a female with one bad spine and one good spine.

5.4.3 Incorporate Multiple Predictors

In the linear regression setting, you created models with multiple predictors. This is useful in the logistic regression setting as well. Let's create a model with both the female's carapace width and her weight. We expand our **glm** formula in the same way as you would expand the **lm** formula.

```
multimod <- glm(y ~ weight + width, data = crabs, family = binomial)
coef(multimod)
```

```
##   (Intercept)        weight         width
## -9.3547261192  0.0008337917  0.3067892044
```

This produces the following logistic regression equation:

$$\log\left(\frac{p_i}{1-p_i}\right) = -9.35 + 0.0008337917\ \text{weight}_i + 0.3067892044\ \text{width}_i.$$

Equivalently, our model could be written as

$$\frac{p_i}{1-p_i} = \exp(-9.35)\ \exp\left(0.0008337917\ \text{weight}_i\right)\ \exp\left(0.3067892044\ \text{width}_i\right).$$

The exponentiated regression coefficients

```
round(exp(coef(multimod)),3)
```

```
## (Intercept)      weight       width
##       0.000       1.001       1.359
```

tell us the following:

- Holding weight constant, a one centimeter increase in carapace width is associated with a 1.359 multiplicative change in the odds of having satellites.
- Holding carapace width constant, a one gram increase in weight is associated with a 1.001 multiplicative change in the odds of having satellites.

Of course, a one gram increase in weight is practically imperceptible. Let's instead consider a 100 gram increase in weight.

```
beta1 <- coef(multimod)[2]
exp(beta1 * 100)
```

```
##   weight
## 1.086954
```

Holding carapace width constant, a 100 gram increase in weight is associated with a 1.087 multiplicative change in the odds of having satellites.

5.4.4 Test a Regression Coefficient (Again!)

We discussed how to test a regression coefficient for a logistic model with a single predictor. You learned how to test a regression coefficient in multiple linear regression. Now it is time to test a logistic regression coefficient in models with multiple predictors.

Let's test whether a logistic regression model with both carapace width and weight is better than a model with carapace width only. That is, we want to know whether the female's weight contributes to the model. Again, we check the summary and look at the row labeled **weight**.

```
summary(multimod)
```

```
##
## Call:
## glm(formula = y ~ weight + width, family = binomial, data = crabs)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1127  -1.0344   0.5304   0.9006   1.7207
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -9.3547261  3.5280465  -2.652  0.00801 **
## weight       0.0008338  0.0006716   1.241  0.21445
## width        0.3067892  0.1819473   1.686  0.09177 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 225.76  on 172  degrees of freedom
## Residual deviance: 192.89  on 170  degrees of freedom
## AIC: 198.89
##
## Number of Fisher Scoring iterations: 4
```

The p-value is reported as 0.21445. This is larger than any standard significance level. Therefore, we should remove the female's weight from the model as it does not contribute a significant amount of information, after accounting for the female's carapace width. We can state our conclusion more formally: after accounting for the female's carapace width, we find no significant linear relationship between the female's weight and her log odds of having satellites.

5.5 More Exercises

5.5.1 Crabs, Revisited

Use the output below to determine whether, after accounting for the female's weight, there is a linear relationship between carapace width and the log odds of satellites.

5.5.2 Beer, Revisited

Create a logistic regression for the log odds of a beer being "Good" using both ABV and IBU as predictors.

- Holding IBU constant, are beers with higher ABVs **more** or **less** likely to be "Good"?
- After accounting for a beer's IBU, is there a relationship between ABV and a beer's log odds of being "Good"? Compare your p-value to a significance level of .1.

5.5.3 State Colors, Revisited

Use **IncomeBracket** to model the log odds of a state being red.

- Find the odds of a high income state being red.
- Fill in the blank: the odds of a high income state being red are _____ times the odds of a low income state being red.

5.6 Partial Solutions

5.6.1 Crab Weight

We create the model and look at the summary to find the p-value for the coefficient on weight The p-value is small (1.45×10^{-6}) so we can conclude that a female crab's weight **does** have a significant linear relationship with the log odds of her having one or more satellites.

```
weightmod <- glm(y ~ weight, family = binomial, data = crabs)
summary(weightmod)
```

```
##
## Call:
## glm(formula = y ~ weight, family = binomial, data = crabs)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1108  -1.0749   0.5426   0.9122   1.6285
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept) -3.6947264  0.8801975  -4.198 2.70e-05 ***
## weight       0.0018151  0.0003767   4.819 1.45e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 225.76  on 172  degrees of freedom
## Residual deviance: 195.74  on 171  degrees of freedom
## AIC: 199.74
##
## Number of Fisher Scoring iterations: 4
```

Now we can use our regression equation to calculate the probability of a 2000 gram female having one or more satellites.

```
logodds <- sum(coef(weightmod) * c(1, 2000))
exp(logodds)/(1+exp(logodds))
```

```
## [1] 0.4838963
```

The probability of a 2000 gram female having satellites is about half (.48).

5.6.2 Boundary Water Blowdown

We create the model, isolate the coefficients, and find the summary using the code below.

```
treemod <- glm(y ~ D, data = blowdown, family = binomial)
coef(treemod)
```

```
## (Intercept)           D
## -1.70211206  0.09755846
```

```
summary(treemod)
```

```
##
## Call:
## glm(formula = y ~ D, family = binomial, data = blowdown)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6309  -0.9616  -0.7211   1.1495   1.7172
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.702112   0.082798  -20.56   <2e-16 ***
## D            0.097558   0.004846   20.13   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5057.9  on 3665  degrees of freedom
## Residual deviance: 4555.6  on 3664  degrees of freedom
## AIC: 4559.6
##
```

```
## Number of Fisher Scoring iterations: 4
```

First, the coefficient on the diameter is positive. This tells us that larger trees were more likely to die. Moreover, the relationship between a trees diameter and its log odds of death is statistically significant: the p-value is quite small (smaller than 2×10^{-16}).

Finally, the probability of death for a tree that was 20 cm in diameter is calculated below.

```
(logodds <- coef(treemod) %*% c(1, 20))
```

```
##           [,1]
## [1,] 0.2490571
```

```
exp(logodds)/(1+exp(logodds))
```

```
##           [,1]
## [1,] 0.5619444
```

5.6.3 Beer

We create the model and look at the summary to find the p-value for the coefficient on ABV. The p-value is small (0.00747) so we can conclude that ABV **does** have a significant linear relationship with the log odds of a beer earning a score of at least 90.

```
beermod <- glm(Good ~ ABV, family = binomial, data = beer)
summary(beermod)
```

```
##
## Call:
## glm(formula = Good ~ ABV, family = binomial, data = beer)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.3847  -0.8206  -0.4663   0.7230   2.1320
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -9.7881     3.3959  -2.882  0.00395 **
## ABV           1.4662     0.5481   2.675  0.00747 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 51.564  on 43  degrees of freedom
## Residual deviance: 42.108  on 42  degrees of freedom
## AIC: 46.108
##
## Number of Fisher Scoring iterations: 5
```

Next, we create our model with both ABV and IBU.

```
beermod2 <- glm(Good ~ ABV + IBU, family = binomial, data = beer)
coef(beermod2)
```

```
##  (Intercept)          ABV          IBU
## -9.307699299  1.308989140  0.008248125
```

```
summary(beermod2)
```

```
##
## Call:
## glm(formula = Good ~ ABV + IBU, family = binomial, data = beer)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2930  -0.7759  -0.4435   0.7678   2.1315
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -9.307699   3.678245  -2.530   0.0114 *
## ABV          1.308989   0.721717   1.814   0.0697 .
## IBU          0.008248   0.025139   0.328   0.7428
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 51.564  on 43  degrees of freedom
## Residual deviance: 42.000  on 41  degrees of freedom
## AIC: 48
##
## Number of Fisher Scoring iterations: 5
```

The coefficient on ABV is positive: this tells us that, after accounting for IBU, beers with higher ABV are more likely to be "Good." The p-value for this regression coefficient is $.0697 < .1$, therefore we can conclude that, even after accounting for IBU, there is a significant linear relationship between a beers ABV and its log odds of being "Good."

5.6.4 State Colors

We create the model and look at the summary to find the p-value for the coefficient on per capita income. The p-value is very small (smaller than 2.2×10^{-16}) so we can conclude that per capita income **does** have a significant linear relationship with the log odds of a state being red.

```
electionmod <- glm(Red ~  per_capita_income, family = binomial, data = election)
summary(electionmod)
```

```
##
## Call:
## glm(formula = Red ~ per_capita_income, family = binomial, data = election)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.617  -1.146  -0.753   1.157   2.073
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)        1.649e+00  1.722e-01   9.575   <2e-16 ***
## per_capita_income -7.339e-05  7.214e-06 -10.173   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4352.6  on 3142  degrees of freedom
## Residual deviance: 4237.1  on 3141  degrees of freedom
## AIC: 4241.1
##
## Number of Fisher Scoring iterations: 4
```

```
lowhighmod <- glm(Red ~ IncomeBracket, data = election, family = binomial)
exp(coef(lowhighmod))
```

```
##      (Intercept) IncomeBracketlow
##        0.6197836        1.8217084
```

The odds of a high income state being red is about .62. The odds of a low income state being red are about 1.82 times the odds of a high income state being red. That is, low income states are more likely to be red than high income states.

Chapter 6

Using R Reproducibly

Author: Alicia Hofelich Mohr

Most scientific work is **iterative**

source: Wickham & Grolemund: R for Data Science

The process between input and output needs to be **transparent** and **repeatable**.

6.1 Types of Reproducibility

Many fields, most notably biomedicine and psychology, have been undergoing a **replication crisis** - a name given to the finding that many published research results can not be reproduced.

But what is meant by "reproducible"? Given the diversity of disciplines involved, not everyone uses the term in the same way. Scholars such as Victoria Stodden and Brian Nosek, who leads the Center for Open Science, distinguish between several types of reproducibility:

Computational Reproducibility: Given the author's data and statistical code, can someone produce the same results?

Empirical Reproducibility: Is there enough information communicated about the study (e.g., design, experimental procedure, exclusion criteria, etc) for another researcher to repeat it in the exact same way?

Statistical Reproducibility: Is adequate information provided about the choice of statistical tests, threshold values, corrections, and a priori hypotheses to protect against statistical biases (such as "p-hacking")?

Replicability: If someone re-runs the study with the same methods and analysis, collecting new data, do they get the same results?

6.2 General Principles of Reproducibility

6.2.1 Don't do things by hand

Everything that happens between the raw data and final output should be captured in the script. Point and click edits can't be tracked, and it may be impossible to reverse any errors (rounding, copy/paste,

overwriting).

It can also be time consuming to re-do "by hand" edits.

6.2.2 Automate where you can

Repetitive tasks are prone to errors (and are inefficient).

As a programmatic language, R is useful for many automation tasks.

- **File manipulation** *file.create(), dir.create(), list.files(), file.copy(), write()*
- **Looped processing** *for(), while()*
- **Vectorized functions** *apply(), lappy(), Vectorize()*

For example, our lab had an oscilloscope that collected data in around 100 separate .wav files. Before analysis, we needed to down sample, cut the time to the first five seconds, and remove the baseline each of the collected files.

You could do this by opening each of the 100 files one at a time in an audio program, doing all the steps, and resaving (not so fun!).

Or, automate with a few lines of R code:

```
library(tuneR)
setwd("~/Documents/wav_files")
for (i in list.files()) {
  wavdata <- readWave(paste0(getwd(), "/", i), header=F)
  #take correct channel
  wavdata <- ts(attr(wavdata, "left"), deltat=(1/64000))
  #downsample
  wavdata <- downsample(wavdata, samp.rate = 2004)
  #cut file
  wavdata <- window(wavdata,start=1, end=5)
  #Take out the moving baseline with a spline
  wavdata <- smooth.spline(x=wavdata, nknots=100)

  #save processed file
  save(wavdata, file=paste0(gsub(".wav", "", i), "_processed", ".Rdata"))
}
```

6.2.3 Use Version Control

In addition to using tools such as Git and Github, think about how you will identify milestone versions of a project.

Can you easily identify which version of your code and data was used for your submitted article? The revised article? A related conference presentation?

6.2.4 Document your environment

source: tlvincent, Rbloggers; Data: CRAN archive

The open source nature of R is what makes it a powerful tool for reproducible research, but it can also be part of the challenge. Although many packages allow backwards compatibility, not all do, and code run with one version of a package may not run (or worse, produce different results) than another.

To ensure your code can be run in long after your current laptop is functioning, documentation is key. At a minimum, think about documenting:

- **Infrastructure** What software and hardware are you using to run your code?
- **Packages** What version of packages are used in your code?
- **Workflow** How did your data get to this script, where is it going?
- **Functional requirements** What is your script supposed to do?

Capturing your session info does a lot of this for you:

```
sessionInfo()
```

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.1 LTS
##
## Matrix products: default
## BLAS: /home/geyer/local/current/lib/R/lib/libRblas.so
## LAPACK: /home/geyer/local/current/lib/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_3.5.1  backports_1.1.2 bookdown_0.7    magrittr_1.5
##  [5] rprojroot_1.3-2 tools_3.5.1     htmltools_0.3.6 Rcpp_0.12.18
##  [9] stringi_1.2.4   rmarkdown_1.10  knitr_1.20      xfun_0.3
## [13] stringr_1.3.1   digest_0.6.15   evaluate_0.11
```

6.2.5 Document your analysis

While code is technically "self-documenting", it doesn't always make sense to another user (or yourself six months after you wrote it!). Comment code heavily, including instructions on the order different scripts should be executed.

```
#Comment

#Comment

#Comment!
```

Using a "literate programming" tool, such as R Markdown, that combines analysis and interpretation can make it easy to document your analysis as you go.

6.2.6 Make your data open

One essential part of reproducibility is that your code (and data, where appropriate) are made available for verification or reuse. Many grants or journals may also require you to make your data and code available.

There are many options for sharing:

Github

If you are a student, faculty, or staff at the University of Minnesota, you have access to two versions of Github:

Pros of sharing with github:

- Version control/backup
- Easy to "open up" a private repo
- Great for reuse or contribution from others

Cons of github:

- Clunky to point users to past milestone versions
- Limited project metadata (indexing, searchability)
- No formal archival actions

Data & Archival Repositories

Data repositories are another option for sharing datasets and related documents, such as statistical code, procedures, or other supplemental material. These sites are specifically designed for hosting data, and many include services to make the data more findable, accessible, and preserved in the long-term. There are many different kinds of repositories for different kinds of data, and some journals or grants may require data to be shared in a specific repository

Pros of data repositories:

- Enhance findability (metadata, Google Scholar indexing)
- Data citations, Digital Object Identifiers (DOIs) for persistent access
- May offer curation and archival services
- Some manage access to data, allowing access to data that may not be suitable for public access

Cons of data repositories:

- Not every discipline has a dedicated repository
- Data services vary widely
- Can take a lot of work to prepare data for submission

6.3 Starting a Reproducible Workflow in R

Want to get started with a more reproducible workflow in R? Here are some quick tips and suggestions for using good R style

6.3.1 Organization

Create an R project for the main folder for a research project. This will make it easy to add git and github to your project.

Use consistent directory organization in your project folder (e.g., separate folders for raw data, scripts, plots, reports), and reference the relative paths to these folders in your scripts.

Give files descriptive and meaningful names. Avoid special characters (/ , . # ?), spaces between words, or very lengthy file names.

6.3.2 Anatomy of a script

Order matters in a script; data used in an analysis needs to be loaded at an earlier point in the script.

At the beginning of the script:

Start with metadata (data about the script or dataset). This helps you quickly identify what the script does without pouring through the code.

```
##########################################
## Analysis Script for Project A
##   2017-09-01
##
##  Script takes in "A_processed_data.csv"
##  Creates Tables 1-4, Figures 1, 2 for submission to
##    Journal of Cool Results
##########################################
```

Load all packages you use in one spot. Consider using package management tools such as pacman or packrat.

- *pacman* Makes it easy to load required packages, and install any that are missing. Very useful when sending scripts to collaborators.

```
# "p_load" will load packages, installing any that are missing
pacman::p_load("ggplot2", "devtools", "dplyr")
```

- *packrat* Creates a project-specific library for packages, so that the same versions of packages used in the script stay packaged with the script.

```
#Set up a private package library for a project
packrat::init("~/Documents/MyProject")
```

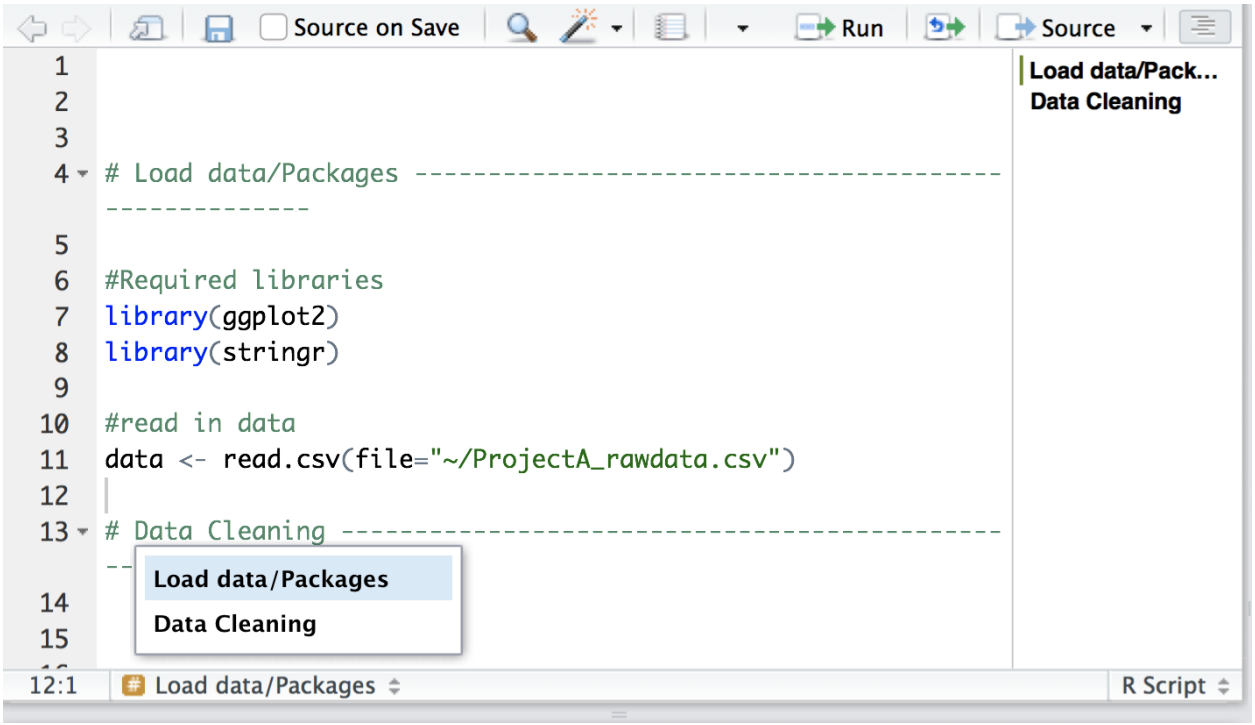
Set the working directory at start, and use relative file paths throughout. If you do this through RStudio's interface, copy the generated syntax into your script.

```
setwd("~/Documents/MyProject")
rawdata <- read.csv("raw_data/Myrawdatafile.csv")
```

Load all data in one spot. Again, if you are loading through the interface windows, copy the syntax generated in the console.

Throughout your script:

Use sections in R Studio to organize and quickly navigate your code. Press cmd-shift-R on a Mac or cntrl-shift-R on a PC to insert a section.



Use lots of white space (it's free). Use spaces between arguments and between different lines of code.

```
# Good
average <- mean(feet / 12 + inches, na.rm = TRUE)

# Bad
average<-mean(feet/12+inches,na.rm=TRUE)
```

Comment extensively.

```
#What does this do again?
x[i][[1]] <- do.call(rbind, y)
```

Use sensible names for data objects and variables. Keep it simple but meaningful.

```
# Good
day_one
day1

# Bad
first_day_of_the_month
DayOne
dayone
djm1
```

Set seed when doing any random number generations

```
#take a random sample twice
sample(1:100, size=10)
```

```
##  [1] 56 69 79 29 48 74 76 31 72 75
```

```
sample(1:100, size=10)
```

```
##  [1] 19 13 27 70 22 31 81 40 18 43
```

```
#set seed before hand to ensure the random numbers are consistent
set.seed(seed=42)
sample(1:100, size=10)
```

```
##  [1] 92 93 29 81 62 50 70 13 61 65
```

```
set.seed(seed=42)
sample(1:100, size=10)
```

```
##  [1] 92 93 29 81 62 50 70 13 61 65
```

If you save any outputs, make sure they are done by your script. ggsave(), png(), pdf()

```
# Base R: Creates "Figure1.png" in the working directory
(g <- ggplot(election, aes(x=winrep_2016)) +
    geom_bar() +
    labs(x="Trump win", y="Number of counties"))

png(file="Figure1.png")
g
dev.off()

#ggplot2: Creates "Figure1.png" in working directory
ggsave(filename="Figure1.png", plot=g, device="png")
```

At the end of your R session

Do not save your "R workspace" at the end of your R sessions. The R workspace contains all the objects currently loaded in your R environment. Instead, your script should generate all the objects needed. You can set R options to ensure it does not get saved.

You can also clear your environment at any time with the broom icon in the Environment window, or by typing rm(list=ls()) in the console.

If you use R studio, it's a good idea to start your script with rm(list=ls()) to make sure you are working from a clean environment.

Clear the console in RStudio with cntl-l.

Chapter 7

An R Markdown Demo

Author: Charles J. Geyer

7.1 License

7.2 R

- The version of R used to make this document is 3.5.1.
- The version of the `rmarkdown` package used to make this document is 1.10.
- The version of the `knitr` package used to make this document is 1.20.
- The version of the `ggplot2` package used to make this document is 3.0.0.

```
library(rmarkdown)
library(ggplot2)
```

7.3 Rendering

The source for this file is https://raw.githubusercontent.com/IRSAAtUMn/RWorkshop18/master/07-rmark.Rmd.

This is a demo for using the R package `rmarkdown`. To turn this file into HTML, after you have downloaded it, use the R command

```
render("07-rmark.Rmd")
```

The same rendering can be accomplished in RStudio by loading the document into Rstudio and clicking the "Knit" button.

R markdown can also be converted to output formats other than HTML. Among these are PDF, Microsoft Word, and e-book formats. Other output formats are explained in the Rmarkdown documentation.

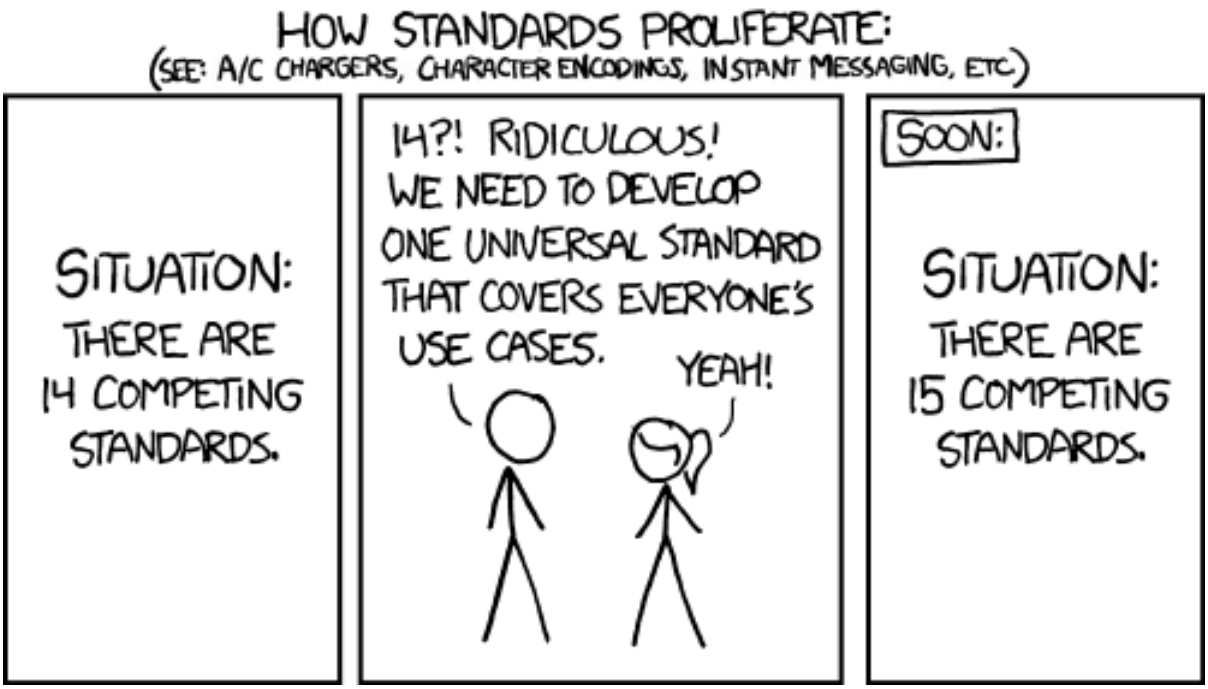


Figure 7.1: xkcd:927 Standards

7.4 Markdown

Markdown (Wikipedia page) is "markup language" like HTML and LaTeX but much simpler. Variants of the original markdown language are used by GitHub (https://github.com) for formatting README files and other documentation, by reddit (https://www.reddit.com/) for formatting posts and comments, and by R Markdown (http://rmarkdown.rstudio.com/) for making documents that contain R computations and graphics.

Markdown is fast becoming an internet standard for "easy" markup.

Markdown is way too simple to replace HTML or LaTeX, but it gets the job done, even if the result isn't as pretty as one might like.

7.5 R Markdown

7.5.1 Look

If you really need your documents to look exactly the way you want them to look, then you will have to learn LaTeX and `knitr`. But if you are willing to accept the look that R Markdown gives you, then it is satisfactory for most purposes.

7.5.2 What Does It Do?

R Markdown allows you to include R computations and graphics in documents *reproducibly*. That means that anyone anywhere who gets your R markdown file can satisfy themselves that R did what you say it did.

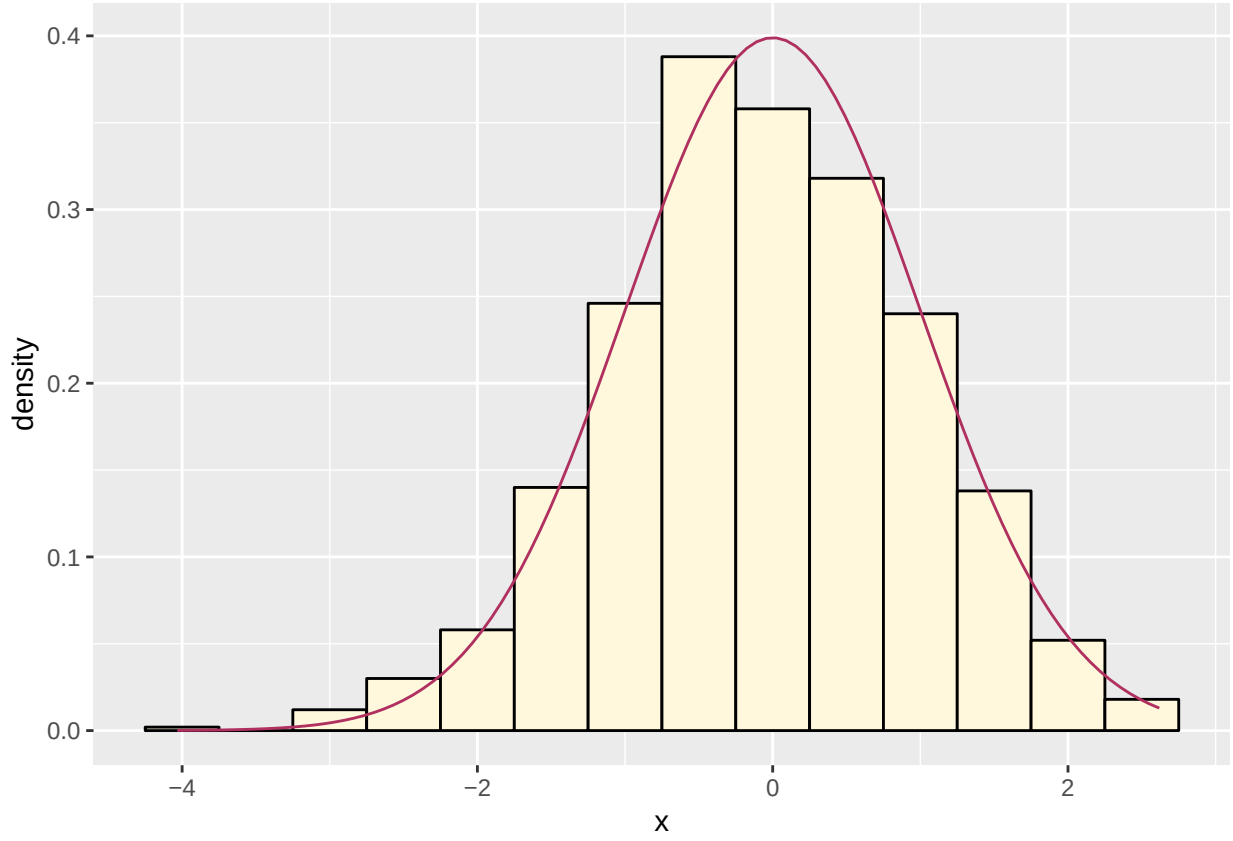


Figure 7.2: Histogram with probability density function.

This is very different from the old fashioned way of putting results in documents, where one just cut-and-pastes (snarf-and-barfs) the results into the document. Then there is zero evidence that these numbers or figures were produced the way you claim.

For concreteness, here is a simple example.

```
2 * pi
```

```
## [1] 6.283185
```

The result here was not cut-and-pasted into the document. Instead the R expression `2 * pi` was executed in R and the result was put in the document *by R Markdown, automatically.*

This happens *every time the document is generated* so the result always matches the R code that generated it. There is no way they can fail to match (which can happen and often does happen with snarf-and-barf).

Here is another concrete example.

```
# set.seed(42) # uncomment to always get the same plot
mydata <- data.frame(x = rnorm(1000))
ggplot(mydata, aes(x)) +
    geom_histogram(aes(y = ..density..), binwidth = 0.5,
        fill = "cornsilk", color = "black") +
    stat_function(fun = dnorm, color = "maroon")
```

Every time the document is generated, this figure is different because the random sample produced by `rnorm(1000)` is different. (If I wanted it to be the same, I could uncomment the `set.seed(42)` statement.)

Again, there is no way the figure can fail to match the R code that generates it. The source code for the document (the `Rmd` file) is *proof* that everything is as the document claims. Anyone can verify the results by generating the document again.

7.5.3 Why Do We Want It?

7.5.3.1 The "Replication Crisis"

Many are now saying there is a "replication crisis" in science (Wikipedia page).

There are many issues affecting this "crisis."

- There are scientific issues, such as what experiments are done and how they are interpreted.
- There are statistical issues, such as too small sample sizes, multiple hypothesis testing without correction, and publication bias.
- And there are computational issues: what data analysis was done and was it correct?

R Markdown only directly helps with the computational issues. It also helps to document your statistical analyses (*Duh!* Do you think generating a document that includes a statistical analysis may help to document it?)

Indirectly, R Markdown also helps with the other issues. Having the whole analysis from raw data to scientific findings publicly available — as many scientific journals now require — tends to make you a lot more careful and a lot more thoughtful in doing the analysis.

7.5.3.2 Business and Consulting

Outside of science there is no buzz about "replication crisis," at least not yet. The hype about "big data" is so strong that hardly anyone is questioning whether results are correct or actually support conclusions people draw from them.

But even if the results are never made public, R Markdown can still help a lot. Imagine you have spent weeks doing a very complicated analysis. The day before it is finished, a co-worker tells you there is a mistake in the data that has to be fixed. If you are generating your report the old-fashioned way, it will take almost as much time to redo the report as to do it in the first place. If you have done the report with R Markdown, correct the data, rerun R Markdown, and you're done.

R Markdown takes some time to learn. And it always takes more time to do a data analysis while simultaneously documenting it than to do it while skipping the documentation. But it takes a lot less time than doing the analysis without documentation and adding documentation later. Also

- R Markdown documentation is far more likely to be correct, and
- the analysis itself is far more likely to be correct.

7.5.3.3 Newbie Data Analysis

The way most newbies use R or any other statistical package is to dive right in

- typing commands into R,
- typing commands into a file and cut-and-pasting them into R, or
- using RStudio.

None of these actually document what was done because commands get edited a lot.

If you are in the habit of saving your workspace when you leave R or RStudio, can you explain *exactly* how every R object in there was created? *Starting from raw data?* Probably not.

7.5.3.4 Expert Data Analysis

The way experts use R is

- type commands into a file, say `foo.R`. Use

```
R CMD BATCH --vanilla foo.R
```

to run R to do the analysis.

- type commands with explanations into an R Markdown file, and render it in a clean R environment (empty global environment). Either start R with a clean global environment (with `R --vanilla`) and do

```
library(rmarkdown)
render("foo.Rmd")
```

or start RStudio with a clean global environment (on the "Tools" menu, select "Global Options" and uncheck "Restore .RData into workspace at startup", then close and restart) load the R Markdown file and click "Knit".

The important thing is using a clean R environment so all calculations are fully reproducible. Same results every time the analysis is rerun by you or by anybody, anywhere, on any computer that has R.

7.6 Examples Not Involving R

One of the purposes of this document is to serve as an example of how to use R Markdown.

We have already exemplified a lot, but there's more.

7.6.1 Title, Author, Date

Because this document is a chapter in a "book" done by the `bookdown` R package, it does not serve as a complete example of a standalone R markdown document.

So we have made a toy example https://raw.githubusercontent.com/IRSAAtUMn/RWorkshop18/master/foo/foo.Rmd, which renders as http://htmlpreview.github.io/?https://github.com/IRSAAtUMn/RWorkshop18/blob/master/foo/foo.html

That document shows how to put in a title, author, and date using the YAML comment block (set off by lines of hyphens at the top of the document). There does not seem to be any reference where all of the stuff that can be put in a YAML comment is documented. What can be done with YAML comments is scattered in many places in the R Markdown documentation.

7.6.2 Headers

Headers are indicated by hash marks (one for each level), as with all the headers in this document. There is also an alternative format that you can use if you want.

7.6.3 Paragraphs

Paragraph breaks are indicated by blank lines.

7.6.4 Fonts

Font changes are indicated by *stars for italic,* **double stars for boldface,** and `backticks for typewriter font (for code)` as here.

7.6.5 Lists

Bulleted lists examples are shown above. Here is a numbered list

1. item one,
2. item two, and
3. item three.

Note that one does not have to do the numbers oneself. R Markdown figures them out (so when you insert a new item in the list it gets the numbers right without you doing anything to make that happen).

The reference material for lists shows how to make sublists and more.

7.6.6 HTML Links

Links have also already been exemplified.

If you want the link text to be the same as the link URL, you can just put the URL in plain text. R Markdown will make it a link. It can also go in angle brackets.

If you want the link text to be different from the link URL, you can just put the link text in square brackets followed by the URL in round brackets (with no space in between the two). For example, `[CRAN](https://cran.r-project.org)` makes the link CRAN.

For more about links, here is the documentation.

7.6.7 Tables

There is Markdown syntax for tables, but we won't illustrate it here (documentation).

We will be more interested in tables created by R.

7.7 Examples Involving R

7.7.1 Code Chunks

Code chunks begin with ```` ```{r} ```` and end with ```` ``` ```` (documentation). These delimiters have to begin in column one (I think).

Here is an example.

```
2 + 2
```

```
## [1] 4
```

This is a "code chunk" processed by `rmarkdown`. When `rmarkdown` hits such a thing, it processes it, runs R to get the results, and stuffs the results (by default) in the output file it is creating. The text between code chunks is Markdown.

7.7.2 Code Chunks With Options

Lots of things can be made to happen by including options in the code chunk beginning delimiter. AFAIK all `knitr` chunk options can be used.

Here are some simple ones.

The option `eval=FALSE` says to show the chunk but do not evaluate it, as here

```
2 + 2
```

The option `echo=FALSE` says to to **not** show the chunk but **do** evaluate it (just the opposite of `eval=FALSE`), as here (nothing appears in the output document because of `echo=FALSE` but the code chunk is executed).

If you look at the document source you will see a hidden code chunk that assigns a value to the variable `hide` which we can see in a code chunk with no options

```
hide
```

```
## [1] 3
```

This example also shows that all code chunks are executed in the same R session, so R objects assigned names in earlier chunks can be used in later chunks.

Many more examples of options for code chunks are exemplified below.

7.7.3 Plots

We showed a simple plot above, here is a more complicated one.

7.7.3.1 Make Up Data

Simulate regression data, and do the regression.

```
n <- 50
x <- seq(1, n)
a.true <- 3
b.true <- 1.5
y.true <- a.true + b.true * x
s.true <- 17.3
y <- y.true + s.true * rnorm(n)
out1 <- lm(y ~ x)
summary(out1)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -58.087  -8.457   2.870  12.573  41.063
##
## Coefficients:
```

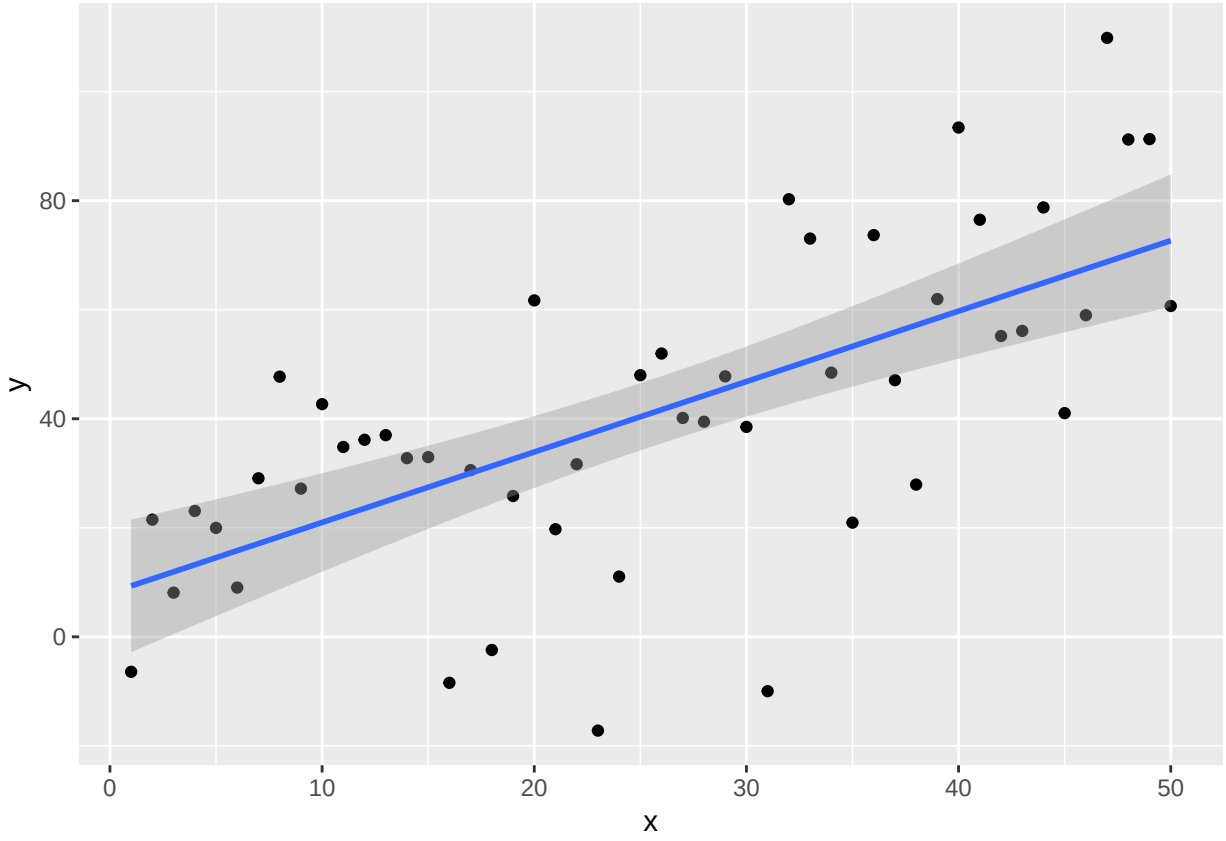


Figure 7.3: Simple Linear Regression

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   8.0516     6.2230   1.294    0.202
## x             1.2924     0.2124   6.085 1.86e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 21.67 on 48 degrees of freedom
## Multiple R-squared:  0.4355, Adjusted R-squared:  0.4237
## F-statistic: 37.03 on 1 and 48 DF,  p-value: 1.857e-07
```

7.7.3.2 Figure with Code to Make It Shown

The following figure is produced by the following code

```
mydata <- data.frame(x, y)
ggplot(mydata, aes(x = x, y = y)) + geom_point() +
    geom_smooth(method = "lm")
```

Here we use the chunk options `fig.align='center', fig.cap='Simple Linear Regression'` to center the figure and to get the figure legend.

Note that options are comma separated.

7.7.3.3 Figure with Code to Make It Not Shown

For this example we do a cubic regression on the same data.

```
out3 <- lm(y ~ x + I(x^2) + I(x^3))
summary(out3)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -51.688 -11.042   3.798  11.154  37.126
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 14.9635099 13.0663787   1.145    0.258
## x            1.0805831  2.1968906   0.492    0.625
## I(x^2)      -0.0307443  0.0995639  -0.309    0.759
## I(x^3)       0.0007654  0.0012840   0.596    0.554
##
## Residual standard error: 21.39 on 46 degrees of freedom
## Multiple R-squared:  0.4731, Adjusted R-squared:  0.4388
## F-statistic: 13.77 on 3 and 46 DF,  p-value: 1.538e-06
```

Then we plot this figure with a hidden code chunk (so the R commands to make it do not appear in the document).

This plot is made by a hidden code chunk that uses the option `echo=FALSE` in addition to `fig.align` and `fig.caption` that were also used in the preceding section.

Also note that, as with the figure in the section titled **What Does It Do?** above, every time we rerun `rmarkdown` these two figures change because the simulated data are random. (We could use `set.seed` to make the simulated data always the same, if we wanted to.)

7.7.4 R in Text

The *no snarf and barf* rule must be adhered to strictly. None at all!

When you want to refer to some number in R printout, either make a code chunk that contains the printout you want, or, much nicer looking, you can "inline" R printout.

Here we show how to do that. The quadratic and cubic regression coefficients in the preceding regression were -0.0307443 and 7.6538278×10^{-4}. Magic!

See the source for this document to see how the magic works. The dollar signs around the in-line R chunks are required by the way R markdown handles scientific notation. It outputs LaTeX, which is explained in Section 7.8 below. It thus forces you to understand at least that much LaTeX. (Sorry about that!)

If you never snarf and barf, and everything in your document is computed by R, then everything is always as claimed.

7.7.5 Tables

The same goes for tables. Here is a "table" of sorts in some R printout.

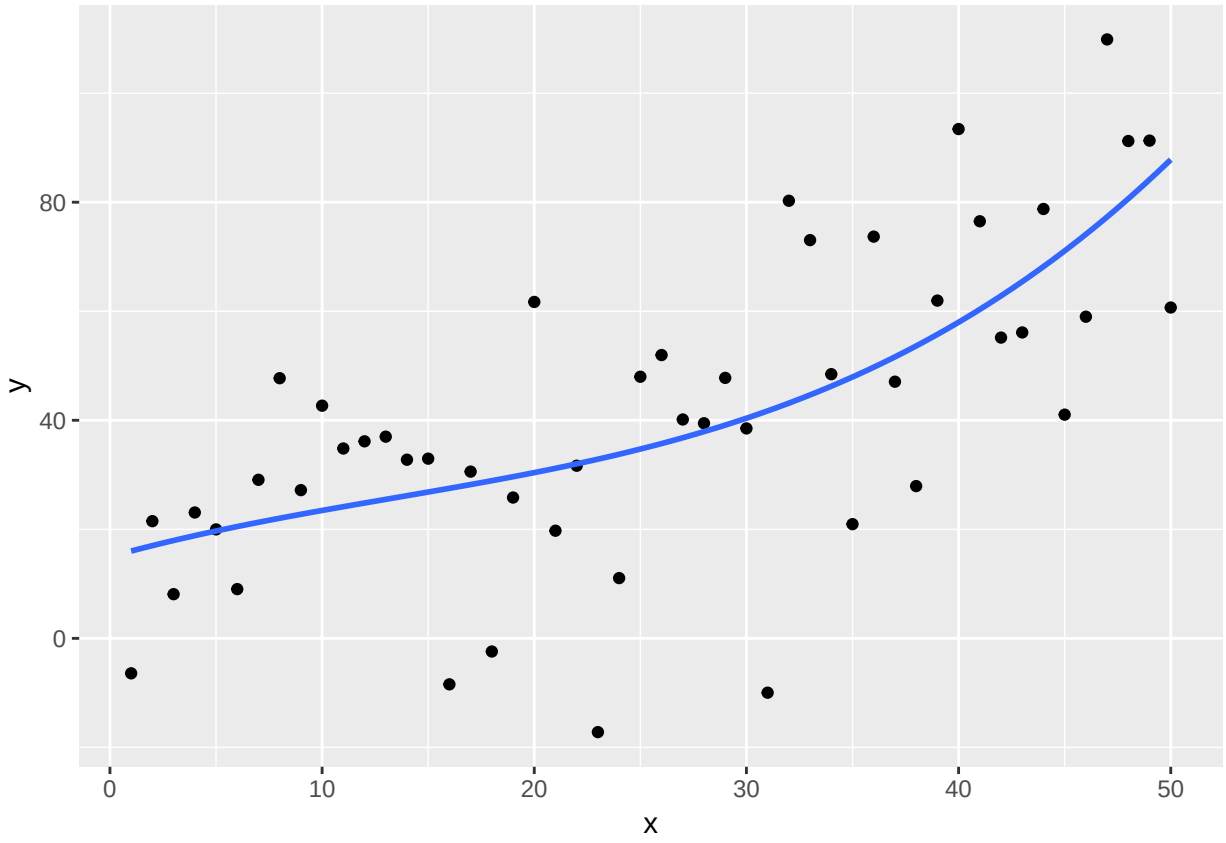


Figure 7.4: Scatter Plot with Cubic Regression Curve

Table 7.1: ANOVA Table

Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
48	22545.5				
47	21205.7	1	1339.78	2.929	0.094
46	21043.2	1	162.54	0.355	0.554

```
out2 <- lm(y ~ x + I(x^2))
anova(out1, out2, out3)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ x
## Model 2: y ~ x + I(x^2)
## Model 3: y ~ x + I(x^2) + I(x^3)
##   Res.Df   RSS Df Sum of Sq      F  Pr(>F)
## 1     48 22546
## 2     47 21206  1   1339.78 2.9287 0.09375 .
## 3     46 21043  1    162.54 0.3553 0.55405
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We want to turn that into a table in output format we are creating. First we have to figure out what the output of the R function `anova` is and capture it so we can use it.

```
foo <- anova(out1, out2, out3)
class(foo)
```

```
## [1] "anova"      "data.frame"
```

So now we are ready to turn the data frame `foo` into a table and the simplest way to do that seems to be the `kable` option on our R chunk

7.8 LaTeX Math

You can put real math in R Markdown documents. The way you do it mimics LaTeX (Wikipedia page), which is far and away the best document preparation system for ink on paper documents (it doesn't work so well for e-readers).

To actually learn LaTeX math, you have to read Section 3.3 of the LaTeX book and should also read the User's Guide for the `amsmath` Package.

Here are some examples to illustrate how good it is and how it works with R Markdown.

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/(2\sigma^2)}, \qquad -\infty < x < \infty.$$

If

$$f(x, y) = \frac{1}{2}, \qquad 0 < x < y < 1,$$

then

$$E(X) = \int_0^1 dy \int_0^y x f(x, y)\, dx \tag{7.1}$$

$$= \frac{1}{2} \int_0^1 dy \int_0^y x\, dx \tag{7.2}$$

$$= \frac{1}{2} \int_0^1 dy \left[\frac{x^2}{2} \right]_0^y \tag{7.3}$$

$$= \frac{1}{2} \int_0^1 \frac{y^2}{2}\, dy \tag{7.4}$$

$$= \frac{1}{2} \cdot \left[\frac{y^3}{6} \right]_0^1 \tag{7.5}$$

$$= \frac{1}{2} \cdot \frac{1}{6} \tag{7.6}$$

$$= \frac{1}{12} \tag{7.7}$$

7.9 Caching Computation

If computations in an R Markdown take so much time that editing the document becomes annoying, you can "cache" the computations by adding the option `cache=TRUE` to time consuming code chunks.

This feature is rather smart. If anything changes in code for the cached computations, then the computations will be redone. But if nothing has changed, the computations will not be redone (the cached results will be used again) and no time is lost.

My Stat 3701 course notes have an example that does caching:

- R Markdown source and
- HTML output.

There are also other examples of lots of other things at http://www.stat.umn.edu/geyer/3701/notes/.

7.10 Summary

Rmarkdown is terrific, so important that we cannot get along without it or its older competitors `Sweave` and `knitr`.

Its virtues are

- The numbers and graphics you report are actually what they are claimed to be.
- Your analysis is reproducible. Even years later, when you've completely forgotten what you did, the whole write-up, every single number or pixel in a plot is reproducible.
- Your analysis actually works—at least in this particular instance. The code you show actually executes without error.
- Toward the end of your work, with the write-up almost done you discover an error. Months of rework to do? No! Just fix the error and rerun Rmarkdown. One single problem like this and you will have all the time invested in Rmarkdown repaid.
- This methodology provides discipline. There's nothing that will make you clean up your code like the prospect of actually revealing it to the world.

Whether we're talking about homework, a consulting report, a textbook, or a research paper. If they involve computing and statistics, this is the way to do it.

Chapter 8

Git (Version Control)

Author: Charles J. Geyer

8.1 License

8.2 R

- The version of R used to make this document is 3.5.1.
- The version of the `rmarkdown` package used to make this document is 1.10.
- The version of the `knitr` package used to make this document is 1.20.

8.3 Version Control

8.3.1 What is It?

- revision control
- version control
- version control system (VCS)
- source code control
- source code management (SCM)
- content tracking

all mean the same thing.

It isn't just for source code. It's for any "content."

I use it for

- classes (slides, handouts, homework assignments, tests, solutions),

- papers (versions of the paper, tech reports, data, R scripts for data analysis),
- R packages (the traditional use of source code control, although there is a lot besides source code in an R package),
- notes (long before they turn into papers, I put them under version control).

8.3.2 Very Old Fashioned Version Control

```
asterLA.7-3-09.tex    asterLA.10-3.tex
asterLA.7-16.tex      asterLA.10-3c.tex
asterLA.7-16c.tex     asterLA10-4.tex
asterLA.7-19.tex      asterLA10-4c.tex
asterLA.7-19c.tex     asterLA10-4d.tex
asterLA.7-19z.tex     asterLA01-12.tex
asterLA8-19.tex       asterLA01-12c.tex
asterLA8-19c.tex      asterLA01-20.tex
asterLA.9-1.tex       asterLA01-20c.tex
asterLA.9-1c.tex      asterLA02-22.tex
asterLA.9-11.tex
```

A real example, file names of versions of a paper I wrote with a co-author without version control. This works but is highly sub-optimal. Only one author can work at a time because there is no good way to merge simultaneous changes. Nor is there any obvious ownership of the versions. It is hard to tell who did what when.

Using Dropbox for version control isn't any better.

8.3.3 A Plethora of Version Control Systems

Ripped off from Wikipedia (under "Version control").

Local Only	Client-Server	Distributed
SCCS (1972)	CVS (1986)	BitKeeper (1998)
RCS (1982)	ClearCase (1992)	GNU arch (2001)
	Perforce (1995)	Darcs (2002)
	Subversion (2000)	Monotone (2003)
		Bazaar (2005)
		Git (2005)
		Mercurial (2005)

There were more, but I've only kept the ones I'd heard of.

Don't worry. We're only going to talk about one (git).

8.3.4 Why?

Section 28.1.1.1 of the GNU Emacs manual.

Version control systems provide you with three important capabilities:

- **Reversibility:** the ability to back up to a previous state if you discover that some modification you did was a mistake or a bad idea.

- **Concurrency:** the ability to have many people modifying the same collection of files knowing that conflicting modifications can be detected and resolved.
- **History:** the ability to attach historical data to your data, such as explanatory comments about the intention behind each change to it. Even for a programmer working solo, change histories are an important aid to memory; for a multi-person project, they are a vitally important form of communication among developers.

8.3.5 Mindshare

Starting from nowhere in 2005, `git` and GitHub (https://github.com/) have gotten dominant mindshare, at least among free and open source systems.

In Google Trends, the only searches that are trending up are for `git` and github. Searches for competing version control systems are trending down.

Many well known open-source projects are on git. The Linux kernel (of course since Linus Torvalds wrote git to be the VCS for the kernel), android (as in phones), Python, Go, Ruby on Rails, Gnome, Qt, KDE, X, Perl, Vim, and GNU Emacs.

Some aren't. R is still on Subversion. Firefox is still on Mercurial.

8.4 Git

8.4.1 Git versus GitHub

Git is a version control system. It is a command line program that can be used by itself (and, in fact, is best used that way).

GUI applications are available on all platforms to dumb down Git with menus and buttons.

GitHub is a company (recently purchased by Microsoft) that is just as famous as Facebook or Google in the geek world. It offers "social coding." Thus it is something like Facebook for software developers. You can use `git` without GitHub (or one of its competitors) but GitHub helps you show the world what you do and collaborate with the world.

One of the GitHub developers wrote the best book introducing Git; Pro Git.

Still, I want to emphasize that you do not need GitHub to use Git any more than you need RStudio to use R.

8.4.2 Git and GitHub and RStudio

RStudio facilitates using Git for R projects. It seems to require also using GitHub (but this can be worked around).

But if you only know how to use Git via RStudio, you will not know how to use Git for non-R projects.

8.4.3 Getting Git

Type "git" into Google and follow the first link it gives you http://git-scm.com/.

Under "downloads" it tells you how to get Git for Windows and for Mac OS X.

If you have Linux, it just comes with (use the installer for your distribution).

E. g., on Ubuntu

```
sudo apt-get update
sudo apt-get install git
```

8.4.4 What?

From Wikipedia under Git:

- Torvalds has quipped about the name *git,* which is British English slang meaning "unpleasant person." Torvalds said: "I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'git'."
- The man page describes Git as "the stupid content tracker".

From memory:

- Linus has also said that it was a short name that hadn't already been used for a UNIX command.

8.4.5 Using Git

8.4.5.1 From the Command Line

There is only one command to use Git and it is called `git`. The first command line argument specifies one of many subcommands.

`git help`

shows the frequently used subcommands (many more than we will discuss here).

As this also says at the bottom `git help -a` shows all subcommands and `git help -g` shows "concept guides."

For example

`git help everyday`

shows about 20 git subcommands that are enough for straightforward use of git. (We won't even discuss that many.)

8.4.5.2 With a GUI Application

We won't discuss this. They are all different.

If you are using a GUI application there is usually a way to get a UNIX command line. So anything that cannot be done graphically (menus and buttons, drag and drop) can still be done. The GUI is not as helpful as you might think.

8.4.5.3 With RStudio

It is hopeless to try to describe how to do anything with a WIMP application like RStudio. If it isn't obvious, you're in trouble.

Nevertheless, I made detailed instructions for how to avoid almost everything below and just use RStudio to do the simplest baby project. They are https://github.com/IRSAAtUMn/RWorkshop18/blob/master/notes/08-git.md.

8.5 Tell Git Who You Are

Before doing anything else. You have to tell Git who you are.

The UNIX commands

```
git config --global user.name "Charles J. Geyer"
git config --global user.email charlie@stat.umn.edu
```

do this. Of course, replace my name and e-mail address with yours.

If you are using a GUI application or RStudio, you probably need to escape to the command line to do this.

8.6 Starting a New Project Locally

Create a directory (a. k. a. folder) that will contain your project. Change directory into it and do `git init`. In UNIX (e. g. Linux or OS X), if the directory is `Foo`, then

```
cd Foo
git init
```

does the job.

The UNIX command

```
ls -A
```

shows that there is a new directory `.git` that is the hallmark of a git repository ("repo" for short). Now all Git subcommands work in this directory. The `.git` subdirectory is where Git stores all information it keeps track of. The rest of the directory is where you work.

If you have created the project using RStudio (as discussed above), then RStudio has run `git init` for you so you do not have to do it yourself.

8.7 Starting an R Project that will be a CRAN Package

If you are starting an R Project that will provide an R package, especially one intended to eventually be on CRAN, put the directory that contains the package *in* the Git repo. Do not try to make the Git repo the same directory as the R package. A Git repo contains a lot of stuff that an R package is not allowed to have.

8.8 Cloning an Existing Project

```
git clone git://github.com/cjgeyer/foo.git
```

clones an already existing project on which a lot of work has been done (and you get it all, including all of the version control history).

This requires that you have set up SSH for use with Git, which will be discussed below (Section GitHub and SSH keys).

Alternatively,

```
git clone https://github.com/cjgeyer/foo.git
```

(This is easier to use in the beginning but rapidly becomes a nuisance.)

This (clone an existing project) is what RStudio wants us to do and what we discussed above.

8.9 GitHub and SSH keys

To use GitHub productively, you need to make an SSH key pair and tell GitHub about the public key. This allows you to only type the SSH passphrase for the key once per reboot of your computer. This is secure. Your computer remembers what it needs to remember to access github (or use SSH in other ways without typing a password or passphrase, but we won't cover that).

If you don't do this, you will have to type your GitHub password frequently to use Git. This is a nuisance.

Recent updates of Microsoft Windows 10 have an OpenSSH client. AFAICS it works like it does on UNIX (Linux or OS X). It is a command line tool.

To make keys, you do from the command line

```
ssh-keygen
```

(this is one UNIX command, the hyphen is part of the name). Do what it says (successfully type the same passphrase twice and after having accepted the default location for the keys or having specified another location).

If you manage to do this successfully, there will be two new files

```
id_rsa
id_rsa.pub
```

created somewhere (the default is the `.ssh` subdirectory of your home directory). The first is the private key; the second is the public key.

Upload the public key to GitHub. Log in to your account; on the menu triangle next to your picture (upper right corner) choose "Settings"; on the settings page, choose "SSH and GPG keys" in the navigation area on the left; then on the page that takes you to click the green button labeled "New SSH key" and fill in the resulting text area. The title can be anything. Into the "Key" textarea you cut-and-paste the contents of the `id_rsa.pub` file. To get the contents in UNIX, do

```
cat id_rsa.pub
```

at the command line. What you see is what you cut-and-paste into the GitHub textarea. Then click the green button labeled "Add SSH key". If you did this successfully, it will show the "fingerprint" of the key.

8.10 Halftime Summary

We are perhaps not halfway done, but the worst is over. Everything we have talked about so far only needs to be done once. It is painfully complicated, but once it is done you only need to do it again very rarely.

- You do `git init` at most once per project.
- You do `git clone` at most once per project on each computer you want to use for development.
- You create an SSH key at most once per computer you buy or get an account on (or after each reinstall of the operating system that blows away your account, if any).
- You tell GitHub about the SSH key once per SSH key.
- You tell Git your name and e-mail address similarly, at most once per per each computer you buy or get an account on.

Now we are ready to talk about everyday use of Git.

8.11 Everyday Git: One Computer, No Collaboration

8.11.1 Commits

What Git remembers is *commits* created by the git subcommand `git commit`.

If you never commit, then Git never does anything.

A Git commit consists of exactly what you say it does. Git allows precise control over what it does.

Most of the time you do not want such precise control. You just want Git to do the obvious. It can do that too.

So first the hard way, and then the easy way.

8.11.1.1 The Edit, Add, Commit Cycle.

There is no need to commit unless the Git repo has changed. So you edit some file in the repo so it is different from what Git remembers (in a previous commit).

Now if you want Git to remember the new version, you tell it first that you want it to remember the new version and second to do the commit. For example, if the file is `foo.R`, then

```
git add foo.R
git commit -m "A message about what the changes are"
```

does these two steps. If there are multiple files that have been edited, then you must do `git add` for each one you want changes remembered in the commit.

The `-m` argument to `git commit` is not necessary if you have told either UNIX or git what editor you use.

```
git config --global core.editor vim
```

or whatever editor you use instead of vim will do that job (this too is only done once per computer account you have). Then

```
git commit
```

will pop up an editor instance to type in the commit message.

8.11.1.2 The Cycle without Add Most of the Time

Doing a bunch of `git add` commands before each commit is what allows precise control of what you want the commit to remember.

But usually, you want the commit to remember all the changes to files Git is already tracking. One command does this:

```
git commit -a
```

(the `-a` flag tells Git to add all files it is already tracking).

When you use this pattern, you only need to say `git add` filename for new files either created since the last commit or not tracked before (not added in any previous commit).

8.11.1.3 Using RStudio

The edit-add-commit cycle can also be done entirely with RStudio.

One can edit the file in the upper left window and commit in the upper right window. Clicking the button labeled "Commit" will pop up a window in which you can select files to add, provide a commit message, and commit.

8.11.2 Status

Before doing a commit, it is often useful to do

```
git status
```

This tells you what would be committed (either with or without `-a`) if you were to do a commit. It also lists untracked files that will not be committed unless you do `git add` on them.

8.11.3 The File `.gitignore`

The file `.gitignore` in the top directory of the repo list files that `git status` and other git subcommands should ignore.

It uses wildcards (the gitignore documentation explains). For example, a `.gitignore` file that contains

```
*.html
*.Rout
*.pdf
```

ignores all files with those extensions (the character `*` is the wildcard character: it matches any string).

RStudio provides a `.gitignore` file in any Git repo it creates, but you may want to add to it.

8.11.4 Log

The command

```
git log
```

lists all commits that have been done, showing (by default) all the commit messages, the date and time, and the committer.

It provides an overview of the whole history of the repo.

8.11.5 Diff

The command

```
git diff
```

shows (by default) all of the changes between the "staging area," which contains any results of `git add` that you may have done, and the working directory.

If have not done any `git add` since the last commit, then it shows all of your edits to tracked files since the last commit.

This can be very useful.

- If working with collaborators, this is Git's competitor to "track changes" in Microsoft Office or Libre Office. It allows you to just look at changes.
- Even if there are no collaborators, it is often useful just to look at changes you made, especially just before a commit (to look even harder for mistakes).

8.12 Everyday Git: GitHub or Multiple Computers

If you made the repo by cloning a GitHub repository, then

```
git push origin master
```

will copy the latest commit to the GitHub repo. Now it is backed up there.

If you are using multiple computers, then you can use GitHub for communication between them. To start working on another computer, do all of the one time stuff (tell Git your name and e-mail address, make SSH keys, and put the public key on GitHub (there is a different public key for each computer, or at least for each computer account if different computers share file storage)).

Then clone (with `git clone`) the GitHub repo on this computer.

And you are ready to work on this other computer.

When you are finished working on this computer, do a commit, then

```
git push origin master
```

to push the changes to GitHub. Then on another computer (assuming no changes have been made to the working directory since the last commit on this computer; `git status` will tell you)

```
git pull origin master
```

will get all of the changes from GitHub.

Warning: Either of these (`git push` or `git pull`) can fail because you did effectively simultaneous (as far as Git can tell) editing on the two computers. Then you have to figure out how to fix the issue. The commands `git merge` and `git stash` may help. The former will be discussed below, but the latter we will skip. Most of the time, there will be no trouble (there is only trouble when you make changes on both computers with no commits; then you have to somehow reconcile the changes).

8.13 Everyday Git: Collaboration

8.13.1 Setup

When working with collaborators, the simplest workflow is for each collaborator to have their own GitHub repository and for everyone to pull from everyone else.

Suppose you are working with one collaborator https://github.com/jfmuggs who has an already existing GitHub repo named `blurfle`.

You clone it

```
git clone git://github.com/jfmuggs/blurfle.git
```

Now you have the `blurfle` repo on your computer. You have all of the code and all of the history. But you want to be able to push changes to where your collaborator (J. Fred Muggs, call him Fred) can see them. So you need a GitHub repo in your account.

So create an empty public repo (using the button labeled "New"). Don't put a "README" file or a Gitignore file or anything. Then follow GitHub's instructions to push the local repo to GitHub

```
git remote add origin https://github.com/cjgeyer/blurfle.git
git push -u origin master
```

except that doesn't work because there is already a remote repository name `origin` (in the `jfmuggs` account). So let's change that, and while we are at it, call neither of the GitHub repos `origin`.

```
git remote rename origin fred
git remote add me https://github.com/cjgeyer/blurfle.git
git push -u me master
```

Now there are two GitHub repos, both exactly the same (one in the `jfmuggs` account and one in the `cjgeyer` account). Of course, if you were doing this, you would replace my account name with yours.

Now Fred and I each have two repos, one local and one remote (on GitHub) that we can commit and push to. I also have another remote repo named (locally) `fred` that I can pull from. Fred should also give my GitHub repo a name; say he does

```
git remote add charlie https://github.com/cjgeyer/blurfle.git
```

8.13.2 Workflow

I make some changes and do a commit, then I push them

```
git push me master
```

Then I tell Fred "pull from me" (either by e-mail or by a GitHub pull request).

Then Fred can pull my changes

```
git pull charlie master
```

As always `git push` and `git pull` should only be done in a clean repo (just after a commit).

It gets complicated when we both make changes simultaneously. Then someone may have to reconcile any incompatible changes.

I get a "Pull from me" from Fred. I have uncommitted changes, so I do a commit (never pull when there are uncommitted changes). Then I do

```
git pull fred master
```

and it doesn't work. Git says

```
Auto-merging foo.txt
CONFLICT (content): Merge conflict in foo.txt
Automatic merge failed; fix conflicts and then commit the result.
```

(if the file in which merge conflicts exist is named `foo.txt` and there may be multiple such files).

When we edit this file we see something like this

```
<<<<<<< HEAD
Foo that Foos is not static Foo.
=======
Foo! Bar! Baz! Qux!
>>>>>>> a26ce42b351548fb0efd61f7902358b1f69d3266
```

Schematically, this is

```
<<<<<<< HEAD
```

then some text that was my version of what goes here, then

```
=======
```

then some text that was Fred's version of what goes here, then

```
>>>>>>> [some very long hexadecimal string, the SHA1 hash for the remote commit]
```

I edit the file, replacing all of this with what I think the text should be here (naturally, I think I'm right, although not always), in this case just the one line

```
Foo that Foos is not static Foo.
```

and then I have to do a commit (because the pull that failed with merge conflicts did not do a commit)

```
git commit -a
```

accepting the default commit message that Git suggests

```
Merge branch 'master' of github.com/jfmuggs/blurfle
```

Then I push and tell Fred to "Pull from me".

Most pulls will work. If changes are made to two different parts of the file, Git just makes them automatically and commits the merge. It is only overlapping changes that must be fixed manually.

8.14 Wrapup

The Git subcommands

```
git init
git clone
git add
git commit
git diff
git status
git log
git help
git push
git pull
git remote
```

Allow you to do most of what Git can do for you. There are many other subcommands and a huge number of options to all subcommands, including the ones listed above (and we only discussed one such option, the `-a` option to `git commit`).

So there is a lot more to learn about Git. But this is enough for a start.