Codes for STEM

Noushin Nabavi

2020-09-19

Contents

Chapter 1

Coding for STEM

Tools and capabilities of data science is changing everyday!

This is how I understand it today:

Data can: * Describe the current state of an organization or process
* Detec anomalous events
* Diagnose the causes of events and behaviors
* Predict future events

Data Science workflows can be developed for:
* Data collection and management
* Exploration and visualization
* Experimentation and prediction

Applications of data science can include:
* Traditional machine learning: e.g. finding probabilities of events, labeled data, and algorithms
* Deep learning: neurons work together for image and natural language recognition but requires more training data
* Internet of things (IOT): e.g. smart watch algorithms to detect and analyze motion sensors

Data science teams can consist of: * Data engineers: SQL, Java, Scala, Python
* Data analysts: Dashboards, hypothesis tests and visualization using spreadsheets, SQL, BI (Tableau, power BI, looker)
* Machine learning scientists: predictions and extrapolations, classification, etc. and use R or python * Data employees can be isolated, embedded, or hybrid

Data use can come with risks of identification of personal information. Policies for personally identifiable information may need to consider:

* sensitivity and caution
* pseudonymization and anonymization

Preferences can be stated or revealed through the data so questions need to be specific, avoid loaded language, calibrate, require actionable results.

Data storage and retrieval may include: * parallel storage solutions (e.g. cluster or server)
* cloud storage (google, amazon, azure)
* types of data: 1) unstructured (email, text, video, audio, web, and social media = document database); 2) structured = relational databases
* Data querying: NoSQL and SQL

Communication of data can include:
* Dashboards
* Markdowns
* BI tools
* rshiny or d3.js

Team management around data can use: * Trello, slack, rocket chat, or JIRA to communicate due data and priority

A/B Testing: * Control and Variation in samples
* 4 steps in A/B testing: pick metric to track, calculate sample size, run the experiment, and check significance

Machine learning (ML) can be used for time series forecasting (investigate seasonality on any time scale), natural language processing (word count, word embeddings to create features that group similar words), neural networks, deep learning, and AI.
Learning can be classified into: *Supervised*: labels and features/ Model evaluation on test and train data with applications in: * recommendation systems
* subscription predictions
* email subject optimization
Unsupervised: unlabeled data with only features
* clustering

Deep learning and AI requirements: * prediction is more feasible than explanations
* lots of very large amount of training data

Chapter 2

Introduction

Importing data into R

working with excel, csv, txt, and tsv files in R

```
library(readr)
library(data.table)
library(readxl)
library(gdata)
# library(XLConnect)
library(httr)
library(rvest)
library(xml2)
library(rlist)
library(jsonlite)
library(dplyr)
```

Importing csv file: pools <- read.csv(“swimming_pools.csv”, stringsAsFactors = FALSE)

With stringsAsFactors, you can tell R whether it should convert strings in the flat file to factors.

Import txt file with read.delim: hotdogs <- read.delim(“hotdogs.txt”, header = FALSE)

Import txt file with read.table: hotdogs <- read.table(path, sep = “”, col.names = c(”type“,”calories“,”sodium”))

Import with readr functions: - read_csv, read_tsv, and read_delim are part of this package

Can specify column names before import: properties <- c(“area”, “temp”, “size”, “storage”, “method”, “texture”, “flavor”, “moistness”)

Import potatoes.txt: potatoes <- read_tsv(“potatoes.txt”, col_names = properties)

Import potatoes.txt using read_delim(): potatoes <- read_delim(“potatoes.txt”, delim = ””, col_names = properties)

Import 5 observations from potatoes.txt: potatoes_fragment <- read_tsv(“potatoes.txt”, skip = 6, n_max = 5, col_names = properties)

Import all data, but force all columns to be character: potatoes_char potatoes_char <- read_tsv(“potatoes.txt”, col_types = “cccccccc”, col_names = properties)

Import without col_types hotdogs <- read_tsv(“hotdogs.txt”, col_names = c(“type”, “calories”, “sodium”))

The collectors you will need to import the data fac <- col_factor(levels = c(“Beef”, “Meat”, “Poultry”)) int <- col_integer()

Edit the col_types argument to import the data correctly: hotdogs_factor <- read_tsv(“hotdogs.txt”, col_names = c(“type”, “calories”, “sodium”), col_types = list(fac, int, int))

Import potatoes.csv with fread() from data.table: potatoes <- fread(“potatoes.csv”)

Import columns 6 and 8 of potatoes.csv: potatoes <- fread(“potatoes.csv”, select = c(6, 8))

Plot texture (x) and moistness (y) of potatoes: plot(potatoes*texture*, *potatoes*moistness)

Print the names of all worksheets using readxl library: excel_sheets(“urbanpop.xlsx”)

Read the sheets, one by one pop_1 <- read_excel(“urbanpop.xlsx”, sheet = 1) pop_2 <- read_excel(“urbanpop.xlsx”, sheet = 2) pop_3 <- read_excel(“urbanpop.xlsx”, sheet = 3)

Put pop_1, pop_2 and pop_3 in a list: pop_list <- list(pop_1, pop_2, pop_3)

Read all Excel sheets with lapply(): pop_list <- lapply(excel_sheets(“urbanpop.xlsx”), read_excel, path = “urbanpop.xlsx”)

Import the first Excel sheet of urbanpop_nonames.xlsx (R gives names): pop_a <- read_excel(“urbanpop_nonames.xlsx”, col_names = FALSE)

Import the first Excel sheet of urbanpop_nonames.xlsx (specify col_names): cols <- c(“country”, paste0(“year_”, 1960:1966)) pop_b <- read_excel(“urbanpop_nonames.xlsx”, col_names = cols)

Import the second sheet of urbanpop.xlsx, skipping the first 21 rows: urbanpop_sel <- read_excel(“urbanpop.xlsx”, sheet = 2, col_names = FALSE, skip = 21)

Print out the first observation from urbanpop_sel urbanpop_sel[1,]

Import a local file Similar to the readxl package, you can import single Excel sheets from Excel sheets to start your analysis in R.

Import the second sheet of urbanpop.xls: urban_pop <- read.xls("urbanpop.xls", sheet = "1967-1974")

Print the first 11 observations using head() head(urban_pop, n = 11)

Column names for urban_pop columns <- c("country", paste0("year_", 1967:1974))

Finish the read.xls call urban_pop <- read.xls("urbanpop.xls", sheet = 2, skip = 50, header = FALSE, stringsAsFactors = FALSE, col.names = columns)

Import all sheets from urbanpop.xls path <- "urbanpop.xls" urban_sheet1 <- read.xls(path, sheet = 1, stringsAsFactors = FALSE) urban_sheet2 <- read.xls(path, sheet = 2, stringsAsFactors = FALSE) urban_sheet3 <- read.xls(path, sheet = 3, stringsAsFactors = FALSE)

Extend the cbind() call to include urban_sheet3: urban_all urban <- cbind(urban_sheet1, urban_sheet2[-1], urban_sheet3[-1])

Remove all rows with NAs from urban: urban_clean urban_clean <- na.omit(urban)

Print out a summary of urban_clean summary(urban_clean)

When working with XLConnect, the first step will be to load a workbook in your R session with loadWorkbook(); this function will build a "bridge" between your Excel file and your R session: Here using the XLConnect package

Build connection to urbanpop.xlsx: my_book <- loadWorkbook("urbanpop.xlsx")

List the sheets in my_book getSheets(my_book)

Import the second sheet in my_book readWorksheet(my_book, sheet = 2)

Import columns 3, 4, and 5 from second sheet in my_book: urbanpop_sel urbanpop_sel <- readWorksheet(my_book, sheet = 2, startCol = 3, endCol = 5)

Import first column from second sheet in my_book: countries countries <- readWorksheet(my_book, sheet = 2, startCol = 1, endCol = 1)

cbind() urbanpop_sel and countries together: selection selection <- cbind(countries, urbanpop_sel)

Add a worksheet to my_book, named "data_summary" createSheet(my_book, "data_summary")

Use getSheets() on my_book getSheets(my_book)

Create data frame: sheets <- getSheets(my_book)[1:3] dims <- sapply(sheets, function(x) dim(readWorksheet(my_book, sheet = x)), USE.NAMES =

FALSE) summ <- data.frame(sheets = sheets, nrows = dims[1,], ncols = dims[2,])

Add data in summ to "data_summary" sheet writeWorksheet(my_book, summ, "data_summary")

Rename "data_summary" sheet to "summary" renameSheet(my_book, "data_summary", "summary")

Remove the fourth sheet removeSheet(my_book, 4)

Save workbook to "renamed.xlsx" saveWorkbook(my_book, file = "renamed.xlsx")

Download various files with download.file() Here are the URLs! As you can see they're just normal strings:

```
csv_url <- "http://s3.amazonaws.com/assets.datacamp.com/production/course_1561/datasets
tsv_url <- "http://s3.amazonaws.com/assets.datacamp.com/production/course_3026/datasets

# Read a file in from the CSV URL and assign it to csv_data
csv_data <- read.csv(file = csv_url)

# Read a file in from the TSV URL and assign it to tsv_data
tsv_data <- read.delim(file = tsv_url)

# Examine the objects with head()
head(csv_data, n = 2)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
```

```
head(tsv_data, n = 2)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
```

Download the file with download.file()

```
download.file(url = csv_url, destfile = "feed_data.csv")

# Read it in with read.csv()
csv_data <- read.csv(file = "feed_data.csv")


# Add a new column: square_weight
csv_data$square_weight <- (csv_data$weight ^ 2)
```

Save it to disk with saveRDS() saveRDS(object = csv_data, file = "modified_feed_data.RDS")

Read it back in with readRDS() modified_feed_data <- readRDS(file = "modified_feed_data.RDS")

Using data from API clients

example 1 Load pageviews library for wikipedia

```
library(pageviews)

# Get the pageviews for "Hadley Wickham"
hadley_pageviews <- article_pageviews(project = "en.wikipedia", article = "Hadley Wickham")

# Examine the resulting object
str(hadley_pageviews)
```

```
## 'data.frame': 1 obs. of  8 variables:
##  $ project    : chr "wikipedia"
##  $ language   : chr "en"
##  $ article    : chr "Hadley_Wickham"
##  $ access     : chr "all-access"
##  $ agent      : chr "all-agents"
##  $ granularity: chr "daily"
##  $ date       : POSIXct, format: "2015-10-01"
##  $ views      : num 53
```

Load the httr package:

```
library(httr)

# Make a GET request to http://httpbin.org/get
get_result <- GET(url = "http://httpbin.org/get")

# Print it to inspect it
get_result
```

```
## Response [http://httpbin.org/get]
##   Date: 2020-09-19 13:40
##   Status: 200
##   Content-Type: application/json
##   Size: 365 B
## {
##   "args": {},
##   "headers": {
##     "Accept": "application/json, text/xml, application/xml, */*",
##     "Accept-Encoding": "deflate, gzip",
##     "Host": "httpbin.org",
```

```
##     "User-Agent": "libcurl/7.54.0 r-curl/4.3 httr/1.4.2",
##     "X-Amzn-Trace-Id": "Root=1-5f660a60-44335da95e01d75b0329c10d"
##   },
##   "origin": "99.229.26.120",
## ...
```

Make a POST request to http://httpbin.org/post with the body "this is a test"

```
post_result <- POST(url = "http://httpbin.org/post", body = "this is a test")

# Print it to inspect it
post_result
```

```
## Response [http://httpbin.org/post]
##   Date: 2020-09-19 13:40
##   Status: 200
##   Content-Type: application/json
##   Size: 472 B
## {
##   "args": {},
##   "data": "this is a test",
##   "files": {},
##   "form": {},
##   "headers": {
##     "Accept": "application/json, text/xml, application/xml, */*",
##     "Accept-Encoding": "deflate, gzip",
##     "Content-Length": "14",
##     "Host": "httpbin.org",
## ...
```

Make a GET request to url and save the results: Handling http failures

```
fake_url <- "http://google.com/fakepagethatdoesnotexist"

# Make the GET request
request_result <- GET(fake_url)
```

Example start to finish using httr package: The API url

```
base_url <- "https://en.wikipedia.org/w/api.php"

# Set query parameters
query_params <- list(action = "parse",
                     page = "Hadley Wickham",
                     format = "xml")
```

Read page contents as HTML: library(rvest) # Extract page name element from infobox: library(xml2) Create a dataframe for full name Reproducibility

```r
get_infobox <- function(title){
  base_url <- "https://en.wikipedia.org/w/api.php"

# Change "Hadley Wickham" to title

query_params <- list(action = "parse",
                      page = title,
                      format = "xml")}

resp <- GET(url = base_url, query = query_params)
resp_xml <- content(resp)

page_html <- read_html(xml_text(resp_xml))
infobox_element <- html_node(x = page_html, css =".infobox")
page_name <- html_node(x = infobox_element, css = ".fn")
```

Construct a directory-based API URL to `http://swapi.co/api`, looking for person `1` in `people`:

```r
directory_url <- paste("http://swapi.co/api", "people", "1", sep = "/")

# Make a GET call with it
result <- GET(directory_url)

# Create list with nationality and country elements
query_params <- list(nationality = "americans",
                    country = "antigua")

# Make parameter-based call to httpbin, with query_params
parameter_response <- GET("https://httpbin.org/get", query = query_params)

# Print parameter_response
parameter_response
```

```
## Response [https://httpbin.org/get?nationality=americans&country=antigua]
##   Date: 2020-09-19 13:40
##   Status: 200
##   Content-Type: application/json
##   Size: 465 B
## {
##   "args": {
##     "country": "antigua",
##     "nationality": "americans"
##   },
##   "headers": {
##     "Accept": "application/json, text/xml, application/xml, */*",
```

```
##     "Accept-Encoding": "deflate, gzip",
##     "Host": "httpbin.org",
##     "User-Agent": "libcurl/7.54.0 r-curl/4.3 httr/1.4.2",
## ...
```

Using user agents Informative user-agents are a good way of being respectful of the developers running the API you're interacting with. They make it easy for them to contact you in the event something goes wrong. I always try to include: My email address; A URL for the project the code is a part of, if it's got a URL.

Do not change the url:

```r
url <- "https://wikimedia.org/api/rest_v1/metrics/pageviews/per-article/en.wikipedia/al
```

Add the email address and the test sentence inside user_agent() server_response <- GET(url, user_agent("my@email.address this is a test"))

Rate-limiting The next stage of respectful API usage is rate-limiting: making sure you only make a certain number of requests to the server in a given time period. Your limit will vary from server to server, but the implementation is always pretty much the same and involves a call to Sys.sleep(). This function takes one argument, a number, which represents the number of seconds to "sleep" (pause) the R session for. So if you call Sys.sleep(15), it'll pause for 15 seconds before allowing further code to run.

Construct a vector of 2 URLs: for(url in urls){ Send a GET request to url result <- GET(url) Delay for 5 seconds between requests Sys.sleep(5) }

```r
urls <- c("http://httpbin.org/status/404",
          "http://httpbin.org/status/301")
```

Tying it all together:

```r
get_pageviews <- function(article_title){
  url <- paste(
    "https://wikimedia.org/api/rest_v1/metrics/pageviews/per-article/en.wikipedia/all-a
    article_title,
    "daily/2015100100/2015103100",
    sep = "/"
  )

response <- GET(url, user_agent("my@email.com this is a test"))
  # Is there an HTTP error?
  if(http_error(response)){
    # Throw an R error
    stop("the request failed")
  }
  # Return the response's content
  content(response)
```

```
}
```

working with JSON files (for more information see: www.json.org) While JSON is a useful format for sharing data, your first step will often be to parse it into an R object, so you can manipulate it with R.

web scraping 101 The first step with web scraping is actually reading the HTML in. This can be done with a function from xml2, which is imported by rvest - read_html(). This accepts a single URL, and returns a big blob of XML that we can use further on.

Hadley Wickham's Wikipedia page:

```
test_url <- "https://en.wikipedia.org/wiki/Hadley_Wickham"

# Read the URL stored as "test_url" with read_html()
test_xml <- read_html(test_url)

# Print test_xml
test_xml
```

```
## {html_document}
## <html class="client-nojs" lang="en" dir="ltr">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-subject  ...
```

html_node(), which extracts individual chunks of HTML from a HTML document. There are a couple of ways of identifying and filtering nodes, and for now we're going to use XPATHs: unique identifiers for individual pieces of a HTML document.

Extract the element of table_element referred to by second_xpath_val and store it as page_name page_name <- html_node(x = table_element, xpath = second_xpath_val)

Extract the text from page_name:

```
page_title <- html_text(page_name)

# Print page_title
page_title
```

```
## [1] "Hadley Wickham"
```

```
# Turn table_element into a data frame and assign it to wiki_table
# wiki_table <- rvest::html_table(table_element)

# Print wiki_table
# wiki_table
```

Cleaning a data frame Rename the columns of wiki_table:

CSS web scraping CSS is a way to add design information to HTML, that instructs the browser on how to display the content. You can leverage these design instructions to identify content on the page.

Chapter 3

Useful R Functions + Examples

This is *NOT* intended to be fully comprehensive list of every useful R function that exists, but is a practical demonstration of selected relevant examples presented in user-friendly format, all available in base R. For a wider collection to work through, this Reference Card is recommended: https://cran.r-project.org/doc/contrib/Baggott-refcard-v2.pdf

Additional CRAN reference cards and R guides (including non-English documentation) found here: https://cran.r-project.org/other-docs.html

3.1 Contents

A. Essentials
* 1. `getwd()`, `setwd()`
* 2. `?foo`, `help(foo)`, `example(foo)`
* 3. `install.packages("foo")`, `library("foo")`
* 4. `devtools::install_github("username/packagename")`
* 5. `data("foo")`
* 6. `read.csv`, `read.table`
* 7. `write.table()`
* 8. `save()`, `load()`

B. Basics
* 9. `c()`, `cbind()`, `rbind()`, `matrix()`
* 10. `length()`, `dim()`
* 11. `sort()`, `'vector'[]`, `'matrix'[]`

* 12. `data.frame()`, `class()`, `names()`, `str()`, `summary()`, `View()`, `head()`, `tail()`, `as.data.frame()`

C. Core
* 13. `df[order(),]`
* 14. `df[,c()]`, `df[which(),]`
* 15. `table()`
* 16. `mean()`, `median()`, `sd()`, `var()`, `sum()`, `min()`, `max()`, `range()`
* 17. `apply()`
* 18. `lapply()` using `list()`
* 19. `tapply()`

D. Common
* 20. `if` statement, `if...else` statement
* 21. `for` loop
* 22. `function()...`

3.2 R Syntax

REMEMBER: KEY R LANGUAGE SYNTAX

- **Case Sensitivity**: as per most UNIX-based packages, R is case sensitive, hence `X` and `x` are different symbols and would refer to different variables.

- **Expressions vs Assignments**: an expression, like `3 + 5` can be given as a command which will be evaluated and the value immediately printed, but not stored. An assignment however, like `sum <- 3 + 5` using the assignment operator `<-` also evaluates the expression `3 + 5`, but instead of printing and not storing, it stores the value in the object `sum` but doesn't print the result. The object `sum` would need to be called to print the result.

- **Reserved Words**: choice for naming objects is almost entirely free, except for these reserved words: https://stat.ethz.ch/R-manual/R-devel/library/base/html/Reserved.html

- **Spacing**: outside of the function structure, spaces don't matter, e.g. `3+5` is the same as `3+        5` is the same as `3 + 5`. For more best-practices for R code Hadley Wickham's Style Guide is a useful reference: http://adv-r.had.co.nz/Style.html
- **Comments**: add comments within your code using a hastag, `#`. R will ignore everything to the right of the hashtag within that line

3.3 Functional examples

1. Working Directory management

- `getwd()`, `setwd()` R/RStudio is always pointed at a specific directory on your computer, so it's important to be able to check what's the current directory using `getwd()`, and to be able to change and specify a different directory to work in using `setwd()`.

#check the directory R is currently pointed at getwd()

2. Bring up help documentation & examples

- `?foo`, `help(foo)`, `example(foo)`

```
?boxplot
help(boxplot)
example(boxplot)
```

---

3. Load & Call CRAN Packages

- `install.packages("foo")`, `library("foo")` Packages are add-on functionality built for R but not pre-installed (base R), hence you need to install/load the packages you want yourself. The majority of packages you'd want have been submitted to and are available via CRAN. At time of writing, the CRAN package repository featured 8,592 available packages.

4. Load & Call Packages from GitHub

- `devtools::install_github("username/packagename")` Not all packages you'll want will be available via CRAN, and you'll likely need to get certain packages from GitHub accounts. This example shows how to install the `shinyapps` package from RStudio's GitHub account.
- install.packages("devtools") #pre-requisite for `devtools...` function
- devtools::install_github("rstudio/shinyapps") #install specific package from specific GitHub account
- library("shinyapps") #Call package

5. Load datasets from base R & Loaded Packages

- `data("foo")`

```
#AIM: show available datasets
data()

#AIM: load an available dataset
data("iris")
```

---

6. I/O Loading Existing Local Data

- `read.csv`, `read.table`

(a) I/O When already in the working directory where the data is

Import a local **csv** file (i.e. where data is separated by **commas**), saving it as an object: - object <- read.csv("xxx.csv")

Import a local tab delimited file (i.e. where data is separated by **tabs**), saving it as an object: - object <- read.csv("xxx.csv", header = FALSE) —

(b) I/O When NOT in the working directory where the data is

For example to import and save a local **csv** file from a different working directory you either need to specify the file path (operating system specific), e.g.:

on a mac: - object <- read.csv("~/Desktop/R/data.csv")

on windows: = object <- read.csv("C:/Desktop/R/data.csv")

OR

You can use the file.choose() command which will interactively open up the file dialog box for you to browse and select the local file, e.g.: - object <- read.csv(file.choose())

(c) I/O Copying & Pasting Data

For relatively small amounts of data you can do an equivalent copy paste (operating system specific):

on a mac: - object <- read.table(pipe("pbpaste"))

on windows: - object <- read.table(file = "clipboard")

(d) I/O Loading Non-Numerical Data - character strings

Be careful when loading text data! R may assume character strings are statistical factor variables, e.g. "low", "medium", "high", when are just individual labels like names. To specify text data NOT to be converted into factor variables, add `stringsAsFactor = FALSE` to your `read.csv/read.table` command: - object <- read.table("xxx.txt", stringsAsFactors = FALSE)

(e) I/O Downloading Remote Data

For accessing files from the web you can use the same `read.csv/read.table` commands. However, the file being downloaded does need to be in an R-friendly format (maximum of 1 header row, subsequent rows are the equivalent of one data record per row, no extraneous footnotes etc.). Here is an example downloading an online csv file of coffee harvest data used in a Nature study: - object <- read.csv("http://sumsar.net/files/posts/2014-02-04-bayesian-first-aid-one-sample-t-test/roubik_2002_coffe_yield.csv")

7. I/O Exporting Data Frame

- `write.table()`

Navigate to the working directory you want to save the data table into, then run the command (in this case creating a tab delimited file): - write.table(object, "xxx.txt", sep = "")

8. I/O Saving Down & Loading Objects

- `save()`, `load()`

These two commands allow you to save a named R object to a file and restore that object again.
Navigate to the working directory you want to save the object in then run the command: - save(object, file = "xxx.rda")

reload the object: - load("xxx.rda")

9. Vector & Matrix Construction

- `c()`, `cbind()`, `rbind()`, `matrix()` Vectors (lists) & Matrices (two-dimensional arrays) are very common R data structures.

```
#use c() to construct a vector by concatenating data
foo <- c(1, 2, 3, 4) #example of a numeric vector
oof <- c("A", "B", "C", "D") #example of a character vector
ofo <- c(TRUE, FALSE, TRUE, TRUE) #example of a logical vector

#use cbind() & rbind() to construct matrices
coof <- cbind(foo, oof) #bind vectors in column concatenation to make a matrix
roof <- rbind(foo, oof) #bind vectors in row concatenation to make a matrix

#use matrix() to construct matrices
moof <- matrix(data = 1:12, nrow=3, ncol=4) #creates matrix by specifying set of values, no. of r
```

10. Vector & Matrix Explore

- `length()`, `dim()`

```
length(foo) #length of vector

dim(coof) #returns dimensions (no. of rows & columns) of vector/matrix/dataframe
```

11. Vector & Matrix Sort & Select

- `sort()`, `'vector'[]`, `'matrix'[]`

```
#create another numeric vector
jumble <- c(4, 1, 2, 3)
sort(jumble) #sorts a numeric vector in ascending order (default)
sort(jumble, decreasing = TRUE) #specify the decreasing arg to reverse default order

#create another character vector
mumble <- c( "D", "B", "C", "A")
```

```
sort(mumble) #sorts a character vector in alphabetical order (default)
sort(mumble, decreasing = TRUE) #specify the decreasing arg to reverse default order

jumble[1] #selects first value in our jumble vector
tail(jumble, n=1) #selects last value
jumble[c(1,3)] #selects the 1st & 3rd values
jumble[-c(1,3)] #selects everything except the 1st & 3rd values

coof[1,] #selects the 1st row of our coof matrix
coof[,1] #selects the 1st column
coof[2,1] #selects the value in the 2nd row, 1st column
coof[,"oof"] #selects the column named "oof"
coof[1:3,] #selects columns 1 to 3 inclusive
coof[c(1,2,3),] #selects the 1st, 2nd & 3rd rows (same as previous)
```

12. Create & Explore Data Frames

    - `data.frame()`, `class()`, `names()`, `str()`, `summary()`, `View()`, `head()`, `tail()`, `as.data.frame()` A data frame is a matrix-like data structure made up of lists of variables with the same number of rows, which can be of differing data types (numeric, character, factor etc.) - matrices must have columns all of the same data type.

```
#create a data frame with 3 columns with 4 rows each
doof <- data.frame("V1"=1:4, "V2"=c("A","B","C","D"), "V3"=5:8)

class(doof) #check data frame object class
names(doof) # returns column names
str(doof) #see structure of data frame
summary(doof) #returns basic summary stats
View(doof) #invokes spreadsheet-style viewer
head(doof, n=2) #shows first parts of object, here requesting the first 2 rows
tail(doof, n=2) #shows last parts of object, here requesting the last 2 rows

convert <- as.data.frame(coof) #convert a non-data frame object into a data frame
```

13. Data Frame Sort

    - `df[order(),]`

```
#use 'painters' data frame
library("MASS") #call package with the required data
data("painters") #load required data
View(painters) #scan dataset

#syntax for using a specific variable: df=data frame, '$', V1=variable name
df$V1
```

```
#AIM: print the 'School' variable column
painters$School

#syntax for df[order(),]
df[order(df$V1, df$V2...),] #function arguments: df=data frame, in square brackets specify within

#AIM: order the dataset rows based on the painters' Composition Score column, in Ascending order
painters[order(painters$Composition),] #Composition is the sorting variable

#AIM: order the dataset rows based on the painters' Composition Score column, in Descending order
painters[order(-painters$Composition),] #append a minus sign in front of the variable you want to

#AIM: order the dataset rows based on the painters' Composition Score column, in Descending order
painters[order(-painters$Composition), c(1:3)]
```

14. Data Frame Select & Deselect

- `df[,c()]`, `df[which(),]`

```
#use 'painters' data frame

#syntax for select & deselect based on column variables
df[, c("V1", "V2"...)] #function arguments: df=data frame, in square brackets specify columns to

#AIM: select the Composition & Drawing variables based on their column name
painters[, c("Composition", "Drawing")] #subset the df, selecting just the named columns (and all

#AIM: select the Composition & Drawing variables based on their column positions in the painters
painters[, c(1,2)] #subset the df, selecting just the 1st & 2nd columns (and all the rows)

#AIM: drop the Expression variable based on it's column position in the painters data frame and r
painters[c(1:5), -4] #returns the subsetted df having deselected the 4th column, Expression and t


#syntax for select & deselect based on row variable values
df[which(),] #df=data frame, specify the variable value within the `which()` to subset the df on.

#AIM: select all rows where the painters' School is the 'A' category
painters[which(painters$School == "A"),] #returns the subsetted df where equality holds true, i.e

#AIM: deselect all rows where the painters' School is the 'A' category, i.e. return df subset wit
painters[which(painters$School != "A" & painters$Colour > 10),] #returns the subsetted df where e
```

15. Data Frame Frequency Calculations

- `table()`

```
#create new data frame
flavour <- c("choc", "strawberry", "vanilla", "choc", "strawberry", "strawberry")
gender <- c("F", "F", "M", "M", "F", "M")
icecream <- data.frame(flavour, gender) #icecream df made up of 2 factor variables, fl

#AIM: create a frequency distribution table which shows the count of each gender in th
table(icecream$gender)

#AIM: create a frequency distribution table which shows the count of each flavour in th
table(icecream$flavour)

#AIM: create Contingency/2-Way Table showing the counts for each combination of flavou
table(icecream$flavour, icecream$gender)
```

16. Descriptive/Summary Stats Functions

- `mean()`, `median()`, `sd()`, `var()`, `sum()`, `min()`, `max()`, `range()`

```
#re-using the jumble vector from before
jumble <- c(4, 1, 2, 3)

mean(jumble)
median(jumble)
sd(jumble)
var(jumble)
sum(jumble)
min(jumble)
max(jumble)
range(jumble)
```

17. Apply Functions

- `apply()` `apply()` returns a vector, array or list of values where a specified function has been applied to the 'margins' (rows/cols combo) of the original vector/array/list.

```
#re-using the moof matrix from before
moof <- matrix(data = 1:12, nrow=3, ncol=4)

#apply syntax
apply(X, MARGIN, FUN,...) #function arguments: X=an array, MARGIN=1 to apply to rows/2

#AIM: using the moof matrix, apply the sum function to the rows
apply(moof, 1, sum)

#AIM: using the moof matrix, apply the sum function to the columns
apply(moof, 2, sum)
```

18. Apply Functions

- `lapply()` using `list()` A list, a common data structure, is a generic vector containing objects of any types. `lapply()` returns a list where each element returned is the result of applying a specified function to the objects in the list.

```
#create list of various vectors and matrices
bundle <- list(moof, jumble, foo)

#lapply syntax
lapply(X, FUN,...) #function arguments: X=a list, FUN=function to apply

#AIM: using the bundle list, apply the mean function to each object in the list
lapply(bundle, mean)
```

19. Apply Functions

- `tapply()` `tapply()` applies a specified function to specified groups/subsets of a factor variable.

```
#tapply syntax
tapply(X, INDEX, FUN,...) #function arguments: X=an atomic object, INDEX=list of 1+ factors of X

#AIM: calculate the mean Drawing Score of the painters, but grouped by School category
tapply(painters$Drawing, painters$School, mean) #grouping the data by the 8 different Schools, ap
```

20. Programming Tools

- `if` statement, `if...else` statement An `if` statement is used when certain computations are conditional and only execute when a specific condition is met - if the condition is not met, nothing executes. The `if...else` statement extends the `if` statement by adding on a computation to execute when the condition is not met, i.e. the 'else' part of the statement.

```
#if-statement syntax
if ('test expression')
    {
    'statement'
    }

#if...else statement
if ('test expression')
    {
    'statement'
    }else{
    'another statement'
    }
```

```r
#AIM: here we want to test if the object, 'condition_to_test' is smaller than 10. If i

#specify the 'test expression'
condition_to_test <- 7

#write your 'if...else' function based on a 'statement' or 'another statement' depende
if (condition_to_test > 5)
    {
    result_after_test = 'Above Average'
    }else{
    result_after_test = 'Below Average'
    }

#call the resulting 'statement' as per the instruction of the 'if...else' statement
result_after_test
```

21. Programming Tools

- `for` loop A `for` loop is an automation method for repeating (looping) a specific set of instructions for each element in a vector.

```r
#for loop syntax requires a counter, often called 'i' to denote an index
for ('counter' in 'looping vector')
    {
    'instructions'
    }

#AIM: here we want to print the phrase "In the Year yyyy" 6x, once for each year betwe
#this for loop executes the code chunk 'print(past("In the Year", i)) for each of the
for (i in 2010:2015)
    {
    print(paste("In the Year", i))
    }

#AIM: create an object which contains 10 items, namely each number between 1 and 10 sq
#to store rather than just print results, an empty storage container needs to be creat
container <- NULL
for (i in 1:10)
    {
    container[i] = i^2
    }

container #check results: the loop is instructed to square every element of the loopin
```

22. Programming Tools

- `function()...` User-programmed functions allow you to specify customised arguments and returned values.

```
#AIM: to create a simplified take-home pay calculator (single-band), called 'takehome_pay'. Our f
takehome_pay <- function(tax_rate, income)
    {
    tax = tax_rate * income
    return(income - tax)
    }

takehome_pay(tax_rate = 0.2, income = 25000) #call our function to calculate 'takehome_pay' on a
```

23. Strings

- `grep()`, `tolower()`, `nchar()`

24. Further Data Selection

- `quantile()`, `cut()`, `which()`, `na.omit()`, `complete.cases()`, `sample()`

25. Further Data Creation

- `seq()`, `rep()`

26. Other Apply-related functions

- `split()`, `sapply()`, `aggregate()`

27. More Loops

- `while` loop, `repeat` loop

…..Ad Infinitum!!

Chapter 4

Demo for dplyr

```
# Load data and dependencies:
library(dplyr)

data(iris)
```

Explore the iris data

```
head(iris)
pairs(iris)
str(iris)
summary(iris)
```

A. **Select**: keeps only the variables you mention

```
select(iris, 1:3)
select(iris, Petal.Width, Species)
select(iris, contains("Petal.Width"))
select(iris, starts_with("Species"))
```

B. **Arrange**: sort a variable in descending order

```
arrange(iris, Sepal.Length)
arrange(iris, desc(Sepal.Length))
arrange(iris, Sepal.Length, desc(Sepal.Width))
```

C. **Filter**: find rows/cases where conditions are true Note: rows where the condition evaluates to NA are dropped

```
filter(iris, Petal.Length > 5)
filter(iris, Petal.Length > 5 & Species == "setosa")
filter(iris, Petal.Length > 5, Species == "setosa") #the comma is a shorthand for &
filter(iris, !Species == "setosa")
```

D. **Pipe Example with MaggriteR** (ref: Rene Magritte This is not a pipe) The long Way, before nesting or multiple variables

```
data1 <- filter(iris, Petal.Length > 6)
data2 <- select(data1, Petal.Length, Species)
```

With **DPLYR**:

```
select(
  filter(iris, Petal.Length > 6),
  Petal.Length, Species) %>%
  head()
```

```
##   Petal.Length   Species
## 1          6.6 virginica
## 2          6.3 virginica
## 3          6.1 virginica
## 4          6.7 virginica
## 5          6.9 virginica
## 6          6.7 virginica
```

Using pipes with the data variable

```
iris %>%
  filter(Petal.Length > 6) %>%
  select(Petal.Length, Species) %>%
  head()
```

```
##   Petal.Length   Species
## 1          6.6 virginica
## 2          6.3 virginica
## 3          6.1 virginica
## 4          6.7 virginica
## 5          6.9 virginica
## 6          6.7 virginica
```

Using the . to specify where the incoming variable will be piped to: - myFunction(arg1, arg2 = .)

```
iris %>%
  filter(., Species == "versicolor")
```

Other magrittr examples:

```
iris %>%
  filter(Petal.Length > 2.0) %>%
  select(1:3)

iris %>%
  select(contains("Width")) %>%
```

```
  arrange(Petal.Width) %>%
  head()

iris %>%
  filter(Petal.Width == "versicolor") %>%
  arrange(desc(Sepal.Width))

iris %>%
  filter(Sepal.Width > 1) %>%
  View()

iris %>%
  filter(Petal.Width  == 0.1) %>%
  select(Sepal.Width) %>%
  unique()
```

a second way to get the unique values:

```
iris %>%
  filter(Petal.Width  == 0.1) %>%
  distinct(Sepal.Width)
```

```
##   Sepal.Width
## 1         3.1
## 2         3.0
## 3         4.1
## 4         3.6
```

E. **Mutate**: adds new variables and preserves existing; transmute() drops existing variables

```
iris %>%
  mutate(highSpecies = Sepal.Width > 6) %>%
  head()


iris %>%
  mutate(size = Sepal.Width + Petal.Width) %>%
  head()


iris %>%
  mutate(MeanPetal.Width = mean(Petal.Width, na.rm = TRUE),
         greaterThanMeanPetal.Width = ifelse(Petal.Width > MeanPetal.Width, 1, 0)) %>%
  head()
```

```
iris %>%
  mutate(buckets = cut(Petal.Width, 3)) %>%
  head()


iris %>%
  mutate(Petal.WidthBuckets = case_when(Petal.Width < 1 ~ "Low",
                                Petal.Width >= 2 & Sepal.Width < 3 ~ "Med",
                                Petal.Width >= 4 ~ "High")) %>%
  head()
```

E. **Group_by and Summarise**: used on grouped data created by group_by(). The output will have one row for each group.

```
iris %>%
  summarise(Petal.WidthMean = mean(Petal.Width),
            Petal.WidthSD = sd(Petal.Width))

iris %>%
  group_by(Petal.Width) %>%
  mutate(Petal.WidthMean = mean(Petal.Width))

iris %>%
  group_by(Petal.Width) %>%
  summarise(Petal.WidthMean = mean(Petal.Width))

iris %>%
  group_by(Petal.Width, Species) %>%
  summarise(count = n())
```

F. **Slice**: Slice does not work with relational databases because they have no intrinsic notion of row order. If you want to perform the equivalent operation, use filter() and row_number().

```
iris %>%
  slice(2:4) %>%
  head()
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          4.9         3.0          1.4         0.2  setosa
## 2          4.7         3.2          1.3         0.2  setosa
## 3          4.6         3.1          1.5         0.2  setosa
```

Other verbs within DPLYR: **Scoped verbs**

```
# ungroup
iris %>%
  group_by(Petal.Width, Species) %>%
```

```
  summarise(count = n()) %>%
  ungroup()

# Summarise_all
iris %>%
  select(1:4) %>%
  summarise_all(mean)


iris %>%
  select(1:4) %>%
  summarise_all(funs(mean, min))


iris %>%
  summarise_all(~length(unique(.)))

# summarise_at
iris %>%
  summarise_at(vars(-Petal.Width), mean)


iris %>%
  summarise_at(vars(contains("Petal.Width")), funs(mean, min))

# summarise_if
iris %>%
  summarise_if(is.numeric, mean)

iris %>%
  summarise_if(is.factor, ~length(unique(.)))

# other verbs:
iris %>%
  mutate_if(is.factor, as.character) %>%
  str()


iris %>%
  mutate_at(vars(contains("Width")), ~ round(.))


iris %>%
  filter_all(any_vars(is.na(.)))
```

```
iris %>%
  filter_all(all_vars(is.na(.)))

# Rename
iris %>%
  rename("sp" = "Species") %>%
  head()

# And finally: make a test and save
test <- iris %>%
  group_by(Petal.Width) %>%
  summarise(MeanPetal.Width = mean(Petal.Width))
```

Chapter 5

Demo for Data.table

Load libraries:

```
# Load data.table
library(data.table)
library(bikeshare14)
library(tidyverse)
```

Create the data.table:

```
X <- data.table(id = c("a", "b", "c"), value = c(0.5, 1.0, 1.5))

print(X)
```

```
##    id value
## 1:  a   0.5
## 2:  b   1.0
## 3:  c   1.5
```

Get number of columns in batrips:

```
batrips <- as.data.table(batrips)
col_number <- ncol(batrips)
col_number
```

```
## [1] 11
```

Print the first 4 rows:

```
head(batrips, 4)
```

```
##    trip_id duration          start_date          start_station start_terminal
## 1:  139545      435 2014-01-01 00:14:00 San Francisco City Hall             58
## 2:  139546      432 2014-01-01 00:14:00 San Francisco City Hall             58
```

```
## 3:  139547     1523 2014-01-01 00:17:00  Embarcadero at Sansome             60
## 4:  139549     1620 2014-01-01 00:23:00       Steuart at Market             74
##                end_date        end_station end_terminal bike_id
## 1: 2014-01-01 00:21:00    Townsend at 7th           65     473
## 2: 2014-01-01 00:21:00    Townsend at 7th           65     395
## 3: 2014-01-01 00:42:00    Beale at Market           56     331
## 4: 2014-01-01 00:50:00 Powell Street BART           39     605
##    subscription_type zip_code
## 1:        Subscriber    94612
## 2:        Subscriber    94107
## 3:        Subscriber    94112
## 4:          Customer    92007
```

Print the last 4 rows:

```
tail(batrips, 4)
```

```
##    trip_id duration          start_date                   start_station
## 1:  588911      422 2014-12-31 23:19:00 Grant Avenue at Columbus Avenue
## 2:  588912     1487 2014-12-31 23:31:00        South Van Ness at Market
## 3:  588913     1458 2014-12-31 23:32:00        South Van Ness at Market
## 4:  588914      364 2014-12-31 23:33:00           Embarcadero at Bryant
##    start_terminal            end_date
## 1:             73 2014-12-31 23:26:00
## 2:             66 2014-12-31 23:56:00
## 3:             66 2014-12-31 23:56:00
## 4:             54 2014-12-31 23:40:00
##                                  end_station end_terminal bike_id
## 1: Yerba Buena Center of the Arts (3rd @ Howard)           68     604
## 2:                           Steuart at Market           74     480
## 3:                           Steuart at Market           74     277
## 4:                               Howard at 2nd           63      56
##    subscription_type zip_code
## 1:        Subscriber    94133
## 2:          Customer    94109
## 3:          Customer    94109
## 4:        Subscriber    94105
```

Print the structure of batrips:

```
str(batrips)
```

```
## Classes 'data.table' and 'data.frame': 326339 obs. of  11 variables:
##  $ trip_id         : int  139545 139546 139547 139549 139550 139551 139552 139553 1
##  $ duration        : int  435 432 1523 1620 1617 779 784 721 624 574 ...
##  $ start_date      : POSIXct, format: "2014-01-01 00:14:00" "2014-01-01 00:14:00"
##  $ start_station   : chr  "San Francisco City Hall" "San Francisco City Hall" "Emba
##  $ start_terminal  : int  58 58 60 74 74 74 74 74 57 57 ...
```

```
##  $ end_date         : POSIXct, format: "2014-01-01 00:21:00" "2014-01-01 00:21:00" ...
##  $ end_station      : chr  "Townsend at 7th" "Townsend at 7th" "Beale at Market" "Powell Stree
##  $ end_terminal     : int  65 65 56 39 39 46 46 46 68 68 ...
##  $ bike_id          : int  473 395 331 605 453 335 580 563 358 365 ...
##  $ subscription_type: chr  "Subscriber" "Subscriber" "Subscriber" "Customer" ...
##  $ zip_code         : chr  "94612" "94107" "94112" "92007" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Filter third row:

```
row_3 <- batrips[3]
row_3 %>%
  head(3)
```

```
##    trip_id duration          start_date          start_station start_terminal
## 1:  139547     1523 2014-01-01 00:17:00 Embarcadero at Sansome             60
##                end_date     end_station end_terminal bike_id subscription_type
## 1: 2014-01-01 00:42:00 Beale at Market           56     331        Subscriber
##    zip_code
## 1:    94112
```

Filter rows 1 through 2:

```
rows_1_2 <- batrips[1:2]
rows_1_2 %>%
  head(2)
```

```
##    trip_id duration          start_date           start_station start_terminal
## 1:  139545      435 2014-01-01 00:14:00 San Francisco City Hall             58
## 2:  139546      432 2014-01-01 00:14:00 San Francisco City Hall             58
##                end_date     end_station end_terminal bike_id subscription_type
## 1: 2014-01-01 00:21:00 Townsend at 7th           65     473        Subscriber
## 2: 2014-01-01 00:21:00 Townsend at 7th           65     395        Subscriber
##    zip_code
## 1:    94612
## 2:    94107
```

Filter the 1st, 6th and 10th rows:

```
rows_1_6_10 <- batrips[c(1, 6, 10)]
rows_1_6_10 %>%
  head()
```

```
##    trip_id duration          start_date           start_station start_terminal
## 1:  139545      435 2014-01-01 00:14:00 San Francisco City Hall             58
## 2:  139551      779 2014-01-01 00:24:00       Steuart at Market             74
## 3:  139555      574 2014-01-01 00:25:00           5th at Howard             57
##                end_date                                        end_station
## 1: 2014-01-01 00:21:00                                    Townsend at 7th
## 2: 2014-01-01 00:37:00                              Washington at Kearney
```

```
## 3: 2014-01-01 00:35:00 Yerba Buena Center of the Arts (3rd @ Howard)
##    end_terminal bike_id subscription_type zip_code
## 1:           65     473        Subscriber    94612
## 2:           46     335          Customer    94109
## 3:           68     365          Customer    94941
```

Select all rows except the first two:

```
not_first_two <- batrips[-(1:2)]
not_first_two %>%
  head(2)
```

```
##    trip_id duration          start_date          start_station start_terminal
## 1:  139547     1523 2014-01-01 00:17:00 Embarcadero at Sansome             60
## 2:  139549     1620 2014-01-01 00:23:00      Steuart at Market             74
##               end_date        end_station end_terminal bike_id
## 1: 2014-01-01 00:42:00    Beale at Market           56     331
## 2: 2014-01-01 00:50:00 Powell Street BART           39     605
##    subscription_type zip_code
## 1:        Subscriber    94112
## 2:          Customer    92007
```

Select all rows except 1 through 5 and 10 through 15:

```
exclude_some <- batrips[-c(1:5, 10:15)]
exclude_some %>%
  head(2)
```

```
##    trip_id duration          start_date     start_station start_terminal
## 1:  139551      779 2014-01-01 00:24:00 Steuart at Market             74
## 2:  139552      784 2014-01-01 00:24:00 Steuart at Market             74
##               end_date           end_station end_terminal bike_id
## 1: 2014-01-01 00:37:00 Washington at Kearney           46     335
## 2: 2014-01-01 00:37:00 Washington at Kearney           46     580
##    subscription_type zip_code
## 1:          Customer    94109
## 2:          Customer
```

Select all rows except the first and last:

```
not_first_last <- batrips[-c(1, .N)]
# Or
# batrips[-c(1, nrow(batrips))]

not_first_last %>%
  head(2)
```

```
##    trip_id duration          start_date           start_station start_terminal
## 1:  139546      432 2014-01-01 00:14:00 San Francisco City Hall             58
## 2:  139547     1523 2014-01-01 00:17:00  Embarcadero at Sansome             60
```

```
##                end_date     end_station end_terminal bike_id subscription_type
## 1: 2014-01-01 00:21:00 Townsend at 7th           65     395        Subscriber
## 2: 2014-01-01 00:42:00 Beale at Market           56     331        Subscriber
##    zip_code
## 1:    94107
## 2:    94112
```

Filter all rows where start_station is "Market at 10th":

```
trips_mlk <- batrips[start_station == "Market at 10th"]
trips_mlk %>%
  head(2)
```

```
##    trip_id duration          start_date  start_station start_terminal
## 1:  139605     1352 2014-01-01 07:40:00 Market at 10th             67
## 2:  139609     1130 2014-01-01 08:08:00 Market at 10th             67
##                end_date    end_station end_terminal bike_id subscription_type
## 1: 2014-01-01 08:03:00 Market at 10th           67     545        Subscriber
## 2: 2014-01-01 08:27:00 Market at 10th           67     545        Subscriber
##    zip_code
## 1:    94590
## 2:    94590
```

Filter all rows where start_station is "MLK Library" AND duration > 1600:

```
trips_mlk_1600 <- batrips[start_station == "MLK Library" & duration > 1600]
trips_mlk_1600 %>%
  head(2)
```

```
##    trip_id duration          start_date start_station start_terminal
## 1:  147733     1744 2014-01-09 11:47:00   MLK Library             11
## 2:  158900    61848 2014-01-19 16:42:00   MLK Library             11
##                end_date           end_station end_terminal bike_id
## 1: 2014-01-09 12:16:00    San Jose City Hall           10     691
## 2: 2014-01-20 09:52:00 San Jose Civic Center            3      86
##    subscription_type zip_code
## 1:        Subscriber    95112
## 2:          Customer    95608
```

Filter all rows where `subscription_type` is not `"Subscriber"`::

```
customers <- batrips[subscription_type != "Subscriber"]
customers %>%
  head(2)
```

```
##    trip_id duration          start_date     start_station start_terminal
## 1:  139549     1620 2014-01-01 00:23:00 Steuart at Market             74
## 2:  139550     1617 2014-01-01 00:23:00 Steuart at Market             74
##                end_date        end_station end_terminal bike_id
## 1: 2014-01-01 00:50:00 Powell Street BART           39     605
```

```
## 2: 2014-01-01 00:50:00 Powell Street BART           39     453
##    subscription_type zip_code
## 1:          Customer    92007
## 2:          Customer    92007
```

Filter all rows where start_station is "Ryland Park" AND subscription_type is not "Customer":

```
ryland_park_subscribers <- batrips[start_station == "Ryland Park" & subscription_type
ryland_park_subscribers %>%
  head(2)
```

```
##    trip_id duration          start_date start_station start_terminal
## 1:  243456      330 2014-04-10 09:10:00   Ryland Park             84
## 2:  244497      594 2014-04-11 07:28:00   Ryland Park             84
##                end_date                      end_station end_terminal bike_id
## 1: 2014-04-10 09:16:00                        Japantown            9      23
## 2: 2014-04-11 07:38:00 San Jose Diridon Caltrain Station            2      54
##    subscription_type zip_code
## 1:        Subscriber    95110
## 2:        Subscriber    95110
```

Filter all rows where end_station contains "Market":

```
any_markets <- batrips[end_station %like% "Market"]
any_markets %>%
  head(2)
```

```
## Empty data.table (0 rows and 11 cols): trip_id,duration,start_date,start_station,sta
```

Filter all rows where trip_id is 588841, 139560, or 139562:

```
filter_trip_ids <- batrips[trip_id %in% c(588841, 139560, 139562)]
filter_trip_ids %>%
  head(2)
```

```
##    trip_id duration          start_date     start_station start_terminal
## 1:  139560     3793 2014-01-01 00:32:00 Steuart at Market             74
## 2:  139562     3626 2014-01-01 00:33:00 Steuart at Market             74
##                end_date       end_station end_terminal bike_id subscription_type
## 1: 2014-01-01 01:35:00 Steuart at Market           74     311          Customer
## 2: 2014-01-01 01:33:00 Steuart at Market           74     271          Customer
##    zip_code
## 1:    55417
## 2:    94070
```

Filter all rows where duration is between [5000, 6000]:

```
duration_5k_6k <- batrips[duration %between% c(5000, 6000)]
duration_5k_6k %>%
```

```r
  head(2)
```

```
##    trip_id duration          start_date     start_station start_terminal
## 1:  139607     5987 2014-01-01 07:57:00 Market at Sansome             77
## 2:  139608     5974 2014-01-01 07:57:00 Market at Sansome             77
##               end_date                     end_station end_terminal bike_id
## 1: 2014-01-01 09:37:00 Grant Avenue at Columbus Avenue           73     591
## 2: 2014-01-01 09:37:00 Grant Avenue at Columbus Avenue           73     596
##    subscription_type zip_code
## 1:          Customer    75201
## 2:          Customer    75201
```

Filter all rows with specific start stations:

```r
two_stations <- batrips[start_station %chin% c("San Francisco City Hall", "Embarcadero at Sansome
two_stations %>%
  head(2)
```

```
##    trip_id duration          start_date           start_station start_terminal
## 1:  139545      435 2014-01-01 00:14:00 San Francisco City Hall             58
## 2:  139546      432 2014-01-01 00:14:00 San Francisco City Hall             58
##               end_date     end_station end_terminal bike_id subscription_type
## 1: 2014-01-01 00:21:00 Townsend at 7th           65     473        Subscriber
## 2: 2014-01-01 00:21:00 Townsend at 7th           65     395        Subscriber
##    zip_code
## 1:    94612
## 2:    94107
```

Selecting columns from a data.table Select bike_id and trip_id using a character vector:

```r
df_way <- batrips[, c("bike_id", "trip_id")]
df_way %>%
  head(2)
```

```
##    bike_id trip_id
## 1:     473  139545
## 2:     395  139546
```

Select start_station and end_station cols without a character vector:

```r
dt_way <- batrips[, .(start_station, end_station)]
dt_way %>%
  head(2)
```

```
##               start_station     end_station
## 1: San Francisco City Hall Townsend at 7th
## 2: San Francisco City Hall Townsend at 7th
```

Deselect start_terminal and end_terminal columns:

```
drop_terminal_cols <- batrips[, !c("start_terminal", "end_terminal")]
drop_terminal_cols %>%
  head(2)
```

```
##    trip_id duration          start_date           start_station
## 1:  139545      435 2014-01-01 00:14:00 San Francisco City Hall
## 2:  139546      432 2014-01-01 00:14:00 San Francisco City Hall
##               end_date     end_station bike_id subscription_type zip_code
## 1: 2014-01-01 00:21:00 Townsend at 7th     473        Subscriber    94612
## 2: 2014-01-01 00:21:00 Townsend at 7th     395        Subscriber    94107
```

Calculate median duration using the j argument:

```
median_duration <- batrips[, median(duration)]
median_duration %>%
  head()
```

```
## [1] 511
```

Get median duration after filtering:

```
median_duration_filter <- batrips[end_station == "Market at 10th" & subscription_type =
median_duration_filter %>%
  head()
```

```
## [1] 651
```

Compute duration of all trips:

```
trip_duration <- batrips[, difftime(end_date, start_date, units = "min")]
head(trip_duration) %>%
  head(2)
```

```
## Time differences in mins
## [1] 7 7
```

Have the column mean_durn:

```
mean_duration <- batrips[, .(mean_durn = mean(duration))]
mean_duration %>%
  head(2)
```

```
##    mean_durn
## 1:  1131.967
```

Get the min and max duration values:

```
min_max_duration <- batrips[, .(min(duration), max(duration))]
min_max_duration %>%
  head(2)
```

```
##    V1       V2
```

```
## 1: 60 17270400
```

Calculate the number of unique values:

```
other_stats <- batrips[, .(mean_duration = mean(duration),
                           last_ride = max(end_date))]
other_stats %>%
  head(2)
```

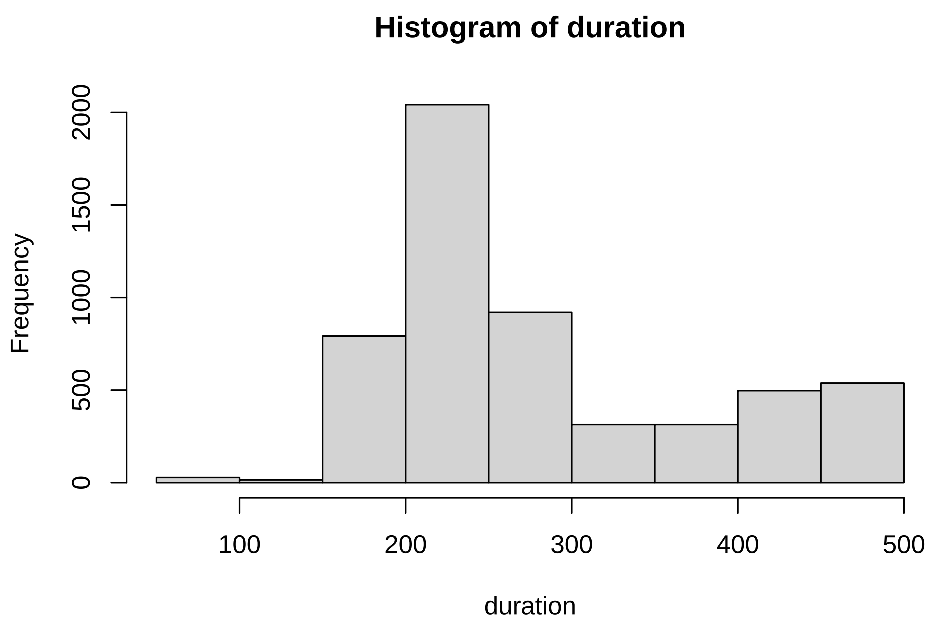
```
##    mean_duration           last_ride
## 1:      1131.967 2015-06-24 20:18:00
```

```
duration_stats <- batrips[start_station == "Townsend at 7th" & duration < 500,
                          .(min_dur = min(duration),
                            max_dur = max(duration))]
duration_stats
```

```
##    min_dur max_dur
## 1:      62     499
```

Plot the histogram of duration based on conditions:

```
batrips[start_station == "Townsend at 7th" & duration < 500, hist(duration)]
```




```
## $breaks
##  [1]  50 100 150 200 250 300 350 400 450 500
##
## $counts
## [1]   28   15  792 2042  920  314  314  497  538
```

```
##
## $density
## [1] 1.025641e-04 5.494505e-05 2.901099e-03 7.479853e-03 3.369963e-03
## [6] 1.150183e-03 1.150183e-03 1.820513e-03 1.970696e-03
##
## $mids
## [1]  75 125 175 225 275 325 375 425 475
##
## $xname
## [1] "duration"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

Computations by groups Compute the mean duration for every start_station:

```
mean_start_stn <- batrips[, .(mean_duration = mean(duration)), by = start_station]
mean_start_stn %>%
  head(2)
```

```
##              start_station mean_duration
## 1: San Francisco City Hall      1893.936
## 2:  Embarcadero at Sansome      1418.182
```

Compute the mean duration for every start and end station:

```
mean_station <- batrips[, .(mean_duration = mean(duration)), by = .(start_station, end_
mean_station %>%
  head(2)
```

```
##              start_station     end_station mean_duration
## 1: San Francisco City Hall Townsend at 7th      678.6364
## 2:  Embarcadero at Sansome Beale at Market      651.2367
```

Compute the mean duration grouped by start_station and month:

```
mean_start_station <- batrips[, .(mean_duration = mean(duration)), by = .(start_statio
mean_start_station %>%
  head(2)
```

```
##              start_station month mean_duration
## 1: San Francisco City Hall     1     1548.2591
## 2:  Embarcadero at Sansome     1      952.1756
```

Compute mean of duration and total trips grouped by start and end stations:

```r
aggregate_mean_trips <- batrips[, .(mean_duration = mean(duration),
                                    total_trips = .N),
                                by = .(start_station, end_station)]
aggregate_mean_trips %>%
  head(2)
```

```
##                start_station     end_station mean_duration total_trips
## 1: San Francisco City Hall Townsend at 7th      678.6364         121
## 2:  Embarcadero at Sansome Beale at Market      651.2367         545
```

Compute min and max duration grouped by start station, end station, and month:

```r
aggregate_min_max <- batrips[, .(min_duration = min(duration),
                                 max_duration = max(duration)),
                             by = .(start_station, end_station,
                                    month(start_date))]
aggregate_min_max %>%
  head(2)
```

```
##                start_station     end_station month min_duration max_duration
## 1: San Francisco City Hall Townsend at 7th     1          370          661
## 2:  Embarcadero at Sansome Beale at Market     1          345         1674
```

Chaining data.table expressions: Compute the total trips grouped by start_station and end_station

```r
trips_dec <- batrips[, .N, by = .(start_station,
                                  end_station)]
trips_dec %>%
  head(2)
```

```
##                start_station     end_station   N
## 1: San Francisco City Hall Townsend at 7th 121
## 2:  Embarcadero at Sansome Beale at Market 545
```

Arrange the total trips grouped by start_station and end_station in decreasing order:

```r
trips_dec <- batrips[, .N, by = .(start_station,
                                  end_station)][order(-N)]
trips_dec %>%
  head(2)
```

```
##                                start_station
## 1:                           Townsend at 7th
## 2: San Francisco Caltrain 2 (330 Townsend)
##                                 end_station    N
## 1: San Francisco Caltrain (Townsend at 4th) 3158
```

```
## 2:                              Townsend at 7th 2937
```

Top five most popular destinations:

```
top_5 <- batrips[, .N, by = end_station][order(-N)][1:5]
top_5
```

```
##                                  end_station     N
## 1: San Francisco Caltrain (Townsend at 4th) 33213
## 2:     Harry Bridges Plaza (Ferry Building) 15692
## 3:  San Francisco Caltrain 2 (330 Townsend) 15333
## 4:                         Market at Sansome 14816
## 5:                           2nd at Townsend 14064
```

Compute most popular end station for every start station:

```
popular_end_station <- trips_dec[, .(end_station = end_station[1]),
                                 by = start_station]
popular_end_station %>%
  head(2)
```

```
##                               start_station
## 1:                          Townsend at 7th
## 2: San Francisco Caltrain 2 (330 Townsend)
##                                  end_station
## 1: San Francisco Caltrain (Townsend at 4th)
## 2:                           Townsend at 7th
```

Find the first and last ride for each start_station:

```
first_last <- batrips[order(start_date),
                      .(start_date = start_date[c(1, .N)]),
                      by = start_station]
first_last
```

```
##                        start_station          start_date
##   1:         San Francisco City Hall 2014-01-01 00:14:00
##   2:         San Francisco City Hall 2014-12-31 22:06:00
##   3:          Embarcadero at Sansome 2014-01-01 00:17:00
##   4:          Embarcadero at Sansome 2014-12-31 22:08:00
##   5:               Steuart at Market 2014-01-01 00:23:00
##  ---
## 144: Santa Clara County Civic Center 2014-12-31 15:32:00
## 145:                     Ryland Park 2014-04-10 09:10:00
## 146:                     Ryland Park 2014-12-31 07:56:00
## 147:        Stanford in Redwood City 2014-09-03 19:41:00
## 148:        Stanford in Redwood City 2014-12-22 16:56:00
```

Using .SD (I)

```r
relevant_cols <- c("start_station", "end_station",
                   "start_date", "end_date", "duration")
```

Find the row corresponding to the shortest trip per month:

```r
shortest <- batrips[, .SD[which.min(duration)],
                    by = month(start_date),
                    .SDcols = relevant_cols]
shortest %>%
  head(2)
```

```
##    month                         start_station
## 1:     1                      2nd at Townsend
## 2:     2 San Francisco Caltrain (Townsend at 4th)
##                                end_station          start_date
## 1:                         2nd at Townsend 2014-01-21 13:01:00
## 2: San Francisco Caltrain (Townsend at 4th) 2014-02-08 14:28:00
##               end_date duration
## 1: 2014-01-21 13:02:00       60
## 2: 2014-02-08 14:29:00       61
```

Using .SD (II) Find the total number of unique start stations and zip codes per month:

```r
unique_station_month <- batrips[, lapply(.SD, uniqueN),
                                by = month(start_date),
                                .SDcols = c("start_station", "zip_code")]
unique_station_month %>%
  head(2)
```

```
##    month start_station zip_code
## 1:     1            68      710
## 2:     2            69      591
```

Adding and updating columns by reference Add a new column, duration_hour:

```r
batrips[, duration_hour := duration / 3600]
```

Fix/edit spelling in the second row of start_station:

```r
batrips[2, start_station := "San Francisco City Hall 2"]
```

Replace negative duration values with NA:

```r
batrips[duration < 0, duration := NA]
```

Add a new column equal to total trips for every start station:

```r
batrips[, trips_N := .N, by = start_station]
```

Add new column for every start_station and end_station:

```
batrips[, duration_mean := mean(duration), by = .(start_station, end_station)]
```

Calculate the mean duration for each month:

```
batrips[, mean_dur := mean(duration, na.rm = TRUE),
            by = month(start_date)]
```

Replace NA values in duration with the mean value of duration for that month:

```
batrips[, mean_dur := mean(duration, na.rm = TRUE),
            by = month(start_date)][is.na(duration),
                                    duration := mean_dur]
```

Delete the mean_dur column by reference:

```
batrips[, mean_dur := mean(duration, na.rm = TRUE),
            by = month(start_date)][is.na(duration),
                                    duration := mean_dur][, mean_dur := NULL]
```

Add columns using the LHS := RHS form LHS := RHS form. In the LHS, you specify column names as a character vector and in the RHS, you specify values/expressions to be added inside list() (or the alias, .()):

```
batrips[, c("mean_duration",
            "median_duration") := .(mean(duration), median(duration)),
        by = start_station]
```

Add columns using the functional form:

```
batrips[, `:=`(mean_duration = mean(duration),
               median_duration = median(duration)),
        by = start_station]
```

Add the mean_duration column:

```
batrips[duration > 600, mean_duration := mean(duration),
        by = .(start_station, end_station)]
```

Use read.csv() to import batrips Fread is much faster!

- system.time(read.csv("batrips.csv"))
- system.time(fread("batrips.csv"))

Import using read.csv():

```
csv_file <- read.csv("data/sample.csv", fill = NA, quote = "",
                     stringsAsFactors = FALSE, strip.white = TRUE,
                     header = TRUE)
csv_file %>%
  head(2)
```

```
##   YEAR    GEO Age_group  Sex                                        Element
```

```
## 1 1980 Canada         0 Both           Number of survivors at age x (lx)
## 2 1980 Canada         0 Both Number of deaths between age x and x+1 (dx)
##   AVG_VALUE
## 1    100000
## 2       976
```

Import using fread():

```r
csv_file <- fread("data/sample.csv")
csv_file %>%
  head(2)
```

```
##    YEAR    GEO Age_group  Sex                                     Element
## 1: 1980 Canada         0 Both           Number of survivors at age x (lx)
## 2: 1980 Canada         0 Both Number of deaths between age x and x+1 (dx)
##    AVG_VALUE
## 1:    100000
## 2:       976
```

Check the class of Sex column:

```r
class(csv_file$Sex)
```

```
## [1] "character"
```

Import using read.csv with defaults:

```r
str(csv_file)
```

```
## Classes 'data.table' and 'data.frame': 1048575 obs. of  6 variables:
##  $ YEAR     : int  1980 1980 1980 1980 1980 1980 1980 1980 1980 1980 ...
##  $ GEO      : chr  "Canada" "Canada" "Canada" "Canada" ...
##  $ Age_group: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Sex      : chr  "Both" "Both" "Both" "Both" ...
##  $ Element  : chr  "Number of survivors at age x (lx)" "Number of deaths between age x and x+1
##  $ AVG_VALUE: num  1.00e+05 9.76e+02 9.76e-03 1.80e-04 9.90e-01 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Select “id” and “val” columns:

```r
select_columns <- fread("data/sample.csv", select = c("GEO", "Sex"))
select_columns %>%
  head(2)
```

```
##       GEO  Sex
## 1: Canada Both
## 2: Canada Both
```

Drop the “val” column:

```
drop_column <- fread("data/sample.csv", drop = "Sex")
drop_column %>%
  head(2)
```

```
##    YEAR    GEO Age_group                                     Element AVG_VALUE
## 1: 1980 Canada         0           Number of survivors at age x (lx)    100000
## 2: 1980 Canada         0 Number of deaths between age x and x+1 (dx)       976
```

Import the file while avoiding the warning:

```
only_data <- fread("data/sample.csv", nrows = 3)
only_data
```

```
##    YEAR    GEO Age_group  Sex                                     Element
## 1: 1980 Canada         0 Both            Number of survivors at age x (lx)
## 2: 1980 Canada         0 Both  Number of deaths between age x and x+1 (dx)
## 3: 1980 Canada         0 Both Death probability between age x and x+1 (qx)
##    AVG_VALUE
## 1:  1.00e+05
## 2:  9.76e+02
## 3:  9.76e-03
```

Import only the metadata:

```
only_metadata <- fread("data/sample.csv", skip = 7)
only_metadata %>%
  head(2)
```

```
##      V1     V2 V3   V4                                                     V5
## 1: 1980 Canada  0 Both Cumulative number of life years lived beyond age x (Tx)
## 2: 1980 Canada  0 Both                Life expectancy (in years) at age x (ex)
##           V6
## 1: 7543058.0
## 2:      75.4
```

Import using read.csv:

```
base_r <- read.csv("data/sample.csv",
                   colClasses = c(rep("factor", 4),
                                  "character",
                                  "numeric"))
str(base_r)
```

```
## 'data.frame': 1048575 obs. of  6 variables:
##  $ YEAR     : Factor w/ 35 levels "1980","1981",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ GEO      : Factor w/ 10 levels "Alberta","British Columbia",..: 3 3 3 3 3 3 3 3 3
##  $ Age_group: Factor w/ 111 levels "0","1","10","100",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Sex      : Factor w/ 3 levels "Both","F","M": 1 1 1 1 1 1 1 1 1 3 ...
##  $ Element  : chr  "Number of survivors at age x (lx)" "Number of deaths between age
```

```
## $ AVG_VALUE: num  1.00e+05 9.76e+02 9.76e-03 1.80e-04 9.90e-01 ...
```

Import using fread:

```
import_fread <- fread("data/sample.csv",
                      colClasses = list(factor = 1:4, numeric = 7:10))
```

```
## Warning in fread("data/sample.csv", colClasses = list(factor = 1:4, numeric =
## 7:10)): Column number 7 (colClasses[[2]][1]) is out of range [1,ncol=6]

## Warning in fread("data/sample.csv", colClasses = list(factor = 1:4, numeric =
## 7:10)): Column number 8 (colClasses[[2]][2]) is out of range [1,ncol=6]

## Warning in fread("data/sample.csv", colClasses = list(factor = 1:4, numeric =
## 7:10)): Column number 9 (colClasses[[2]][3]) is out of range [1,ncol=6]

## Warning in fread("data/sample.csv", colClasses = list(factor = 1:4, numeric =
## 7:10)): Column number 10 (colClasses[[2]][4]) is out of range [1,ncol=6]
```

```
str(import_fread)
```

```
## Classes 'data.table' and 'data.frame': 1048575 obs. of  6 variables:
##  $ YEAR     : Factor w/ 35 levels "1980","1981",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ GEO      : Factor w/ 10 levels "Alberta","British Columbia",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ Age_group: Factor w/ 111 levels "0","1","10","100",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Sex      : Factor w/ 3 levels "Both","F","M": 1 1 1 1 1 1 1 1 1 3 ...
##  $ Element  : chr  "Number of survivors at age x (lx)" "Number of deaths between age x and x+1
##  $ AVG_VALUE: num  1.00e+05 9.76e+02 9.76e-03 1.80e-04 9.90e-01 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Import the file correctly, use the fill argument to ensure all rows are imported correctly:

```
correct <- fread("data/sample.csv", fill = TRUE)
correct %>%
  head(2)
```

```
##    YEAR    GEO Age_group  Sex                                     Element
## 1: 1980 Canada         0 Both           Number of survivors at age x (lx)
## 2: 1980 Canada         0 Both Number of deaths between age x and x+1 (dx)
##    AVG_VALUE
## 1:    100000
## 2:       976
```

Import the file using na.strings The missing values are encoded as "##". Note that fread() handles an empty field „ by default as NA

```
missing_values <- fread("data/sample.csv", na.strings = "##")
missing_values %>%
  head(2)
```

```
##    YEAR    GEO Age_group  Sex                                     Element
```

```
## 1: 1980 Canada         0 Both           Number of survivors at age x (lx)
## 2: 1980 Canada         0 Both Number of deaths between age x and x+1 (dx)
##    AVG_VALUE
## 1:  1.00E+05
## 2:       976
```

Write dt to fwrite.txt: - fwrite(dt, “fwrite.txt”)

Import the file using readLines():

```
readLines("data/sample.csv") %>%
  head(2)
```

```
## Warning in readLines("data/sample.csv"): incomplete final line found on 'data/
## sample.csv'
```

```
## [1] "YEAR,GEO,Age_group,Sex,Element,AVG_VALUE"
## [2] "1980,Canada,0,Both,Number of survivors at age x (lx),1.00E+05"
```

Write batrips_dates to file using “ISO” format: - fwrite(batrips_dates, “iso.txt”, dateTimeAs = “ISO”)

Write batrips_dates to file using “squash” format: - fwrite(batrips_dates, “squash.txt”, dateTimeAs = “squash”)

Chapter 6

Tests for experiments

Prior to performing experiments, we need to set the dependent variables (outcome), and independent variables (explanatory variables).

Other experimental components to consider include randomization, replication, blocking

```
# load dependencies
library(ggplot2)
library(broom)
library(tidyverse)
library(pwr)
library(haven)
library(simputation)
library(sampling)
library(agricolae)
library(naniar)
library(DescTools)
library(mice)
```

load data: Dataset is on the Effect of Vitamin C on Tooth Growth in Guinea Pigs:

```
data(ToothGrowth)

ToothGrowth %>%
  head(2)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
```

Perform a two-sided t-test:

```
t.test(x = ToothGrowth$len, alternative = "two.sided", mu = 18)
```

```
##
##  One Sample t-test
##
## data:  ToothGrowth$len
## t = 0.82361, df = 59, p-value = 0.4135
## alternative hypothesis: true mean is not equal to 18
## 95 percent confidence interval:
##  16.83731 20.78936
## sample estimates:
## mean of x
##  18.81333
```

Perform a t-test

```
ToothGrowth_ttest <- t.test(len ~ supp, data = ToothGrowth)
ToothGrowth_ttest
```

```
##
##  Welch Two Sample t-test
##
## data:  len by supp
## t = 1.9153, df = 55.309, p-value = 0.06063
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.1710156  7.5710156
## sample estimates:
## mean in group OJ mean in group VC
##         20.66333         16.96333
```

Tidy ToothGrowth_ttest:

```
tidy(ToothGrowth_ttest)
```

```
## # A tibble: 1 x 10
##   estimate estimate1 estimate2 statistic p.value parameter conf.low conf.high
##      <dbl>     <dbl>     <dbl>     <dbl>   <dbl>     <dbl>    <dbl>     <dbl>
## 1     3.70      20.7      17.0      1.92  0.0606      55.3   -0.171      7.57
## # ... with 2 more variables: method <chr>, alternative <chr>
```

Replication: Count number of observations for each combination of supp and dose

```
ToothGrowth %>%
  count(supp, dose)
```

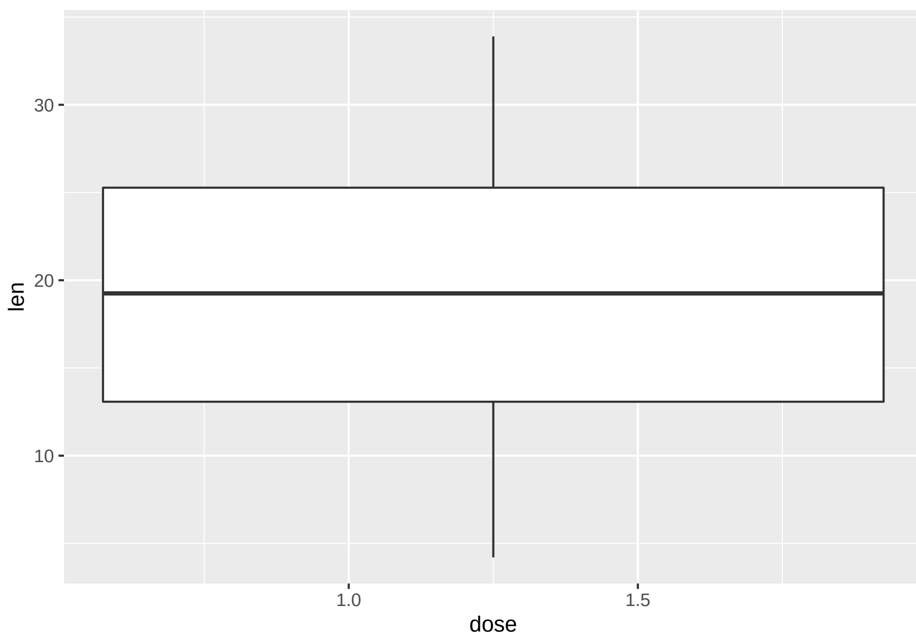
```
##   supp dose  n
## 1   OJ  0.5 10
```

```
## 2   OJ  1.0 10
## 3   OJ  2.0 10
## 4   VC  0.5 10
## 5   VC  1.0 10
## 6   VC  2.0 10
```

Blocking: Create a boxplot with geom_boxplot() aov() creates a linear regression model by calling lm() and examining results with anova() all in one function call.

```r
ggplot(ToothGrowth, aes(x = dose, y = len)) +
  geom_boxplot()
```

```
## Warning: Continuous x aesthetic -- did you forget aes(group=...)?
```



Create ToothGrowth_aov and Examine ToothGrowth_aov with summary():

```r
ToothGrowth_aov <- aov(len ~ dose + supp, data = ToothGrowth)
summary(ToothGrowth_aov)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)    
## dose         1 2224.3  2224.3  123.99 6.31e-16 ***
## supp         1  205.3   205.3   11.45   0.0013 ** 
## Residuals   57 1022.6    17.9                     
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Hypothesis Testing (null and alternative) with `pwr` package one sided and two sided tests: - type ?t.test to find out more

```
#Less than
t.test(x = ToothGrowth$len,
       alternative = "less",
       mu = 18)
```

```
##
##  One Sample t-test
##
## data:  ToothGrowth$len
## t = 0.82361, df = 59, p-value = 0.7933
## alternative hypothesis: true mean is less than 18
## 95 percent confidence interval:
##      -Inf 20.46358
## sample estimates:
## mean of x
##  18.81333
```

```
# Greater than
t.test(x = ToothGrowth$len,
       alternative = "greater",
       mu = 18)
```

```
##
##  One Sample t-test
##
## data:  ToothGrowth$len
## t = 0.82361, df = 59, p-value = 0.2067
## alternative hypothesis: true mean is greater than 18
## 95 percent confidence interval:
##  17.16309      Inf
## sample estimates:
## mean of x
##  18.81333
```

It turns out the mean of len is actually very close to 18, so neither of these tests tells us much about the mean of tooth length. ?pwr.t.test()

Calculate sample size:

```
pwr.t.test(n = NULL,
           d = 0.25,  # small effect size of 0.25
           sig.level = 0.05,
           type = "one.sample",
           alternative = "greater",
           power = 0.8)
```

```
##
```

```
##      One-sample t test power calculation
##
##               n = 100.2877
##               d = 0.25
##       sig.level = 0.05
##           power = 0.8
##     alternative = greater
```

Calculate power:

```r
pwr.t.test(n = 100,
           d = 0.35,
           sig.level = 0.1,
           type = "two.sample",
           alternative = "two.sided",
           power = NULL)
```

```
##
##      Two-sample t test power calculation
##
##               n = 100
##               d = 0.35
##       sig.level = 0.1
##           power = 0.7943532
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

power for multiple groups:

```r
pwr.anova.test(k = 3,
               n = 20,
               f = 0.2, #effect size
               sig.level = 0.05,
               power = NULL)
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##               k = 3
##               n = 20
##               f = 0.2
##       sig.level = 0.05
##           power = 0.2521043
##
## NOTE: n is number in each group
```

Anova tests (for multiple groups) can be done in two ways

Basic Experiments for exploratory data analysis including A/B testing

get data:

```
data(txhousing)

txhousing %>%
  head(2)
```

```
## # A tibble: 2 x 9
##   city     year month sales  volume median listings inventory  date
##   <chr>   <int> <int> <dbl>   <dbl>  <dbl>    <dbl>     <dbl> <dbl>
## 1 Abilene  2000     1    72 5380000  71400      701       6.3 2000
## 2 Abilene  2000     2    98 6505000  58700      746       6.6 2000.
```

remove NAs:

```
tx_housing <- na.omit(txhousing)

# Examine the variables with glimpse()
glimpse(tx_housing)
```

```
## Rows: 7,126
## Columns: 9
## $ city      <chr> "Abilene", "Abilene", "Abilene", "Abilene", "Abilene", "A...
## $ year      <int> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 200...
## $ month     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6, ...
## $ sales     <dbl> 72, 98, 130, 98, 141, 156, 152, 131, 104, 101, 100, 92, 7...
## $ volume    <dbl> 5380000, 6505000, 9285000, 9730000, 10590000, 13910000, 1...
## $ median    <dbl> 71400, 58700, 58100, 68600, 67300, 66900, 73500, 75000, 6...
## $ listings  <dbl> 701, 746, 784, 785, 794, 780, 742, 765, 771, 764, 721, 65...
## $ inventory <dbl> 6.3, 6.6, 6.8, 6.9, 6.8, 6.6, 6.2, 6.4, 6.5, 6.6, 6.2, 5....
## $ date      <dbl> 2000.000, 2000.083, 2000.167, 2000.250, 2000.333, 2000.41...
```

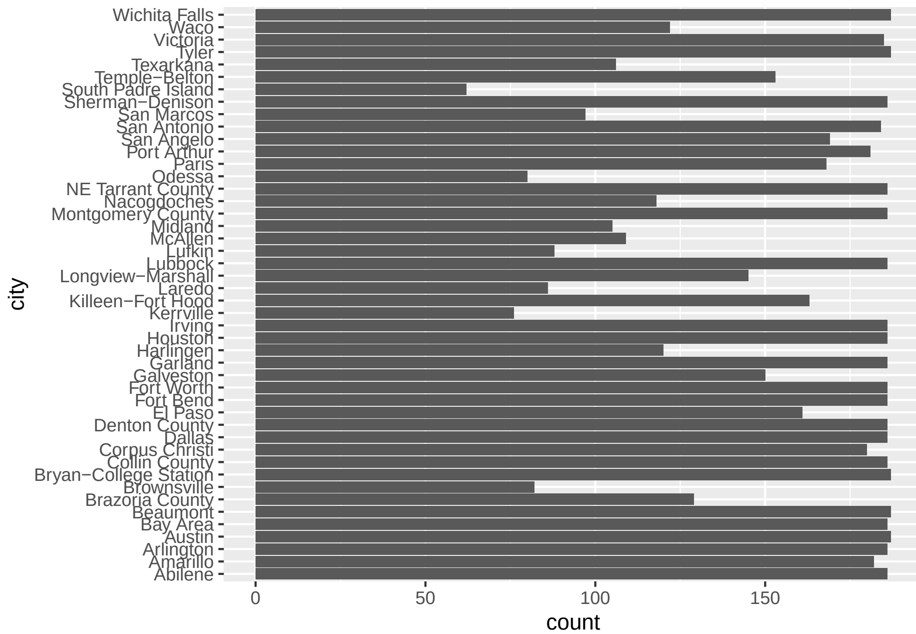
Find median and means with summarize():

```
tx_housing %>%
  summarize(median(volume), mean(sales), mean(inventory))
```

```
## # A tibble: 1 x 3
##   `median(volume)` `mean(sales)` `mean(inventory)`
##              <dbl>         <dbl>             <dbl>
## 1        26240116.          603.              7.17
```

Use ggplot2 to build a bar chart of purpose:

```
ggplot(data=tx_housing, aes(x = city)) +
  geom_bar() +
  coord_flip()
```



Use recode() to create the new purpose_recode variable

```r
tx_housing$city_recode <- tx_housing$city %>%
  recode("Bay Area" = "California",
         "El Paso" = "California")
```

Build a linear regression model, purpose_recode_model:

```r
purpose_recode_model <- lm(sales ~ city_recode, data = tx_housing)

# Examine results of purpose_recode_model
summary(purpose_recode_model)
```

```
##
## Call:
## lm(formula = sales ~ city_recode, data = tx_housing)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2938.1   -40.2    -2.5    30.5  3353.9
##
## Coefficients:
##                                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)                         150.462     23.162   6.496 8.80e-11 ***
## city_recodeAmarillo                  87.680     32.936   2.662 0.007781 **
## city_recodeArlington                272.425     32.756   8.317  < 2e-16 ***
## city_recodeAustin                  1846.227     32.712  56.438  < 2e-16 ***
```

```
## city_recodeBeaumont                   26.596     32.712   0.813 0.416221
## city_recodeBrazoria County           -62.400     36.194  -1.724 0.084743 .
## city_recodeBrownsville               -92.975     41.873  -2.220 0.026425 *
## city_recodeBryan-College Station      36.281     32.712   1.109 0.267428
## city_recodeCalifornia                338.886     28.706  11.805  < 2e-16 ***
## city_recodeCollin County             931.871     32.756  28.449  < 2e-16 ***
## city_recodeCorpus Christi            194.427     33.028   5.887 4.12e-09 ***
## city_recodeDallas                   4205.000     32.756 128.373  < 2e-16 ***
## city_recodeDenton County             476.242     32.756  14.539  < 2e-16 ***
## city_recodeFort Bend                 669.758     32.756  20.447  < 2e-16 ***
## city_recodeFort Worth                622.441     32.756  19.002  < 2e-16 ***
## city_recodeGalveston                 -65.862     34.666  -1.900 0.057484 .
## city_recodeGarland                    42.683     32.756   1.303 0.192600
## city_recodeHarlingen                 -85.571     36.987  -2.314 0.020721 *
## city_recodeHouston                  5440.672     32.756 166.096  < 2e-16 ***
## city_recodeIrving                    -30.478     32.756  -0.930 0.352161
## city_recodeKerrville                -106.291     43.005  -2.472 0.013475 *
## city_recodeKilleen-Fort Hood          64.930     33.892   1.916 0.055430 .
## city_recodeLaredo                    -61.776     41.192  -1.500 0.133732
## city_recodeLongview-Marshall          35.689     34.995   1.020 0.307840
## city_recodeLubbock                   113.511     32.756   3.465 0.000533 ***
## city_recodeLufkin                   -103.349     40.871  -2.529 0.011471 *
## city_recodeMcAllen                     5.969     38.104   0.157 0.875530
## city_recodeMidland                    -4.081     38.559  -0.106 0.915706
## city_recodeMontgomery County         410.027     32.756  12.518  < 2e-16 ***
## city_recodeNacogdoches              -120.412     37.177  -3.239 0.001206 **
## city_recodeNE Tarrant County         532.387     32.756  16.253  < 2e-16 ***
## city_recodeOdessa                    -59.912     42.235  -1.419 0.156075
## city_recodeParis                    -115.998     33.622  -3.450 0.000564 ***
## city_recodePort Arthur               -83.849     32.982  -2.542 0.011034 *
## city_recodeSan Angelo                -37.368     33.570  -1.113 0.265688
## city_recodeSan Antonio              1574.038     32.845  47.923  < 2e-16 ***
## city_recodeSan Marcos               -128.040     39.563  -3.236 0.001216 **
## city_recodeSherman-Denison           -46.134     32.756  -1.408 0.159050
## city_recodeSouth Padre Island       -121.366     46.324  -2.620 0.008814 **
## city_recodeTemple-Belton             -19.992     34.477  -0.580 0.562030
## city_recodeTexarkana                 -73.142     38.443  -1.903 0.057132 .
## city_recodeTyler                      97.971     32.712   2.995 0.002755 **
## city_recodeVictoria                  -80.922     32.800  -2.467 0.013645 *
## city_recodeWaco                       36.947     36.802   1.004 0.315437
## city_recodeWichita Falls             -13.232     32.712  -0.405 0.685850
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 315.9 on 7081 degrees of freedom
## Multiple R-squared:  0.9269,Adjusted R-squared:  0.9264
```

```
## F-statistic:  2040 on 44 and 7081 DF,  p-value: < 2.2e-16
```

Get anova results and save as purpose_recode_anova:

```r
purpose_recode_anova <- anova(purpose_recode_model)

# Print purpose_recode_anova
purpose_recode_anova
```

```
## Analysis of Variance Table
##
## Response: sales
##                Df     Sum Sq   Mean Sq F value    Pr(>F)
## city_recode    44 8955309044 203529751  2039.7 < 2.2e-16 ***
## Residuals    7081  706580734     99785
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Examine class of purpose_recode_anova:

```r
class(purpose_recode_anova)
```

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

Use aov() to build purpose_aov:

```r
# Analysis of variance
purpose_aov <- aov(sales ~ city_recode, data = tx_housing)
```

Conduct Tukey's HSD test to create tukey_output:

```r
tukey_output <- TukeyHSD(purpose_aov, "city_recode", conf.level = 0.95)

# Tidy tukey_output to make sense of the results
tidy(tukey_output)
```

```
## # A tibble: 990 x 7
##    term     contrast          null.value estimate conf.low conf.high adj.p.value
##    <chr>    <chr>                  <dbl>    <dbl>    <dbl>     <dbl>       <dbl>
##  1 city_re~ Amarillo-Abilene           0     87.7    -42.3      218.    8.41e- 1
##  2 city_re~ Arlington-Abilene          0    272.     143.       402.    8.51e-12
##  3 city_re~ Austin-Abilene             0   1846.    1717.      1975.    7.92e-12
##  4 city_re~ Beaumont-Abilene           0     26.6   -102.       156.    1.00e+ 0
##  5 city_re~ Brazoria County--~         0    -62.4   -205.        80.4   1.00e+ 0
##  6 city_re~ Brownsville-Abil~          0    -93.0   -258.        72.2   9.86e- 1
##  7 city_re~ Bryan-College St~          0     36.3    -92.8      165.    1.00e+ 0
##  8 city_re~ California-Abile~          0    339.     226.       452.    7.92e-12
##  9 city_re~ Collin County-Ab~          0    932.     803.      1061.    7.92e-12
## 10 city_re~ Corpus Christi-A~          0    194.      64.1      325.    4.00e- 6
## # ... with 980 more rows
```

Multiple factor experiments: Use aov() to build purpose_emp_aov

```
purpose_emp_aov <- aov(sales ~ city_recode + volume , data = tx_housing)

# Print purpose_emp_aov to the console
# purpose_emp_aov

#Call summary() to see the p-values:
summary(purpose_emp_aov)
```

```
##               Df    Sum Sq   Mean Sq F value Pr(>F)
## city_recode   44 8.955e+09 203529751   12992 <2e-16 ***
## volume         1 5.957e+08 595663462   38022 <2e-16 ***
## Residuals   7080 1.109e+08     15666
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Model validation Pre-modeling exploratory data analysis Examine the summary of sales

```
summary(tx_housing$sales)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       6      95     187     603     527    8945
```

Examine sales by volume:

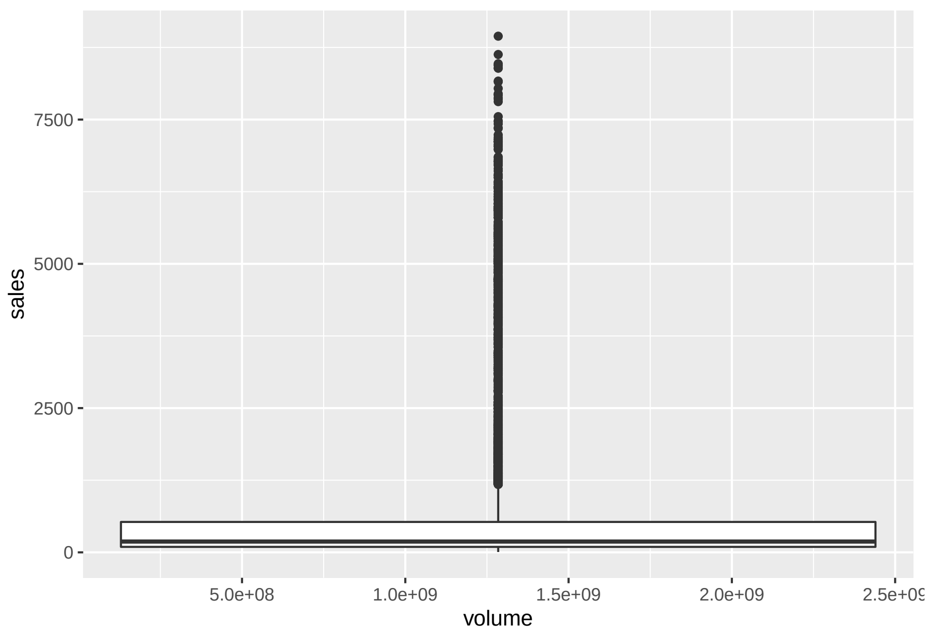
```
tx_housing %>%
  group_by(volume) %>%
  summarize(mean = mean(sales), var = var(sales), median = median(sales))
```

```
## # A tibble: 6,855 x 4
##     volume  mean   var median
##      <dbl> <dbl> <dbl>  <dbl>
##  1  835000    14    NA     14
##  2 1018825    14    NA     14
##  3 1110000     9    NA      9
##  4 1156999     6    NA      6
##  5 1165000    18    NA     18
##  6 1215000    11    NA     11
##  7 1260000    23    NA     23
##  8 1305000    16    NA     16
##  9 1419500    25    NA     25
## 10 1434950    22    NA     22
## # ... with 6,845 more rows
```

Make a boxplot of sales by volume

```
ggplot(tx_housing, aes(x = volume, y = sales)) +
  geom_boxplot()
```

```
## Warning: Continuous x aesthetic -- did you forget aes(group=...)?
```



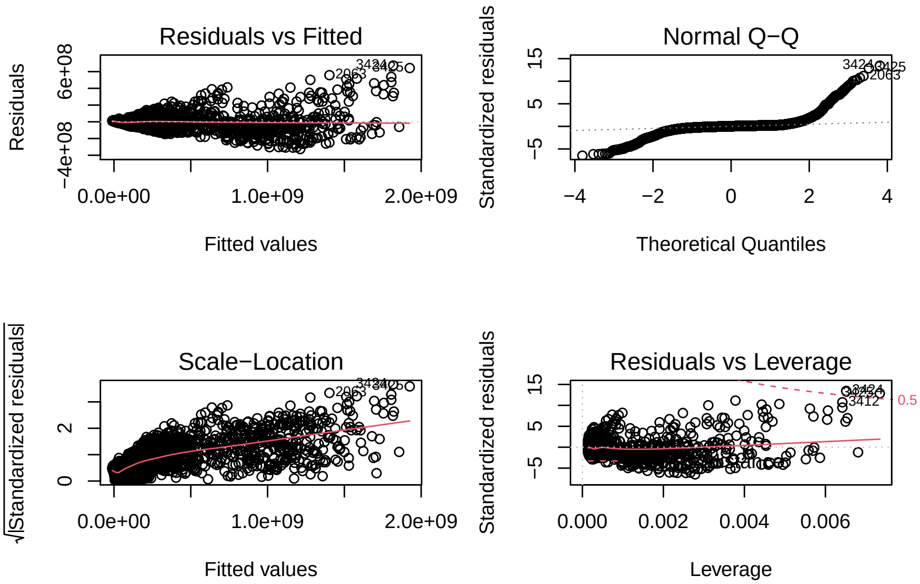
Use aov() to create volume_aov plus call summary() to print results

```
volume_aov <- aov(volume ~ sales, data = tx_housing)
summary(volume_aov)
```

```
##                Df    Sum Sq   Mean Sq F value Pr(>F)    
## sales           1 4.535e+20 4.535e+20  180283 <2e-16 ***
## Residuals    7124 1.792e+19 2.515e+15                   
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Post-modeling validation plots + variance For a 2x2 grid of plots:

```
par(mfrow = c(2, 2))

# Plot grade_aov
plot(volume_aov)
```



Bartlett's test for homogeneity of variance We can test for homogeneity of variances using bartlett.test(), which takes a formula and a dataset as inputs: - bartlett.test(volume ~ sales, data = tx_housing)

Conduct the Kruskal-Wallis rank sum test: kruskal.test() to examine whether volume varies by sales when a non-parametric model is employed

```
kruskal.test(volume ~ sales,
             data = tx_housing)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  volume by sales
## Kruskal-Wallis chi-squared = 6877.9, df = 1702, p-value < 2.2e-16
```

The low p-value indicates that based on this test, we can be confident in our result, which we found across this experiment, that volume varies by sales

Sampling [randomized experiments]

load data from NHANES dataset https://wwwn.cdc.gov/nchs/nhanes/continuousnhanes/default.aspx?BeginYear=2015

Import the three datasets using read_xpt():

```
nhanes_demo <- read_xpt(url("https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/DEMO_I.XPT"))
nhanes_bodymeasures <- read_xpt(url("https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/BMX_I.X
nhanes_medical <- read_xpt(url("https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/MCQ_I.XPT"))
```

Merge the 3 datasets you just created to create nhanes_combined:

```r
nhanes_combined <- list(nhanes_demo, nhanes_medical, nhanes_bodymeasures) %>%
  Reduce(function(df1, df2) inner_join(df1, df2, by = "SEQN"), .)
```

Fill in the dplyr code:

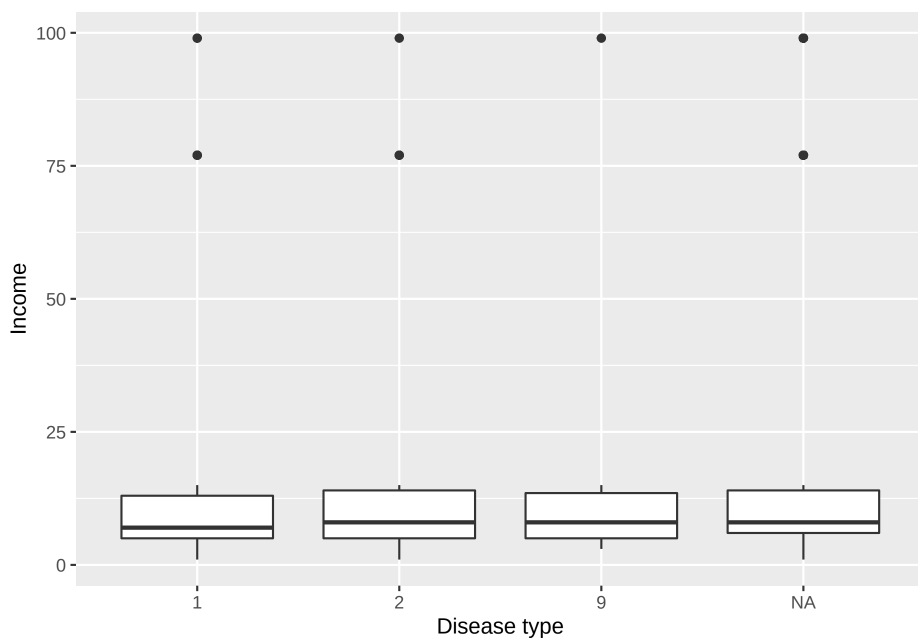
```r
nhanes_combined %>%
  group_by(MCQ035) %>%
  summarize(mean = mean(INDHHIN2, na.rm = TRUE))
```

```
## # A tibble: 4 x 2
##   MCQ035  mean
##    <dbl> <dbl>
## 1      1  9.89
## 2      2 10.4
## 3      9 17.8
## 4     NA 11.7
```

Fill in the ggplot2 code:

```r
nhanes_combined %>%
  ggplot(aes(as.factor(MCQ035), INDHHIN2)) +
  geom_boxplot() +
  labs(x = "Disease type",
       y = "Income")
```

```
## Warning: Removed 273 rows containing non-finite values (stat_boxplot).
```



NHANES Data Cleaning Filter to keep only those greater than 16:

```
nhanes_filter <- nhanes_combined %>% filter(RIDAGEYR > 16)
```

Load simputation & impute bmxwt by riagendr: library(simputation)

```
nhanes_final <- simputation::impute_median(nhanes_filter, INDHHIN2 ~ RIDAGEYR)
```

Recode mcq365d with recode() & examine with count():

```
nhanes_final$mcq365d <- recode(nhanes_final$MCQ035,
                               `1` = 1,
                               `2` = 2,
                               `9` = 2)
nhanes_final %>% count(MCQ035)
```

```
## # A tibble: 4 x 2
##   MCQ035     n
##    <dbl> <int>
## 1      1   522
## 2      2   369
## 3      9    15
## 4     NA  4981
```

Resampling NHANES data: Use sample_n() to create nhanes_srs:

```
nhanes_srs <- nhanes_final %>% sample_n(2500)
```

Create nhanes_stratified with group_by() and sample_n()

```
nhanes_stratified <- nhanes_final %>% group_by(RIDAGEYR) %>% sample_n(2000, replace = T

nhanes_stratified %>%
  count(RIDAGEYR)
```

```
## # A tibble: 64 x 2
## # Groups:   RIDAGEYR [64]
##    RIDAGEYR     n
##       <dbl> <int>
##  1       17  2000
##  2       18  2000
##  3       19  2000
##  4       20  2000
##  5       21  2000
##  6       22  2000
##  7       23  2000
##  8       24  2000
##  9       25  2000
## 10       26  2000
## # ... with 54 more rows
```

Load sampling package and create nhanes_cluster with cluster(): library(sampling)

```
nhanes_cluster <- cluster(nhanes_final, c("INDHHIN2"), 6, method = "srswor")
```

Randomized complete block designs (RCBD): use library(agricolae) block = experimental groups are blocked to be similar (e.g. by sex) complete = each treatment is used the same of times in every block randomized = the treatment is assigned randomly inside each block

Create designs using ls():

```
designs <- ls("package:agricolae", pattern = "design")
print(designs)
```

```
##  [1] "design.ab"      "design.alpha"   "design.bib"     "design.crd"
##  [5] "design.cyclic"  "design.dau"     "design.graeco"  "design.lattice"
##  [9] "design.lsd"     "design.mat"     "design.rcbd"    "design.split"
## [13] "design.strip"   "design.youden"
```

Use str() to view design.rcbd's criteria:

```
str(design.rcbd)
```

```
## function (trt, r, serie = 2, seed = 0, kinds = "Super-Duper", first = TRUE,
##     continue = FALSE, randomization = TRUE)
```

Build treats and rep

```
treats <- LETTERS[1:5]
blocks <- 4
blocks
```

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

NHANES RCBD: Build my_design_rcbd and view the sketch

```
my_design_rcbd <- design.rcbd(treats, r = blocks, seed = 42)
my_design_rcbd$sketch
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] "D"  "A"  "C"  "B"  "E"
## [2,] "E"  "A"  "C"  "D"  "B"
## [3,] "D"  "B"  "E"  "A"  "C"
## [4,] "B"  "D"  "E"  "C"  "A"
```

Use aov() to create nhanes_rcbd:

```
nhanes_rcbd <- aov(INDHHIN2 ~ MCQ035 + RIDAGEYR, data = nhanes_final)
```

Check results of nhanes_rcbd with summary():

```r
summary(nhanes_rcbd)
```

```
##              Df Sum Sq Mean Sq F value  Pr(>F)
## MCQ035        1   1399  1398.5   8.242 0.00419 **
## RIDAGEYR      1    312   312.4   1.841 0.17519
## Residuals   903 153221   169.7
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 4981 observations deleted due to missingness
```

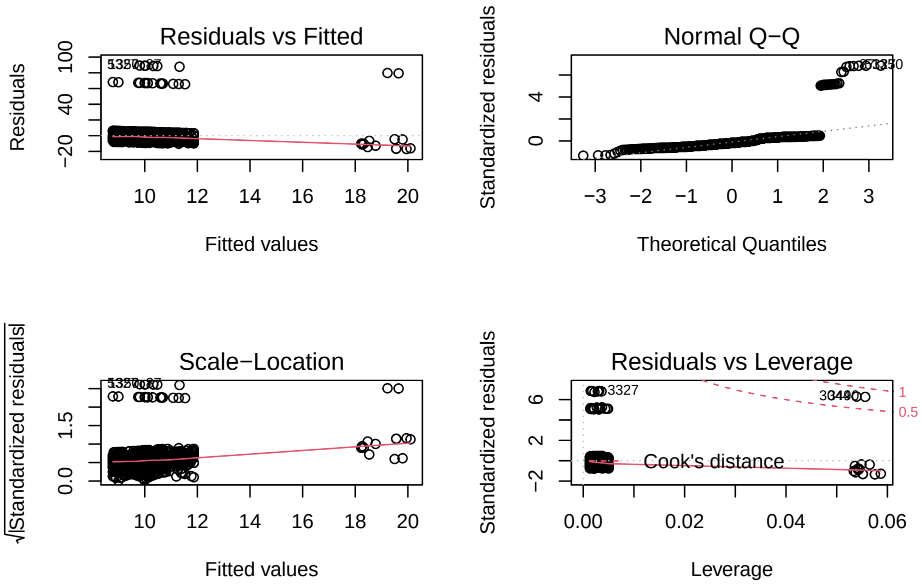
Print mean weights by mcq365d and riagendr:

```r
nhanes_final %>%
  group_by(MCQ035, RIDAGEYR) %>%
  summarize(mean_ind = mean(INDHHIN2, na.rm = TRUE))
```

```
## # A tibble: 204 x 3
## # Groups:   MCQ035 [4]
##    MCQ035 RIDAGEYR mean_ind
##     <dbl>    <dbl>    <dbl>
##  1      1       17    10.3
##  2      1       18     8
##  3      1       19     5.38
##  4      1       20     8
##  5      1       21     8.75
##  6      1       22     6.27
##  7      1       23     7.18
##  8      1       24    14
##  9      1       25     9.11
## 10      1       26    10.6
## # ... with 194 more rows
```

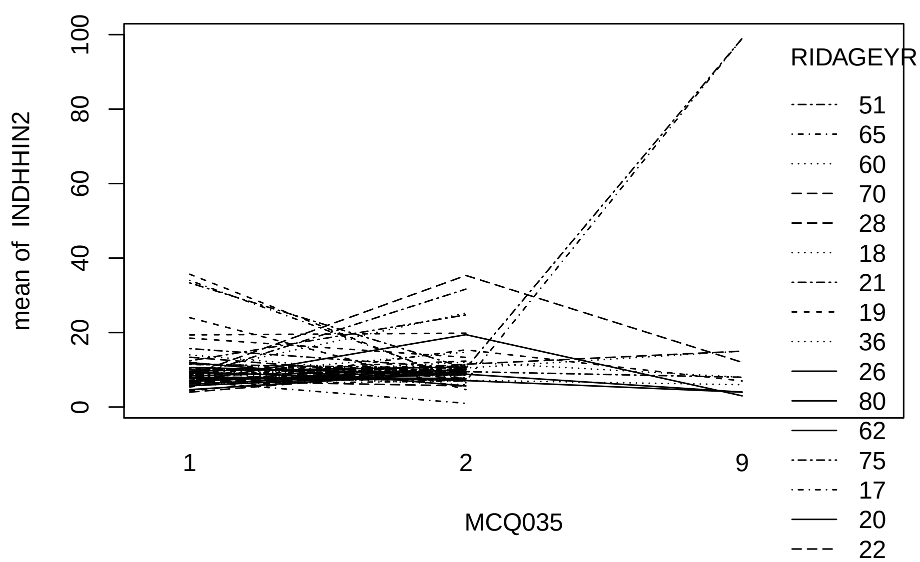
RCBD Model Validation Set up the 2x2 plotting grid and plot nhanes_rcbd

```r
par(mfrow = c(2, 2))
plot(nhanes_rcbd)
```



Run the code to view the interaction plots:

```
with(nhanes_final, interaction.plot(MCQ035, RIDAGEYR, INDHHIN2))
```



Balanced incomplete block design (BIBD) Balanced = each pair of treatment occur together in a block an equal of times Incomplete = not every treatment will appear in every block

Use str() to view design.bibd's criteria str(design.bib)

Columns are a blocking factor

create my_design_bibd_1

```
my_design_bibd_1 <- agricolae::design.bib(LETTERS[1:3], k = 3, seed = 42)
```

```
##
## Parameters BIB
## ==============
## Lambda     : 2
## treatmeans : 3
## Block size : 3
## Blocks     : 2
## Replication: 2
##
## Efficiency factor 1
##
## <<< Book >>>
```

create my_design_bibd_2

```
my_design_bibd_2 <- design.bib(LETTERS[1:8], k = 8, seed = 42)
```

```
##
## Parameters BIB
## ==============
## Lambda     : 2
## treatmeans : 8
## Block size : 8
## Blocks     : 2
## Replication: 2
##
## Efficiency factor 1
##
## <<< Book >>>
```

create my_design_bibd_3:

```
my_design_bibd_3 <- design.bib(LETTERS[1:4], k = 4, seed = 42)
```

```
##
## Parameters BIB
## ==============
## Lambda     : 2
## treatmeans : 4
## Block size : 4
## Blocks     : 2
## Replication: 2
##
## Efficiency factor 1
##
```

```
## <<< Book >>>
```

```r
my_design_bibd_3$sketch
```

```
##      [,1] [,2] [,3] [,4]
## [1,] "C"  "A"  "D"  "B"
## [2,] "C"  "D"  "B"  "A"
```

Build the data.frame:

```r
creatinine <- c(1.98, 1.97, 2.35, 2.09, 1.87, 1.95, 2.08, 2.01, 1.84, 2.06, 1.97, 2.22)
food <- as.factor(c("A", "C", "D", "A", "B", "C", "B", "C", "D", "A", "B", "D"))
color <- as.factor(rep(c("Black", "White", "Orange", "Spotted"), each = 3))
cat_experiment <- as.data.frame(cbind(creatinine, food, color))
```

Create cat_model and examine with summary():

```r
cat_model <- aov(creatinine ~ food + color, data = cat_experiment)
summary(cat_model)
```

```
##             Df  Sum Sq  Mean Sq F value Pr(>F)
## food         1 0.01204 0.012042   0.530  0.485
## color        1 0.00697 0.006971   0.307  0.593
## Residuals    9 0.20461 0.022735
```

Calculate lambda, where lamdba is a measure of proportional reduction in error in cross tabulation analysis:

```r
DescTools::Lambda(cat_experiment, direction = c("symmetric", "row", "column"), conf.level = NA)
```

```
## [1] 0.08636925
```

Create weightlift_model & examine results:

```r
weightlift_model <- aov(MCQ035 ~ INDHHIN2 + RIDAGEYR, data = nhanes_final)
summary(weightlift_model)
```

```
##              Df Sum Sq Mean Sq F value  Pr(>F)
## INDHHIN2      1    9.6   9.614   8.256 0.00416 **
## RIDAGEYR      1    3.9   3.868   3.321 0.06872 .
## Residuals   903 1051.6   1.165
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 4981 observations deleted due to missingness
```

Latin Squares Design Key assumption: the treatment and two blocking factors do NOT interact Two blocking factors (instead of one) Analyze like RCBD

Mean, var, and median of Math score by Borough:

```r
sat_scores <- read.csv(url("https://data.ct.gov/api/views/kbxi-4ia7/rows.csv?accessType=DOWNLOAD'
```

```r
sat_scores %>%
  group_by(District, Test.takers..2012) %>%
  summarize(mean = mean(Test.takers..2012, na.rm = TRUE),
            var = var(Test.takers..2012, na.rm = TRUE),
            median = median(Test.takers..2012, na.rm = TRUE)) %>%
  head()
```

```
## # A tibble: 6 x 5
## # Groups:   District [6]
##   District                  Test.takers..2012  mean   var median
##   <chr>                                 <int> <dbl> <dbl>  <dbl>
## 1 Amistad Academy District                 34    34    NA     34
## 2 Ansonia                                 118   118    NA    118
## 3 Avon                                    254   254    NA    254
## 4 Berlin                                  216   216    NA    216
## 5 Bethel                                  200   200    NA    200
## 6 Bloomfield                               14    14    NA     14
```

Dealing with Missing Test Scores Examine missingness with miss_var_summary() and library(mice):

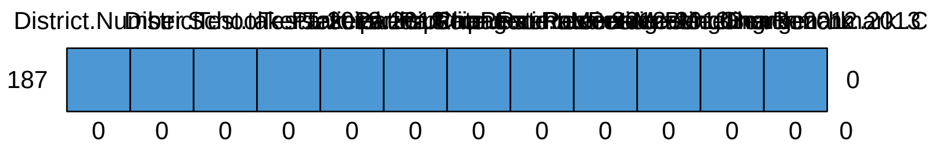
```r
sat_scores %>% miss_var_summary()
```

```
## # A tibble: 12 x 3
##    variable                              n_miss pct_miss
##    <chr>                                  <int>    <dbl>
##  1 Test.takers..2013                          9     4.57
##  2 Test.takers..Change.                       9     4.57
##  3 Participation.Rate..estimate...Change.     8     4.06
##  4 Percent.Meeting.Benchmark..Change.         8     4.06
##  5 Test.takers..2012                          7     3.55
##  6 Participation.Rate..estimate...2012        7     3.55
##  7 Participation.Rate..estimate...2013        7     3.55
##  8 Percent.Meeting.Benchmark..2012            7     3.55
##  9 Percent.Meeting.Benchmark..2013            7     3.55
## 10 District.Number                            0     0
## 11 District                                   0     0
## 12 School                                     0     0
```

```r
sat_scores <- na.omit(sat_scores)

mice::md.pattern(sat_scores)
```

```
##  /\     /\
## {  `---'  }
## {  O   O  }
## ==>  V <==  No need for mice. This data set is completely observed.
```

```
##  \  \|/  /
##   `-----'
```




```
##     District.Number District School Test.takers..2012 Test.takers..2013
## 187               1        1      1                 1                 1
##                   0        0      0                 0                 0
##     Test.takers..Change. Participation.Rate..estimate...2012
## 187                    1                                   1
##                        0                                   0
##     Participation.Rate..estimate...2013 Participation.Rate..estimate...Change.
## 187                                   1                                      1
##                                       0                                      0
##     Percent.Meeting.Benchmark..2012 Percent.Meeting.Benchmark..2013
## 187                               1                               1
##                                   0                               0
##     Percent.Meeting.Benchmark..Change.
## 187                                  1 0
##                                      0 0
```

Impute the Math score by Borough:

```
sat_scores_2 <- simputation::impute_median(sat_scores, Test.takers..2012 ~ District)
#Convert Math score to numeric
sat_scores$Average_testtakers2012 <- as.numeric(sat_scores$Test.takers..2012)
```

Examine scores by Borough in both datasets, before and after imputation:

```
sat_scores %>%
  group_by(District) %>%
  summarize(median = median(Test.takers..2012, na.rm = TRUE),
            mean = mean(Test.takers..2012, na.rm = TRUE))
```

```
## # A tibble: 129 x 3
##    District                 median  mean
##    <chr>                     <dbl> <dbl>
##  1 Amistad Academy District     34    34
##  2 Ansonia                     118   118
##  3 Avon                        254   254
##  4 Berlin                      216   216
##  5 Bethel                      200   200
##  6 Bloomfield                   65    65
##  7 Bolton                       62    62
##  8 Branford                    196   196
```

```
##  9 Bridgeport                    155   202
## 10 Bristol                       211   211
## # ... with 119 more rows
```

```
sat_scores_2 %>%
  group_by(District) %>%
  summarize(median = median(Test.takers..2012),
            mean = mean(Test.takers..2012))
```

```
## # A tibble: 129 x 3
##    District                  median  mean
##    <chr>                      <dbl> <dbl>
##  1 Amistad Academy District     34    34
##  2 Ansonia                     118   118
##  3 Avon                        254   254
##  4 Berlin                      216   216
##  5 Bethel                      200   200
##  6 Bloomfield                   65    65
##  7 Bolton                       62    62
##  8 Branford                    196   196
##  9 Bridgeport                  155   202
## 10 Bristol                     211   211
## # ... with 119 more rows
```

Drawing Latin Squares with agricolae

Design a LS with 5 treatments A:E then look at the sketch

```
my_design_lsd <- design.lsd(trt = LETTERS[1:5], seed = 42)
my_design_lsd$sketch
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] "E"  "D"  "A"  "C"  "B"
## [2,] "D"  "C"  "E"  "B"  "A"
## [3,] "A"  "E"  "B"  "D"  "C"
## [4,] "C"  "B"  "D"  "A"  "E"
## [5,] "B"  "A"  "C"  "E"  "D"
```

Build nyc_scores_ls_lm:

```
sat_scores_ls_lm <- lm(Test.takers..2012 ~ Test.takers..2013 + District,
                       data = sat_scores)

# Tidy the results with broom
tidy(sat_scores_ls_lm) %>%
  head()
```

```
## # A tibble: 6 x 5
##   term              estimate std.error statistic  p.value
```

```
##   <chr>                 <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)           3.42    23.7         0.144 8.86e- 1
## 2 Test.takers..2013     0.987    0.0601     16.4   2.85e-23
## 3 DistrictAnsonia      12.0     33.8         0.355 7.24e- 1
## 4 DistrictAvon         10.9     35.8         0.303 7.63e- 1
## 5 DistrictBerlin       -4.45    35.3        -0.126 9.00e- 1
## 6 DistrictBethel        9.14    34.8         0.263 7.94e- 1
```

Examine the results with anova:

```r
anova(sat_scores_ls_lm)
```

```
## Analysis of Variance Table
##
## Response: Test.takers..2012
##                    Df  Sum Sq Mean Sq   F value Pr(>F)
## Test.takers..2013   1 2144936 2144936 3830.0419 <2e-16 ***
## District          128   46850     366    0.6536 0.9749
## Residuals          57   31922     560
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Graeco-Latin Squares three blocking factors (when there is treatments) Key assumption: the treatment and two blocking factors do NOT interact Analyze like RCBD

Drawing Graeco-Latin Squares with agricolae

Create trt1 and trt2 Create my_graeco_design

```r
trt1 <- LETTERS[1:5]
trt2 <- 1:5
my_graeco_design <- design.graeco(trt1, trt2, seed = 42)
```

Examine the parameters and sketch:

```r
my_graeco_design$parameters
```

```
## $design
## [1] "graeco"
##
## $trt1
## [1] "A" "B" "C" "D" "E"
##
## $trt2
## [1] 1 2 3 4 5
##
## $r
## [1] 5
##
```

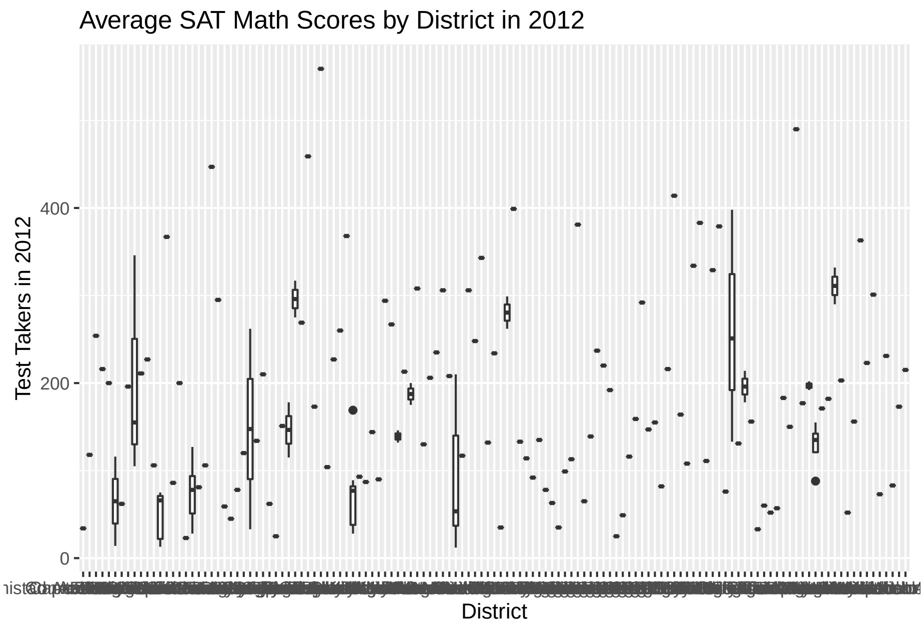
```
## $serie
## [1] 2
##
## $seed
## [1] 42
##
## $kinds
## [1] "Super-Duper"
##
## [[8]]
## [1] TRUE
```

```
my_graeco_design$sketch
```

```
##      [,1]  [,2]  [,3]  [,4]  [,5]
## [1,] "D 5" "A 1" "C 3" "B 4" "E 2"
## [2,] "A 3" "C 4" "B 2" "E 5" "D 1"
## [3,] "C 2" "B 5" "E 1" "D 3" "A 4"
## [4,] "B 1" "E 3" "D 4" "A 2" "C 5"
## [5,] "E 4" "D 2" "A 5" "C 1" "B 3"
```

Create a boxplot of scores by District, with a title and x/y axis labels:

```
ggplot(sat_scores, aes(District, Test.takers..2012)) +
  geom_boxplot() +
  labs(title = "Average SAT Math Scores by District in 2012",
       x = "District",
       y = "Test Takers in 2012")
```



Build sat_scores_gls_lm:

```
sat_scores_gls_lm <- lm(Test.takers..2012 ~ Test.takers..2013 + District + School,
                        data = sat_scores)

# Tidy the results with broom
tidy(sat_scores_gls_lm) %>%
  head()
```

```
## # A tibble: 6 x 5
##   term                estimate std.error statistic p.value
##   <chr>                  <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)            -9.24       NaN       NaN     NaN
## 2 Test.takers..2013       1.39       NaN       NaN     NaN
## 3 DistrictAnsonia       -17.8        NaN       NaN     NaN
## 4 DistrictAvon          -75.7        NaN       NaN     NaN
## 5 DistrictBerlin        -81.6        NaN       NaN     NaN
## 6 DistrictBethel        -55.8        NaN       NaN     NaN
```

Examine the results with anova

```
anova(sat_scores_gls_lm)
```

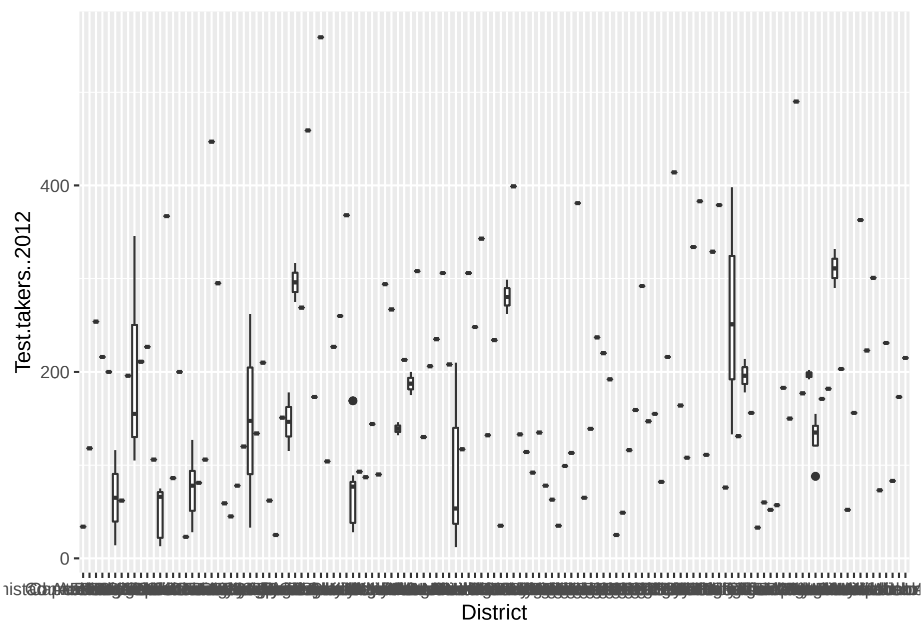
```
## Warning in anova.lm(sat_scores_gls_lm): ANOVA F-tests on an essentially perfect
## fit are unreliable

## Analysis of Variance Table
##
```

```
## Response: Test.takers..2012
##                     Df  Sum Sq Mean Sq F value Pr(>F)
## Test.takers..2013    1 2144936 2144936
## District           128   46850     366
## School              57   31922     560
## Residuals            0       0
```

Factorial Experiment Design 2 or more factor variables are combined and crossed All of the possible interactions between factors are considered as effect on outcome e.g. high/low water on high/low light

Build the boxplot for the district vs. test taker score:

```
ggplot(sat_scores,
       aes(District, Test.takers..2012)) +
  geom_boxplot()
```



Create sat_scores_factorial and examine the results:

```
sat_scores_factorial <- aov(Test.takers..2012 ~ Test.takers..2013 * District * School,

tidy(sat_scores_factorial) %>%
  head()
```

```
## # A tibble: 3 x 4
##   term                 df    sumsq   meansq
##   <chr>             <dbl>    <dbl>    <dbl>
## 1 Test.takers..2013     1 2144936. 2144936.
```

```
## 2 District            128   46850.     366.
## 3 School               57   31922.     560.
```

Evaluating the sat_scores Factorial Model

Use shapiro.test() to test the outcome:

```r
shapiro.test(sat_scores$Test.takers..2013)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  sat_scores$Test.takers..2013
## W = 0.91495, p-value = 6.28e-09
```

Chapter 7

Demo for A/B testing

```
# load dependencies
library(tidyverse)
library(powerMediation)
library(broom)
library(pwr)
library(gsDesign)
library(powerMediation)
```

Read in data:

```
fileLocation <- "http://stat.columbia.edu/~rachel/datasets/nyt1.csv"
click_data <- read.csv(url(fileLocation))
```

Find oldest and most recent age:

```
min(click_data$Age)
```

```
## [1] 0
```

```
max(click_data$Age)
```

```
## [1] 108
```

Compute baseline conversion rates:

```
click_data %>%
  summarize(impression_rate = mean(Impressions))
```

```
##   impression_rate
## 1        5.007316
```

determine baseline for genders:

```
click_data %>%
  group_by(Gender) %>%
  summarize(impression_rate = mean(Impressions))
```

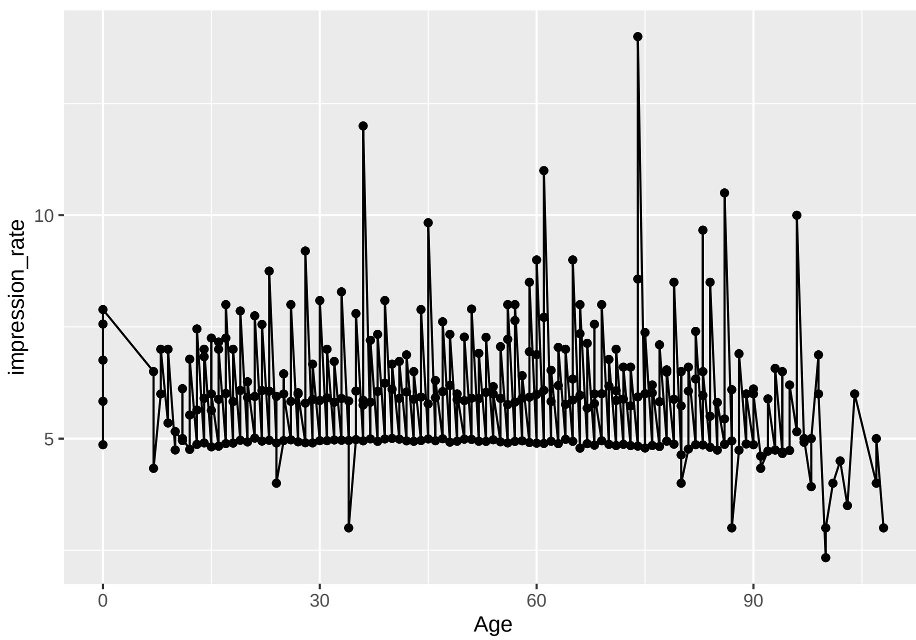
```
## # A tibble: 2 x 2
##   Gender impression_rate
##    <int>           <dbl>
## 1      0            5.01
## 2      1            5.01
```

determine baseline for clicks:

```
click_data_age<- click_data %>%
  group_by(Clicks, Age) %>%
  summarize(impression_rate = mean(Impressions))
```

visualize baselines:

```
ggplot(click_data_age, aes(x = Age, y = impression_rate)) +
  geom_point() +
  geom_line()
```



Experimental design, power analysis, and t-tests

run power analysis: learn more here: help(SSizeLogisticBin)

```
total_sample_size <- SSizeLogisticBin(p1 = 0.2, # conversion rate for control conditio
                                      p2 = 0.3, # conversion rate for expected convers
```

```
                                    B = 0.5, # most commonly used
                                    alpha = 0.05, # most commonly used
                                    power = 0.8) # most commonly used
total_sample_size
```

```
## [1] 587
```

```
total_sample_size /2 # per condition
```

```
## [1] 293.5
```

can use a ttest or linear regression for statistical tests: lm is used when more variables are in data but similar to t-test

```
lm(Gender ~ Clicks, data = click_data) %>%
  summary()
```

```
##
## Call:
## lm(formula = Gender ~ Clicks, data = click_data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -0.375 -0.375 -0.375  0.625  0.884
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.3750325  0.0007418  505.56   <2e-16 ***
## Clicks      -0.0863451  0.0022930  -37.66   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4813 on 458439 degrees of freedom
## Multiple R-squared:  0.003083,Adjusted R-squared:  0.003081
## F-statistic:  1418 on 1 and 458439 DF,  p-value: < 2.2e-16
```

```
# t.test(Gender ~ Clicks, data = click_data) %>%
#  summary()
```

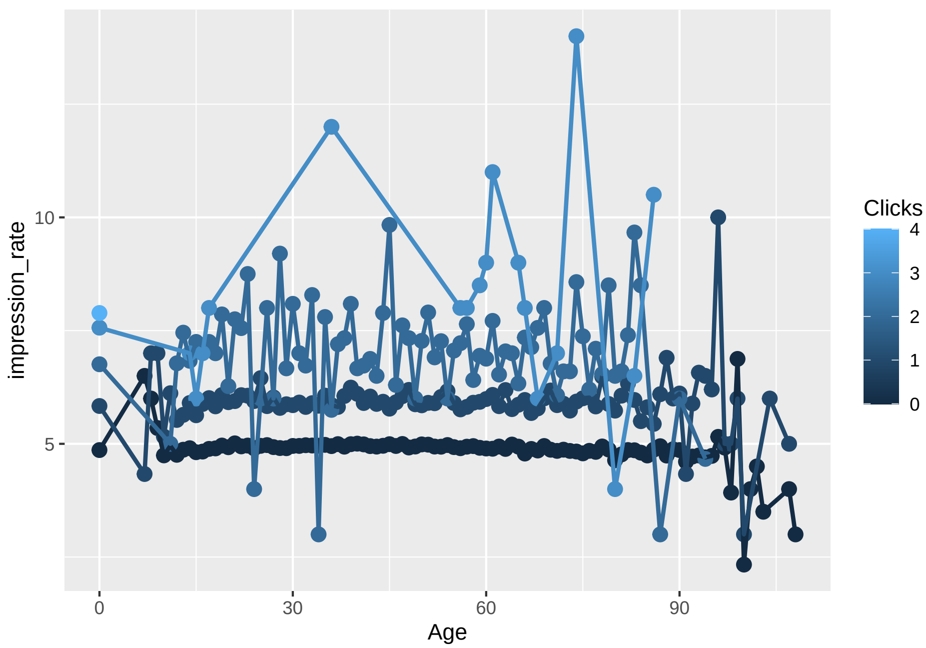
Analyzing results Group and summarize

```
click_data_groups <- click_data %>%
  group_by(Clicks, Age) %>%
  summarize(impression_rate = mean(Impressions))
```

Make plot of conversion rates for clicks:

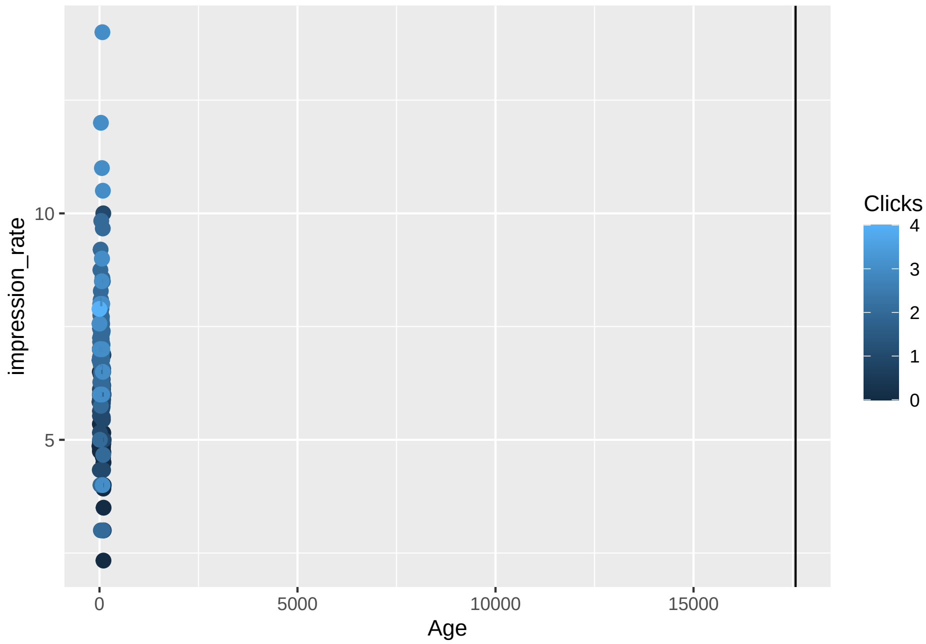
```
ggplot(click_data_groups,
       aes(x = Age,
```

```
           y = impression_rate,
           color = Clicks,
           group = Clicks)) +
  geom_point(size = 3) +
  geom_line(lwd = 1)
```



Make plot of conversion rates for clicks (can add intercepts and interaction of two variables):

```
ggplot(click_data_groups,
       aes(x = Age,
           y = impression_rate,
           color = Clicks,
           group = interaction(Clicks, impression_rate))) +
  geom_point(size = 3) +
  geom_line(lwd = 1) +
  geom_vline(xintercept = as.numeric(as.Date("2018-02-15")))
```



Check for glm documentation family can be used to express different error distributions. ?glm

Run logistic regression to analyze model outputs:

```
experiment_results <- glm(Gender ~ Clicks,
                          family = "binomial",
                          data = click_data) %>%
  tidy()

experiment_results
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic   p.value
##   <chr>          <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)   -0.510   0.00319    -160.   0.
## 2 Clicks        -0.400   0.0107      -37.3 4.10e-304
```

Follow-up experimentations to test assumptions

Designing follow-up experiments since A/B testing is a continuous loops i.e. make new dataframes and compute various other conversion rate differences can use spread() to reformat data

```
click_data_new_groups <- click_data %>%
  group_by(Clicks, Age) %>%
  summarize(impression_rate = mean(Impressions)) %>%
  spread(Clicks, impression_rate)
```

Compute summary statistics:

```
mean(click_data_new_groups$Age, na.rm = TRUE)
```

```
## [1] 55.9802
```

```
sd(click_data_new_groups$Age, na.rm = TRUE)
```

```
## [1] 29.4771
```

Run logistic regression and power analysis Run power analysis for logistic regression

```
total_sample_size <- SSizeLogisticBin(p1 = 0.49,
                                      p2 = 0.64,
                                      B = 0.5,
                                      alpha = 0.05,
                                      power = 0.8)
total_sample_size
```

```
## [1] 341
```

View summary of data:

```
new_data <- click_data %>%
  group_by(Clicks) %>%
  summarize(impression_rate = mean(Impressions)/10)

# Run logistic regression to analyze model outputs
followup_experiment_sep_results <- glm(impression_rate ~ Clicks,
                                       family = "binomial",
                                       data = new_data) %>%
  tidy()
```

```
## Warning in eval(family$initialize): non-integer #successes in a binomial glm!
```

```
followup_experiment_sep_results
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)  0.00899     1.59    0.00566   0.995
## 2 Clicks       0.361       0.710   0.509     0.611
```

Specifics of A/B Testing= use of experimental design to compare 2 or more variants of a design Test Types: A/B, A/A, A/B/N test conditions Assumptions to test: within group vs. between group experiments

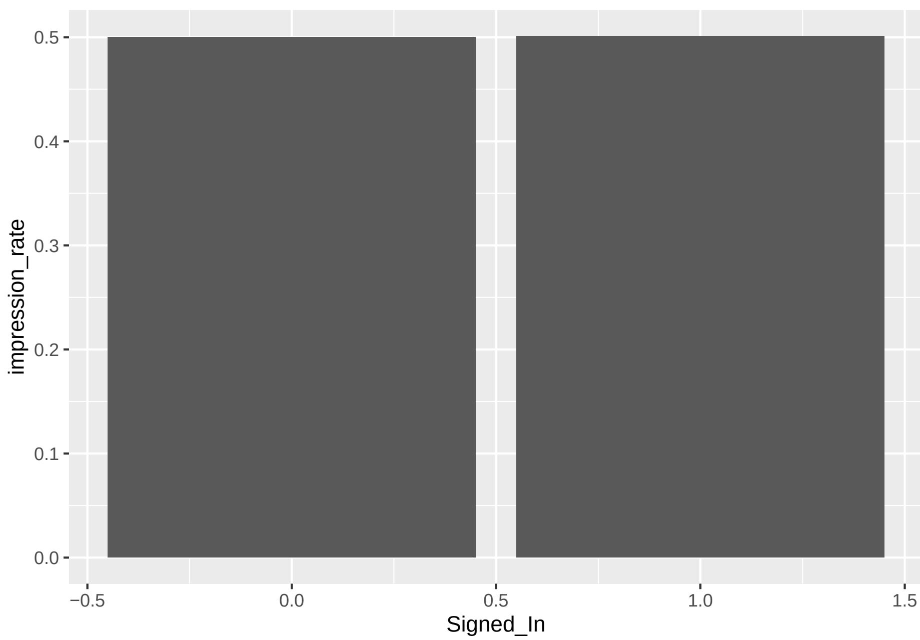
e.g. plotting A/A data Compute conversion rates for A/A experiment:

```r
click_data_sum <- click_data %>%
  group_by(Signed_In) %>%
  summarize(impression_rate = mean(Impressions)/10)
click_data_sum
```

```
## # A tibble: 2 x 2
##   Signed_In impression_rate
##       <int>           <dbl>
## 1         0           0.500
## 2         1           0.501
```

Plot conversion rates for two conditions:

```r
ggplot(click_data_sum,
       aes(x = Signed_In, y = impression_rate)) +
  geom_bar(stat = "identity")
```




```r
#Based on these bar plots the two A conditions look very similar. That's good!
```

Run logistic regression to analyze model outputs:

```r
aa_experiment_results <- glm(Signed_In ~ impression_rate,
                             family = "binomial",
                             data = click_data_sum) %>%
  tidy()
aa_experiment_results
```

```
## # A tibble: 2 x 5
##   term             estimate  std.error statistic p.value
##   <chr>               <dbl>      <dbl>     <dbl>   <dbl>
## 1 (Intercept)       -21589.  51475133. -0.000419    1.00
## 2 impression_rate    43135. 102844880.  0.000419    1.00
```

Confounding variables: element that can affect the truth of A/B exp change one element at a time to know the change you are testing Need to also consider the side effects procedures are the same as above

Power analysis requires 3 variables: power (1-beta) , significance level (alpha or p-value), effect size as power goes up, so does the of data points needed as significance level goes up (i.e. more significant), so do of data points needed as effect sizw increase, of data points decrease ttest (linear regression) can be used for continuous dependent variable (e.g. time spent on a website)

```r
pwr.t.test(power = 0.8,
           sig.level = 0.05,
           d = 0.6)  # d = effect size
```

```
##
##      Two-sample t test power calculation
##
##               n = 44.58577
##               d = 0.6
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

```r
pwr.t.test(power = 0.8,
           sig.level = 0.05,
           d = 0.2) #(see more on experimental design)
```

```
##
##      Two-sample t test power calculation
##
##               n = 393.4057
##               d = 0.2
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

Load package to run power analysis: library(powerMediation)

logistic regression can be used for categorical dependent variable (e.g. click or not click) Run power analysis for logistic regression

```
total_sample_size <- SSizeLogisticBin(p1 = 0.17, # assuming a control value of 17%
                                      p2 = 0.27, # assuming 10% increase in the test condition
                                      B = 0.5,
                                      alpha = 0.05,
                                      power = 0.8)
total_sample_size
```

```
## [1] 537
```

Stopping rules and sequential analysis procedures that allow interim analyses in pre-defined points = sequential analysis

```
seq_analysis <- gsDesign(k=4, # number of times you want to look at the data
                         test.type = 1,
                         alpha = 0.05,
                         beta = 0.2, # power = 1-beta so power is 0.8
                         sfu = "Pocock") # spending function to figure out how to update p-values
seq_analysis
```

```
## One-sided group sequential design with
## 80 % power and 5 % Type I Error.
##            Sample
##             Size
##   Analysis Ratio*  Z   Nominal p  Spend
##          1  0.306 2.07    0.0193 0.0193
##          2  0.612 2.07    0.0193 0.0132
##          3  0.918 2.07    0.0193 0.0098
##          4  1.224 2.07    0.0193 0.0077
##      Total                       0.0500
##
## ++ alpha spending:
##  Pocock boundary.
## * Sample size ratio compared to fixed design with no interim
##
## Boundary crossing probabilities and expected sample size
## assume any cross stops the trial
##
## Upper boundary (power or Type I Error)
##           Analysis
##    Theta      1      2      3      4 Total   E{N}
##   0.0000 0.0193 0.0132 0.0098 0.0077  0.05 1.1952
##   2.4865 0.2445 0.2455 0.1845 0.1255  0.80 0.7929
```

```
max_n <- 1000
max_n_per_group <- max_n / 2
stopping_points <- max_n_per_group * seq_analysis$timing
```

```
stopping_points
```

```
## [1] 125 250 375 500
```

Run sequential analysis:

```
seq_analysis_3looks <- gsDesign(k = 3,
                                test.type = 1,
                                alpha = 0.05,
                                beta = 0.2,
                                sfu = "Pocock")
seq_analysis_3looks
```

```
## One-sided group sequential design with
## 80 % power and 5 % Type I Error.
##            Sample
##             Size
##   Analysis Ratio*  Z   Nominal p  Spend
##          1  0.394 1.99    0.0232 0.0232
##          2  0.789 1.99    0.0232 0.0155
##          3  1.183 1.99    0.0232 0.0113
##      Total                       0.0500
##
## ++ alpha spending:
##  Pocock boundary.
## * Sample size ratio compared to fixed design with no interim
##
## Boundary crossing probabilities and expected sample size
## assume any cross stops the trial
##
## Upper boundary (power or Type I Error)
##           Analysis
##    Theta      1      2      3 Total   E{N}
##   0.0000 0.0232 0.0155 0.0113  0.05 1.1591
##   2.4865 0.3334 0.2875 0.1791  0.80 0.8070
```

Fill in max number of points and compute points per group and find stopping points

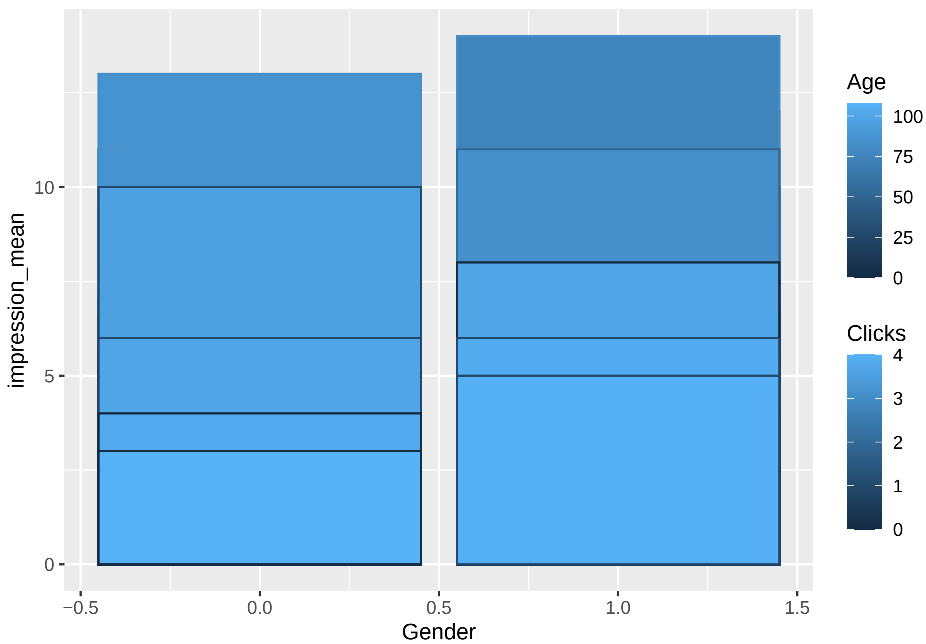
```
max_n <- 3000
max_n_per_group <- max_n / 2
stopping_points = max_n_per_group * seq_analysis_3looks$timing
stopping_points
```

```
## [1]  500 1000 1500
```

Multivariate testing (i.e. more than one independent variable in the experiment)

Compute summary values for four conditions

```
new_click_data <- click_data %>%
  group_by(Age, Gender, Clicks) %>%
  summarize(impression_mean = mean(Impressions))

# Plot summary values for four conditions
ggplot(new_click_data,
       aes(x = Gender,
           y = impression_mean,
           color = Clicks,
           fill = Age)) +
  geom_bar(stat = "identity", position = "dodge")
```




```
multivar_results <- lm(Age ~ Gender * Clicks, data = click_data) %>%
  tidy()

multivar_results$p.value #none are significant
```

```
## [1]  0.000000e+00  0.000000e+00  0.000000e+00 3.569988e-236
```

```
multivar_results
```

```
## # A tibble: 4 x 5
##   term          estimate std.error statistic   p.value
##   <chr>            <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)      23.4     0.0427     549.  0.
## 2 Gender           17.2     0.0699     246.  0.
```

```
## 3 Clicks           -5.00    0.123      -40.8 0.
## 4 Gender:Clicks     7.72    0.235       32.8 3.57e-236
```

Organize variables and run logistic regression:

```
new_click_data_results <- click_data %>%
  mutate(gender = factor(Gender,
                         levels = c("0", "1"))) %>%
  mutate(clicks = factor(Clicks,
                         levels = c("1", "0"))) %>%
  glm(gender ~ gender * clicks,
      family = "binomial",
      data = .) %>%
  tidy()
```

```
## Warning in model.matrix.default(...): the response appeared on the right-hand
## side and was dropped
```

```
## Warning in model.matrix.default(...): problem with term 1 in model.matrix: no
## columns are assigned
```

```
new_click_data_results
```

```
## # A tibble: 3 x 5
##   term             estimate std.error statistic p.value
##   <chr>               <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)        -0.894    0.0114   -78.5     0
## 2 clicks0           -21.7     94.2       -0.230   0.818
## 3 gender1:clicks0    45.1    154.         0.293   0.769
```

Chapter 8

R for Reporting

Possible ways to report your findings include e-mailing figures and tables around with some explanatory text or creating reports in Word, LaTeX or HTML.

R code used to produce the figures and tables is typically not part of these documents. So in case the data changes, e.g., if new data becomes available, the code needs to be re-run and all the figures and tables updated. This can be rather cumbersome. If code and reporting are not in the same place, it can also be a bit of a hassle to reconstruct the details of the analysis carried out to produce the results.

To enable reproducible data analysis and research, the idea of dynamic reporting is that data, code and results are all in one place. This can for example be a R Markdown document like this one. Generating the report automatically executes the analysis code and includes the results in the report.

8.1 Usage demonstrations

8.1.1 Inline code

Simple pieces of code can be included inline. This can be handy to, e.g., include the number of observations in your data set dynamically. The *cars* data set, often used to illustrate the linear model, has 50 observations.

8.1.2 Code chunks

You can include typical output like a summary of your data set and a summary of a linear model through code chunks.

```
summary(cars)
```

```
##      speed           dist
```

```
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

```
m <- lm(dist ~ speed, data = cars)
summary(m)
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511,Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

8.1.2.1 Include tables

The estimated coefficients, as well as their standard errors, t-values and p-values can also be included in the form of a table, for example through **knitr**'s `kable` function.

```
library("knitr")
kable(summary(m)$coef, digits = 2)
```

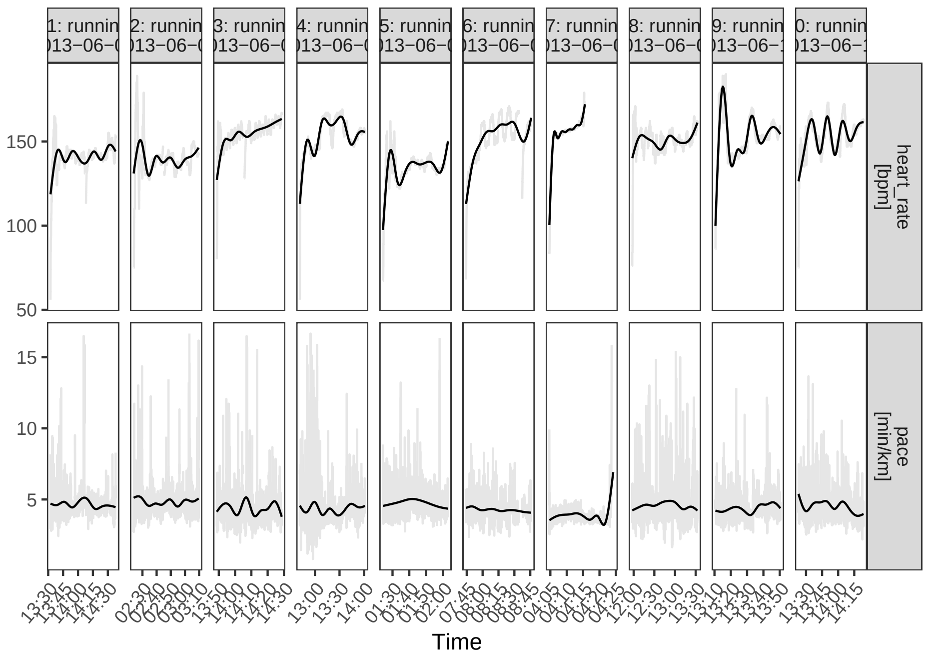
	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-17.58	6.76	-2.60	0.01
speed	3.93	0.42	9.46	0.00

8.1.2.2 Include figures

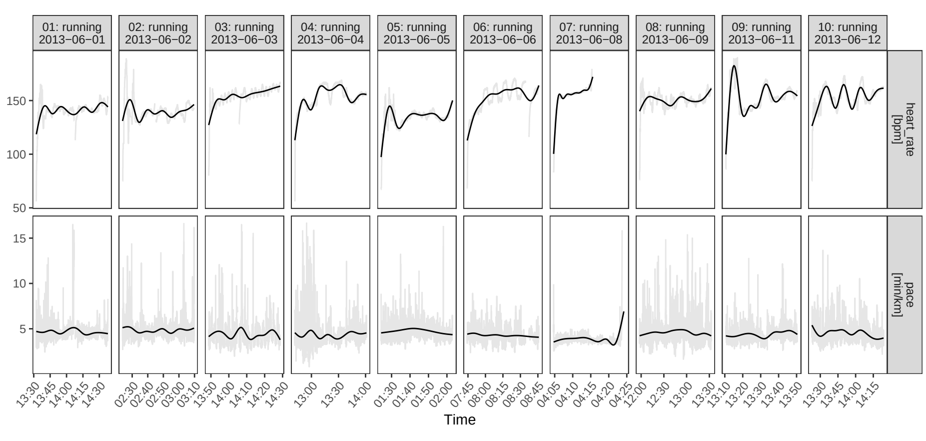
The **trackeR** package provides infrastructure for running and cycling data in **R** and is used here to illustrate how figures can be included.

```r
## install.packages("devtools")
## devtools::install_github("hfrick/trackeR")
library("trackeR")
data("runs", package = "trackeR")
```

A plot of how heart rate and pace evolve over time in 10 training sessions looks like this



but the plot looks better with a wider plotting window.



8.2 Resources

- Markdown main page
- R Markdown
- knitr in a nutshell tutorial by Karl Broman

8.3 Beginner Resources by Topic

8.3.1 Getting Set-Up with R & RStudio

- **Download & Install R:**
  - https://cran.r-project.org
  - For Mac: click on **Download R for (Mac) OS X**, look at the top link under **Files**, which at time of writing is **R-3.2.4.pkg**, and download this if compatible with your current version mac OS (Mavericks 10.9 or higher). Otherwise download the version beneath it which is compatible for older mac OS versions. Then install the downloaded software.
  - For Windows: click on **Download R for Windows**, then click on the link **install R for the first time**, and download from the large link at the top of the page which at time of writing is **Download R 3.2.4 for Windows**. Then install the downloaded software.
- **Download & Install RStudio:**
  - https://www.rstudio.com/products/rstudio/download/
  - For Mac: under the **Installers for Supported Platforms** heading click the link with **Mac OS X** in it. Install the downloaded software.
  - For Windows: under the **Installers for Supported Platforms** heading click the link with **Windows Vista** in it. Install the downloaded software.
- **Exercises in R: swirl (HIGHLY RECOMMENDED):**
  - http://swirlstats.com/students.html
- **Data Prep**:
  - Intro to dplyr: https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html
  - Data Manipulation (detailed): http://www.sr.bham.ac.uk/~ajrs/R/index.html
  - Aggregation and Restructing Data (base & reshape): http://www.r-statistics.com/2012/01/aggregation-and-restructuring-data-from-r-in-action/
- **Data Types intro**: Vectors, Matrices, Arrays, Data Frames, Lists, Factors: http://www.statmethods.net/input/datatypes.html

- **Using Dates and Times**: http://www.cyclismo.org/tutorial/R/time.html
- **Text Data and Character Strings**: http://gastonsanchez.com/Handling_and_Processing_Strings_in_R.pdf
- **Data Mining**: http://www.rdatamining.com

- **Data Viz**:
  - ggplot2 Cheat Sheet (RECOMMENDED): http://zevross.com/blog/2014/08/04/beautiful-plotting-in-r-a-ggplot2-cheatsheet-3/
  - ggplot2 theoretical tutorial (detailed but RECOMMENDED): http://www.ling.upenn.edu/~joseff/avml2012/
  - Examples of base R, ggplot2, and rCharts: http://patilv.com/Replication-of-few-graphs-charts-in-base-R-ggplot2-and-rCharts-part-1-base-R/
  - Intro to ggplot2: http://heather.cs.ucdavis.edu/~matloff/GGPlot2/GGPlot2Intro.pdf
- **Interactive Visualisations**:
  - Interactive graphics (rCharts, jQuery): http://www.computerworld.com/article/2473365/business-intelligence/business-intelligence-106897-how-to-turn-csv-data-into-interactive-visualizations-with-r-and-rchart.html

---

- **Statistics**:
  - Detailed Statistics Primer: http://health.adelaide.edu.au/psychology/ccs/docs/lsr/lsr-0.3.pdf
  - Beginner guide to statistical topics in R: http://www.cyclismo.org/tutorial/R/
- **Linear Models**: http://data.princeton.edu/R/gettingStarted.html
- **Time Series Analysis**: https://www.otexts.org/fpp/resources
- **Little Book of R series**:
  - Time Series: http://a-little-book-of-r-for-time-series.readthedocs.org/en/latest/
  - Biomedical Statistics: http://a-little-book-of-r-for-biomedical-statistics.readthedocs.org/en/latest/
  - Multivariate Statistics: http://little-book-of-r-for-multivariate-analysis.readthedocs.org/en/latest/

- **RStudio Cheat Sheets**:
  - RStudio IDE: http://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf
  - Data Wrangling (dplyr & tidyr): https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

  - Data Viz (ggplot2): https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf
  - Reproducible Reports (markdown): https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf
  - Interactive Web Apps (shiny): https://www.rstudio.com/wp-content/uploads/2015/02/shiny-cheatsheet.pdf

---

8.3.2 Specialist Topics

- **Google Analytics**: http://online-behavior.com/analytics/r
- **Spatial Cheat Sheet**: http://www.maths.lancs.ac.uk/~rowlings/Teaching/UseR2012/cheatsheet.html
- **Translating between R and SQL**: http://www.burns-stat.com/translating-r-sql-basics/
- **Google's R style guide**: https://google.github.io/styleguide/Rguide.xml

---

8.3.3 Operational Basics

- **Working Directory**:
  Example on a mac = `setwd("~/Desktop/R")` or `setwd("/Users/CRT/Desktop/R")`
  Example on windows = `setwd("C:/Desktop/R")`

- **Help**:
  `?functionName`
  `example(functionName)`
  `args(functionName)`
  `help.search("your search term")`

- **Assignment Operator**: `<-`

8.4 Getting Your Data into R

1. Loading Existing Local Data

(a) When already in the working directory where the data is

Import a local **csv** file (i.e. where data is separated by **commas**), saving it as an object:

```
#this will create a data frame called "object"
#the header argument is defaulted to TRUE, i.e. read.csv assumes your file has a heade
```

```
object <- read.csv("xxx.csv")

#if your csv does not have a header row, add header = FALSE to the command
#in this call default column headers will be assigned which can be changed
object <- read.csv("xxx.csv", header = FALSE)
```

Import a local tab delimited file (i.e. where data is separated by **tabs**), saving is as an object:

(b) When NOT in the working directory where the data is

For example to import and save a local **csv** file from a different working directory you can either need to specify the file path (operating system specific), e.g.:

```
#on a mac
object <- read.csv("~/Desktop/R/data.csv")

#on windows
object <- read.csv("C:/Desktop/R/data.csv")
```

OR

You can use the file.choose() command which will interactively open up the file dialog box for you to browse and select the local file, e.g.:

```
object <- read.csv(file.choose())
```

(c) Copying and Pasting Data

For relatively small amounts of data you can do an equivalent copy paste (operating system specific):

```
#on a mac
object <- read.table(pipe("pbpaste"))

#on windows
object <- read.table(file = "clipboard")
```

2. Loading Non-Numerical Data - character strings

Be careful when loading text data! R may assume character strings are statistical factor variables, e.g. "low", "medium", "high", when are just individual labels like names. To specify text data NOT to be converted into factor variables, add `stringsAsFactor = FALSE` to your `read.csv/read.table` command:

```
object <- read.table("xxx.txt", stringsAsFactors = FALSE)
```

3. Downloading Remote Data

For accessing files from the web you can use the same `read.csv/read.table` commands. However, the file being downloaded does need to be in an R-friendly format (maximum of 1 header row, subsequent rows are the equivalent of one data record per row, no extraneous footnotes etc.). Here is an example downloading an online csv file from Pew Research:

```
object <- read.csv("https://vincentarelbundock.github.io/Rdatasets/csv/datasets/AirPas
```

4. Other Formats - Excel, SPSS, SAS etc.

For other file formats, you will need specific R packages to import these data.

Here's a good site for an overview: http://www.statmethods.net/input/importingdata.html

Here's a more detailed site: http://r4stats.com/examples/data-import/

Here's some info on the `foreign` package for loading statistical software file types: http://www.ats.ucla.edu/stat/r/faq/inputdata_R.htm

---

8.5 Getting Your Data out of R

1. Exporting data

Navigate to the working directory you want to save the data table into, then run the command (in this case creating a tab delimited file): - write.table(object, “xxx.txt”, sep = ””)

2. Save down an R object Navigate to the working directory you want to save the object in then run the command:

- save(object, file = “xxx.rda”)

reload the object: - load(“xxx.rda”)

Chapter 9

Importing data into R

working with excel, csv, and tsv files in R

Import swimming_pools.csv correctly: pools pools <- read.csv("swimming_pools.csv", stringsAsFactors = FALSE) #With stringsAsFactors, you can tell R whether it should convert strings in the flat file to factors. Check the structure of pools str(pools)

Import hotdogs.txt: hotdogs hotdogs <- read.delim("hotdogs.txt", header = FALSE)

Summarize hotdogs summary(hotdogs)

Path to the hotdogs.txt file: path path <- file.path("data", "hotdogs.txt")

Import the hotdogs.txt file: hotdogs hotdogs <- read.table(path, sep = "", col.names = c("type","calories","sodium"))

Call head() on hotdogs head(hotdogs)

#————————————————————————

Load the readr package library(readr) #read_csv, read_tsv, and read_delim are part of this package

Import potatoes.csv with read_csv(): potatoes potatoes <- read_csv("potatoes.csv")

Column names properties <- c("area", "temp", "size", "storage", "method", "texture", "flavor", "moistness")

Import potatoes.txt: potatoes potatoes <- read_tsv("potatoes.txt", col_names = properties)

Call head() on potatoes head(potatoes)

Import potatoes.txt using read_delim(): potatoes potatoes <- read_delim("potatoes.txt", delim = "", col_names = properties)

Print out potatoes potatoes

Import 5 observations from potatoes.txt: potatoes_fragment potatoes_fragment <- read_tsv(“potatoes.txt”, skip = 6, n_max = 5, col_names = properties)

Import all data, but force all columns to be character: potatoes_char potatoes_char <- read_tsv(“potatoes.txt”, col_types = “cccccccc”, col_names = properties)

Print out structure of potatoes_char str(potatoes_char)

Import without col_types hotdogs <- read_tsv(“hotdogs.txt”, col_names = c(“type”, “calories”, “sodium”))

Display the summary of hotdogs summary(hotdogs)

The collectors you will need to import the data fac <- col_factor(levels = c(“Beef”, “Meat”, “Poultry”)) int <- col_integer()

Edit the col_types argument to import the data correctly: hotdogs_factor hotdogs_factor <- read_tsv(“hotdogs.txt”, col_names = c(“type”, “calories”, “sodium”), col_types = list(fac, int, int))

#————————————————————

load the data.table package library(data.table)

Import potatoes.csv with fread(): potatoes potatoes <- fread(“potatoes.csv”)

Print out potatoes potatoes

Import columns 6 and 8 of potatoes.csv: potatoes potatoes <- fread(“potatoes.csv”, select = c(6, 8))

Plot texture (x) and moistness (y) of potatoes plot(potatoes$texture, potatoes$moistness)

#————————————————————

Load the readxl package library(readxl)

Print the names of all worksheets excel_sheets(“urbanpop.xlsx”)

Read the sheets, one by one pop_1 <- read_excel(“urbanpop.xlsx”, sheet = 1) pop_2 <- read_excel(“urbanpop.xlsx”, sheet = 2) pop_3 <- read_excel(“urbanpop.xlsx”, sheet = 3)

Put pop_1, pop_2 and pop_3 in a list: pop_list pop_list <- list(pop_1, pop_2, pop_3)

Display the structure of pop_list str(pop_list)

Read all Excel sheets with lapply(): pop_list pop_list <- lapply(excel_sheets(“urbanpop.xlsx”), read_excel, path = “urbanpop.xlsx”)

Import the first Excel sheet of urbanpop_nonames.xlsx (R gives names): pop_a pop_a <- read_excel(“urbanpop_nonames.xlsx”, col_names = FALSE)

Import the first Excel sheet of urbanpop_nonames.xlsx (specify col_names): pop_b cols <- c("country", paste0("year_", 1960:1966)) pop_b <- read_excel("urbanpop_nonames.xlsx", col_names = cols)

Import the second sheet of urbanpop.xlsx, skipping the first 21 rows: urbanpop_sel urbanpop_sel <- read_excel("urbanpop.xlsx", sheet = 2, col_names = FALSE, skip = 21)

Print out the first observation from urbanpop_sel urbanpop_sel[1,]

#——————————————————————

Import a local file Similar to the readxl package, you can import single Excel sheets from Excel sheets to start your analysis in R. Load the gdata package library(gdata)

Import the second sheet of urbanpop.xls: urban_pop urban_pop <- read.xls("urbanpop.xls", sheet = "1967-1974")

Print the first 11 observations using head() head(urban_pop, n = 11)

Column names for urban_pop columns <- c("country", paste0("year_", 1967:1974))

Finish the read.xls call urban_pop <- read.xls("urbanpop.xls", sheet = 2, skip = 50, header = FALSE, stringsAsFactors = FALSE, col.names = columns)

Print first 10 observation of urban_pop head(urban_pop, n = 10)

Import all sheets from urbanpop.xls path <- "urbanpop.xls" urban_sheet1 <- read.xls(path, sheet = 1, stringsAsFactors = FALSE) urban_sheet2 <- read.xls(path, sheet = 2, stringsAsFactors = FALSE) urban_sheet3 <- read.xls(path, sheet = 3, stringsAsFactors = FALSE)

Extend the cbind() call to include urban_sheet3: urban_all urban <- cbind(urban_sheet1, urban_sheet2[-1], urban_sheet3[-1])

Remove all rows with NAs from urban: urban_clean urban_clean <- na.omit(urban)

Print out a summary of urban_clean summary(urban_clean)

#——————————————————————

When working with XLConnect, the first step will be to load a workbook in your R session with loadWorkbook(); this function will build a "bridge" between your Excel file and your R session.

Load the XLConnect package library(XLConnect)

Build connection to urbanpop.xlsx: my_book my_book <- loadWorkbook("urbanpop.xlsx")

Print out the class of my_book class(my_book)

List the sheets in my_book getSheets(my_book)

Import the second sheet in my_book readWorksheet(my_book, sheet = 2)

Import columns 3, 4, and 5 from second sheet in my_book: urbanpop_sel urbanpop_sel <- readWorksheet(my_book, sheet = 2, startCol = 3, endCol = 5)

Import first column from second sheet in my_book: countries countries <- readWorksheet(my_book, sheet = 2, startCol = 1, endCol = 1)

cbind() urbanpop_sel and countries together: selection selection <- cbind(countries, urbanpop_sel)

Add a worksheet to my_book, named "data_summary" createSheet(my_book, "data_summary")

Use getSheets() on my_book getSheets(my_book)

Create data frame: summ sheets <- getSheets(my_book)[1:3] dims <- sapply(sheets, function(x) dim(readWorksheet(my_book, sheet = x)), USE.NAMES = FALSE) summ <- data.frame(sheets = sheets, nrows = dims[1,], ncols = dims[2,])

Add data in summ to "data_summary" sheet writeWorksheet(my_book, summ, "data_summary")

Rename "data_summary" sheet to "summary" renameSheet(my_book, "data_summary", "summary")

Remove the fourth sheet removeSheet(my_book, 4)

Save workbook to "renamed.xlsx" saveWorkbook(my_book, file = "renamed.xlsx")

#———————————————————————————

Download various files with download.file() Here are the URLs! As you can see they're just normal strings csv_url <- "http://s3.amazonaws.com/assets.datacamp.com/production/course_1561/datasets/chickwts.csv" tsv_url <- "http://s3.amazonaws.com/assets.datacamp.com/production/course_3026/datasets/tsv_data.tsv"

Read a file in from the CSV URL and assign it to csv_data csv_data <- read.csv(file = csv_url)

Read a file in from the TSV URL and assign it to tsv_data tsv_data <- read.delim(file = tsv_url)

Examine the objects with head() head(csv_data) head(tsv_data)

Download the file with download.file() download.file(url = csv_url, destfile = "feed_data.csv")

Read it in with read.csv() csv_data <- read.csv(file = "feed_data.csv")

Add a new column: square_weight csv_data$square_{w}eight < -(csv_{d}ata$weight ^ 2)

Save it to disk with saveRDS() saveRDS(object = csv_data, file = “modified_feed_data.RDS”)

Read it back in with readRDS() modified_feed_data <- readRDS(file = “modified_feed_data.RDS”)

Examine modified_feed_data str(modified_feed_data)

#———————————————

Using data from API clients

#example 1 Load pageviews library for wikipedia library(pageviews)

Get the pageviews for “Hadley Wickham” hadley_pageviews <- article_pageviews(project = “en.wikipedia”, article = “Hadley Wickham”)

Examine the resulting object str(hadley_pageviews)

#example 2 Load birdnik library(birdnik)

Get the word frequency for “vector”, using api_key to access it vector_frequency <- word_frequency(key = api_key, words = “vector”)

#———————————————

Load the httr package library(httr)

Make a GET request to http://httpbin.org/get get_result <- GET(url = “http://httpbin.org/get”)

Print it to inspect it get_result

Make a POST request to http://httpbin.org/post with the body “this is a test” post_result <- POST(url = “http://httpbin.org/post”, body = “this is a test”)

Print it to inspect it post_result

Make a GET request to url and save the results pageview_response <- GET(url)

Call content() to retrieve the data the server sent back pageview_data <- content(pageview_response)

Examine the results with str() str(pageview_data)

Handling http failures fake_url <- “http://google.com/fakepagethatdoesnotexist”

Make the GET request request_result <- GET(fake_url)

Check request_result if(http_error(request_result)){ warning(“The request failed”) } else { content(request_result) } #———————————————

example start to finish

Load httr library(httr)

The API url base_url <- "https://en.wikipedia.org/w/api.php"

Set query parameters query_params <- list(action = "parse", page = "Hadley Wickham", format = "xml")

Get data from API resp <- GET(url = base_url, query = query_params)

Parse response resp_xml <- content(resp)

Load rvest library(rvest)

Read page contents as HTML page_html <- read_html(xml_text(resp_xml))

Extract infobox element infobox_element <- html_node(x = page_html, css ="“.infobox")

Extract page name element from infobox page_name <- html_node(x = infobox_element, css = ".fn")

Extract page name as text page_title <- html_text(page_name)

Your code from earlier exercises wiki_table <- html_table(infobox_element) colnames(wiki_table) <- c("key", "value") cleaned_table <- subset(wiki_table, !key == "")

Create a dataframe for full name name_df <- data.frame(key = "Full name", value = page_title)

Combine name_df with cleaned_table wiki_table2 <- rbind(name_df, cleaned_table)

Print wiki_table wiki_table2

Reproducibility

library(httr) library(rvest) library(xml2)

get_infobox <- function(title){ base_url <- "https://en.wikipedia.org/w/api.php"

Change "Hadley Wickham" to title query_params <- list(action = "parse", page = title, format = "xml")

resp <- GET(url = base_url, query = query_params) resp_xml <- content(resp)

page_html <- read_html(xml_text(resp_xml)) infobox_element <- html_node(x = page_html, css ="“.infobox") page_name <- html_node(x = infobox_element, css = ".fn")

#——————————————————

Construct a directory-based API URL to `http://swapi.co/api`, looking for person `1` in `people` directory_url <- paste("http://swapi.co/api", "people", "1", sep = "/")

Make a GET call with it result <- GET(directory_url)

Create list with nationality and country elements query_params <- list(nationality = "americans", country = "antigua")

Make parameter-based call to httpbin, with query_params parameter_response <- GET("https://httpbin.org/get", query = query_params)

Print parameter_response parameter_response

#————————————————————

Using user agents Informative user-agents are a good way of being respectful of the developers running the API you're interacting with. They make it easy for them to contact you in the event something goes wrong. I always try to include: #My email address; #A URL for the project the code is a part of, if it's got a URL.

Do not change the url url <- "https://wikimedia.org/api/rest_v1/metrics/pageviews/per-article/en.wikipedia/all-access/all-agents/Aaron_Halfaker/daily/2015100100/2015103100"

Add the email address and the test sentence inside user_agent() server_response <- GET(url, user_agent("my@email.address this is a test"))

Rate-limiting The next stage of respectful API usage is rate-limiting: making sure you only make a certain number of requests to the server in a given time period. Your limit will vary from server to server, but the implementation is always pretty much the same and involves a call to Sys.sleep(). This function takes one argument, a number, which represents the number of seconds to "sleep" (pause) the R session for. So if you call Sys.sleep(15), it'll pause for 15 seconds before allowing further code to run.

Construct a vector of 2 URLs urls <- c("http://httpbin.org/status/404", "http://httpbin.org/status/301")

for(url in urls){ Send a GET request to url result <- GET(url) Delay for 5 seconds between requests Sys.sleep(5) }

Tying it all together get_pageviews <- function(article_title){ url <- paste("https://wikimedia.org/api/rest_v1/metrics/pageviews/per-article/en.wikipedia/all-access/all-agents", article_title, "daily/2015100100/2015103100", sep = "/")
response <- GET(url, user_agent("my@email.com this is a test")) Is there an HTTP error? if(http_error(response)){ Throw an R error stop("the request failed") } Return the response's content content(response) } #——————

working with JSON files (for more information see: www.json.org) While JSON is a useful format for sharing data, your first step will often be to parse it into an R object, so you can manipulate it with R.

Get revision history for “Hadley Wickham” resp_json <- rev_history(“Hadley Wickham”)

Check http_type() of resp_json http_type(resp_json) confirm the API returned a JSON object

Examine returned text with content() content(resp_json, as = “text”)

Parse response with content() content(resp_json, as = “parsed”)

Parse returned text with fromJSON() library(jsonlite) fromJSON(content(resp_json, as = “text”))

Manipulating parsed JSON Load rlist library(rlist)

Examine output of this code str(content(resp_json), max.level = 4)

Store revision list revs <- content(resp_json)*query*pages`41916270`revisions

Extract the user element user_time <- list.select(revs, user, timestamp)

Print user_time user_time

Stack to turn into a data frame list.stack(user_time)

Load dplyr library(dplyr)

Pull out revision list revs <- content(resp_json)*query*pages`41916270`revisions

Extract user and timestamp revs %>% bind_rows() %>% select(user, timestamp)

#———————————————————————— working with XML files Just like JSON, you should first verify the response is indeed XML with http_type() and by examining the result of content(r, as = “text”). Then you can turn the response into an XML document object with read_xml()

Load xml2 library(xml2)

Get XML revision history resp_xml <- rev_history(“Hadley Wickham”, format = “xml”)

Check response is XML http_type(resp_xml)

Examine returned text with content() rev_text <- content(resp_xml, as = “text”) rev_text

Turn rev_text into an XML document rev_xml <- read_xml(rev_text)

Examine the structure of rev_xml xml_structure(rev_xml)

Extracting XML data Find all nodes using XPATH “/api/query/pages/page/revisions/rev” xml_find_all(rev_xml, “/api/query/pages/page/revisions/rev”)

Find all rev nodes anywhere in document rev_nodes <- xml_find_all(rev_xml, “//rev”)

Use xml_text() to get text from rev_nodes xml_text(rev_nodes)

Extracting XML attributes All rev nodes rev_nodes <- xml_find_all(rev_xml, “//rev”)

The first rev node first_rev_node <- xml_find_first(rev_xml, “//rev”)

Find all attributes with xml_attrs() xml_attrs(first_rev_node)

Find user attribute with xml_attr() xml_attr(first_rev_node, “user”)

Find user attribute for all rev nodes xml_attr(rev_nodes, “user”)

Find anon attribute for all rev nodes xml_attr(rev_nodes, “anon”)

returning nice API output get_revision_history <- function(article_title){ Get raw revision response rev_resp <- rev_history(article_title, format = “xml”) }

Turn the content() of rev_resp into XML rev_xml <- read_xml(content(rev_resp, “text”))

Find revision nodes rev_nodes <- xml_find_all(rev_xml, “//rev”)

Parse out usernames user <- xml_attr(rev_nodes, “user”)

Parse out timestamps timestamp <- readr::parse_datetime(xml_attr(rev_nodes, “timestamp”))

Parse out content content <- xml_text(rev_nodes)

#——————————————————————

web scraping 101 The first step with web scraping is actually reading the HTML in. This can be done with a function from xml2, which is imported by rvest - read_html(). This accepts a single URL, and returns a big blob of XML that we can use further on.

Load rvest library(rvest)

Hadley Wickham’s Wikipedia page test_url <- “https://en.wikipedia.org/wiki/Hadley_Wickham”

Read the URL stored as “test_url” with read_html() test_xml <- read_html(test_url)

Print test_xml test_xml

#html_node(), which extracts individual chunks of HTML from a HTML document. There are a couple of ways of identifying and filtering nodes, and for now we’re going to use XPATHs: unique identifiers for individual pieces of a HTML document.

Use html_node() to grab the node with the XPATH stored as `test_node_xpath` node <- html_node(x = test_xml, xpath = test_node_xpath)

Print the first element of the result node[[1]]

Extract the name of table_element element_name <- html_name(table_element)

Print the name element_name

Extract the element of table_element referred to by second_xpath_val and store it as page_name page_name <- html_node(x = table_element, xpath = second_xpath_val)

Extract the text from page_name page_title <- html_text(page_name)

Print page_title page_title

Turn table_element into a data frame and assign it to wiki_table wiki_table <- html_table(table_element)

Print wiki_table wiki_table

Cleaning a data frame Rename the columns of wiki_table colnames(wiki_table) <- c("key", "value")

Remove the empty row from wiki_table cleaned_table <- subset(wiki_table, !key == "")

Print cleaned_table cleaned_table

#——————————————————————

CSS web scraping CSS is a way to add design information to HTML, that instructs the browser on how to display the content. You can leverage these design instructions to identify content on the page.

Select the table elements html_nodes(test_xml, css = "table")

Select elements with class = "infobox" html_nodes(test_xml, css = ".infobox")

Select elements with id = "firstHeading" html_nodes(test_xml, css = "#firstHeading")

Extract element with class infobox infobox_element <- html_nodes(test_xml, css = ".infobox")

Get tag name of infobox_element element_name <- html_name(infobox_element)

Print element_name element_name

Extract element with class fn page_name <- html_node(x = infobox_element, css = ".fn")

Get contents of page_name page_title <- html_text(page_name)

Print page_title page_title