Codes for STEM

Noushin Nabavi

2020-09-11

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

Chapter 3

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.

3.1 Usage demonstrations

3.1.1 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.

3.1.2 Code chunks

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

```r
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
```

3.1.2.1 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)
```

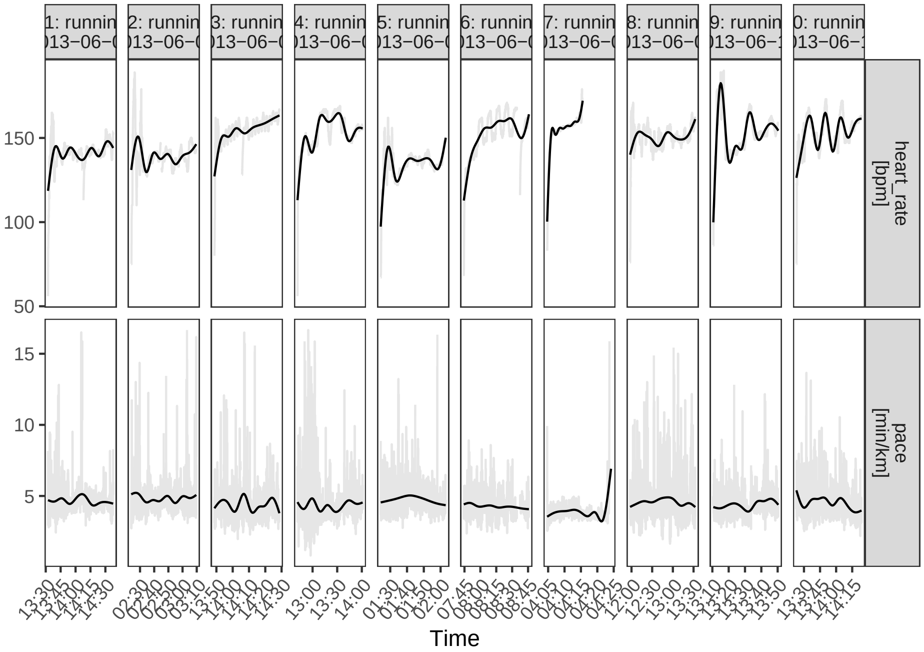
	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-17.58	6.76	-2.60	0.01
speed	3.93	0.42	9.46	0.00

3.1.2.2 Include figures

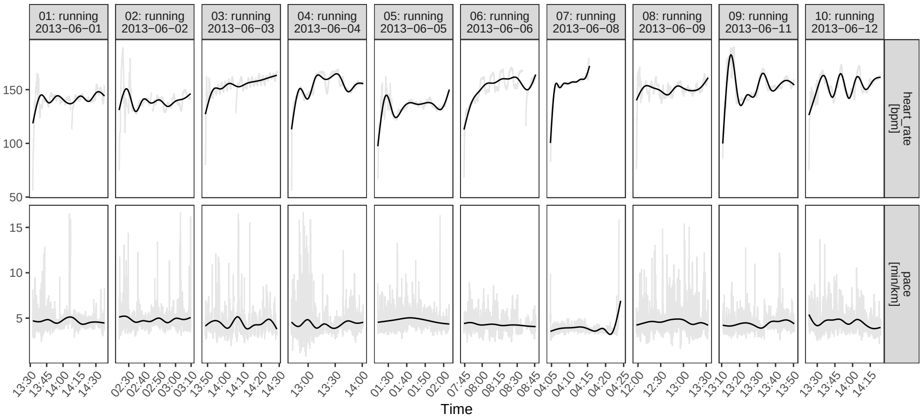
The **trackeR** package provides infrastructure for running and cycling data in **R** and is used here to illustrate how figures can be included.

```
## 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.



3.2 Resources

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

Chapter 4

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

4.1 Contents

A. Essentials

B. Basics

* 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()...`

4.2 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

4.3 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 withi

#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 orde
painters[order(-painters$Composition),] #append a minus sign in front of the variable you want t

#AIM: order the dataset rows based on the painters' Composition Score column, in Descending orde
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 al

#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
painters[c(1:5), -4] #returns the subsetted df having deselected the 4th column, Expression and


#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 wi
painters[which(painters$School != "A" & painters$Colour > 10),] #returns the subsetted df where
```

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 t
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.

```r
#AIM: to create a simplified take-home pay calculator (single-band), called 'takehome_pay'. Our
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 5

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()
```

```r
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.

```r
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().

```r
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**

```r
# 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 6

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
##  $ 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)
```

```
##    trip_id duration          start_date                        start_station
## 1:  139547     1523 2014-01-01 00:17:00               Embarcadero at Sansome
## 2:  139558     1600 2014-01-01 00:28:00 Harry Bridges Plaza (Ferry Building)
##    start_terminal            end_date     end_station end_terminal bike_id
## 1:             60 2014-01-01 00:42:00   Beale at Market           56     331
## 2:             50 2014-01-01 00:54:00 Steuart at Market           74     413
##    subscription_type zip_code
## 1:        Subscriber    94112
## 2:        Subscriber    94102
```

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 %>%
  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:

```
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:

```
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:

```
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)
```

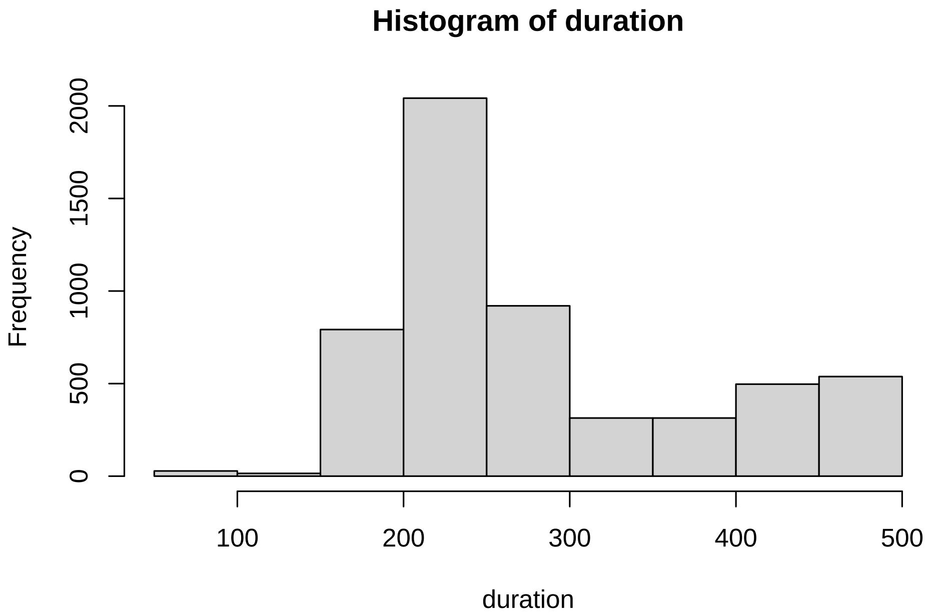
```
##    mean_duration           last_ride
## 1:      1131.967 2015-06-24 20:18:00
```

```
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_station)]
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_station, month(st
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:

```
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:

```
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

```
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:

```
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)

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

Find the row corresponding to the shortest trip per month:

```
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:

```
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:

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

Fix/edit spelling in the second row of start_station:

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

Replace negative duration values with NA:

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

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

```
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():

```
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:

```
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
##  $ 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:

```r
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:

```r
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:

```r
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 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 ...
```

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")

6.1 Beginner Resources by Topic

6.1.1 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

---

6.1.2 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

---

6.1.3 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**: `<-`

6.2 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 header row and
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/AirPass
```

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

---

6.3 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")
