Exploring, Visualizing, and Modeling Big Data with R

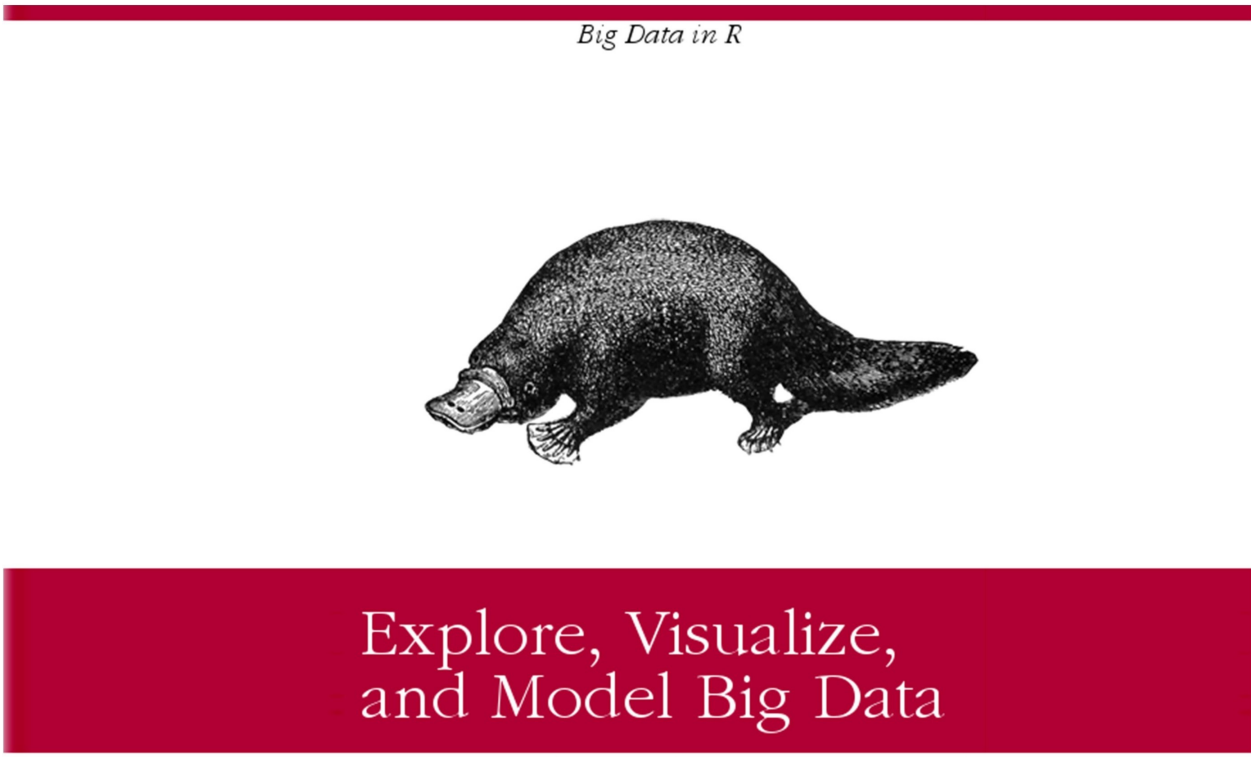
Okan Bulut
Christopher Desjardins

2019-04-04

Contents

Chapter 1

Preface



1.1 Summary

Working with **BIG DATA** requires a particular suite of data analytics tools and advanced techniques, such as machine learning (ML). Many of these tools are readily and freely available in R. This full-day session will provide participants with a hands-on training on how to use data analytics tools and machine learning methods available in R to explore, visualize, and model big data.

The first half of our training session will focus on organizing (manipulating and summarizing) and visualizing (both statically and dynamically) big data in R. The second half will involve a series of short lectures on ML techniques (decision trees, random forests, and support vector machines), as well as hands-on demonstrations

applying these methods in R. Examples will be drawn from the OECD's Programme for International Student Assessment (PISA). Participants will get opportunities to work through several hands-on lab sessions throughout the day.

1.2 Who we are

1. **Okan Bulut – University of Alberta**

    - Associate Professor of educational measurement and psychometrics at the University of Alberta
    - 10+ years using the R programming language for data analysis and visualization
    - Specialized in the analysis and visualization of big data (mostly from large-scale assessments)
    - 6+ years teaching courses and workshops on statistics, psychometrics, and programming with R
    - https://github.com/okanbulut and https://sites.ualberta.ca/~bulut/
    - **E-mail:** bulut@ualberta.ca

2. **Christopher D. Desjardins – University of Minnesota**

    - Research Associate at the Research Methodology Consulting Center.
    - R user since 2006 and contributer since 2009.
    - https://github.com/cddesja and https://cddesja.github.io/
    - **E-mail:** cdesjard@umn.edu

We also co-authored:

- Two R packages – profileR for profile analysis of multivariate data and hemp for psychometric analysis of assessment data
- A recent book titled Handbook of Educational Measurement and Psychometrics Using R

Chapter 2

Introduction

2.1 What is big data?

When we handle a data-related problem, how do we know that we are actually dealing with "big data"? What is "big data"? What characteristics make a dataset big? The following three characteristics (three Vs of big data, source: TechTarget) can help us define the size of data:

1. **Volume**: The number of rows or cases (e.g., students) and the number of columns or variables (e.g., age, gender, student responses, response times)
2. **Variety**: Whether there are secondary sources or data that expand the existing data even further
3. **Velocity**: Whether real-time data are being used

2.2 Why is big data important?

Nowadays nearly every private and public sector of industry, commerce, health, education, and so forth are talking about big data. Data is a strategic and valuable asset when we know which questions we want to answer (see Bernard Marr's article titled Big Data: Too Many Answers, Not Enough Questions). Therefore, it is very important to identify the right questions at the beginning of data collection. More data with appropriate questions can yield quality answers that we can use for better decision-making. However, too much data without any purpose may obfuscate the truth.

Currently big data is used to better understand customers and their behaviors and preferences. Consider Netflix – one of the world's leading subscription services for watching movies and TV shows online. They use big data – such as customers' ratings for each movie and TV show and when customers subscribe/unsubscribe – to make better recommendations for existing customers and to convince more customers to subscribe. Target, a big retailer in the US, implements data mining techniques to predict pregnancies of their shoppers and send them a sale booklet for baby clothes, cribs, and diapers (see this interesting article). Car insurance companies analyze big data from their customers to understand how well their customers actually drive and how much they need to charge each customer to make a profit.

In education, there is no shortage of big data. Student records, teacher observations, assessment results, and other student-related databases make tons of information available to researchers and practitioners. With the advent of new technologies such as facial recognition software and biometric signals, now we get access to a variety of visual and audio data on students. In the context of educational testing and psychometrics, big data can help us to assess students more accurately, while continuously monitoring their progress via learning analytics. We can use log data and response times to understand students' engagement with the test, whether they were cheating, and whether they had pre-knowledge of the items presented on the test.

Figure 2.1: Three Vs of big data

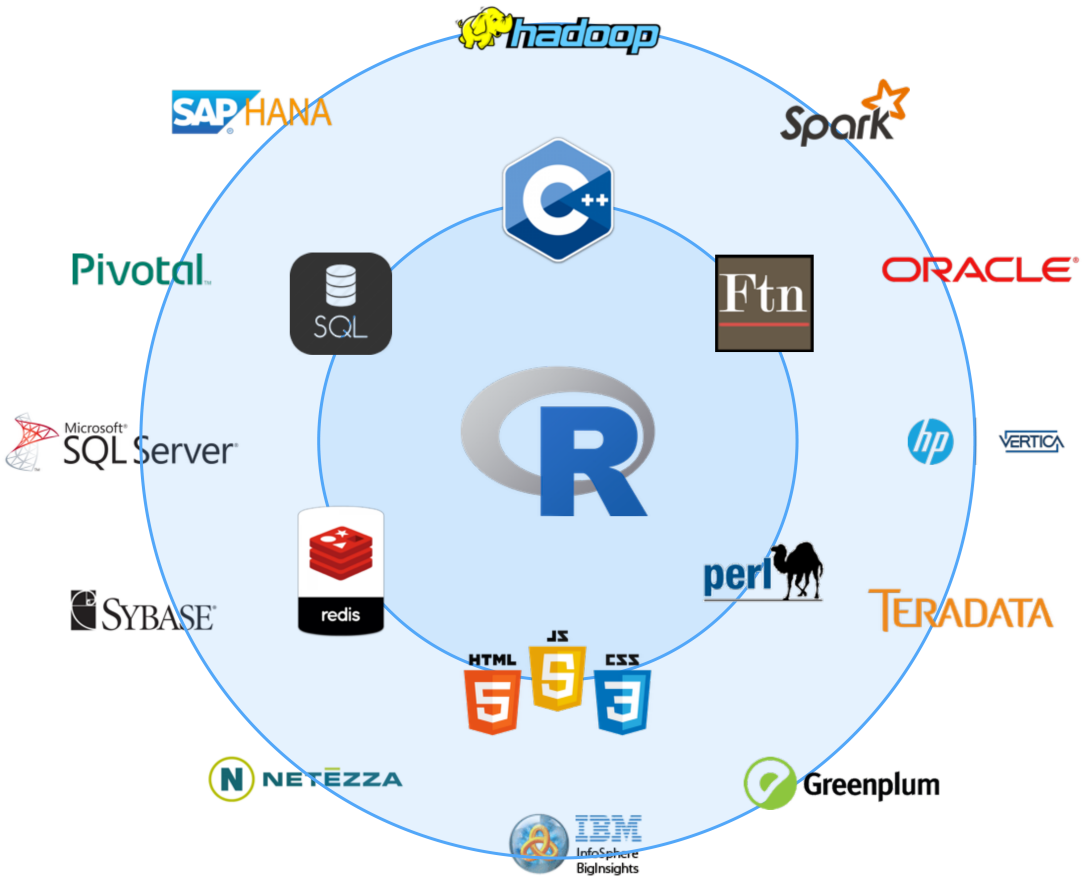


Figure 2.2: Other big data programs integrated with R

2.3 How do we analyze big data?

Big data analysis often begins with reading and then extracting the data. First, we need to read the data into a software program – such as R – and then manage it properly. Second, we need to extract a subset, sample, or summary from the big data. Due to its size, even a subset of the big data might itself be quite large. Third, we need to repeat computation (e.g., fitting a model) for many subgroups of the data (e.g., for each individual or by larger groups that combine individuals based on a particular characteristic). Therefore, we need to use the right tools for our data operations. For example, we may need to store big data in a data warehouse (either a local database or a cloud system) and then pass subsets of data from the warehouse to the local machine where we are analyzing the data.

R, maintained by the R Core Team, has its packages (collect of R functions) available on this The Comprehensive R Archive Network (CRAN). It used to be considered an *inadequate* programming language for big data (see Douglass Merril's article from 2012). Fortunately, today's R, with the help of RStudio and many data scientists, is capable of running most analytic tasks for big data either alone or with the help of other programs and programming languages, such as Spark, Hadoop, SQL, and C++ (see Figure 2.2). R is an amazing data science programming tool, it has a myriad statistical techniques available, and can readily translate the results of our analyses into colourful graphics. There is no doubt that R is one of the most preferred programming tool for statisticians, data scientists, and data analysts who deal with big data on a daily basis.

Some general suggestions on big data analysis include:

1. Obtain a strong computer (multiple and faster CPUs, more memory)
2. If memory is a problem, access the data differently or split up the data
3. Preview a subset of big data using a program, **not** the entire raw data.
4. Visualize either a subset of data or a summary of the big data, **not** the entire raw data.
5. Calculate necessary summary statistics manually, **not** for all variables in big data.
6. Delay computationally expensive operations (e.g., those that require large memory) until you actually need them.
7. Consider using parallel computing – parallel and foreach packages and cloud computing
8. Profile big tasks (in R) to cut down on computational time

```
start_time <- proc.time()

# Do all of your coding here

end_time <- proc.time()
end_time - start_time

# Alternatively,

system.time({

 # Do all of your coding here

})
```

During this training session, we will follow these steps and demonstrate how each one helps us explore, visualize, and model big data in R.

2.4 Additional resources

There are dozens of online resources and books on big data analysis. Here are a few of them that we recommend you check out:

- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). An introduction to statistical learning with applications in R. New York, NY: Springer. (Freely available from the authors' website: http://www-bcf.usc.edu/~gareth/ISL/index.html)
- Grolemund, G., & Wickham, H. (2016). R for data science. Sebastopol, CA: O'Reilly Media, Inc. (Freely available from the authors' website: http://r4ds.had.co.nz/)
- Baumer, B. S., Kaplan, D. T., & Horton, N. J. (2017). Modern data science with R. Boca Raton, FL: CRC Press.
- Romero, C., Ventura, S., Pechenizkiy, M., & Baker, de, R. S. J. (Eds.) (2011). Handbook of educational data mining. (Chapman and Hall/CRC data mining and knowledge discovery series). Boca Raton: CRC Press.
- DataCamp: https://www.datacamp.com/tracks/big-data-with-r
- RStudio: https://www.rstudio.com/resources/webinars/working-with-big-data-in-r/

2.5 PISA dataset

In this training session, we will use the 2015 administration of the OECD's Programme for International Student Assessment (PISA). PISA is a large-scale, international assessment that involves students, parents, teachers, and school principals from all over the world as participants. Every three years, PISA tests 15-year-old students from all over the world in reading, mathematics and science. The tests are designed to gauge how well the students master key subjects in order to be prepared for real-life situations in the adult world.

In addition to assessing students' competencies, PISA also aims to inform educational policies and practices for the participating countries and economies by providing additional information obtained from students, parents, teachers, and school principals through the questionnaires. Students complete a background questionnaire with questions about themselves, their family and home, and their school and learning experiences. School principals complete a questionnaire about the system and learning environment in schools. In some countries, teachers and parents also complete optional questionnaires to provide more information on their perceptions and expectations regarding students. In this training session, we specifically focus on the assessment data and the background questionnaire that all participating students are required to complete.

The 2015 administration of PISA involves approximately 540,000 15-year-old students from 72 participating countries and economies. During this training session, we will sometimes use the entire dataset or take a subset of the PISA dataset to demonstrate the methods used for exploring, visualizing, and modeling big data. For more details about the PISA dataset and its codebooks, please see the PISA website.

The three data files that we will use in this training session can be downloaded using the following links. Please download and unzip the files to follow the examples that we will demonstrate in this training session.

- http://bit.ly/2VleDPZ (all PISA records – 331.65 MB)
- http://bit.ly/2Uf2mQA (only 6 regions with 17 countries – 103.76 MB)
- http://bit.ly/2YNzei0 (randomly selected cases from 6 regions – 22.92 MB)

Chapter 3

Exploratory data analysis

3.1 What is exploratory data analysis?

> Exploratory data analysis is detective work – numerical detective work – or counting detective work – or graphical detective work.
>
> To learn about data analysis, it is right that each of us try many things that do not work

Exploratory data analysis (EDA) is an **iterative, hypothesis-generating framework**. Through EDA, we hope to **uncover new relationships** among the variables in our data. In EDA, our job is the accumulation of evidence, preferably novel evidence and the widespread availability of powerful computers and able statistical tools means accumulating evidence is much easier now than ever before. But everything we find may not be meaningful. Nonetheless, our job is not to evaluate what we've learned but rather discovery and EDA "is an attitude, a state of flexibility, a willingness to look for those things that we believe are not there, as well as those we believe to be there." It is for researchers building on our sleuthing to evaluate whether what we've found is real or not, but it's not our focus in EDA. Regardless of the ultimate fate of what we've found, EDA is vital to science as it challenges our dogmas about the relationships among variables and it forces us to face the fact that our theories may be wrong. It pushes science forwards and provides a framework for designing new studies and experiments to confirm/refute what we've found. We're the detectives, not the judges.

3.2 Confirmatory data analysis

> You can only use an observation once to confirm a hypothesis. As soon as you use it more than once you're back to doing exploratory analysis. This means to do hypothesis confirmation you need to "preregister" (write out in advance) your analysis plan, and not deviate from it even when you have seen the data. Grolemund & Wickham, (2017)

3.3 A framework for EDA

Figure 3.2 shows a framework for data science presented in Grolemund & Wickham (2017). This model is equally applicable for EDA.

The first step in any analysis involves **importing** the data into the software. Depending on how you import data into R, this step can be relatively instantaneous or it can take minutes.

The second step is **tidying** or **reshaping** your data. Grolemund & Wickham's criteria for tidy data are

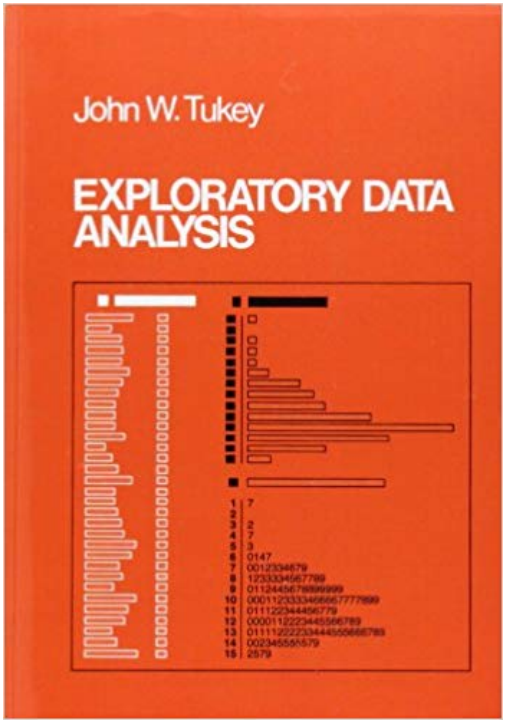


Figure 3.1: The EDA classic.

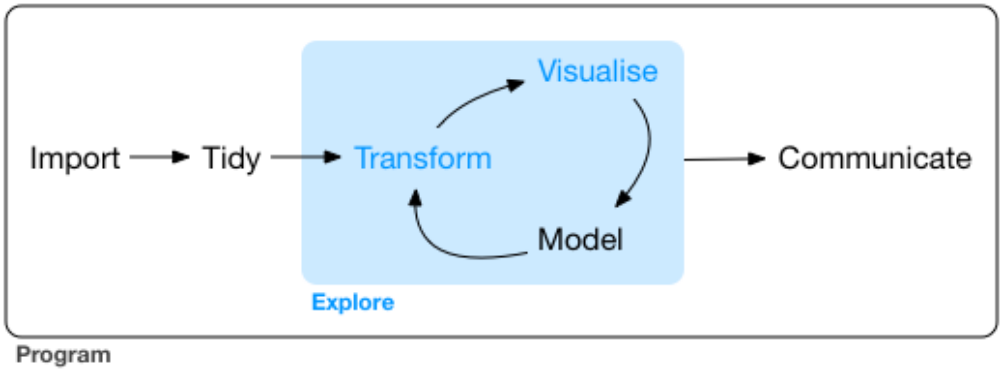


Figure 3.2: Grolemund and Wickham (2017) model of data science.

id	x	y
1	22.19	24.05
2	19.82	22.91
3	19.81	21.19
4	17.49	18.59
5	19.44	19.85

(a) Data for paired t test

id	variable	value
1	x	22.19
2	x	19.82
3	x	19.81
4	x	17.49
5	x	19.44
1	y	24.05
2	y	22.91
3	y	21.19
4	y	18.59
5	y	19.85

(b) Data for mixed effects model

Table 14: Two data sets for performing the same test.

Figure 3.3: Table 11 from Wickham (2014). Which data set is tidy?

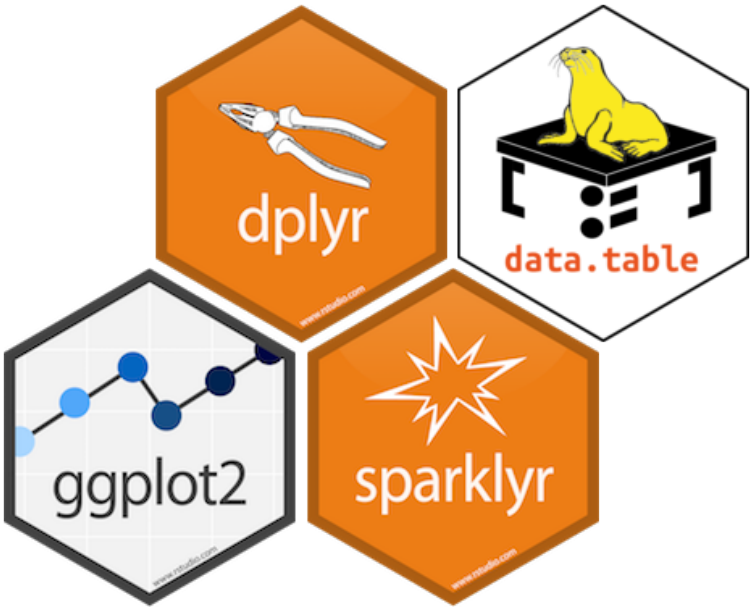
1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

Note that this does not explicit refer to data being in a wide or long format. Table 11 in Wickham (2014) in the Journal of Statistical Software, shown below in Figure 3.3, highlights this issue. In this particular example, without knowing what X and Y are (e.g., do they code for 1) measurements on two occasions? 2) two items on the same assessment? 3) the height and weight of the participants?) we do not know which format is the tidy one.

The next steps in EDA are iterative and they involve **transforming** variables (e.g., changing type, rescaling, creating new ones, etc), **visualizing** these variables (e.g., marginal, bivariate, multivariate plots), and **modeling** the relationships (can these variables predict/classify our outcomes).

Finally, once an interesting relationship has been discovered it must be **communicated** and any, arguably all, of these three would be communicated.

3.4 EDA tools


dplyr
data.table
ggplot2
sparklyr

Chapter 4

Wrangling big data

Data wrangling is a general term that refers to transforming data. Wrangling could involve subsetting, recoding, and transforming variables. For the workshop, we'll also include summarizing data as wrangling as it fits within our discussion of the `data.table` and `sparklyr` packages. However, summarizing might more appropriately occur during data exploration/initial data analysis.

4.1 What is `data.table`?

From the `data.table` wiki

> It is a high-performance version of base R's `data.frame` with syntax and feature enhancements for ease of use, convenience and programming speed.

Its syntax is designed to be concise and consistent. It's somewhat similar to base R, but arguably less intuitive than `tidyverse`. We, and many others, would say that `data.table` is one of the most underrated package out there.

If you're familiar with SQL, then working with a `data.table` (DT) is conceptually similar to querying.

```
DT[i, j, by]

  R:                 i                 j        by
SQL:  where | order by   select | update  group by
```

This should be read as take `DT`, subset (or order) rows using `i`, then calculate `j`, and group by `by`. A graphical depiction of this "grammar", created by one of the developers of `data.table`, is shown in Figure 4.1.

The `data.table` package needs to be installed and loaded throughout the workshop.

```
install.packages("data.table")
library(data.table)
```

Throughout the workshop, we will write DT code as:

```
DT[i,
   j,
   by]
```

That is, we will use write separate lines for the i, j, and by DT statements.

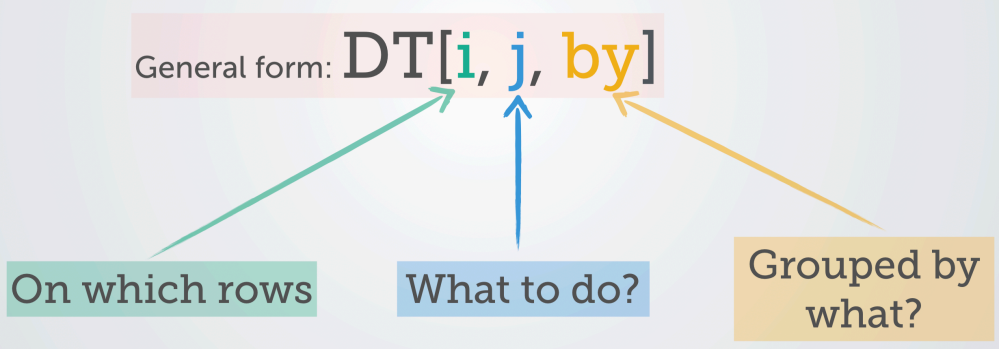


Figure 4.1: Source: https://tinyurl.com/yyepwjpt.

4.1.1 Why use `data.table` over `tidyverse`?

If you're familiar with R, then you might wonder why we are using DT and not `tidyverse`? This has to do with memory management and speed.

```
#
# Benchmark #1 - Reading in data
#

system.time({read.csv("data/pisa2015.csv")})
system.time({fread("data/pisa2015.csv", na.strings = "")})
system.time({read_csv("data/pisa2015.csv")})

#
# Benchmark #2 - Calculating a conditional mean
#

#' Calculate proportion that strongly agreed to an item
#' @param x likert-type item as a numeric vector
getSA <- function(x, ...) mean(x == "Strongly agree", ...)

# read in data using fread()
pisa <- fread("data/pisa2015.csv", na.strings = "")

# calculate conditional means
# This is the proportion of students in each country that
# strongly agree that
# "I want top grades in most or all of my courses."
benchmark(
  "baseR" = {
    X <- aggregate(ST119Q01NA ~ CNTRYID, data = pisa, getSA, na.rm = TRUE)
  },
  "data.table" = {
    X <- pisa[,
              getSA(ST119Q01NA, na.rm = TRUE),
              by = CNTRYID]
  },
  "tidyverse" = {
    X <- pisa %>%
      group_by(CNTRYID) %>%
```

Table 4.1: Comparing base R, data.table, and tidyverse.

Method	Reading in data	Conditional mean (1000 times)
base R	225.51	196.59
data.table	46.80	27.73
tidyverse	233.72	159.22

```
    summarize(getSA(ST119Q01NA, na.rm = TRUE))
},
replications = 1000)
```

Table 4.1 shows the results of this (relatively) unscientific minibenchmark. The first column is the method, the second column is elapsed time (in seconds) to read in the **pisa** data set (only once, though similar results/pattern is found if repeated), and the third column is the elapsed time (in seconds) to calculate the conditional mean 1000 times. We see that `data.table` is substantially faster than base R and the `tidyverse`.

This extends to other data wrangling procedures (e.g., reshaping, recoding). Importantly, `tidyverse` is not designed for big data but instead for data science, more generally. From Grolemund & Wickham (2017)

> "This book (R for Data Science) proudly focuses on small, in-memory datasets. This is the right place to start because you cannot tackle big data unless you have experience with small data. The tools you learn in this book will easily handle hundreds of megabytes of data, and with a little care you can typically use them to work with 1-2 Gb of data. If you are routinely working with larger data (10-100 Gb, say), you should learn more about data.table. This book does not teach data.table because it has a very concise interface which makes it harder to learn since it offers fewer linguistic cues. But if you are working with large data, the performance payoff is worth the extra effort required to learn it."

4.2 Reading/writing data with `data.table`

The `fread` function should always be used when reading in large data sets and arguably when ever you read in a CSV file. As shown above, `read.csv` and `readr::read_csv` are painfully slow with big data.

Throughout the workshop we'll be using the **pisa** data set. Therefore, we begin by reading in (or importing) the data set

```
pisa <- fread("data/pisa2015.csv", na.strings = "")
```

To see the **class** the object `pisa` is and how big it is in R

```
class(pisa)
```

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

```
print(object.size(pisa), unit = "GB")
```

```
## 3.5 Gb
```

We see that objects that are read in with `fread` are of class `data.table` and `data.frame`. That means that methods for data.tables and data.frames will work on these objects. We also see this data set uses up 3.5 Gb of memory and this is all in the memory (RAM) not on the disk and allocated to memory dynamically (this is what SAS does).

If we wanted to write `pisa` back to a CSV to share with a colleague or to use in another program after some wrangling, then we should use the `fwrite` function instead of `write.csv`:

		Laptop SSD 4core/16gb 10m rows		Server 32core/256gb 100m rows		
		Time Sec	Size GB	RamDisk Time	HDD Time	Size GB
fwrite(DT,"fwrite.csv")	csv	2	0.8	9	61	7.5
write_feather(DT, "feather.bin")	bin	5	1.0	27	75	9.1
save(DT,file="save1.Rdata",compress=F)	bin	11	1.2	90	137	12.0
save(DT,file="save2.Rdata",compress=T)	bin	70	0.4	647	679	2.8
write.csv(DT,"write.csv.csv",**)	csv	63	0.8	749	824	7.3
readr::write_csv(DT,"write_csv.csv")	csv	132	0.8	1997	1571	7.3

[**] row.names=F,quote=F

Figure 4.2: Time to write an R object to a file. Source: https://tinyurl.com/y366kvfx.

```
fwrite(pisa, file = "pisa2015.csv")
```

The following image (Figure 4.2), taken from Matt Dowle's blog, shows the speed difference using common ways to save R objects and the differences in sizes of these files.

In the event that you did **just** want to read the data in using the `fread()` function but then wanted to work with a tibble (tidyverse) or a data.frame, you can convert the data set after its been read in:

```
pisa.tib <- tibble::as_tibble(pisa)
pisa.df <- as.data.frame(pisa)
```

However, I strongly recommend against this approach unless you have done some amount of subsetting. If your data set is large enough to benefit appreciably by `fread` then you should try and use the `data.table` package.

For the workshop, we have created two smaller versions of the **pisa** data set for those of you with less beefy computers. The first is a file called `region6.csv` and it was created by

```
region6 <- subset(pisa, CNT %in%  c("United States", "Canada", "Mexico",
                                    "B-S-J-G (China)", "Japan", "Korea",
                                    "Germany", "Italy", "France", "Brazil",
                                    "Colombia", "Uruguay", "Australia",
                                    "New Zealand", "Jordan", "Israel", "Lebanon"))
fwrite(region6, file = "region6.csv")
```

These are the 6 regions that will be covered during data visualization and can be used for the exercises and labs. The other file is a random sample of one country from each regions for even less powerful computers, which can also be used.

```
random6 <- subset(pisa, CNT %in%  c("Mexico", "Uruguay", "Japan",
                                    "Germany", "New Zealand", "Lebanon"))
fwrite(random6, file = "random6.csv")
```

4.2.1 Exercises

1. Read in the **pisa** data set. Either the full data set (recommended to have $>$ 8 Gb of RAM) or one of the smaller data sets.

4.3 Using the i in data.table

One of the first things we need to do when data wrangling is subsetting. Subsetting with `data.table` is very similar to base R but not identical. For example, if we wanted to subset all the students from Mexico who are currently taking Physics, i.e., they checked the item "Which course did you attend? Physics: This year" (ST063Q01NA) we would do the following:

```
pisa[CNTRYID == "Mexico" & ST063Q01NA == "Checked"]

# or (identical to base R)
subset(pisa, CNTRYID == "Mexico" & ST063Q01NA == "Checked")
```

Note that with `data.table` we do not need to use the `$` operator to access a variable in a `data.table` object. This is one improvement to the syntax of a `data.frame`.

Typing the name of a `data.table` won't print all the rows by default like a `data.frame`. Instead it prints just the first and last 5 rows.

```
pisa
```

This is extremely helpful because when we have a object in R, it often defaults to printing the entire object and this has the negative consequence of endless output if we type just the name of a very large object.

Because we have 921 variables, `data.table` will still truncate this output. If we want to view just the rows 10 through 25.

```
pisa[10:25]
```

However, with this many columns it is useless to print all of them and instead we should focus on examining just the columns we're interested in and we will see how to do this when we examine the `j` operator.

Often when data wrangling we would like to perform multiple steps without needing to create intermediate variables. This is known as **chaining**. Chaining can be done in `data.table` via

```
DT[ ...
   ][ ...
     ][ ...
       ]
```

For example, if we wanted to just see rows 17 through 20 after we've done previous subset, we can chain together these commands:

```
pisa[CNTRYID == "Mexico" & ST063Q01NA == "Checked"
     ][17:20]
```

When we're wrangling data, it's common and quite helpful to reorder rows. This can be done using the `order()` function. First, we print the first 6 six elements of the CNTRYID using the default ordering in the **pisa** data. Then we reorder the data by country name in a descending order and then print the first 6 six elements again using chaining.

```
head(pisa$CNTRYID)
```

```
## [1] "Albania" "Albania" "Albania" "Albania" "Albania" "Albania"
```

```
pisa[order(CNTRYID, decreasing = TRUE)
     ][,
       head(CNTRYID)]
```

```
## [1] "Vietnam" "Vietnam" "Vietnam" "Vietnam" "Vietnam" "Vietnam"
```

4.3.1 Exercises

1. Subset all the Female students (ST004D01T) in Germany
2. How many female students are there in Germany?
3. The `.N` function returns the length of a vector/number of rows. Use chaining with the `.N` function to answer Exercise 2.

4.4 Using the j in `data.table`

Using j we can select columns, summarize variables by performing actions on the variables, and create new variables. If we wanted to just select the country identifier:

```
pisa[,
     CNTRYID]
```

However, this returns a vector not a `data.table`. If we wanted instead to return a `data.table`:

```
pisa[,
     list(CNTRYID)]
```

```
##                                       CNTRYID
##      1:                               Albania
##      2:                               Albania
##      3:                               Albania
##      4:                               Albania
##      5:                               Albania
##     ---
## 519330: Argentina (Ciudad Autónoma de Buenos)
## 519331: Argentina (Ciudad Autónoma de Buenos)
## 519332: Argentina (Ciudad Autónoma de Buenos)
## 519333: Argentina (Ciudad Autónoma de Buenos)
## 519334: Argentina (Ciudad Autónoma de Buenos)
```

```
pisa[,
     .(CNTRYID)]
```

```
##                                       CNTRYID
##      1:                               Albania
##      2:                               Albania
##      3:                               Albania
##      4:                               Albania
##      5:                               Albania
##     ---
## 519330: Argentina (Ciudad Autónoma de Buenos)
## 519331: Argentina (Ciudad Autónoma de Buenos)
## 519332: Argentina (Ciudad Autónoma de Buenos)
## 519333: Argentina (Ciudad Autónoma de Buenos)
## 519334: Argentina (Ciudad Autónoma de Buenos)
```

The `.()` is `data.table` shorthand for `list()`. To subset more than one variable, we can just add another variable within the `.()`. For example, if we also wanted to select the science self-efficacy scale (SCIEEFF) as well, we do the following:

```
pisa[,
     .(CNTRYID, SCIEEFF)]
```

```
##                                     CNTRYID SCIEEFF
##      1:                             Albania      NA
##      2:                             Albania      NA
##      3:                             Albania      NA
##      4:                             Albania      NA
##      5:                             Albania      NA
##     ---
## 519330: Argentina (Ciudad Autónoma de Buenos) -0.8799
## 519331: Argentina (Ciudad Autónoma de Buenos)  0.9802
## 519332: Argentina (Ciudad Autónoma de Buenos) -0.5696
## 519333: Argentina (Ciudad Autónoma de Buenos) -0.7065
## 519334: Argentina (Ciudad Autónoma de Buenos) -0.3609
```

If we wanted see how many students took physics in Japan and Mexico, we would do the following:

```
pisa[CNTRYID %in% c("Mexico", "Japan"),
     table(ST063Q01NA)]
```

```
## ST063Q01NA
##     Checked Not checked
##        4283        9762
```

Because `data.table` treats string variables as character variables by default we see that when they are printed they are printed alphabetically, which in this case is fine but is often unhelpful. We can chain together variables and create an intermediate tense variable to get this in the correct format. However, when we want to know how students in Mexico and Japan responded to "I get very tense when I study for a test."

```
pisa[CNTRYID %in% c("Mexico", "Japan"),
     table(ST118Q04NA)]
```

```
## ST118Q04NA
##             Agree          Disagree    Strongly agree Strongly disagree
##              4074              5313              1760              2904
```

We see that the output is unhelpful. Instead, we should convert the character vector into a factor and we will create an intermediate variable called `tense`, which we won't add to our data set.

```
pisa[CNTRYID %in% c("Mexico", "Japan"),
     .(tense = factor(ST118Q04NA, levels = c("Strongly disagree", "Disagree", "Agree", "Strongly agree")
     )[,
       table(tense)
     ]
```

```
## tense
## Strongly disagree          Disagree             Agree    Strongly agree
##              2904              5313              4074              1760
```

Quick digression, in case you were wondering why base R reads strings in as factors and not characters by default (which `data.table` and `readr::read_csv` do),

```
pisa[, .(tense.as.char = ST118Q04NA,
         tense.as.fac = factor(ST118Q04NA, levels = c("Strongly disagree", "Disagree", "Agree", "Strongl
     )[,
       .(character = object.size(tense.as.char),
         factor = object.size(tense.as.fac))
     ]
```

```
##        character        factor
## 1: 4154984 bytes 2078064 bytes
```

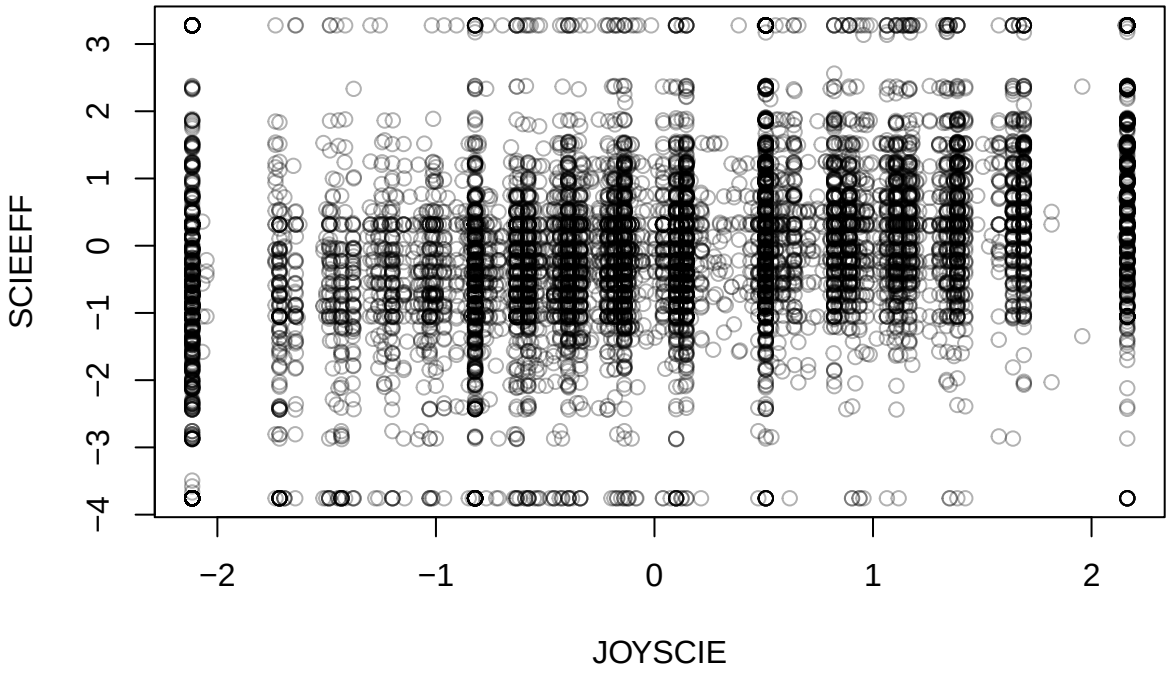
Returning to the science self-efficacy scale, we can request summary information for just these two countries:

```
pisa[CNTRYID %in% c("Mexico","Japan"),
     .(xbar = mean(SCIEEFF, na.rm = T),
       sigma = sd(SCIEEFF, na.rm = T),
       minimum = min(SCIEEFF, na.rm = T),
       med = median(SCIEEFF, na.rm = T),
       maximum = max(SCIEEFF, na.rm = T))]
```

```
##           xbar    sigma minimum     med maximum
## 1: -0.08693672 1.216052 -3.7565 -0.0541  3.2775
```

We can create a quick plot this way, too. For example, if we wanted a create a scatter plot of the science self-efficacy scale against the enjoyment of science scale (JOYSCIE) for just these two countries and print the mean of the enjoyment of science scale, we can do the following:

```
pisa[CNTRYID %in% c("Mexico","Japan"),
     .(plot(y = SCIEEFF, x = JOYSCIE,
            col = rgb(red = 0, green = 0, blue = 0, alpha = 0.3)),
       xbar.joyscie = mean(JOYSCIE, na.rm = T))]
```




```
##    xbar.joyscie
## 1:   0.06140002
```

This example is kind of silly but it shows that j is incredibly flexible and that we can string together a bunch of commands using j without even needing to do chaining.

Let's say we need to recode "After leaving school did you: Eat dinner" from a character variable to a numeric variable. We can do this with a series of if else statements

```
table(pisa$ST078Q01NA)
```

```
##
##     No    Yes
##  23617 373131
```

```
pisa[,
     "eat.dinner" := sapply(ST078Q01NA,
                            function(x) {
                              if (is.na(x)) NA
                              else if (x == "No") 0L
                              else if (x == "Yes") 1L
                            })
     ][,
       table(eat.dinner)
       ]
```

```
## eat.dinner
##      0      1
##  23617 373131
```

In this example we created a new variable called `eat.dinner` using `:=` the function. The `:=` syntax adds this variable directly to the DT. We also specified the `L` to ensure the variable was treated as an integer and not a double, which uses less memory.

We should create a function to do this recoding as there are lots of dichotomous items in the **pisa** data set.

```
#' Convert a dichtomous item (yes/no) to numeric scoring
#' @param x a character vector containing "Yes" and "No" responses.
bin.to.num <- function(x){
  if (is.na(x)) NA
  else if (x == "Yes") 1L
  else if (x == "No") 0L
}
```

Then use this function to create some variables as well as recoding gender to give it a more intuitive variable name.

```
pisa[, `:=`
     (female = ifelse(ST004D01T == "Female", 1, 0),
       sex = ST004D01T,

       # At my house we have ...
       desk = sapply(ST011Q01TA, bin.to.num),
       own.room = sapply(ST011Q02TA, bin.to.num),
       quiet.study = sapply(ST011Q03TA, bin.to.num),
       computer = sapply(ST011Q04TA, bin.to.num),
       software = sapply(ST011Q05TA, bin.to.num),
       internet = sapply(ST011Q06TA, bin.to.num),
       lit = sapply(ST011Q07TA, bin.to.num),
       poetry = sapply(ST011Q08TA, bin.to.num),
       art = sapply(ST011Q09TA, bin.to.num),
       book.sch = sapply(ST011Q10TA, bin.to.num),
       tech.book = sapply(ST011Q11TA, bin.to.num),
       dict = sapply(ST011Q12TA, bin.to.num),
       art.book = sapply(ST011Q16NA, bin.to.num))]
```

Similarly, we can create new variables by combining pre-existing ones. In the later data visualization section, we will use the following variables, so we will create them now. The `rowMeans` function takes a data.frame, so we need to subset the variables from the **pisa** data set and then convert it to a data.frame. This is what the brackets are doing.

```
pisa[, `:=`
     (math = rowMeans(pisa[, c(paste0("PV", 1:10, "MATH"))], na.rm = TRUE),
       reading = rowMeans(pisa[, c(paste0("PV", 1:10, "READ"))], na.rm = TRUE),
       science = rowMeans(pisa[, c(paste0("PV", 1:10, "SCIE"))], na.rm = TRUE))]
```

4.4.1 Exercises

1. The computer and software variables that were created above ask a student whether they had a computer in their home that they can use for school work (computer) and whether they had educational software in their home (software). Find the proportion of students in the Germany and Uruguay that have a computer in their home or have educational software.
2. For just female students, find the proportion of students who have their own room (own.room) or a quiet place to study (quiet.study).

4.5 Summarizing using the by in `data.table`

With the by argument, we can now get conditional responses without the need to subset. If we want to know the proportion of students in each country that have their own room at home.

```
pisa[,
     .(mean(own.room, na.rm = TRUE)),
     by = .(CNTRYID)
     ][1:6,
     ]
```

```
##      CNTRYID        V1
## 1:   Albania       NaN
## 2:   Algeria 0.5187970
## 3: Australia 0.9216078
## 4:   Austria 0.9054462
## 5:   Belgium 0.9153612
## 6:    Brazil 0.7497861
```

Again, we can reorder this using chaining:

```
pisa[,
     .(own.room = mean(own.room, na.rm = TRUE)),
     by = .(country = CNTRYID)
     ][order(own.room, decreasing = TRUE)
       ][1:6
         ]
```

```
##        country  own.room
## 1:     Iceland 0.9862721
## 2: Netherlands 0.9750188
## 3:      Norway 0.9737686
## 4:      Sweden 0.9558879
## 5:     Finland 0.9440994
## 6:     Germany 0.9379274
```

What if we want to compare just the Canada and Iceland on the proportion of students that have books of poetry at home (poetry) or and their mean on the enjoyment of science by student's biological sex?

```
pisa[CNTRYID %in% c("Canada", "Iceland"),
     .(poetry = mean(poetry, na.rm = TRUE),
       enjoy = mean(JOYSCIE, na.rm = TRUE)),
     by = .(country = CNTRYID, sex = sex)]
```

```
##    country    sex    poetry      enjoy
## 1:  Canada Female 0.3632105 0.29635781
## 2:  Canada   Male 0.3123878 0.40950018
## 3: Iceland Female 0.7280806 0.03583745
## 4: Iceland   Male 0.7011494 0.30316273
```

We see a strong country effect on poetry at home, with $> 70\%$ of Icelandic students reporting poetry books at home and just above 30% in Canadian students and we see that Canadian students enjoy science more than Icelandic students and, male students, overall, enjoy science more than females.

Let's examine books of poetry at home by countries and sort it in descending order.

```
pisa[,
     .(poetry = mean(poetry, na.rm = TRUE)),
     by = .(country = CNTRYID)
     ][order(poetry, decreasing = TRUE)
       ][1:6
         ]
```

```
##                country    poetry
## 1:              Kosovo 0.8352507
## 2: Russian Federation 0.8045568
## 3:             Romania 0.8019434
## 4:             Georgia 0.7495615
## 5:    B-S-J-G (China) 0.7442052
## 6:             Estonia 0.7422718
```

Iceland is in the top 10, while Canada is 59.

We can also write more complex functions and provide these to `data.table`. For example, if wanted to fit a regression model to predict a student's score on science self-efficacy scale given their score on the enjoyment of science scale and their sex for just the G7 countries (Canada, France, Germany, Italy, Japan, the United Kingdom, and the United States), we can fit a multiple regression model and return the intercept and slope terms.

```
get.params <- function(cntry){
  mod <- lm(SCIEEFF ~ JOYSCIE + sex, cntry)
  est.params <- list(int = coef(mod)[[1]], enjoy.slope = coef(mod)[[2]], sex.slope = coef(mod)[[3]])
  return(est.params)
}

g7.params <- pisa[CNTRYID %in% c("Canada", "France", "Germany", "Italy",
                                 "Japan", "United Kingdom", "United States"),
                  get.params(.SD),
                  by = .(CNTRYID)]
g7.params
```

```
##           CNTRYID          int enjoy.slope  sex.slope
## 1:         Canada  0.009803357   0.4370945 0.21489577
## 2:         France -0.208698984   0.4760903 0.17743126
## 3:        Germany -0.019150031   0.4316565 0.17971821
## 4:          Italy -0.030880063   0.3309990 0.18831666
```

```
## 5:          Japan -0.353806055   0.3914385 0.04912039
## 6: United Kingdom  0.009711647   0.5182592 0.18981965
## 7:  United States  0.096920721   0.3907848 0.15022008
```

We see a fair bit of variability in these estimated parameters

4.5.1 Exercises

1. Calculate the proportion of students who have art in their home (art) and the average age (AGE) of the students by gender.
2. Within a by argument you can discretize a variable to create a grouping variable. Perform a median split for age within the by argument and assess whether there are age difference associated with having your own room (own.room) or a desk (desk).

4.6 Reshaping data

The `data.table` package provides some very fast methods to reshape data from wide (the current format) to long format. In long format, a single test taker will correspond to multiple rows of data. Some software and `R` packages require data to be in long format (e.g., `lme4` and `nlme`).

Let's begin by creating a student ID and then subsetting this ID and the at-home variables:

```
pisa$id <- 1:nrow(pisa)
athome <- subset(pisa, select = c(id, desk:art.book))
```

To transform the data to long format we *melt* the data.

```
athome.l <- melt(athome,
                 id.vars = "id",
                 measure.vars = c("desk", "own.room", "quiet.study", "lit",
                                  "poetry", "art", "book.sch", "tech.book",
                                  "dict", "art.book"))
athome.l
```

```
##               id variable value
##       1:      1     desk    NA
##       2:      2     desk    NA
##       3:      3     desk    NA
##       4:      4     desk    NA
##       5:      5     desk    NA
##      ---
## 5193336: 519330 art.book     1
## 5193337: 519331 art.book     0
## 5193338: 519332 art.book     1
## 5193339: 519333 art.book     0
## 5193340: 519334 art.book     0
```

We could have also allowed `melt()` to guess the format:

```
athome.guess <- melt(athome)
```

```
## Warning in melt.data.table(athome): To be consistent with reshape2's melt,
## id.vars and measure.vars are internally guessed when both are 'NULL'. All
## non-numeric/integer/logical type columns are considered id.vars, which
## in this case are columns []. Consider providing at least one of 'id' or
```

```
## 'measure' vars in future.
```

```
athome.guess
```

```
##          variable value
##       1:       id     1
##       2:       id     2
##       3:       id     3
##       4:       id     4
##       5:       id     5
##      ---
## 7270672: art.book     1
## 7270673: art.book     0
## 7270674: art.book     1
## 7270675: art.book     0
## 7270676: art.book     0
```

It guessed incorrectly. If id was set as a character vector, then it would have guessed correctly this time. However, you should not allow it to guess the names of the variables.

To go back to wide format we use the `dcast()` function.

```
athome.w <- dcast(athome.l,
                  id ~ variable)
```

Unlike other reshaping packages, `data.table` can also handle reshaping multiple outcomes variables. More about reshaping with `data.table` is available here.

4.7 The sparklyr package

The `sparklyr` package provides an R interface to Apache Spark and a complete `dplyr` backend. Apache Spark "is a unified analytics engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing". Apache Spark can also be interfaced using the `sparkR` package provided by Apache. See here and here for more details.

To use Apache Spark you will need Java 8 JDK installed. It can be installed here. To begin with you need to install `sparklyr` and `dplyr`.

```
install.packages("sparklyr")
install.packages("dplyr")
library("sparklyr")
library("dplyr")
```

We then need to install Spark, which we can do from R.

```
spark_install()
```

Next, we need to setup a connection with Spark and we'll be connecting to a local install of Spark.

```
sc <- spark_connect(master = "local")
```

```
## Re-using existing Spark connection to local
```

Then we need to copy the **pisa** data set to the Spark cluster. However, with this large of a data set, this is a bad idea. We will run into memory issues during the copying process. So, we'll first subset the data before we do this.

```
pisa_sub <- subset(pisa, CNTRYID %in% c("Canada", "France", "Germany",
                                        "Italy", "Japan", "United Kingdom",
```

```
                                    "United States"),
                  select = c("DISCLISCI", "TEACHSUP", "IBTEACH", "TDTEACH",
                             "ENVAWARE", "JOYSCIE", "INTBRSCI", "INSTSCIE",
                             "SCIEEFF", "EPIST", "SCIEACT", "BSMJ", "MISCED",
                             "FISCED", "OUTHOURS", "SMINS", "TMINS",
                             "BELONG", "ANXTEST", "MOTIVAT", "COOPERATE",
                             "PERFEED", "unfairteacher", "HEDRES", "HOMEPOS",
                             "ICTRES", "WEALTH", "ESCS", "math", "reading",
                             "CNTRYID", "sex"))
```

We will use the selected variables in the labs and a description of these variables can be seen below.

Now the data are ready to be copied into Spark.

```
pisa_tbl <- copy_to(sc, pisa_sub, overwrite = TRUE)
```

In tidyverse, you can use the `%>%` to chain together commands or to pass data to functions. With `sparklyr`, we can use the `filter` function instead of subset. For example, if we just want to see the female students' scores on these scales for Germany, we would do the following:

```
pisa_tbl %>%
  filter(CNTRYID == "Germany" & sex == "Female")
```

```
## # Source: spark<?> [?? x 32]
##    DISCLISCI TEACHSUP IBTEACH TDTEACH ENVAWARE JOYSCIE INTBRSCI INSTSCIE
##        <dbl>    <dbl>   <dbl>   <dbl>    <dbl>   <dbl>    <dbl>    <dbl>
##  1   -0.234    -0.804  -0.608 -0.867   -0.536   -0.821  -0.550   NaN
##  2    0.283     0.488  -0.157 -0.685   -0.805   -2.12   -1.13     -1.93
##  3    0.700     1.45    0.988  0.525    0.171   -1.72   -0.225    -0.718
##  4    0.0039    0.568   0.209 -0.0742  -0.234   -0.821  -0.0831   -0.826
##  5    0.763    -0.450   0.535 -0.0057  -0.479    0.613   0.198    -0.304
##  6    0.660    -0.461   0.647  0.450   -0.706   -0.631  -0.551    -1.93
##  7    0.288    -1.82    0.430 -1.32    -0.0217  -0.576  -0.566    -0.670
##  8    0.835    -1.07    0.89  -0.610    0.256    2.16    0.341     1.33
##  9    1.32     -0.246   0.257 -0.867   -0.685   -0.152  -0.509    -0.778
## 10    1.32     -1.29    0.308 -0.790   -0.385   -1.61   -0.399    -1.93
## # ... with more rows, and 24 more variables: SCIEEFF <dbl>, EPIST <dbl>,
## #   SCIEACT <dbl>, BSMJ <int>, MISCED <chr>, FISCED <chr>, OUTHOURS <int>,
## #   SMINS <int>, TMINS <int>, BELONG <dbl>, ANXTEST <dbl>, MOTIVAT <dbl>,
## #   COOPERATE <dbl>, PERFEED <dbl>, unfairteacher <int>, HEDRES <dbl>,
## #   HOMEPOS <dbl>, ICTRES <dbl>, WEALTH <dbl>, ESCS <dbl>, math <dbl>,
## #   reading <dbl>, CNTRYID <chr>, sex <chr>
```

You'll notice the at the top it says `#Source: spark<?>`

If we wanted to calculate the average disciplinary climate in science classes (DISCLISCI) by country and by sex and have it reorder by country than sex, we can do the following:

```
pisa_tbl %>%
  group_by(CNTRYID, sex) %>%
  summarize(ave_disclip = mean(DISCLISCI, na.rm = TRUE)) %>%
  arrange(CNTRYID, sex)
```

```
## # Source:     spark<?> [?? x 3]
## # Groups:     CNTRYID
## # Ordered by: CNTRYID, sex
##    CNTRYID sex    ave_disclip
```

```
##    <chr>   <chr>        <dbl>
##  1 Canada  Female      0.0110
##  2 Canada  Male       -0.0205
##  3 France  Female     -0.236
##  4 France  Male       -0.297
##  5 Germany Female      0.0915
##  6 Germany Male        0.0162
##  7 Italy   Female      0.0708
##  8 Italy   Male       -0.137
##  9 Japan   Female      0.916
## 10 Japan   Male        0.788
## # ... with more rows
```

We can also create new variables using the `mutate` function. If we want to get a measure of home affluence, we could add home educational resources (HEDRES) and home possessions (HOMEPOS)

```
pisa_tbl %>%
  mutate(totl_home = HEDRES + HOMEPOS) %>%
  group_by(CNTRYID) %>%
  summarize(xbar = mean(totl_home, na.rm = TRUE))
```

```
## # Source: spark<?> [?? x 2]
##   CNTRYID          xbar
##   <chr>           <dbl>
## 1 United Kingdom  0.370
## 2 United States   0.113
## 3 Italy           0.324
## 4 Japan          -1.30
## 5 France         -0.332
## 6 Germany         0.279
## 7 Canada          0.430
```

On my computer, the Spark code is slightly faster than `data.table`, but not by much. The real power of using Spark is that we can use its machine learning functions. However, if you're familiar with `tidyverse` (`dplyr`) syntax, then `sparklyr` is a package that is worth investigating for data wrangling with big data sets.

4.8 Lab

This afternoon when we discuss supervised learning, we'll ask you to develop some models to predict the response to the question Do you expect your child will go into a ?" (PA032Q03TA).

1. Recode this variable so that a "Yes" is 1 and a "No" is a -1 and save the variable as `sci_car`.
2. Calculate descriptives for this variable by sex and country. Specifically, the proportion of test takers whose parents said "Yes" or 1.

After you've done this, spend some time investigating the following variables

Label	Description
DISCLISCI	Disciplinary climate in science classes (WLE)
TEACHSUP	Teacher support in a science classes of students choice (WLE)
IBTEACH	Inquiry-based science teaching an learning practices (WLE)
TDTEACH	Teacher-directed science instruction (WLE)
ENVAWARE	Environmental Awareness (WLE)

Label	Description
JOYSCIE	Enjoyment of science (WLE)
INTBRSCI	Interest in broad science topics (WLE)
INSTSCIE	Instrumental motivation (WLE)
SCIEEFF	Science self-efficacy (WLE)
EPIST	Epistemological beliefs (WLE)
SCIEACT	Index science activities (WLE)
BSMJ	Student's expected occupational status (SEI)
MISCED	Mother's Education (ISCED)
FISCED	Father's Education (ISCED)
OUTHOURS	Out-of-School Study Time per week (Sum)
SMINS	Learning time (minutes per week) -
TMINS	Learning time (minutes per week) - in total
BELONG	Subjective well-being: Sense of Belonging to School (WLE)
ANXTEST	Personality: Test Anxiety (WLE)
MOTIVAT	Student Attitudes, Preferences and Self-related beliefs: Achieving motivation (WLE)
COOPERATE	Collaboration and teamwork dispositions: Enjoy cooperation (WLE)
PERFEED	Perceived Feedback (WLE)
unfairteacher	Teacher Fairness (Sum)
HEDRES	Home educational resources (WLE)
HOMEPOS	Home possessions (WLE)
ICTRES	ICT Resources (WLE)
WEALTH	Family wealth (WLE)
ESCS	Index of economic, social and cultural status (WLE)
math	Students' math score in PISA 2015
reading	Students' reading score in PISA 2015

and then do the following using `data.table` and/or `sparklyr`:

3. Means and standard deviations (`sd`) for the variables that you think will be most predictive of `sci_car`.
4. Calculate these same descriptives by groups (by `sci_car` and by `sex`).
5. Calculate correlations between these variables and `sci_car`,
6. Create new variables
    - Discretize the math and reading variables using the OECD means (490 for math and 493) and code them as 1 (at or above the mean) and -1 (below the mean), but do in the `data.table` way without using the `$` operator.
    - Calculate the correlation between these variables and the list of variables above.
7. Chain together a set of operations
    - For example, create an intermediate variable that is the average of JOYSCIE and INTBRSCI, and then calculate the mean by country by `sci_car` through chaining.
8. Transform variables, specifically recode MISCED and FISCED from characters to numeric variables.
9. Examine other variables in the **pisa** data set that you think might be predictive of `PA032Q03TA`.

Chapter 5

Visualizing big data

One of the most effective ways to explore big data, interpret variables, and communicate results obtained from big data analyses to varied audiences is through **data visualization**. When we deal with big data, we can benefit from data visualizations in many ways, such as:

- understanding the distributional characteristics of variables,
- detecting data entry issues,
- identifying outliers in the data,
- understanding relationships among variables,
- selecting suitable variables for data analysis (a.k.a., feature extraction),
- examining the outcomes of predictive models (e.g., accuracy and overfit), and
- communicating the results to various audiences.

Developing effective visualizations requires identifying the goals and design of data analysis clearly. Sometimes we may already know the answers for some questions about the data; in other cases, we may want to explore further and understand the data in order to generate better insights into the next steps of data analysis. In this process, we need to consider many elements, such as types of variables to be used, axes, labels, legends, colors, and so on. Furthermore, if we aim to present the visualization to a particular audience, then we also need to consider the usability and interpretability of the visualization for the target audience.

The development of an effective data visualization typically includes the following steps:

1. Determine the goal of data visualization (e.g., exploring data, relationships, model outcomes)
2. Prepare the data (e.g., clean, organize, and transform data)
3. Identify the ideal visualization tool based on the goal of data visualization
4. Produce the visualization
5. Interpret the information in the visualization and present it to your target audience

Figure 5.1 shows some suggestions for visualizing data based on the type of variables and the purpose of the visualization. In R, almost all of these visualizations can be created very easily, although preparing the data for these visualizations is sometimes quite tedious.

In this section of our session, we will review data visualization tools in R that can help us organize big data, interpret variables, and identify potential variables for predictive models. The first part will focus on data visualizations using the ggplot2 package. Furthermore, we will use other R packages (e.g., GGally, ggExtra, and ggalluvial) that expand the capabilities of `ggplot2` even further (also see http://www.ggplot2-exts.org/gallery/ for more extensions of `ggplot2`). In the second part, we will discuss web-based, interactive visualizations and dashboards using plotly.

As we review data visualization tools, we will also demonstrate how to use each visualization tool in R and produce sample plots and graphics using the **pisa** dataset. Furthermore, we will ask you to work on short

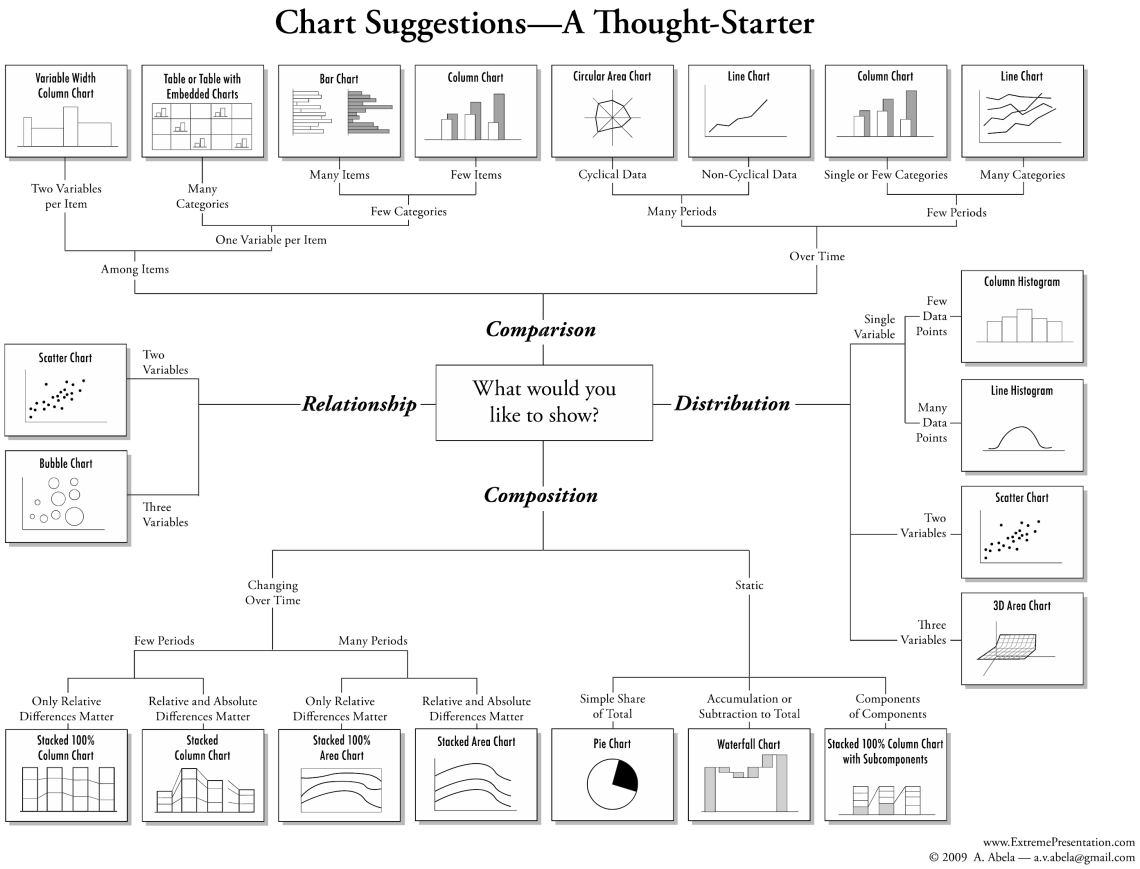


Figure 5.1: Chart suggestions (Source: <https://extremepresentation.com/>)

exercises where you will need to use the functions and packages presented in this section in order to generate your own plots and visualizations using the **pisa** dataset.

Before we begin, let's install and load all of the R packages that we will use in this section:

```
# Install and load the packages one by one.
install.packages("ggplot2")
install.packages("GGally")
install.packages("ggExtra")
install.packages("ggalluvial")
install.packages("plotly")

library("ggplot2")
library("GGally")
library("ggExtra")
library("ggalluvial")
library("plotly")

# Or, just simply run the following to install and load all packages:
dataviz_packages <- c("ggplot2", "GGally", "ggExtra", "ggalluvial", "plotly")
install.packages(dataviz_packages)
lapply(packages, require, character.only = TRUE)

# Load already installed packages
library("data.table")

# we will also use cowplot later in this session.
# Please install it but do not load it for now.
install.packages("cowplot")
```

5.1 Introduction to ggplot2

This section will demonstrate how to visualise your big data using `ggplot2` and other R packages that rely on `ggplot2`. We use 'ggplot2 because it is the most elegant and versatile visualization package in R. Also, it implements a simple grammar of graphics for building a variety of visualizations for either small or large data. This enables creating high-quality plots for publications and presentations easily, with minimal amounts of adjustments and tweaking.

A typical `ggplot2` template ranges from a few layers to many layers, depending on the complexity of the visualization of interest. Layers generate a plot and plot transformations within the plot. We can combine multiple layers using the + operator. Therefore, plots are built step by step by adding new elements in each layer. A simple `ggplot2` template is shown below:

```
ggplot(data = my_data,
       mapping = aes(x = var1, y = var2)) +
  geom_function()
```

where the `ggplot` function uses the two variables (**var1** and **var2**) from a dataset (**my_data**), and draws a new plot based on a particular geom function (**geom_function**). Selecting the variables to be plotted is done through the aesthetic mapping (via the `aes` function). Depending on the aesthetic mapping of interest, we can split the plot, add colors by a group variable, change the labels for each axis, change the font size, and so on. The 'ggplot2 package offers many geom functions to draw different types of plots:

Table 5.1: Variables to be used in the data visualizations

Variable	Description	Variable	Description
CNT	Country	BELONG	Sense of belonging to sc
OECD	OECD membership	EMOSUPS	Parents emotional suppo
CNTSTUID	Student ID	HOMESCH	ICT use outside of schoo
W_FSTUWT	Student weight in the PISA database	ENTUSE	ICT use outside of schoo
ST001D01T	Grade level	ICTHOME	ICT available at home
ST004D01T	Gender (female/male)	ICTSCH	ICT availability at schoo
ST011Q04TA	Possessing a computer at home	WEALTH	Family wealth
ST011Q05TA	Possessing educational software at home	PARED	Highest parental educati
ST011Q06TA	Having internet access at home	TMINS	Total learning time per
ST071Q02NA	Additional time spent for learning math	ESCS	Index of economic, socia
ST071Q01NA	Additional time spent for learning science	TDTEACH	Teacher-directed science
ST123Q02NA	Whether parents support educational efforts and achievements	IBTEACH	Inquiry based science ins
ST082Q01NA	Prefering working as part of a team to working alone	TEACHSUP	Teacher support in scien
ST119Q01NA	Wanting top grades in most or all courses	SCIEEFF	Science self-efficacy
ST119Q05NA	Wanting to the best student in class	math	Students math scores in
ANXTEST	Test anxiety	reading	Students reading scores
COOPERATE	Enjoying cooperation	science	Students science scores i

- `geom_point` for scatter plots, dot plots, etc.
- `geom_boxplot` for boxplots
- `geom_line` for trend lines, time series, etc.

In addition, functions such as `theme_bw()` and `theme()` enable adjusting the theme elements (e.g., font size, font type, background colors) for a given plot. As we create plots in our examples, we will use some of these theme elements to make our plots look nicer.

An important caveat in visualizing big data is that the size of the dataset (*especially the number of rows*) and complexity level of the plot (e.g., additional lines, colors, facets) will influence how quickly and successfully `ggplot2` can render the desired plot. Nobody can absorb the meaning of thousands of data points presented on a single visualization. Therefore, in some cases we will need to find a way to cluster or reduce the magnitude of items to visualize before we render the visualization. Typically we can achieve this by:

- taking smaller, sometimes random, samples from our big data, or
- summarizing our big data using categorical, group variables (e.g., gender, grade, year).

5.2 Marginal plots

We can use marginal plots to examine the distributions of individual variables in a large dataset. A typical marginal plot is a scatter plot that also has histograms or boxplots in the margins of the x- and y-axes. In this section, first we will create histograms and boxplots for the variables in the **pisa** dataset. Then, we will review other options where we will combine multiple variables and different types of plots in a single visualization.

To demonstrate data visualizations, we will first take a subset of the **pisa** dataset by selecting some countries and some variables of interest. The selected variables are shown below.

Here we filter our big data based on a list of countries (we called `country`), select the variables that we have just identified in Table 5.1 and the reading, math, and science scales we created earlier.

```
country <- c("United States", "Canada", "Mexico", "B-S-J-G (China)", "Japan",
             "Korea", "Germany", "Italy", "France", "Brazil", "Colombia", "Uruguay",
             "Australia", "New Zealand", "Jordan", "Israel", "Lebanon")

dat <- pisa[CNT %in% country,
            .(CNT, OECD, CNTSTUID, W_FSTUWT, sex, female,
              ST001D01T, computer, software, internet,
              ST011Q05TA, ST071Q02NA, ST071Q01NA, ST123Q02NA,
              ST082Q01NA, ST119Q01NA, ST119Q05NA, ANXTEST,
              COOPERATE, BELONG,  EMOSUPS, HOMESCH, ENTUSE,
              ICTHOME, ICTSCH, WEALTH, PARED, TMINS, ESCS,
              TEACHSUP, TDTEACH, IBTEACH, SCIEEFF,
              math, reading, science)
            ]
```

Next, we create additional variables by recoding some of the existing variables. The goal is to create some numerical variables out of the character variables in case we want to use them in the modeling stage.

```
# Let's create additional variables that we will use for visualizations
dat <- dat[, `:=` (
  # New grade variable
  grade = (as.numeric(sapply(ST001D01T, function(x) {
  if(x=="Grade 7") "7"
  else if (x=="Grade 8") "8"
  else if (x=="Grade 9") "9"
  else if (x=="Grade 10") "10"
  else if (x=="Grade 11") "11"
  else if (x=="Grade 12") "12"
  else if (x=="Grade 13") NA_character_
  else if (x=="Ungraded") NA_character_}))),
  # Total learning time as hours
  learning = round(TMINS/60, 0),
  # Regions for selected countries
  Region = (sapply(CNT, function(x) {
  if(x %in% c("Canada", "United States", "Mexico")) "N. America"
    else if (x %in% c("Colombia", "Brazil", "Uruguay")) "S. America"
    else if (x %in% c("Japan", "B-S-J-G (China)", "Korea")) "Asia"
    else if (x %in% c("Germany", "Italy", "France")) "Europe"
    else if (x %in% c("Australia", "New Zealand")) "Australia"
    else if (x %in% c("Israel", "Jordan", "Lebanon")) "Middle-East"
    }))
  )]
```

Now, let's see the number of rows in the final dataset and print the first few rows of the selected variables.

```
# N count for the final dataset
dat[,.N] # 158,061 rows
```

```
## [1] 158061
```

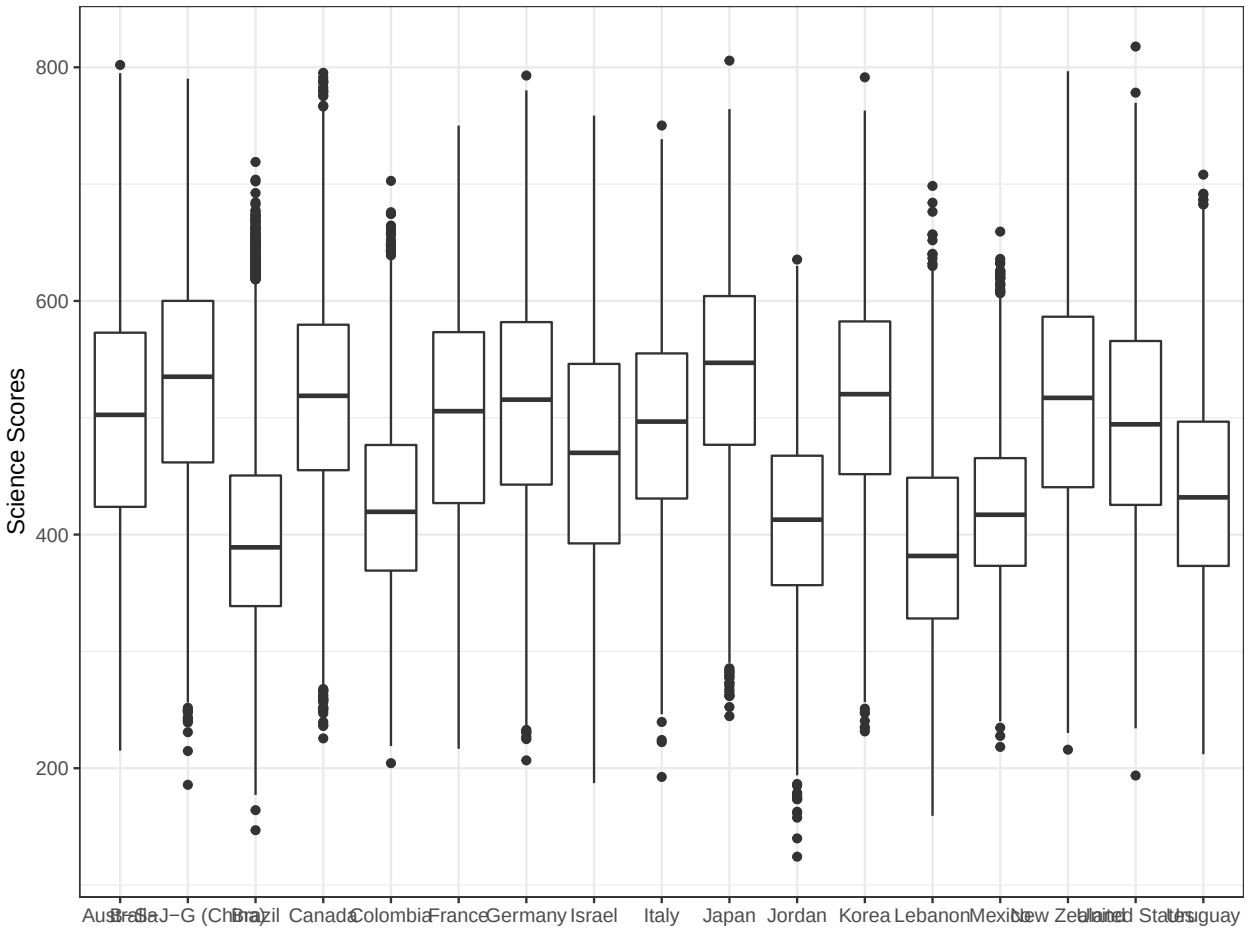
```
# Let's preview the final data
head(dat)
```

```
##          CNT OECD CNTSTUID W_FSTUWT    sex female ST001D01T computer
## 1: Australia  Yes  3610676 28.19991 Female      1  Grade 10        1
## 2: Australia  Yes  3611874 28.19991 Female      1  Grade 10        1
```

```
## 3: Australia  Yes  3601769 28.19991 Female      1  Grade 10        1
## 4: Australia  Yes  3605996 28.19991 Female      1  Grade 10        1
## 5: Australia  Yes  3608147 33.44862   Male      0  Grade 10        1
## 6: Australia  Yes  3610012 33.44862   Male      0  Grade 10        1
##    software internet ST011Q05TA ST071Q02NA ST071Q01NA     ST123Q02NA
## 1:        1        1        Yes          0          1       Disagree
## 2:        1        1        Yes          1          1          Agree
## 3:        1        1        Yes         NA         NA          Agree
## 4:        1        1        Yes          5          7 Strongly agree
## 5:        1        1        Yes          1          1          Agree
## 6:        1        1        Yes          2          2          Agree
##           ST082Q01NA     ST119Q01NA     ST119Q05NA ANXTEST COOPERATE
## 1:          Disagree          Agree Strongly agree -0.1522    0.2085
## 2:             Agree          Agree       Disagree  0.2594   -0.2882
## 3: Strongly disagree Strongly agree       Disagree  2.5493   -1.2109
## 4: Strongly disagree Strongly agree Strongly agree  0.2563    0.3950
## 5:             Agree          Agree       Disagree  0.4517   -1.3606
## 6:             Agree          Agree          Agree  0.5175    0.4252
##     BELONG EMOSUPS HOMESCH  ENTUSE ICTHOME ICTSCH  WEALTH PARED TMINS
## 1:  0.5073 -2.2547 -0.1686 -0.7369       4      5  0.0592    12  1400
## 2: -0.8021 -0.2511  0.0302 -0.1047       9      6  0.7605    12  1100
## 3: -2.4078 -1.9895  1.2836 -1.5403      11     10 -0.1220    11  1960
## 4: -0.3381  1.0991 -0.0498  0.0342      10      7  0.9314    15  2450
## 5: -0.5050 -1.3298 -0.3355  0.2309      NA      7  0.7905    15  1400
## 6: -0.0099 -0.4263  0.1567  0.6896      10      5  0.7054    15  1400
##       ESCS TEACHSUP TDTEACH IBTEACH SCIEEFF     math  reading  science
## 1:  0.4078       NA      NA      NA      NA 545.8999 586.5175 589.5787
## 2:  0.4500   0.3574  0.0615  0.2208 -0.4041 511.6101 570.8238 557.2042
## 3: -0.5889  -1.0718 -0.6102 -0.2198 -0.9003 478.6052 570.0345 569.4709
## 4:  0.6498   0.6375  0.7979 -0.0282  1.2395 506.0904 531.0690 529.0353
## 5:  0.7675   0.8213  0.1990  1.1477 -0.0746 481.8569 506.4988 504.2148
## 6:  1.1151       NA      NA      NA      NA 455.0202 456.4882 472.6377
##    grade learning    Region
## 1:    10       23 Australia
## 2:    10       18 Australia
## 3:    10       33 Australia
## 4:    10       41 Australia
## 5:    10       23 Australia
## 6:    10       23 Australia
```

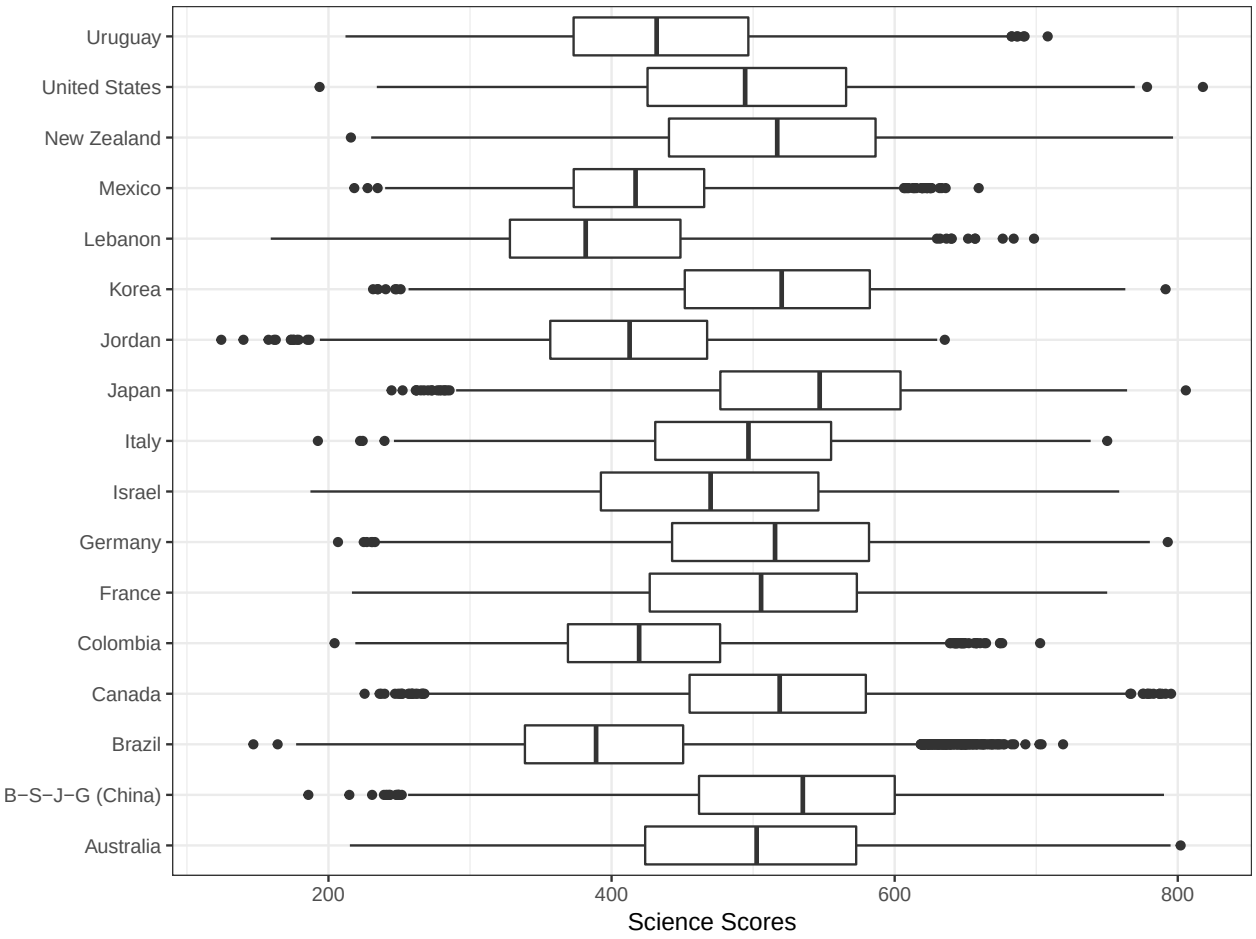
We want to see the distributions of the science scores across the 17 countries in our final dataset. The first line with `ggplot` creates a layout for our figure, the second line draws a box plot using `geom_boxplot`, the fourth line with `labs` creates labels of the axes, and the last line with `theme_bw` removes the default theme with a grey background and activates the dark-on-light `ggplot2` theme – which is much better for publications and presentations (see https://ggplot2.tidyverse.org/reference/ggtheme.html for a complete list of themes available in `ggplot2`).

```
ggplot(data = dat, mapping = aes(x = CNT, y = science)) +
  geom_boxplot() +
  labs(x=NULL, y="Science Scores") +
  theme_bw()
```



The resulting plot is not necessarily nice because all the country names on the x-axis seem to be squeezed together and thus some of the country names are not visible on the x-axis. To correct this, we may want to flip the coordinates of the plot and use country names on the y-axis instead. The `coord_flip()` function allows us to achieve that very easily.

```
ggplot(data = dat,
       mapping = aes(x = CNT, y = science)) +
  geom_boxplot() +
  labs(x=NULL, y="Science Scores") +
  coord_flip() +
  theme_bw()
```



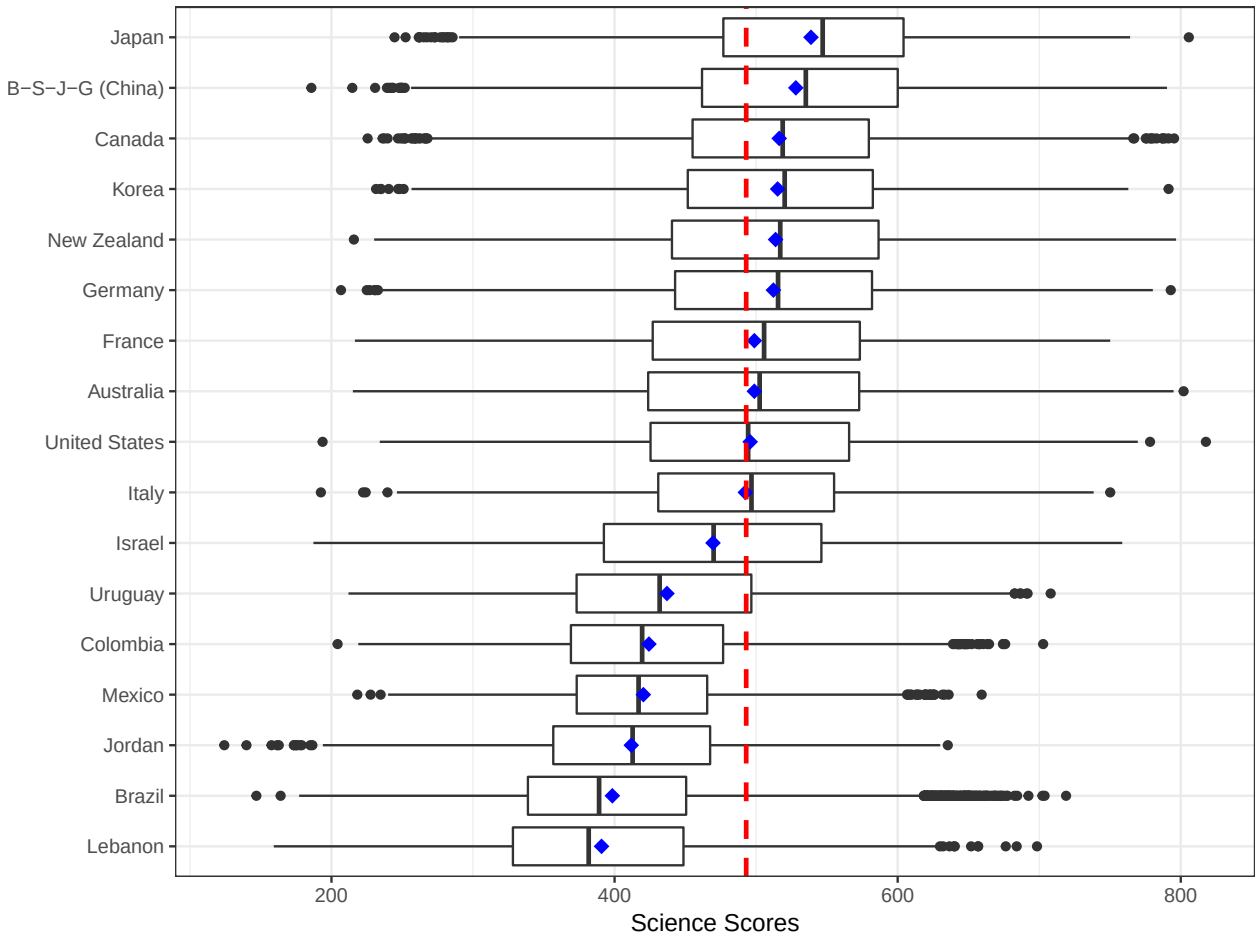
Next, I want to show the mean values in the boxplots since the line in the middle represents the median, not the mean. To achieve this, we first calculate the means by countries.

```
means <- dat[,
             .(science = mean(science)),
             by = CNT]
```

Now we can use `means` to add a point into each boxplot to show the mean score by countries. We will use `stat_summary()` along with the options `colour = "blue", geom = "point"` to create a blue point for the mean. In addition, given that the average science score in PISA 2015 was 493 across all participating countries (see PISA 2015 Results in Focus for more details), we can add a reference line into our plot to identify the average score, which would then allow us to visually examine which countries are above or below the average score. To achieve this, we use `geom_hline` function and specify where it should intersect the plot (i.e., `yintercept = 493`). We also want the reference line to be a red, dashed-line with a thickness level of 1 – to make it more visible in the plot. Finally, to facilitate the interpretation of the plot, we want the boxplots to be ordered based on the average scores for each country and thus we add `reorder(CNT, science)` into the mapping.

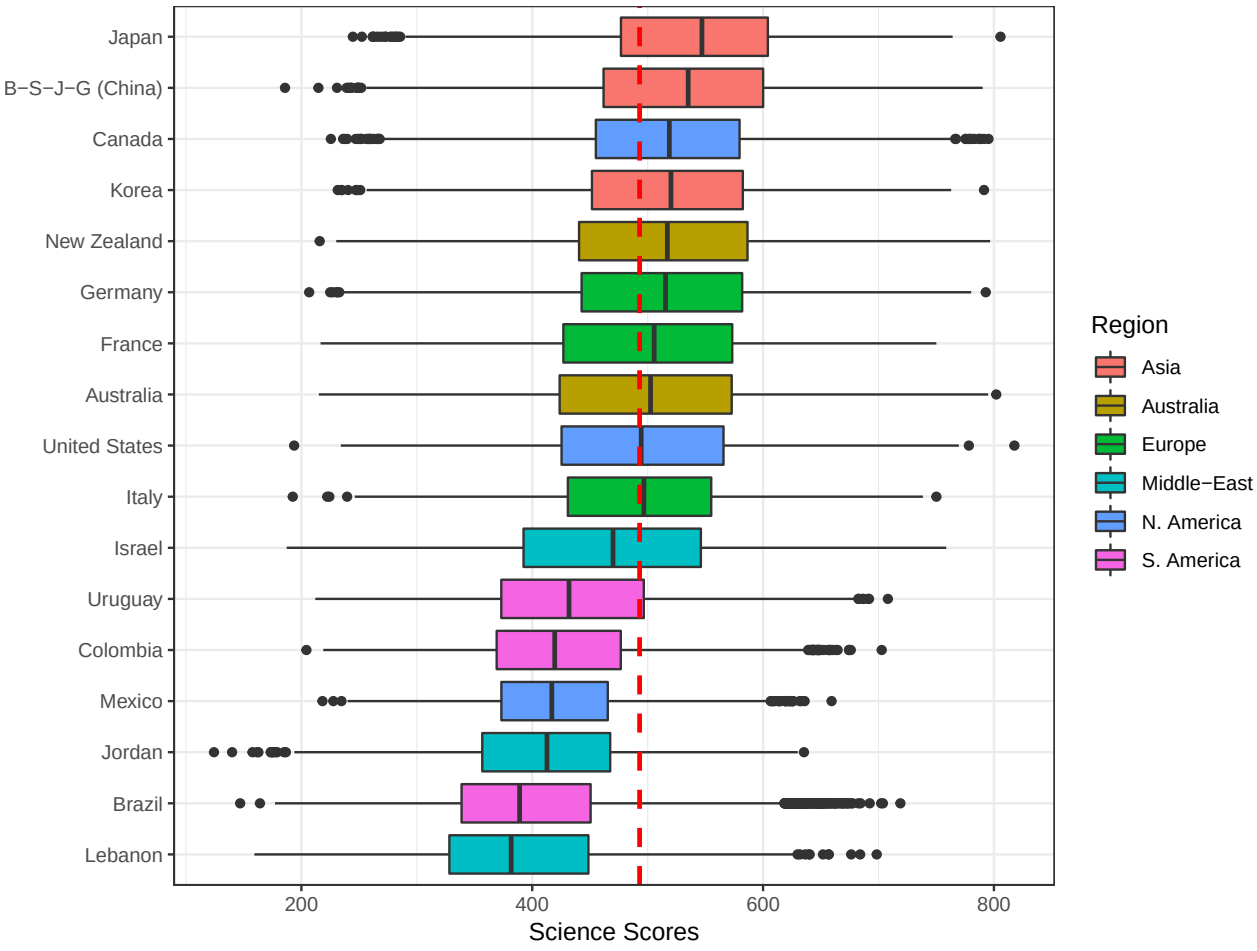
```
ggplot(data = dat,
       mapping = aes(x = reorder(CNT, science), y = science)) +
  geom_boxplot() +
  stat_summary(fun.y = mean, colour = "blue", geom = "point",
               shape = 18, size = 3) +
  labs(x=NULL, y="Science Scores") +
  coord_flip() +
```

```
  geom_hline(yintercept = 493, linetype="dashed", color = "red", size = 1) +
  theme_bw()
```



Now let's add some colors to our figure based on the region where each country is located. In order to do this, we use the region variable to fill the boxplots with color, using `fill = Region`.

```
ggplot(data = dat,
       mapping = aes(x = reorder(CNT, science), y = science, fill = Region)) +
  geom_boxplot() +
  labs(x=NULL, y="Science Scores") +
  coord_flip() +
  geom_hline(yintercept = 493, linetype="dashed", color = "red", size = 1) +
  theme_bw()
```



5.2.1 Exercise

Create a plot of **math** scores over countries with different colors based on region. You need to modify the R code below by replacing `geom_boxplot` with:

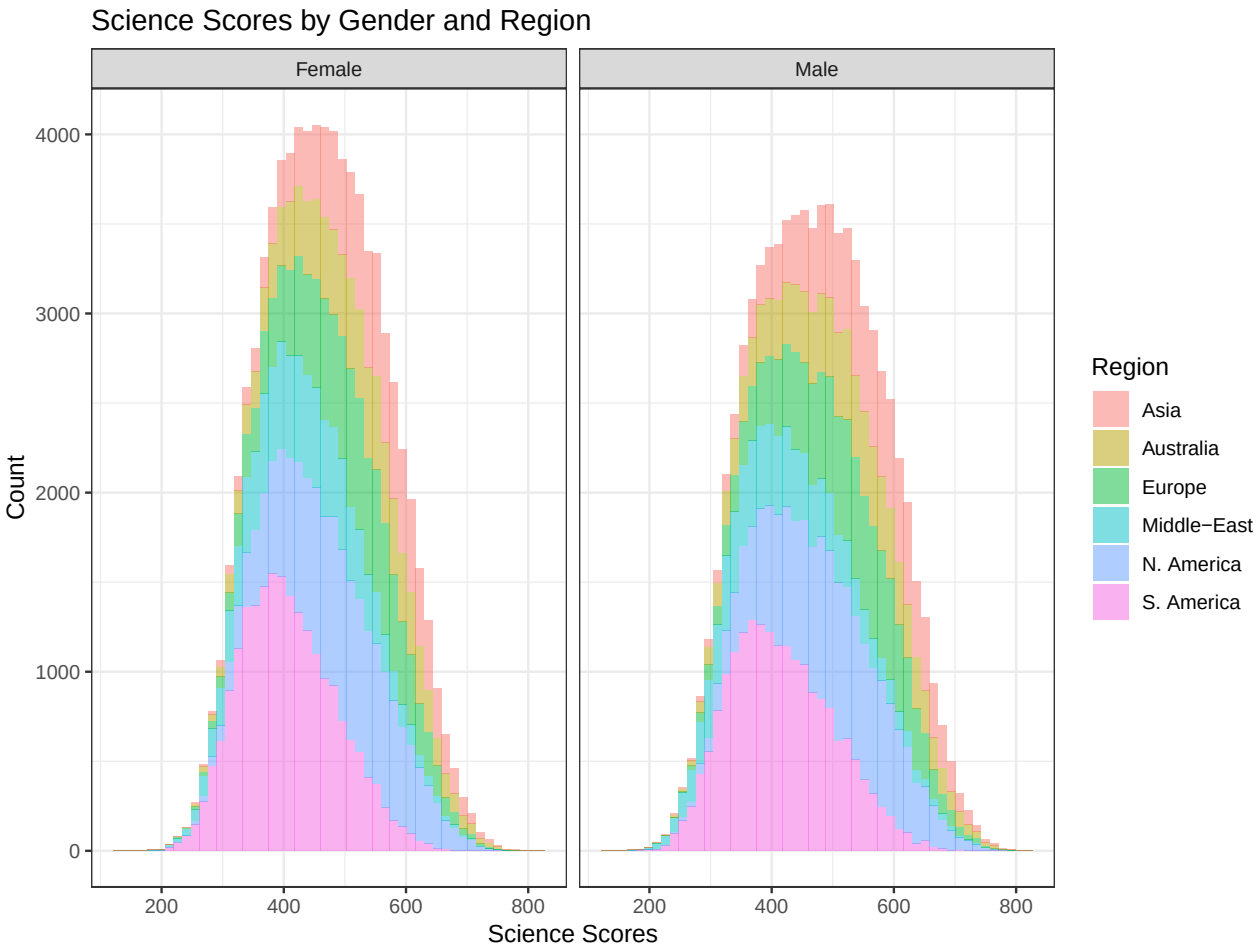
- `geom_point(aes(color = Region))`, and then
- `geom_violin(aes(color = Region))`.

How long did it take to create both plots? Which one is a better way to visualize this type of data?

```
ggplot(data = dat,
       mapping = aes(x = reorder(CNT, math), y = math, fill = Region)) +
  geom_boxplot() +
  labs(x=NULL, y="Math Scores") +
  coord_flip() +
  geom_hline(yintercept = 490, linetype="dashed", color = "red", size = 1) +
  theme_bw()
```

We can also create histograms (or density plots) for a particular variable and split the plot into multiple plots by using a categorical, group variable. In the following example, we use `x = Region` in the mapping in order to identify different regions in the distribution of the science scores. In addition, we use `facet_grid(. ~ sex)` to generate separate histograms by gender. Note that we also added `title = "Science Scores by Gender and Region"` as a title in the `labs` function.

```
ggplot(data = dat,
       mapping = aes(x = science, fill = Region)) +
  geom_histogram(alpha = 0.5, bins = 50) +
  labs(x = "Science Scores", y = "Count",
       title = "Science Scores by Gender and Region") +
  facet_grid(. ~ sex) +
  theme_bw()
```

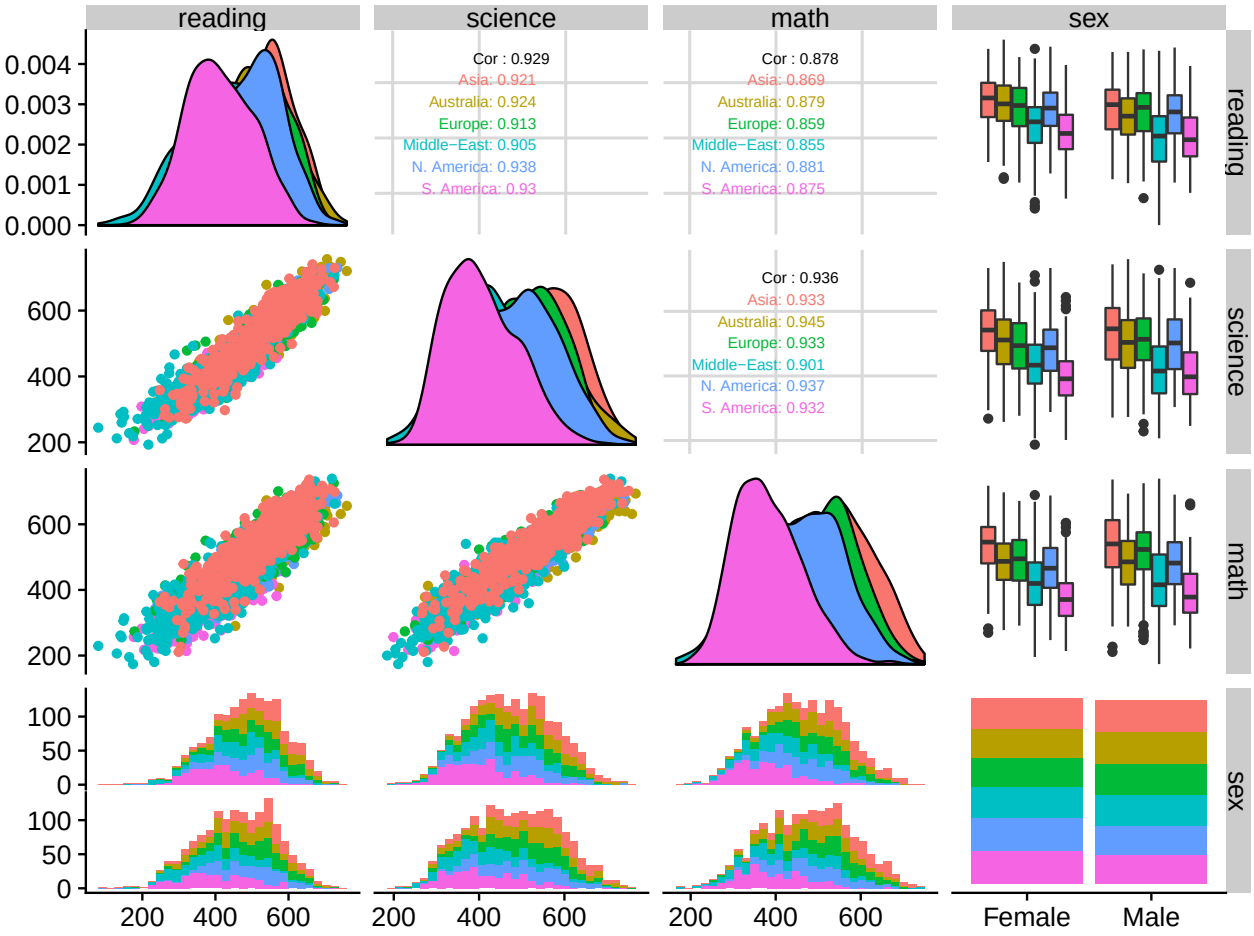


If we are interested in visualizing multiple variables, plotting each variable individually can be time consuming. Therefore, we can use the `ggpairs` function from the `GGally` package to build a more complex, diagnostic plot for multiple variables.

In the following example, we plot reading, science, and math scores as well as gender (i.e., sex) in the same plot. Because our dataset is quite large, plotting all the data points would result in a highly complex plot where most data points would overlap on each other. Therefore, we will take a random sample of 500 cases from each region defined in the data, save this smaller dataset as `dat_small`, and use this dataset inside the `ggpairs` function. We colorize each variable by region (using `mapping = aes(color = Region)`). The resulting plot shows density plots for the continuous variables (by region), a stacked bar chart for gender, and box plots for the continuous variables by region and gender.

```
# Random sample of 500 students from each region
dat_small <- dat[,.SD[sample(.N, min(500,.N))], by = Region]

ggpairs(data = dat_small,
```

```
        mapping = aes(color = Region),
        columns = c("reading", "science", "math", "sex"),
        upper = list(continuous = wrap("cor", size = 2.5))
        )
```



Interpretation:

- What can we say about the regions based on the plots above?
- Do you see any major gender differences for reading, science, or math?
- What is the relationship among reading, science, or math?

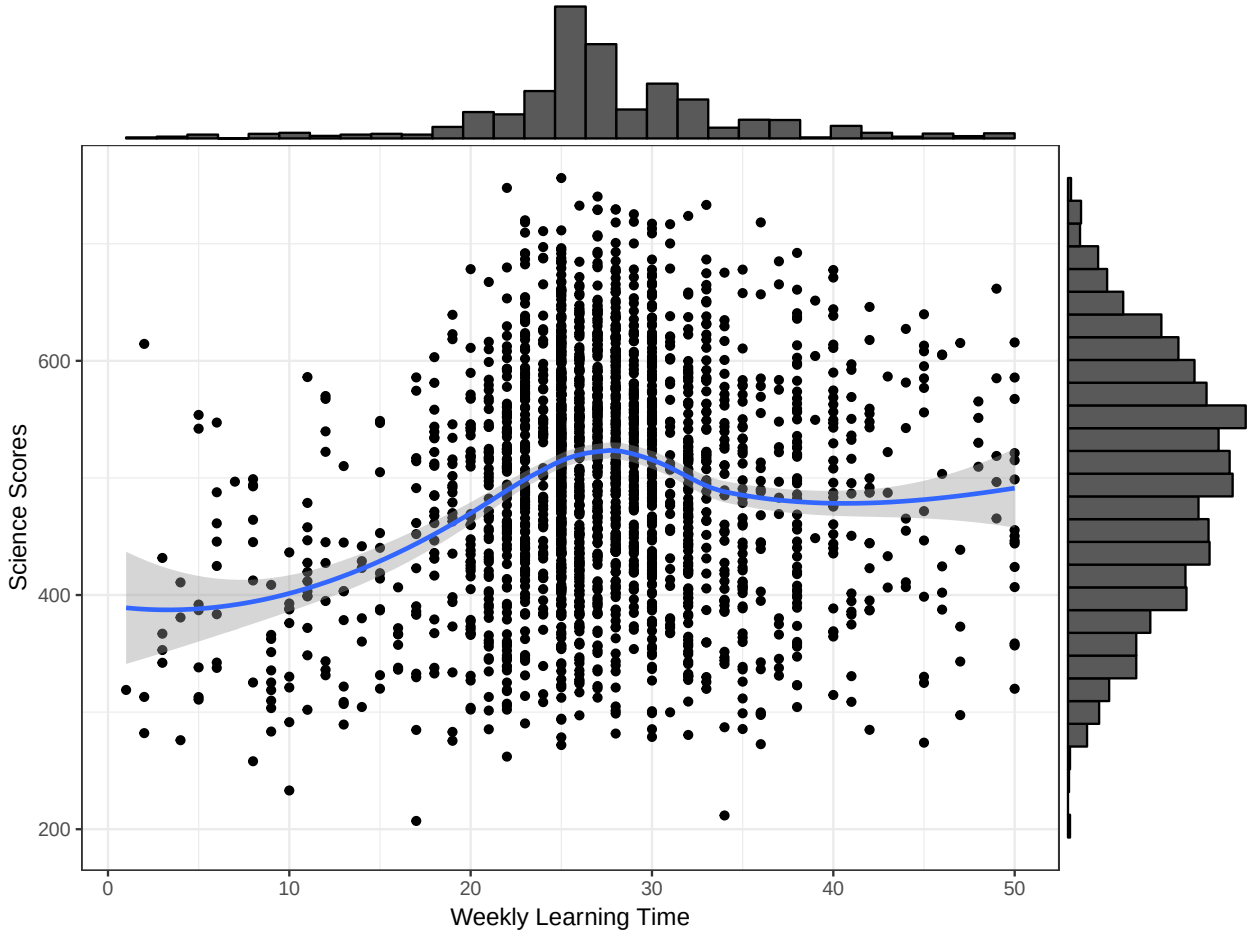
5.3 Conditional plots

When we deal with continuous variables, an effective way to understand the relationship between the variables is to produce conditional plots, such as scatterplots, dotplots, and bubble charts. Simple scatterplots in R can be created using `plot(var1, var2, data = name_of_dataset)`. Using the extended capabilities of `ggplot2` via the `ggExtra` package, we can combine histograms and density plots with scatterplots and visualize them together.

In the following example, we first create a scatterplot of learning time per week and science scores using `ggplot`. We use `geom_point` to draw a plot with points and `geom_smooth(method = "loess")` to add a regression line with loess smoothing (i.e., **Lo**cally **E**stimated **S**catterplot **S**moothing). We save this plot as

`p1` and then pass it to `ggMarginal` to transform the plot into a marginal scatterplot. Inside `ggMarginal`, we use `type = "histogram"` to create histograms for learning time per week and science scores on the x and y axes of the plot. Note that as the plot is created, you may see some warning messages, such as "Removed 750 rows containing missing values", because some variables have missing rows in the dataset.

```
p1 <- ggplot(data = dat_small,
             mapping = aes(x = learning, y = science)) +
  geom_point() +
  geom_smooth(method = "loess") +
  labs(x = "Weekly Learning Time", y = "Science Scores") +
  theme_bw()

# Replace "histogram" with "boxplot" or "density" for other types
ggMarginal(p1, type = "histogram")
```



We can also distinguish male and female students in the plot and create a scatterplot of learning time and science scores with densities by gender. To achieve this, we add `colour = sex` into the mapping of `ggplot` and change the type of plot to `type = "density"` in `ggMarginal`. In addition, we use `groupColour = TRUE, groupFill = TRUE` inside `ggMarginal` to use separate colors for each gender in the density plots.

```
p2 <- ggplot(data = dat_small,
             mapping = aes(x = learning, y = science,
                           colour = sex)) +
  geom_point() +
  geom_smooth(method = "loess") +
```

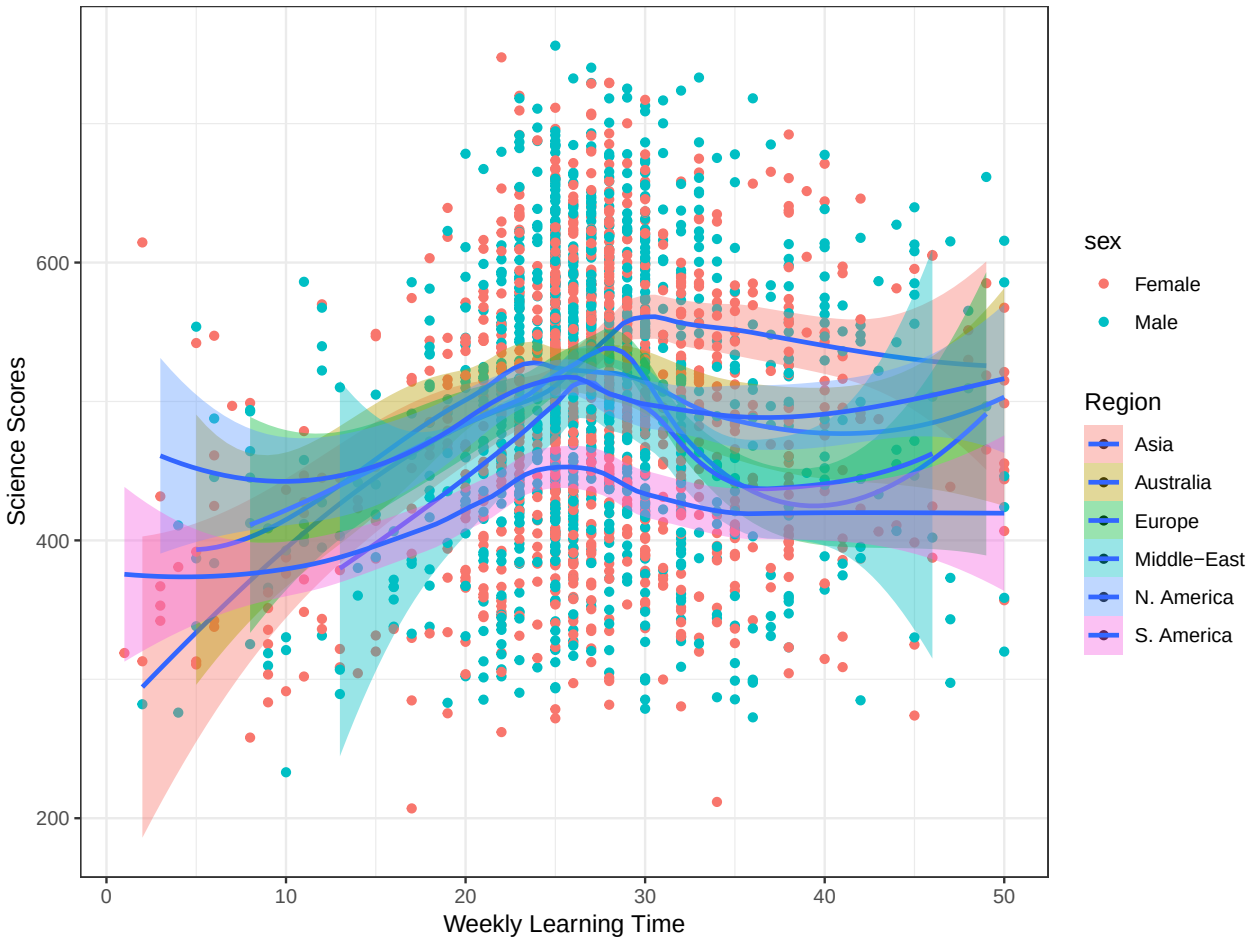
```
  labs(x = "Weekly Learning Time", y = "Science Scores") +
  theme_bw() +
  theme(legend.position = "bottom",
        legend.title = element_blank())

ggMarginal(p2, type = "density", groupColour = TRUE, groupFill = TRUE)
```

Interpretation:

- What can we say about the relationship between weekly learning time and science scores?
- Do you see any gender differences?

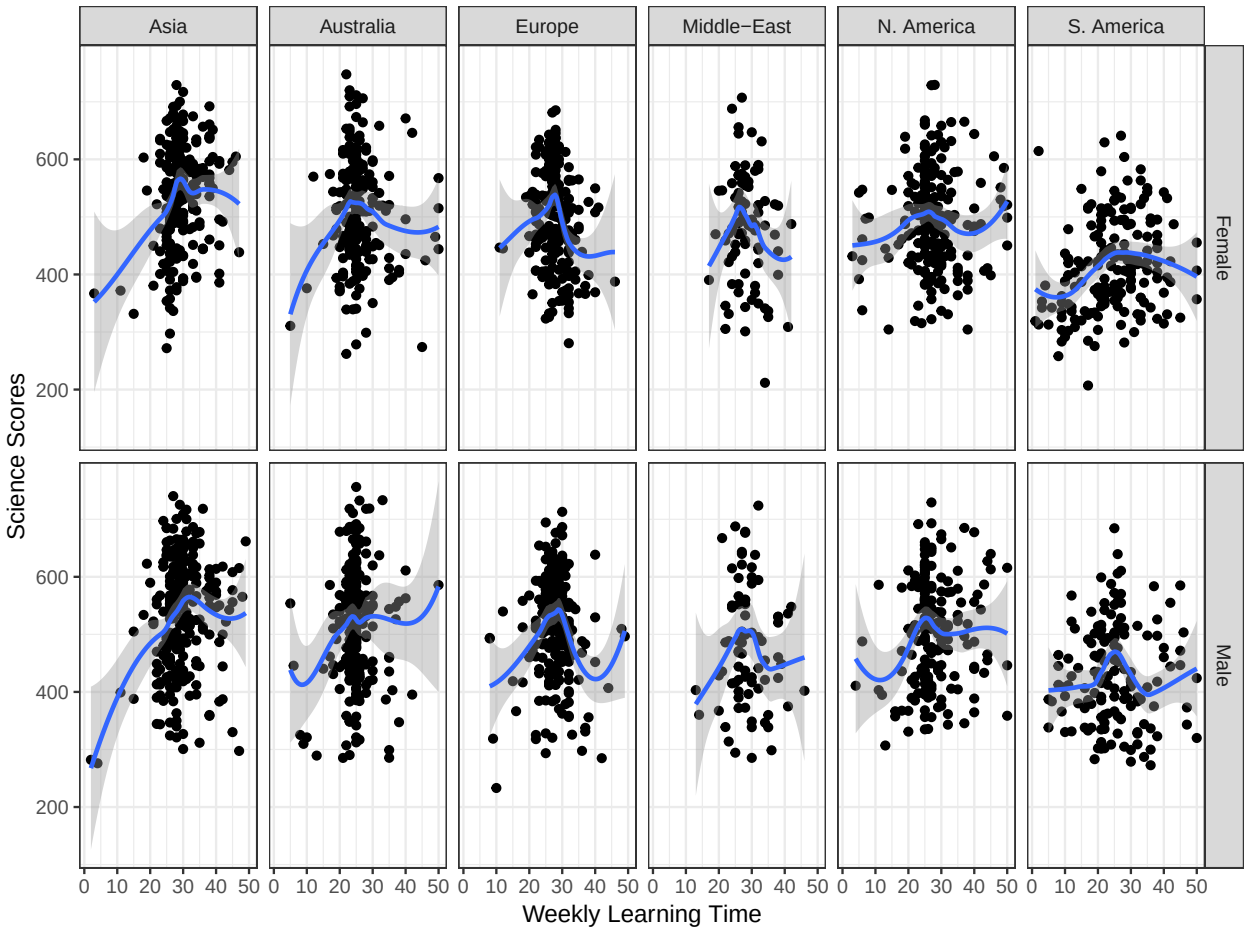
Now let's incorporate more variables into the plot. This time we are not going to use marginal plots. Instead, we will create a regular scatterplot but add other layers to represent additional variables. In the following example, we examine the relationship between students' weekly learning time (learning) and science scores (science) across regions (region) and gender (sex). Adding `fill = Region` into the mapping will allow us to draw regression lines by regions, while adding `aes(colour = sex)` into `geom_point` will allow us to use different colors for male and female students in the plot.

```
ggplot(data = dat_small,
       mapping = aes(x = learning, y = science, fill = Region)) +
  geom_point(aes(colour = sex)) +
  geom_smooth(method = "loess") +
  labs(x = "Weekly Learning Time", y = "Science Scores") +
  theme_bw()
```



The resulting scatterplot is nice but it is hard to compare the results clearly between gender groups and regions. To improve the interpretability of the plot, we will use the faceting option. This will allow us to split the scatterplot into multiple plots based on gender and region. In the following example, we examine the relationship between students' learning time and science scores across regions and gender. We use `facet_grid(sex ~ Region)` to split the plots into multiple rows based on gender and multiple columns based on region.

```
ggplot(data = dat_small,
       mapping = aes(x = learning, y = science)) +
  geom_point() +
  geom_smooth(method = "loess") +
  labs(x = "Weekly Learning Time", y = "Science Scores") +
  theme_bw() +
  theme(legend.title = element_blank()) +
  facet_grid(sex ~ Region)
```



Interpretation:

- Do you see any regional differences?
- Is there any interaction between gender and region?

5.3.1 Exercise

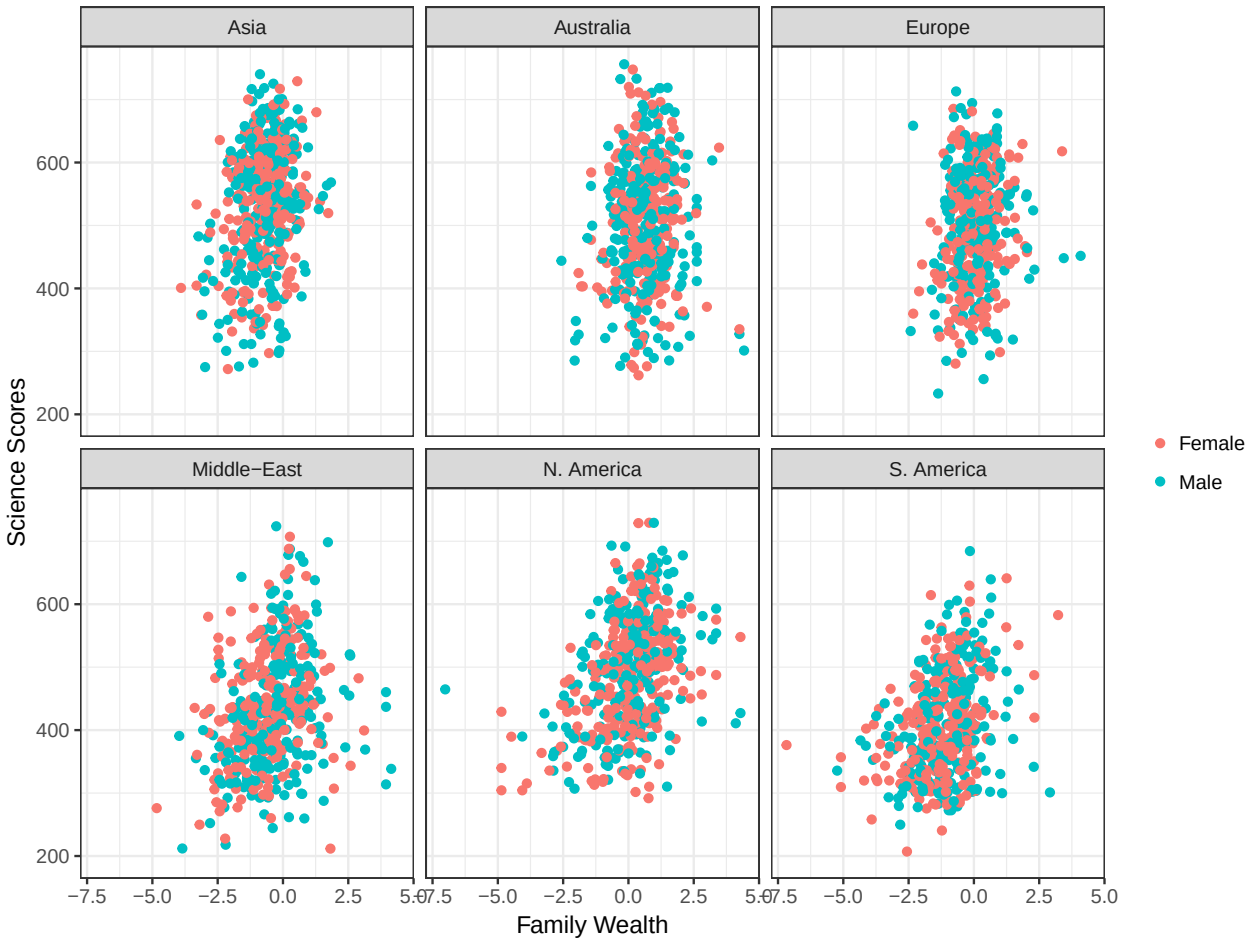
Create a scatterplot of socio-economic status (`ESCS`) and math scores (`math`) across regions (`region`) and gender (`sex`). Use `geom_smooth(method = "lm")` to draw linear regression lines (instead of loess smoothing). Do you think that the relationship between ESCS and math changes across gender and regions?

5.4 Plots for examining correlations

For a simple examination of the correlation between two continuous variables, we could just create a scatterplot matrix. In the following plot, we will create a scatterplot matrix of family wealth (`WEALTH`) and science scores (`science`) by gender (`sex`) and region (`region`). We will use region for facetting and gender for coloring the data points.

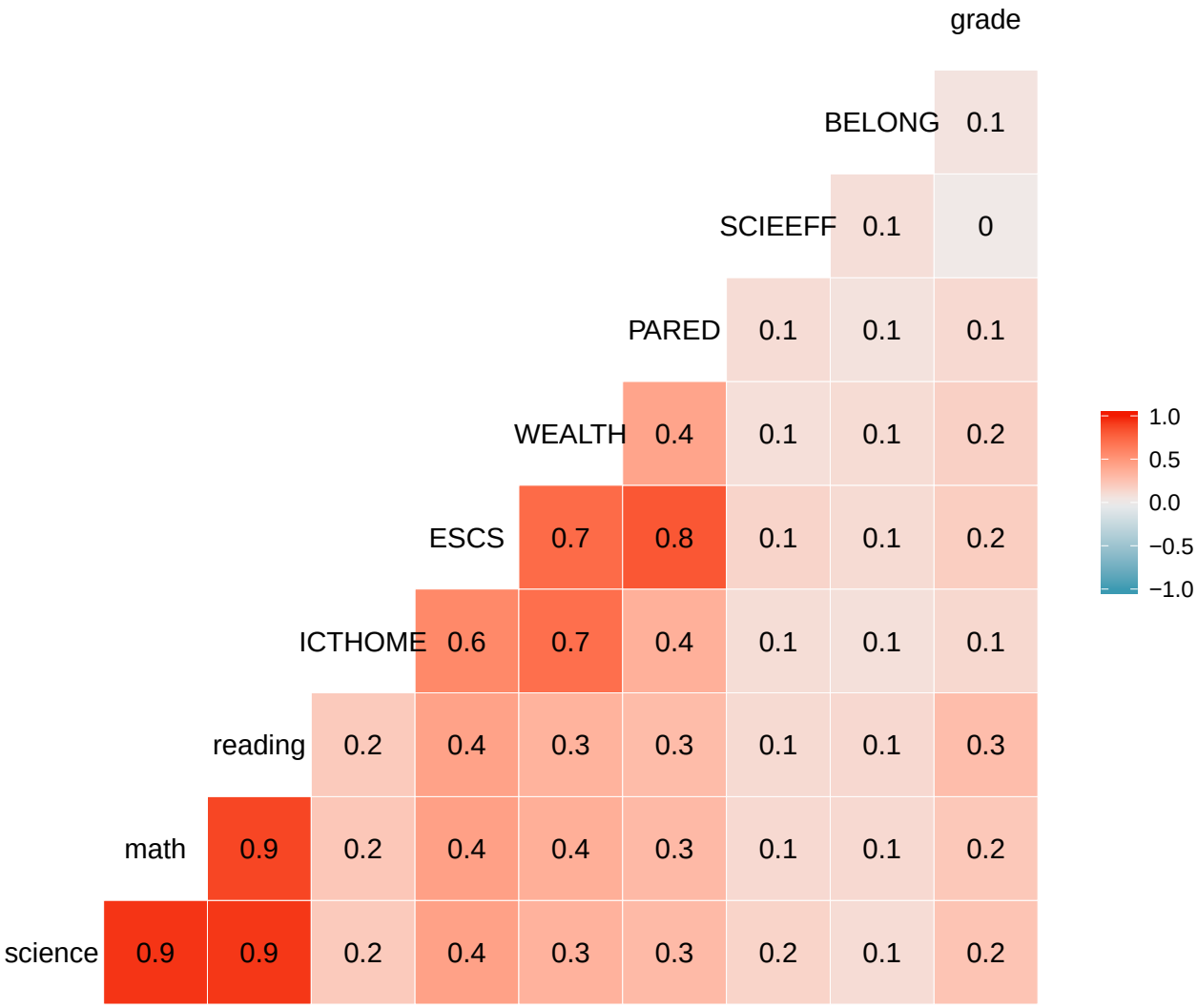
```
ggplot(data = dat_small,
       mapping = aes(x = WEALTH, y = science)) +
  geom_point(aes(color = sex)) +
  facet_wrap( ~ Region) +
  labs(x = "Family Wealth", y = "Science Scores") +
```

```
  theme_bw() +
  theme(legend.title = element_blank())
```



A more effective way for identifying correlated variables in a dataset for further statistical analyses (also known as feature extraction) is to create a correlation matrix plot. The `ggcorr()` function from the `GGally` package provides a quick way to make a correlation matrix plot. In the following example, we will create a correlation matrix plot for science, math, reading, ICT possession at home, socio-economic status, family wealth, highest parental education, science self-efficacy, sense of belonging to school, and grade level.

```
ggcorr(data = dat[,.(science, math, reading, ICTHOME, ESCS,
                     WEALTH, PARED, SCIEEFF, BELONG, grade)],
       method = c("pairwise.complete.obs", "pearson"),
       label = TRUE, label_size = 4)
```



5.5 Plots for examining means by group

Let's assume that we want to see average science scores by gender and country. First, we need to find the average science scores by country and gender and save them in a new dataset. Below we calculate average science scores and N counts by both gender and country and save the dataset as `science_summary`.

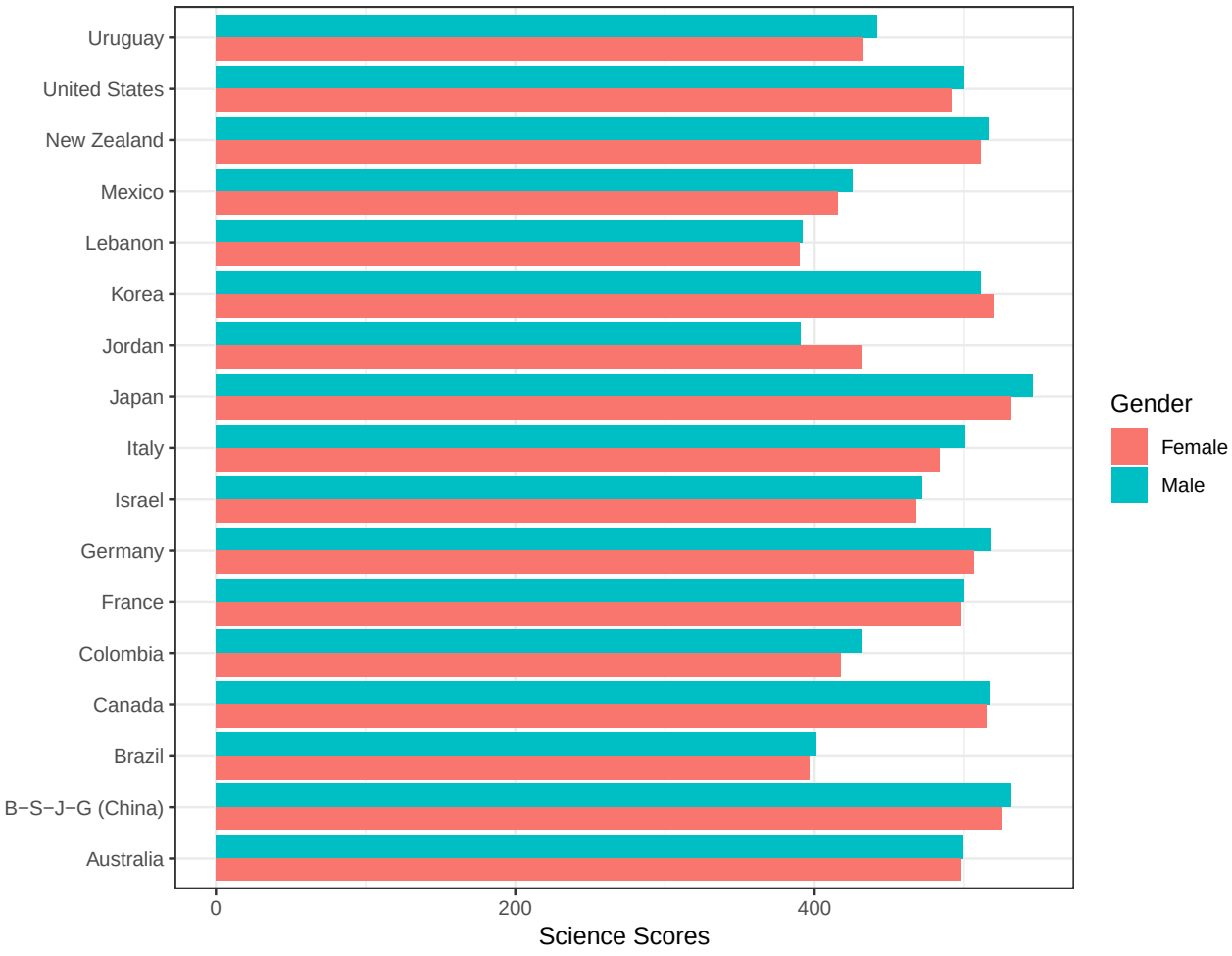
```
science_summary <- dat[,
                       .(Science = mean(science, na.rm = TRUE),
                         Freq = .N),
                       by = c("sex", "CNT")]

head(science_summary)
```

```
##       sex       CNT  Science  Freq
## 1: Female Australia 498.0377  7163
## 2:   Male Australia 499.4419  7367
## 3:   Male    Brazil 400.8134 11068
## 4: Female    Brazil 396.2647 12073
## 5: Female    Canada 515.3443 10022
```

```
## 6:   Male    Canada 517.2765 10036
```

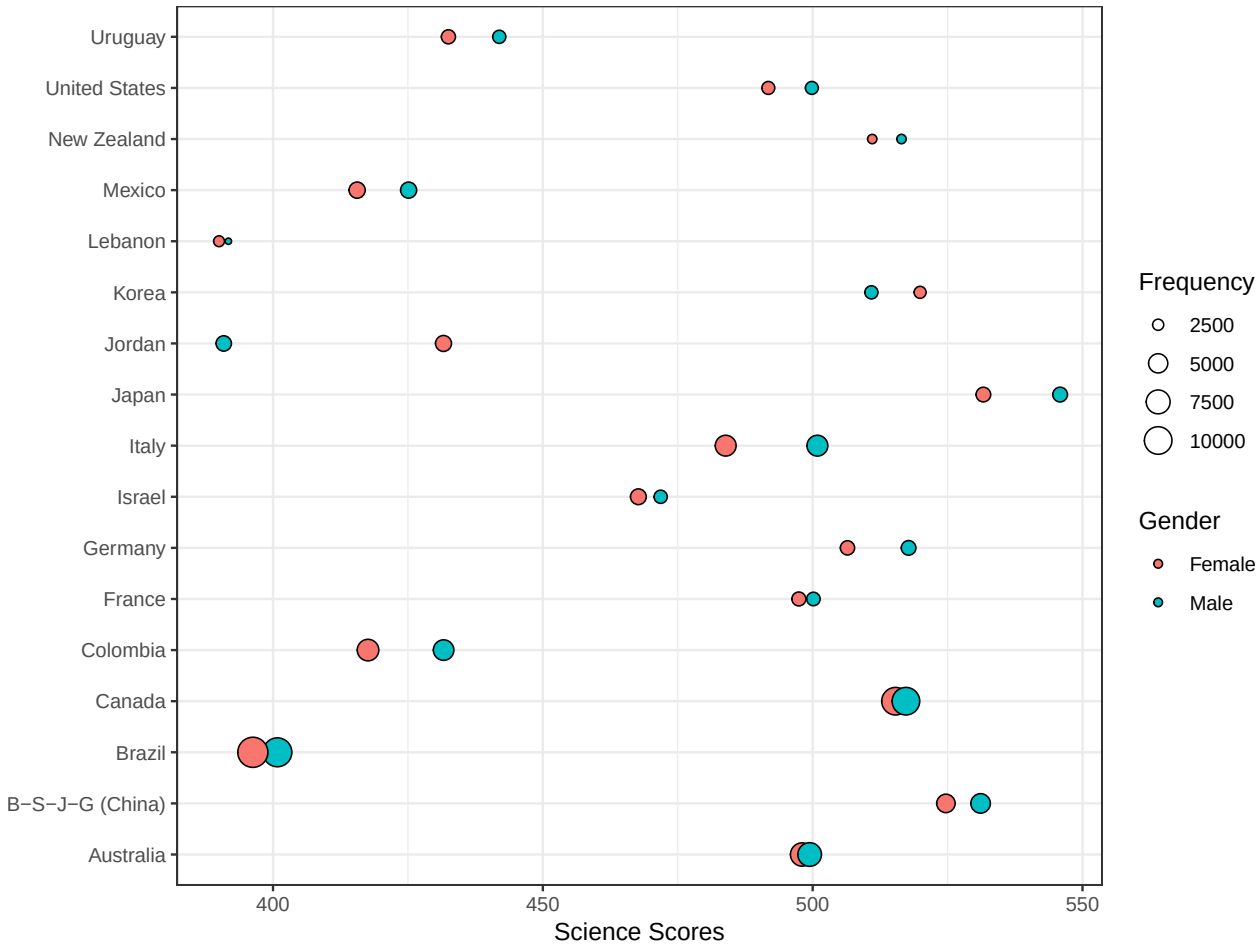
Now, we can create a simple bar graph summarizing the average science performance by gender and country, using our new dataset.

```
ggplot(data = science_summary,
       mapping = aes(x = CNT, y = Science, fill = sex)) +
  geom_bar(stat = "identity", position = "dodge") +
  coord_flip() +
  labs(x = "", y = "Science Scores", fill = "Gender") +
  theme_bw()
```



Despite their easiness and simplicity, bar graphs are not necessarily visually appealing. Thus, we will create a bubble chart to visualize the same information in a different way. A bubble chart is essentially a weighted scatterplot where a third variable determines the size of the dots in the plot. In the following bubble chart, we use *Freq* (i.e., number of students from each country) to determine the size of the dots in the plot, using `size = Freq`.

```
ggplot(data = science_summary,
       mapping = aes(x = CNT, y = Science, size = Freq, fill = sex)) +
  geom_point(shape = 21) +
  coord_flip() +
  theme_bw() +
  labs(x = NULL, y = "Science Scores", fill = "Gender",
       size = "Frequency")
```

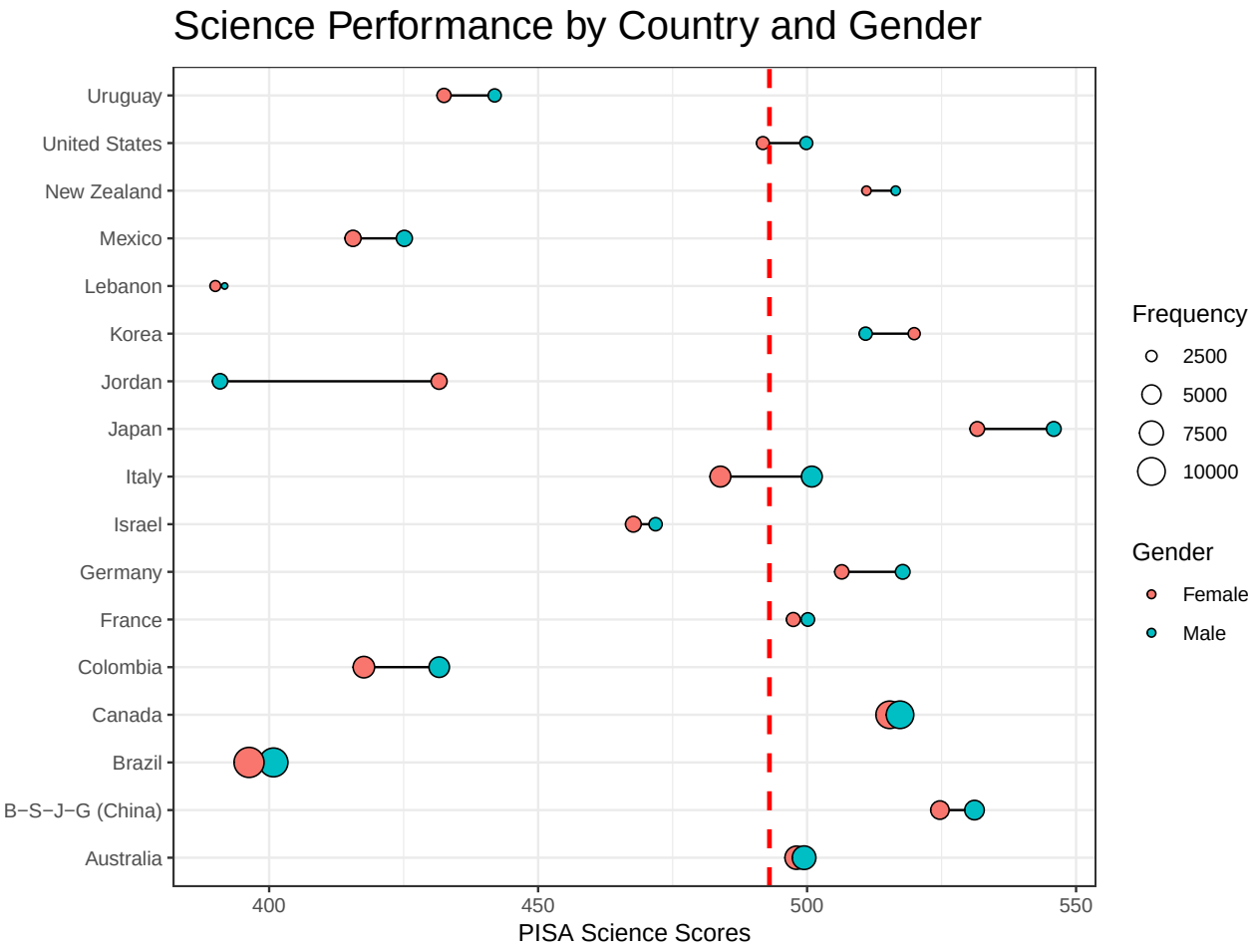


Interpretation:

- Which countries seem to have the highest numbers of students?
- Which countries seem to have the larger achievement gap in science between male and female students?

We can also create a dot plot, which is very similar to the bubble chart when one of the variables is categorical, to convey the same information even more effectively. As you will see, this is a more polished version of the bubble chart with additional titles and subtitles.

```
ggplot(data = science_summary, mapping = aes(x = CNT, y = Science, fill = sex)) +
    geom_line(aes(group = CNT)) + geom_point(aes(size = Freq), shape = 21) +
    geom_hline(yintercept = 493, linetype = "dashed", color = "red", size = 1) +
    labs(x = NULL, y = "PISA Science Scores", fill = "Gender", size = "Frequency",
        title = "Science Performance by Country and Gender") + coord_flip() +
    theme_bw() + theme(plot.title = element_text(size = 18, margin = margin(b = 10)),
    plot.subtitle = element_text(size = 10, color = "darkslategrey"))
```



5.6 Plots for ordinal/categorical variables

An *alluvial plot* can be used to summarize relationships between multiple categorical variables. In the following example, we use region (Region), gender (sex), and a survey item regarding whether parents support educational efforts and achievements (ST123Q02NA). We first create a new dataset called `dat_alluvial` to have frequency counts by region, gender, and our survey item. Because the survey item includes missing values, we label them as "missing" and then recode this variable as a factor with re-ordered levels.

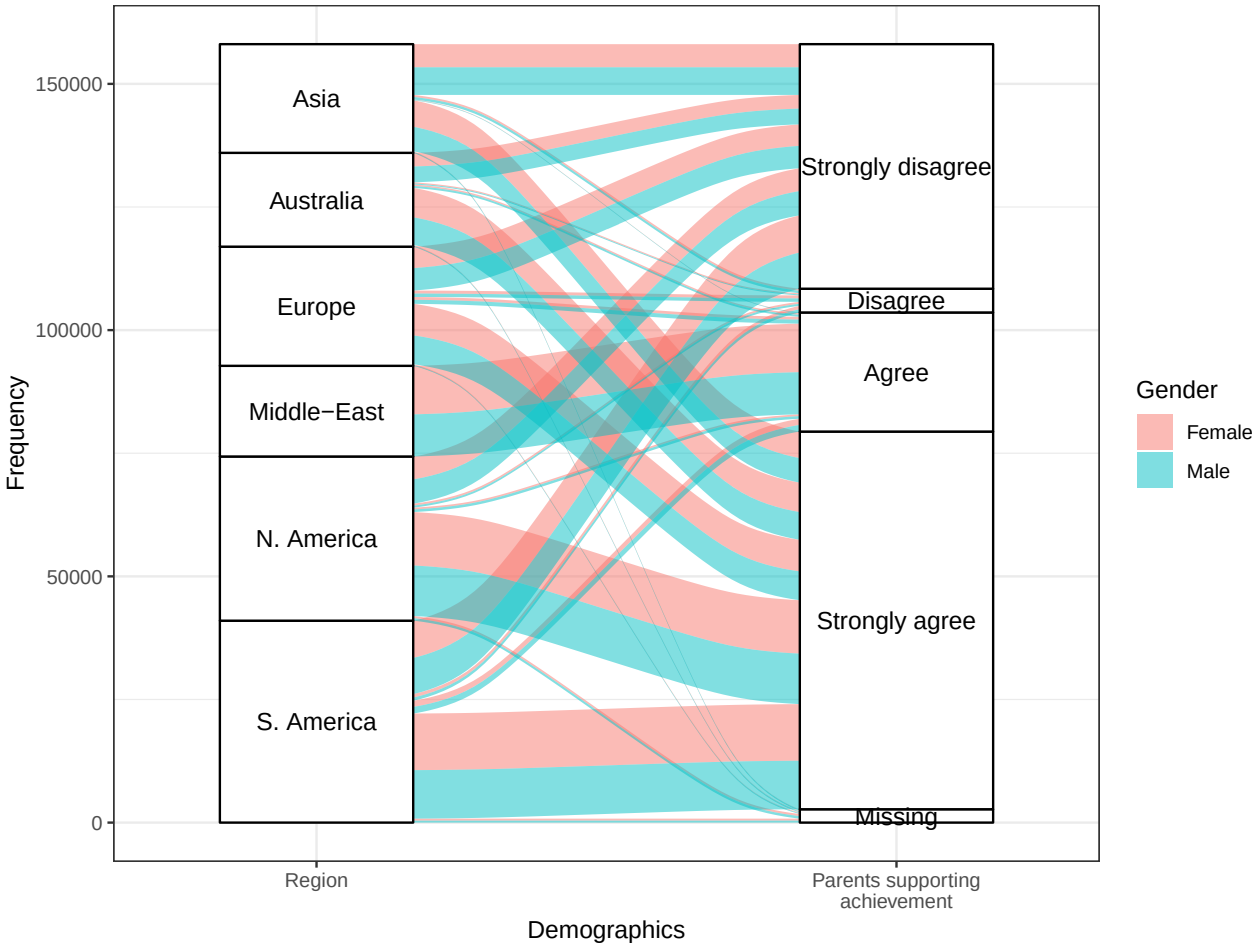
```
dat_alluvial <- dat[,
                    .(Freq = .N),
                    by = c("Region", "sex", "ST123Q02NA")
                    ][,
                      ST123Q02NA := as.factor(ifelse(ST123Q02NA == "", "Missing", ST123Q02NA))
                      ]
levels(dat_alluvial$ST123Q02NA) <- c("Strongly disagree", "Disagree", "Agree",
                                     "Strongly agree", "Missing")

head(dat_alluvial)
```

```
##       Region    sex        ST123Q02NA Freq
## 1: Australia Female          Disagree  232
## 2: Australia Female Strongly disagree 2773
```

```
## 3: Australia Female    Strongly agree 5981
## 4: Australia   Male Strongly disagree 3209
## 5: Australia   Male    Strongly agree 5626
## 6: Australia   Male           Missing  186
```

Unlike the previous visualizations, there is a new layer called `geom_alluvium`, which allows creating an alluvial plot using the `ggplot` function. We use `aes(fill = sex)` inside `geom_alluvium` to differentiate the frequencies by gender.

```
# StatStratum <- StatStratum
ggplot(data = dat_alluvial,
       aes(axis1 = Region, axis2 = ST123Q02NA, y = Freq)) +
  scale_x_discrete(limits = c("Region", "Parents supporting\nachievement"),
                   expand = c(.1, .05)) +
  geom_alluvium(aes(fill = sex)) +
  geom_stratum() +
  geom_text(stat = "stratum", label.strata = TRUE) +
  labs(x = "Demographics", y = "Frequency", fill = "Gender") +
  theme_bw()
```



Interpretation:

- Does parents' support for educational efforts and achievement vary by region and gender?

5.6.1 Exercise

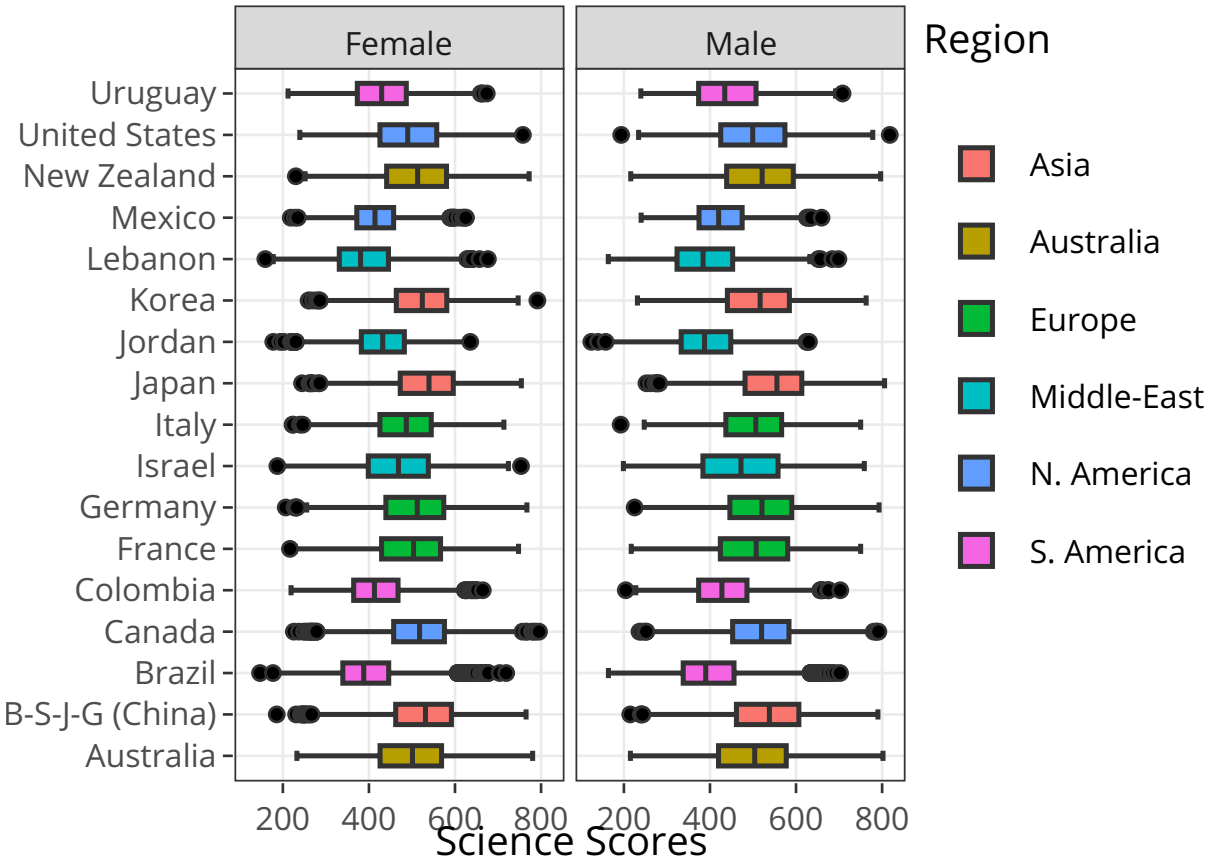
Create an alluvial plot for the survey item (ST119Q01NA) of whether students want top grades in most or all courses by region (Region) and gender (sex). Below we create the summary dataset (dat_alluvial2) for this plot. Use this dataset to draw the alluvial plot plot. How should we interpret the plot (e.g., for each region)?

```
dat_alluvial2 <- dat[,
                     .(Freq = .N),
                     by = c("Region", "sex", "ST119Q01NA")
                     ][,
                       ST119Q01NA := as.factor(ifelse(ST119Q01NA == "", "Missing", ST119Q01NA))]

levels(dat_alluvial2$ST119Q01NA) <- c("Strongly disagree", "Disagree", "Agree",
                                      "Strongly agree", "Missing")
```

5.7 Interactive plots with plotly

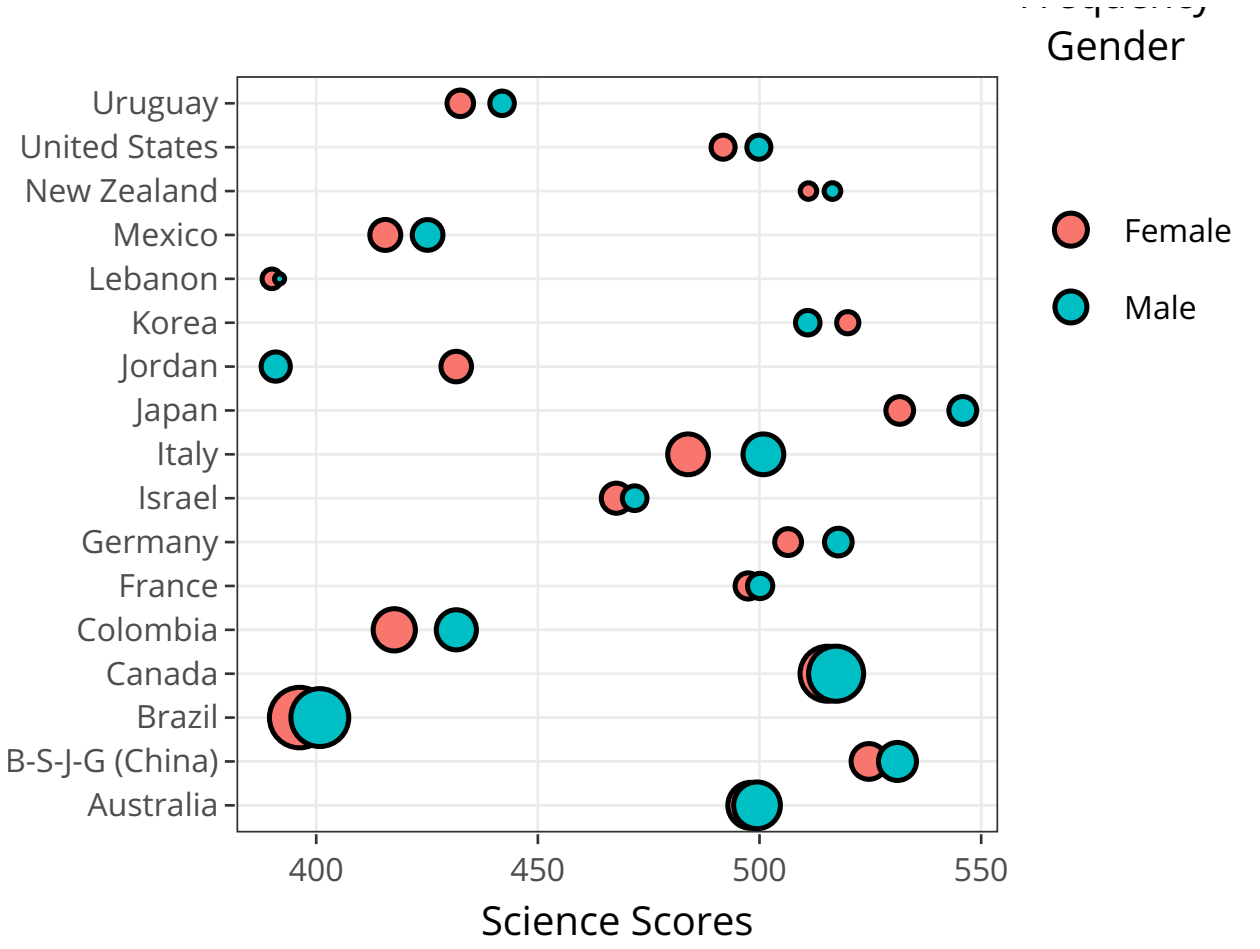
Using the plotly package, we can make more interactive visualizations. The ggplotly function from the plotly package transforms a ggplot2 plot into an interactive plot in the HTML format. In the following example, we first save a boxplot as p3 and then insert this plot into the plotly function in order to generate an interactive plot. As we hover the pointer over the plot area, the plot shows the min, max, q1, q3, and median values.

```
p3 <- ggplot(data = dat,
             mapping = aes(x = CNT, y = science, fill = Region))+
  geom_boxplot() +
  facet_grid(. ~ sex)+
  labs(x = NULL, y = "Science Scores", fill = "Region") +
  coord_flip() +
  theme_bw()

ggplotly(p3)
```



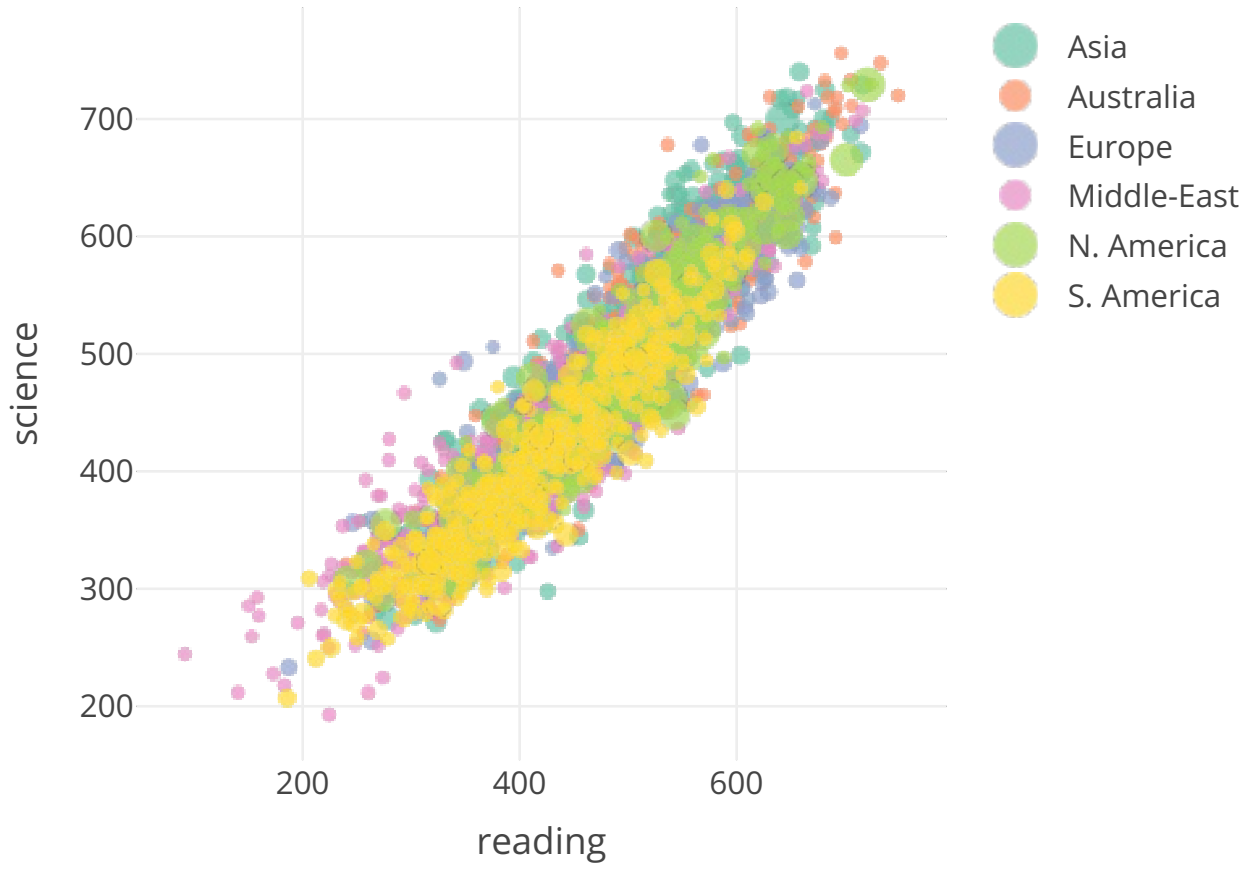
Similarly, we can transform our bubble chart into an interactive plot using `ggplotly()`.

```
p4 <- ggplot(data = science_summary,
       mapping = aes(x = CNT, y = Science, size = Freq, fill = sex)) +
  geom_point(shape = 21) +
  coord_flip() +
  theme_bw() +
  labs(x = NULL, y = "Science Scores", fill = "Gender",
       size = "Frequency")

ggplotly(p4)
```



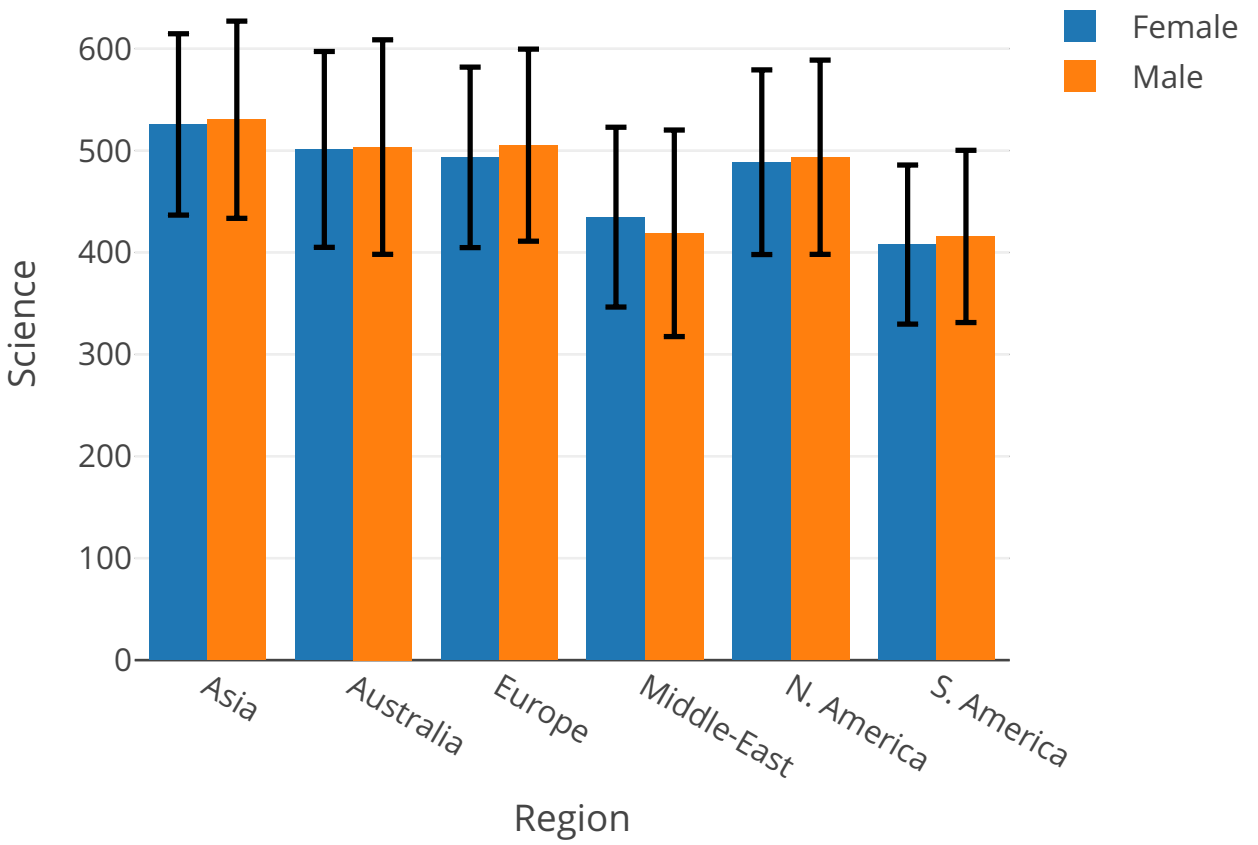
We can also use the `plot_ly` function to create interactive visualizations, without using 'ggplot2. In the following example, we create a scatterplot of reading scores and science scores where the color of the dots will be based on region and the size of the dots will be based on student weight in the PISA database. Because the resulting figure is interactive, we can click on the legend and hide some regions as we review the plot. In addition, we add a hover text (`text = ~paste("Reading: ", reading, '<br>Science:', science)`) into the plot. As we hover on the plot, it will show us a label with reading and science scores.

```
plot_ly(data = dat_small,
        x = ~reading, y = ~science, color = ~Region,
        size = ~W_FSTUWT,
        type = "scatter",
        text = ~paste("Reading: ", reading, '<br>Science:', science))
```



Lastly, we create a bar chart showing average science scores by region and gender. We will also include error bars in the plot. First we will create a new dataset `science_region` with the mean and standard deviation values by gender and region. Then, we will use this summary dataset in `plot_ly()` to draw a bar chart for females and save it as `p5`. Finally, we will add a new layer for males using `add_trace`.

```
science_region <- dat[, .(Science = mean(science, na.rm = TRUE),
                          SD = sd(science, na.rm = TRUE)),
                      by = c("sex", "Region")]

p5 <- plot_ly(data = science_region[which(science_region$sex == 'Female'),],
        x = ~Region,
        y = ~Science,
        type = 'bar',
        name = 'Female',
        error_y = ~list(array = SD, color = 'black'))

add_trace(p5, data = science_region[which(science_region$sex == 'Male'),],
          name = 'Male')
```



Check out the plotly website to see more interesting examples of interactive visualizations and dashboards.

5.7.1 Exercise

Replicate the science-by-region histogram below as a density plot and use`plotly` to make it interactive. You will need to replace `geom_histogram(alpha = 0.5, bins = 50)` with `geom_density(alpha = 0.5)`. Repeat the same process by changing `alpha = 0.5` to `alpha = 0.8`. Which version is better for examining the science score distribution?

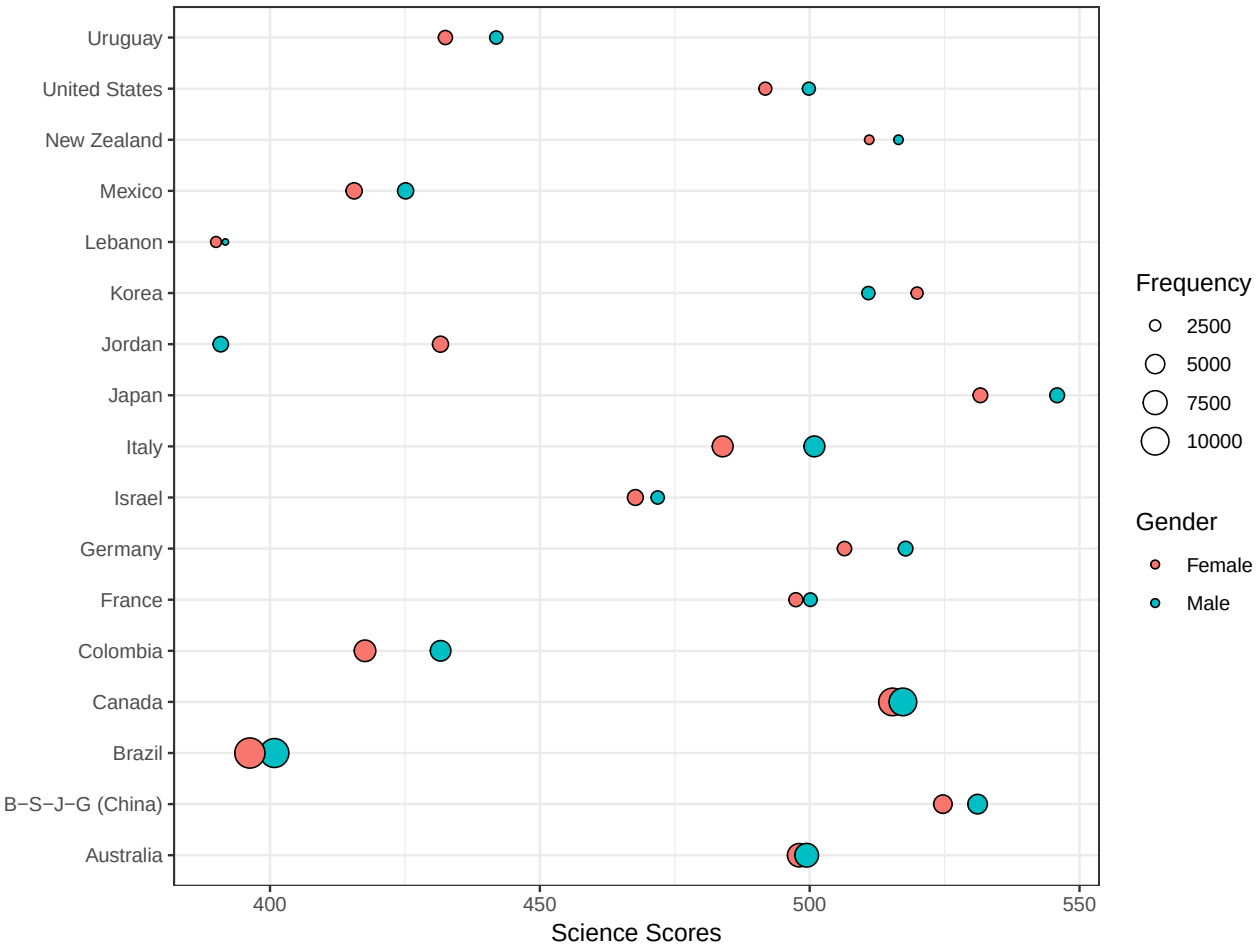
```
ggplot(data = dat,
       mapping = aes(x = science, fill = Region)) +
  geom_histogram(alpha = 0.5, bins = 50) +
  labs(x = "Science Scores", y = "Count",
       title = "Science Scores by Gender and Region") +
  facet_grid(. ~ sex) +
  theme_bw()
```

5.8 Customizing visualizations

Although `ggplot2` has many ways to customize visualizations, sometimes making a plot ready for a publication or a presentation becomes quite tedious. Therefore, we recommend the cowplot package – which is capable of quickly transforming plots created with `ggplot2` into publication-ready plots. The `cowplot` package provides a nice theme that requires a minimum amount of editing for changing sizes of axis labels, plot backgrounds, etc. In addition, we can add custom annotations to `ggplot2` plots using `cowplot` (see the cowplot vignette for more details).

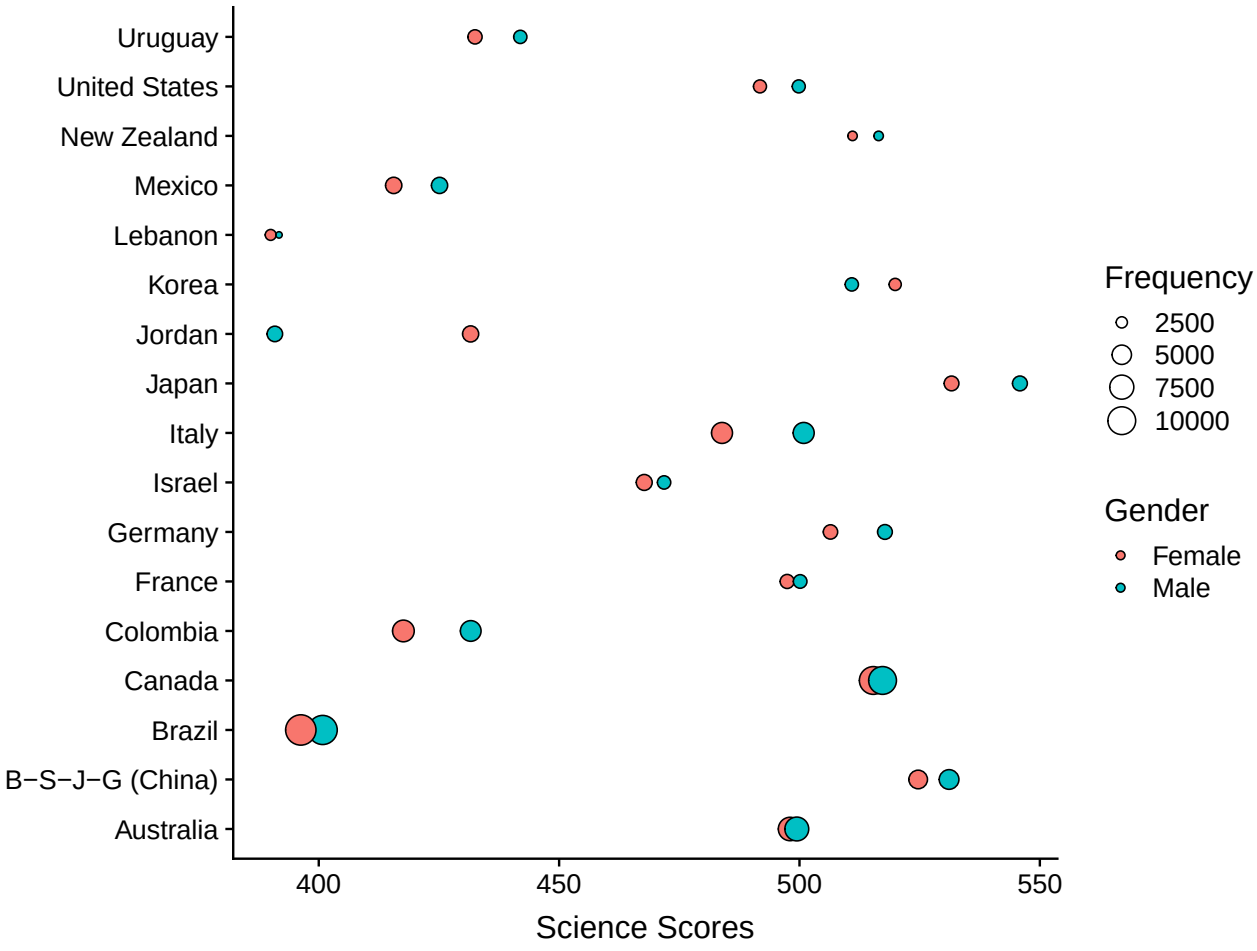
On of the plots that we created earlier was a bubble chart by gender and frequency.

```
ggplot(data = science_summary,
       mapping = aes(x = CNT, y = Science, size = Freq, fill = sex)) +
  geom_point(shape = 21) +
  coord_flip() +
  theme_bw() +
  labs(x = NULL, y = "Science Scores", fill = "Gender",
       size = "Frequency")
```



After we load the `cowplot` package and remove `theme_bw` from the plot, it will change as follows:

```
library("cowplot")

plot1 <-
  ggplot(data = science_summary,
         mapping = aes(x = CNT, y = Science, size = Freq, fill = sex)) +
  geom_point(shape = 21) +
  coord_flip() +
  labs(x = NULL, y = "Science Scores", fill = "Gender",
       size = "Frequency")

plot1
```

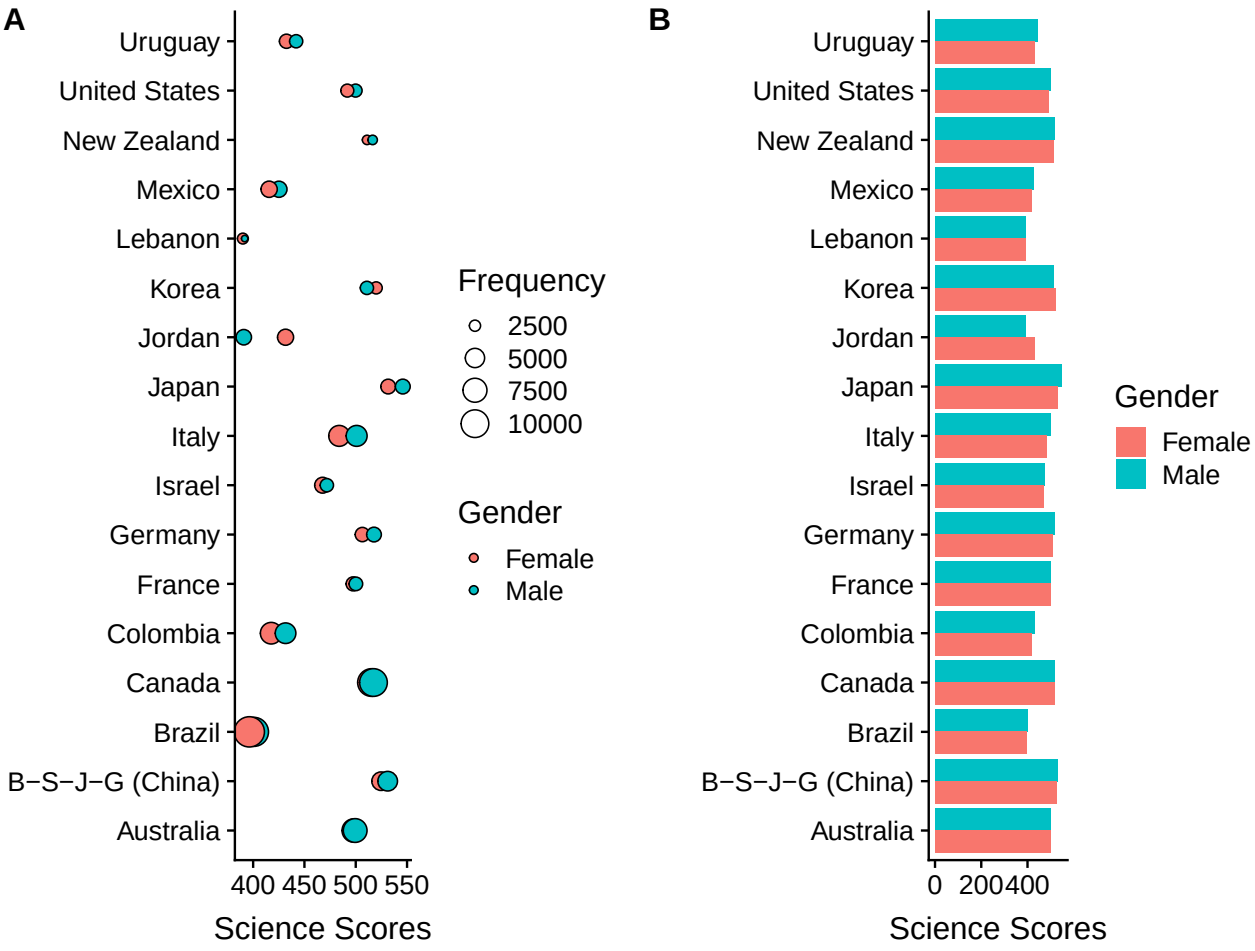


The `cowplot` package removes the gray background, gridlines, and make the axes more visible. If we want to save the plot, we can export it using `save_plot`.

```
save_plot("plot1.png", plot1,
          base_aspect_ratio = 1.6)
```

Also, `cowplot` enables combining two or more plots into one graph via the function `plot_grid`:

```
plot2 <-
  ggplot(data = science_summary,
         mapping = aes(x = CNT, y = Science, fill = sex)) +
  geom_bar(stat = "identity", position = "dodge") +
  coord_flip() +
  labs(x = "", y = "Science Scores", fill = "Gender")


plot_grid(plot1, plot2, labels = c("A", "B"))
```



If you decide not to use the `cowplot` theme, you can just simply unload the package as follows:

```
detach("package:cowplot", unload=TRUE)
```

5.9 Lab

We want to examine the relationships between reading scores and technology-related variables in the `dat` dataset that we created earlier. Create at least two visualizations (either static or interactive) using some of the variables shown below:

- Region
- sex
- grade
- HOMESCH
- ENTUSE
- ICTHOME
- ICTSCH

You can focus on a particular country or region or use the entire dataset for your visualizations.

Chapter 6

Modeling big data

6.1 Introduction to machine learning

> Machine learning is automating the automation – Dr. Pedro Domingos

Machine Learning (ML) is an important aspect of modern business applications and research nowadays. Through advanced mathematical models, ML algorithms can figure out how to perform important tasks either intuitively or by generalizing from existing observations (i.e., sample data). This is often feasible and cost-effective where manual programming is not. ML algorithms utilize sample data – also known as *training data* – to make decisions without being specifically programmed to make those decisions. As more data points become available, ML algorithms assist computer systems in progressively improving their performance so that more ambitious and complex problems can be tackled. As a result, ML has begun to be widely used in computer science and other fields, including educational measurement and psychometrics. Some ML applications include web search, spam filters for e-mails, recommender systems (e.g., Netflix and YouTube), credit scoring, fraud detection, stock trading, and drug design.

Some examples of ML in educational testing and psychometrics include automated essay scoring applications, personalized learning systems, intelligent tutoring systems, and learning analytics applications to inform instructors, students, and other stakeholders.

6.1.1 Focus of machine learning

As an inductive approach, ML focuses on making accurate predictions based on existing data, **NOT** necessarily hypothesis testing (see Figure 6.2).

Also, ML aims to learn from the data to tell you how to utilize the variables for a prediction scenario, **NOT** to give you output for a program that you wrote (see Figure 6.3).

6.1.2 Some concepts underlying machine learning

Here we want to introduce some important ML concepts, based on Dr. Pedro Domingos of University of Washington titled "A Few Useful Things to Know about Machine Learning". According to Dr. Domingos, all machine learning algorithms generally consist of combinations of three elements:

(Statistical) Learning from data = Representation + Evaluation + Optimization

1. **Representation**: A *classifier* is a system that inputs (typically) a vector of discrete and/or continuous feature values (i.e., predictors) and outputs a single discrete or continuous value (i.e., dependent or outcome variable). To build a ML application, a classifier must be represented in some formal language

Figure 6.1: Source: http://tinyurl.com/y95rd2jx

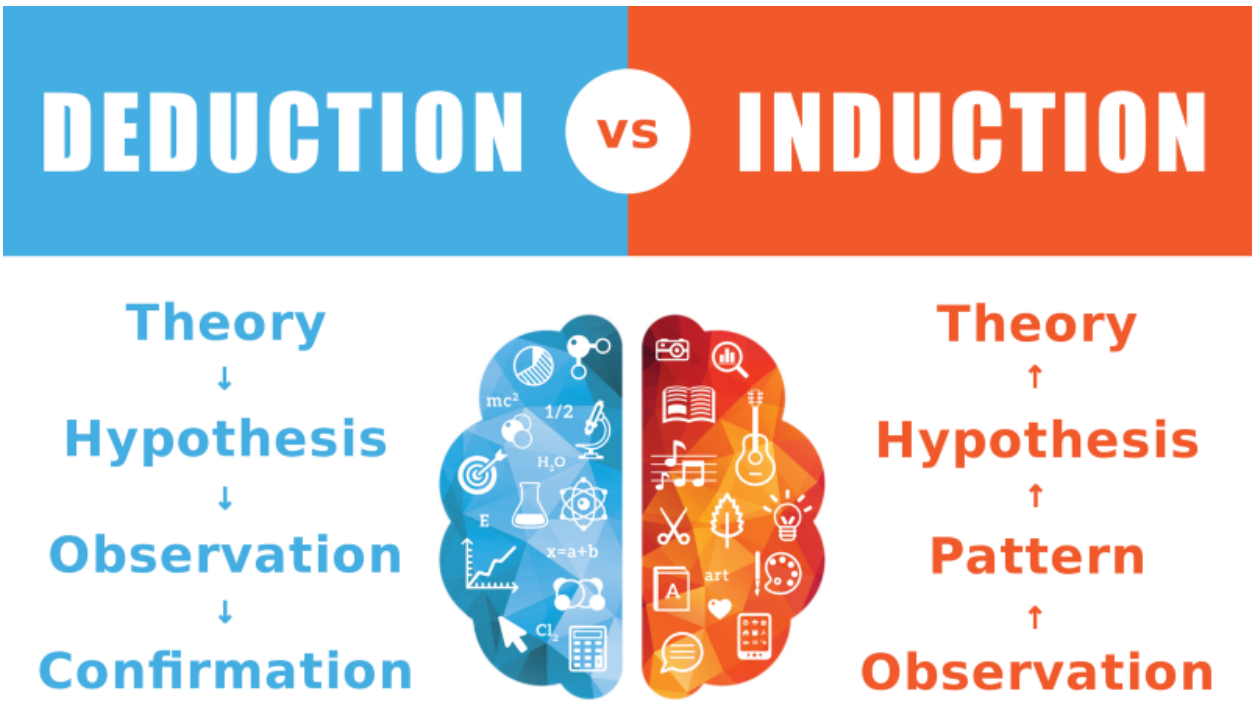


Figure 6.2: Deduction vs. induction (Source: <https://tinyurl.com/yxtt8afm>)

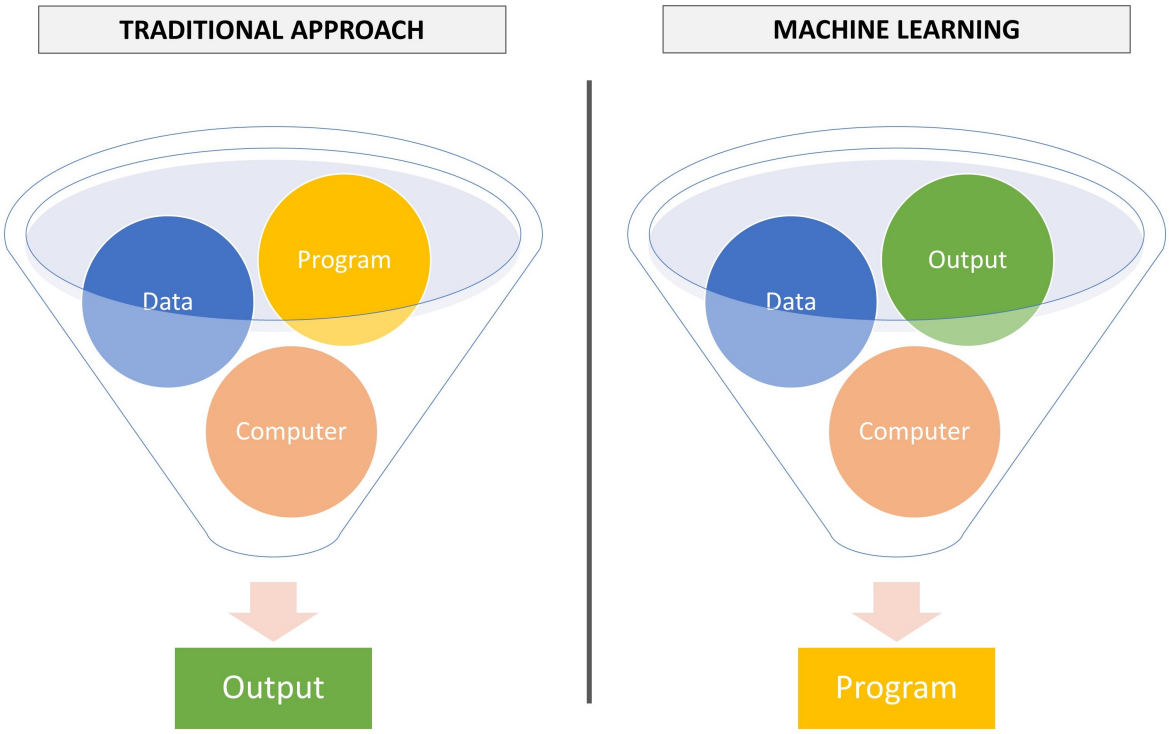


Figure 6.3: Traditional programming vs. machine learning

that the computer can handle. Then we should consider questions such as "how do we present the input data?", "how do we select what features/variables to use?", and so on. We will review some of these classifiers today – such as decision trees, support vector machines, and logistic regression.

2. **Evaluation**: An *evaluation* function is necessary for distinguishing good classifiers from bad ones. The evaluation function used internally by the algorithm may differ from the external one that we want the classifier to optimize. Common evaluation methods include accuracy/error rate, mean squared or absolute error (for continuous outcomes) and precision, accuracy, recall (i.e., sensitivity), and specificity (for categorical outcomes).

3. **Optimization**: An *optimization* method is necessary for searching among the classifiers in the language for the highest-scoring (i.e., most precise) one. The choice of optimization technique is key to the efficiency of the learner. Some optimization methods include greedy search, gradient descent, and linear programming.

6.1.3 Model development

There are several elements that impact the success of model development in ML:

- **Amount of data**: Although there are many sophisticated ML algorithms available to researchers and practitioners, they all rely on the same thing – *data*. Selecting clever ML algorithms that are capable of making the most of the available data and computing resources is important. However, without enough data, even the most sophisticated ML algorithms will return poor-quality results. Nowadays enormous amounts of data are available, but there is not enough time to process all of it.

  You can have data without information, but you cannot have information without data. – Daniel Keys Moran

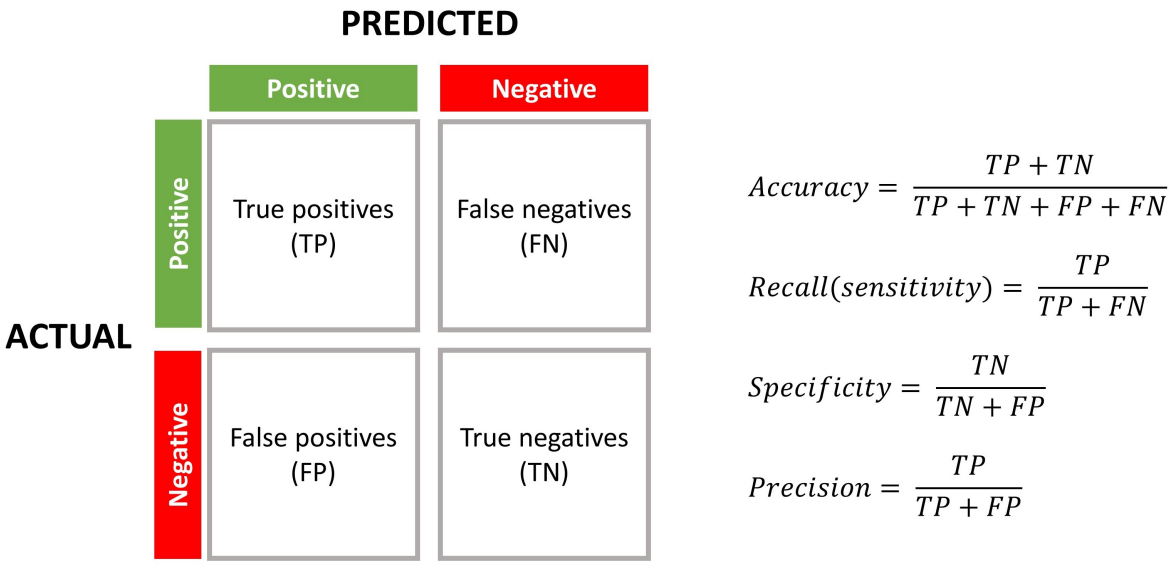


Figure 6.4: Confusion matrix for classification problems

- **Data quality**: Data quality is the essence of ML applications. ML is not magic; it can't get something out of nothing. The better quality data we provide, the more reliable and precise results we can obtain.

  More data beats clever algorithms, but better data beats more data. – Peter Norvig

- **Data wrangling**: Big data are often not in a form that is amenable to learning, but we can construct new features from the data – which is typically where most of the effort in a ML project goes. Data wrangling is the most essential skill for building a successful ML model. The processes of gathering data, integrating it, cleaning it, and pre-processing it are very time-consuming. Furthermore, ML is not a one-time process of building data and running a model to learn from the data, but rather an iterative process of running the model, analyzing the results, modifying the data, tweaking the model, and repeating.

  Data scientists spend 60% of their time on cleaning and organizing data. – Gil Press

- **Feature engineering**: Feature engineering is the key to building a successful ML model. Feature engineering refers to selecting and/or creating the most useful variables in a big dataset for a given purpose (e.g., classification). If there are many independent features that correlate well with the outcome variable, then the learning process is easy. If, however, the outcome variable is a very complex function of the features, then learning doesn't occur very easily. Categorical features nearly always need some treatment where we can use one hot encoding (similar to dummy coding) to convert such features into a form that could be provided to ML algorithms to do a better job in prediction.

  Applied machine learning is basically feature engineering. – Andrew Ng

6.1.4 Model evaluation

ML models that focus on classification problems are often evaluated based on classification accuracy, sensitivity, specificity, and precision (see Figure 6.4).

ML models that focus on regression problems are evaluated based on the fitted regression line and actual data points, as shown in Figure 6.5. Using the difference between observed data points (blue) and predicted

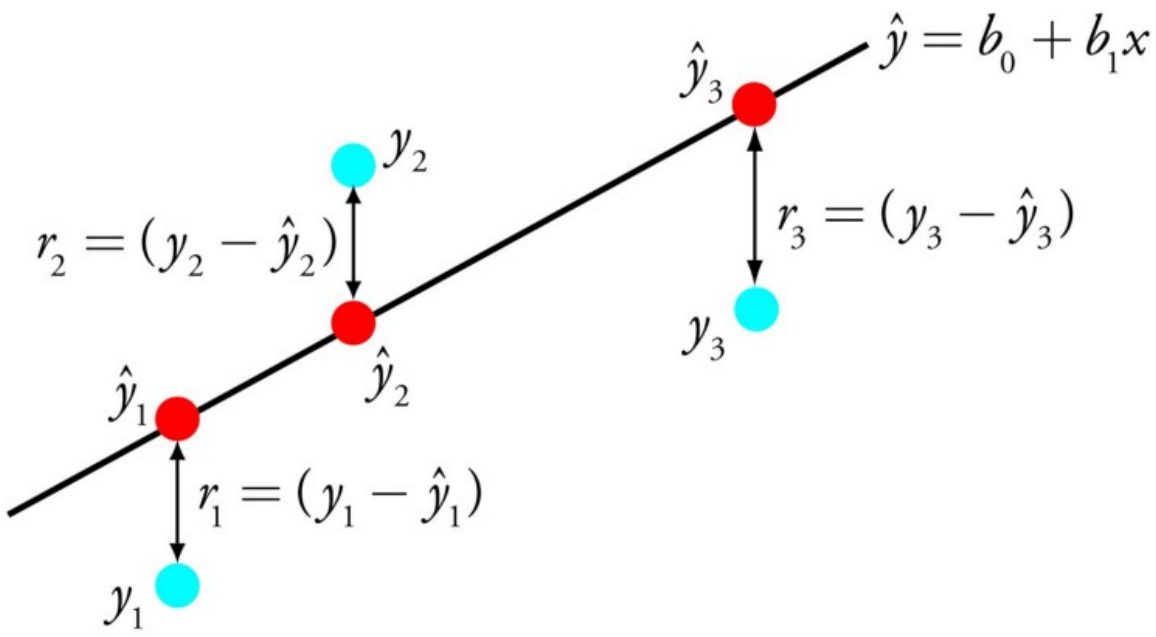


Figure 6.5: A demonstration of simple linear regression

data points (red), we can creata a summary index of error – such as mean absolute error, mean squared error, and root mean squared error.

1. **Mean Absolute Error (MAE):**

$$MAE = \frac{1}{N}\sum_{j=1}^{N}|y_i - \hat{y}_i|$$

where N is the number of observations, y_i is the observed value of the outcome variable for observation i, and $\hat{y}_i$ is the predicted value for the outcome variable for observation i.

2. **Mean Squared Error (MSE):**

$$MSE = \frac{1}{N}\sum_{j=1}^{N}(y_i - \hat{y}_i)^2$$

3. **Root Mean Squared Error (MSE):**

$$RMSE = \sqrt{MSE}$$

Figure 6.6 shows a typical machine learning pipeline that illustrates the flow of the model development and evaluation elements.

6.1.5 Key issues

In ML applications, **generalization** refers to how well the concepts learned by a machine learning model apply to specific examples (i.e., new data that the model hasn't seen yet). Therefore, the fundamental goal of ML is to build a model that can generalize beyond the examples seen in training data. Regardless

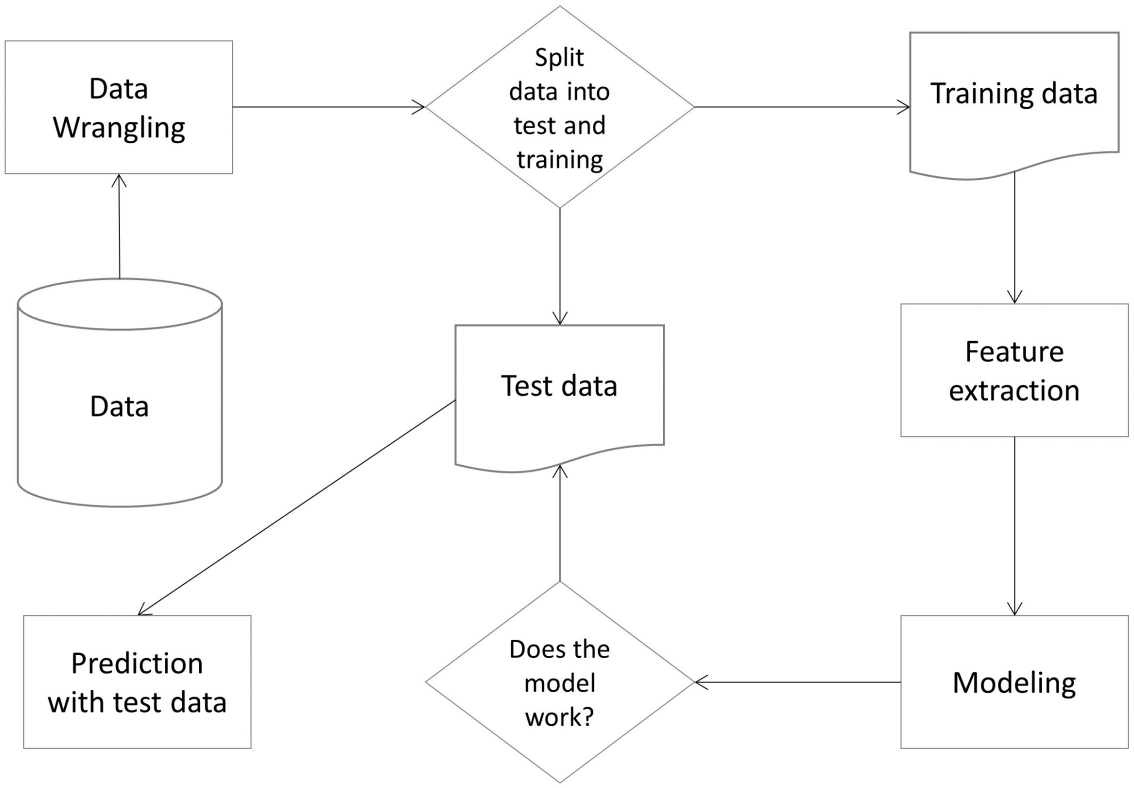


Figure 6.6: A typical machine learning pipeline

of how many observations we have in training, the model will produce inaccurate results for at least some observations in the test data. This is primarily because we are very unlikely to see those exact examples from training data again when testing the model with new (or validation) data. That is, getting highly precise results in training data is easy, whereas generalizing the model beyond training data is hard. Therefore, most machine learning beginners would easily fall for the illusion of success with training data and then get immediately disappointed with the results from new data.

When we talk about how well a ML model learns from training data and generalizes to new data, there are two key issues: **overfitting** and **underfitting**.

- **Overfitting** refers to a model that models the training data too well. It happens when a ML model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. In the context of automated essay scoring, overfitting would occur when all the essays (including words, punctuation, word combinations) from the training data are used to maximize the accuracy of the essay scores. Because the model would be very specific to the words or phrases used by students in the training data, the same ML model would yield very poor results when the essay is given to a different group of students who write essays quite differently (e.g., English language learners). Overfitting typically occurs with ML models that implement nonparametric and nonlinear function to learn from the data (e.g., neural network models).
- **Underfitting** refers to a ML model that can neither model the training data nor generalize to new data – which means that our ML attempt was a complete failure. An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data. The obvious remedy to underfitting is to try alternate ML algorithms. Nevertheless, it does provide a good contrast to the problem of overfitting.

Both overfitting and underfitting may cause poor performance of ML algorithms; but by far the most common problem in ML applications is overfitting. There are two important techniques that we can use when evaluating ML algorithms to limit overfitting:

1. **Use a resampling technique to estimate model accuracy**: The most popular resampling technique is k-fold cross validation. This method allows us to train and test our model k-times on different subsets of training data and build up an estimate of the performance of a ML model on unseen data. Using cross validation is a gold standard in ML applications for estimating model accuracy on unseen data (see Figure 6.7).
2. **Hold back a validation dataset**: If we already have new data (or very large data from which we can spare enough data), using a validation dataset is also an excellent practice.

In conclusion, we ideally want to select a model at the sweet spot between underfitting and overfitting. As we use more data for training the model, we can review the performance of the ML algorithm over time. We can plot both the outcome on the training data and the outcome on the test data we have held back from the training process.

Over time, as the ML algorithm learns, the prediction error for the model on the training data goes down and so does the error on the test data. If we train the model for too long, the performance on the training data may continue to decrease because the model is overfitting and learning irrelevant details and noise in the training dataset. At the same time, the error for the test set starts to increase again as the model's ability to generalize decreases. The sweet spot is the point just before the error on the test data starts to increase where the model has good accuracy on both the training data and the unseen test data.

6.2 Types of machine learning

In general, ML applications can be categorized in two ways:

1. Supervised learning vs. unsupervised learning
2. ML for classification problems vs. ML for regression problems

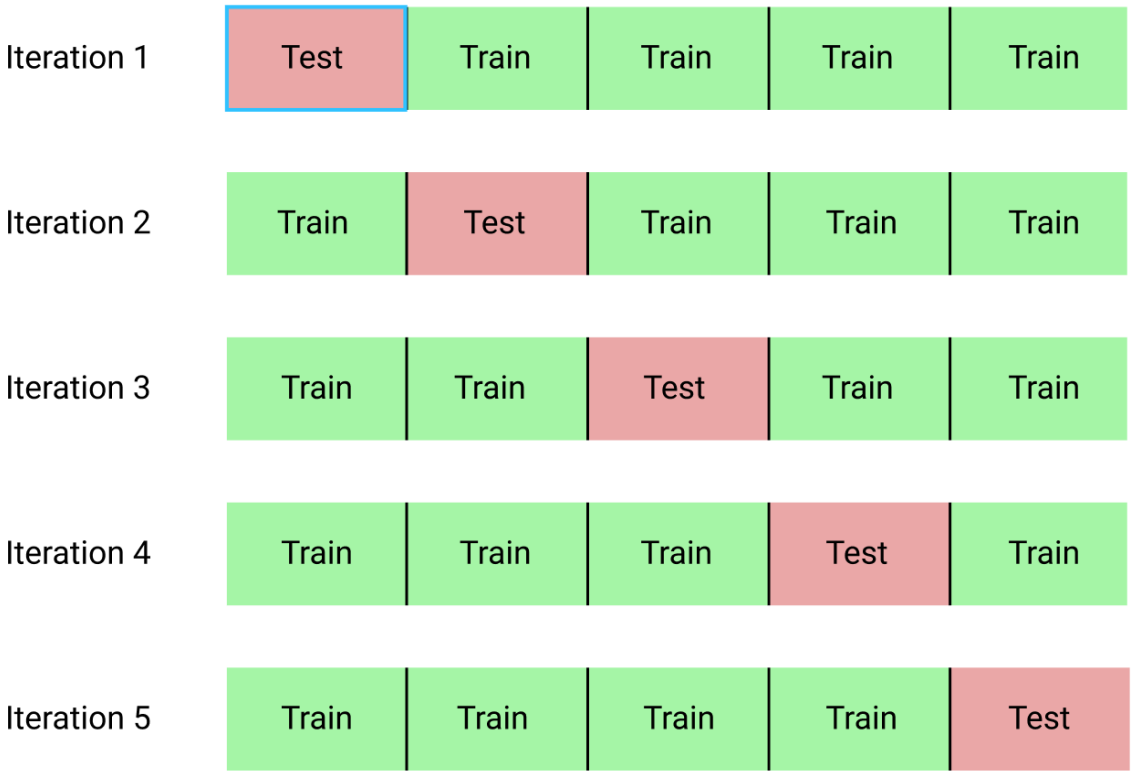


Figure 6.7: An illustration of *k*-fold cross validation

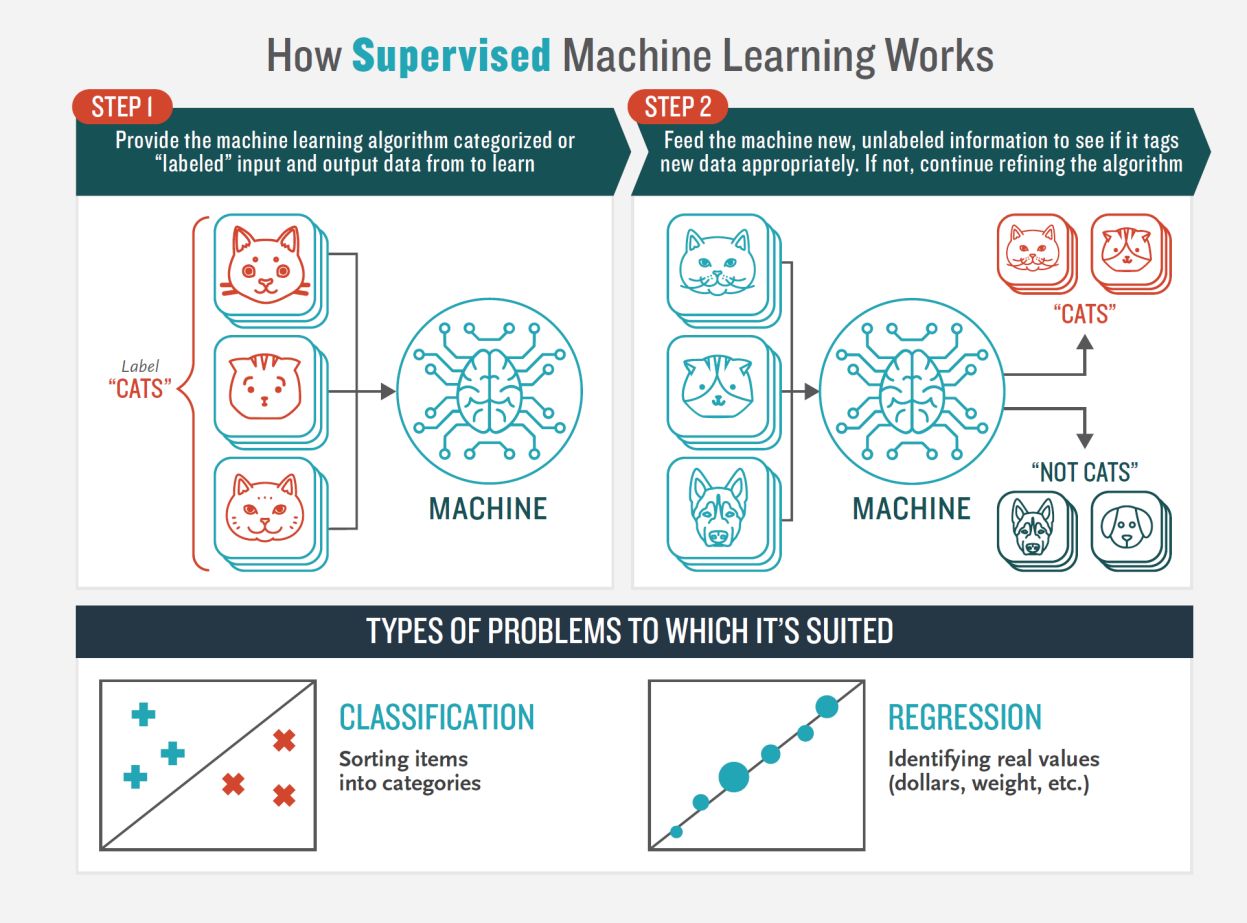


Figure 6.8: How supervised machine learning works

In **supervised learning**, ML algorithms are given **training data** categorized as input variables and output variables from which to learn patterns and make inferences on previously unseen data (**testing data**). The goal of supervised learning is for machines to replicate a mapping function we have identified for them (for example, which students passed or failed the test at the end of the semester). Provided enough examples, ML algorithms can learn to recognize and respond to patterns in data without explicit instructions. Supervised machine learning is typically used for **classification** tasks, in which we segment the data inputs into categories (e.g., for pass/fail decisions, strongly agree/agree/neutral/disagree/strongly disagree), and **regression** tasks, in which the output variable is a real value, such as a test score. The accuracy of supervised learning algorithms typically is easy to evaluate, because there is a known, "ground truth"" (output variable) to which the algorithm is optimizing (see Figure 6.8).

Unsupervised machine learning is an approach to training ML in which the algorithm is given **only input data**, from which it identifies patterns on its own. The goal of unsupervised learning is for algorithms to identify underlying patterns or structures in data to better understand it. Unsupervised learning is closer to how humans learn most things in life: through observation, experience, and analogy. Unsupervised learning is best used for clustering problems – for example, grouping examinees based on their response times and engagement with the items during testing in order to detect anomalies. It is also useful for "association", in which ML algorithms independently discover rules in data; for example, students who tend to answer math items slowly also tend to answer science items slowly. The accuracy of unsupervised learning is harder to evaluate, as there is no predefined ground truth the algorithm is working toward (see Figure 6.9).

Figure 6.10 below shows most widely used algorithms for both supervised and unsupervised ML applications.

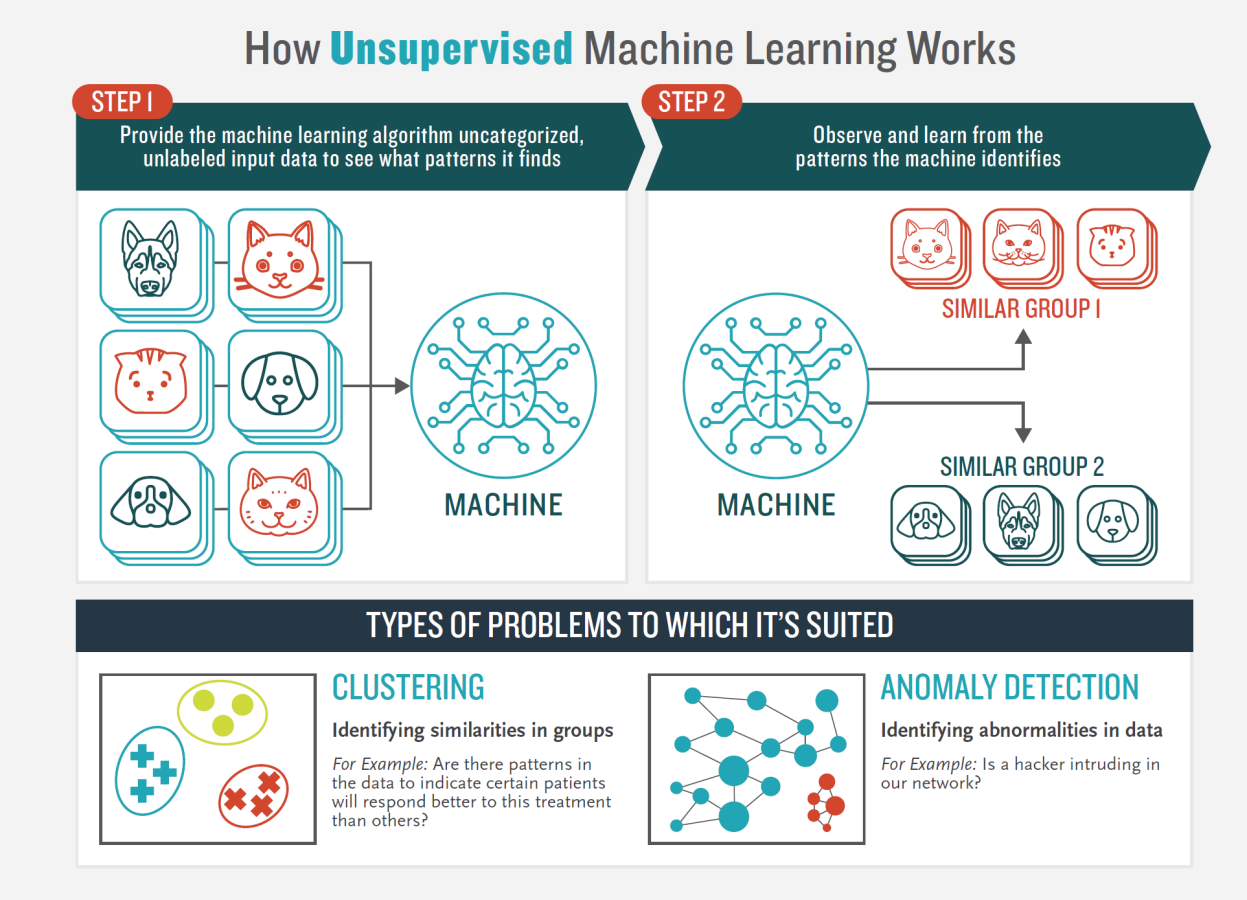


Figure 6.9: How unsupervised machine learning works

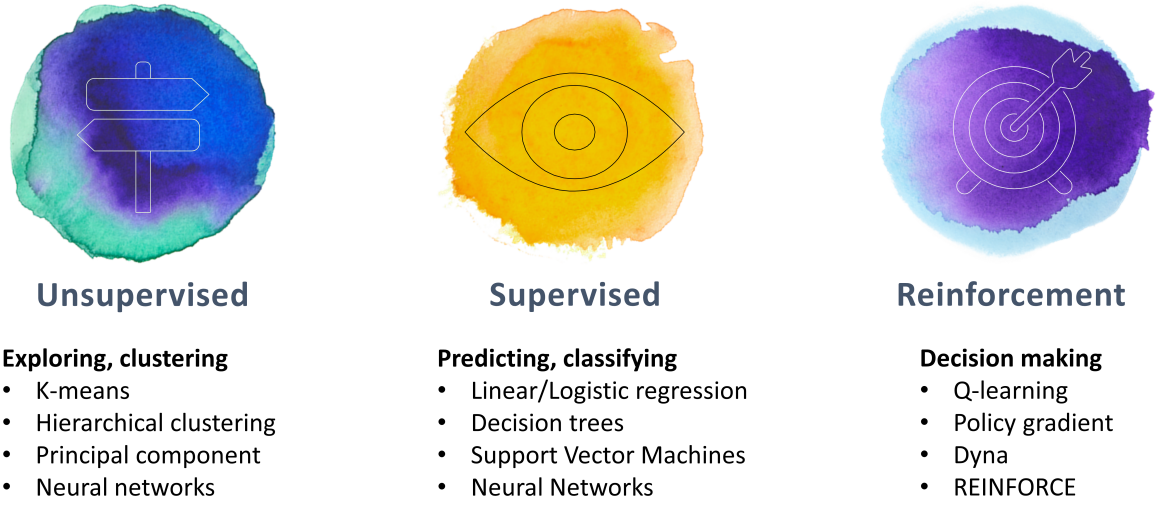


Figure 6.10: Widely used machine learning algorithms

The last column in Figure 6.10 refers to "reinforcement learning" – a more specific type of machine learning – but we will not be covering reinforcement learning in this training session.

Chapter 7

Supervised Machine Learning - Part I

7.1 Decision Trees

Decision trees (also known as classification and regression trees – CART) are an important type of algorithm for predictive modeling and machine learning. In general, the CART approach relies on *stratifying* or *segmenting* the prediction space into a number of simple regions. In order to make regression-based or classification-based predictions, we use the mean or the mode of the training observations in the region to which they belong.

A typical layout of a decision tree model looks like a binary tree. The tree has a root node that represents the starting point of the prediction. There are also decision nodes where we split the data into a smaller subset and leaf nodes where we make a decision. Each node represents a single input variable (i.e., predictor) and a split point on that variable. The leaf nodes of the tree contain an output variable (i.e., dependent variable) for which we make a prediction. Predictions are made by walking the splits of the tree until arriving at a leaf node and output the class value at that leaf node. Figure 7.1 shows an example of a decision tree model in the context of a binary dependent variable (accepting or not accepting a new job offer).

Although decision trees are not highly competitive with the advanced supervised learning approaches, they are still quite popular in ML applications because they:

- are fast to learn and very fast for making predictions.
- are often accurate for a broad range of problems.
- do not require any special preparation for the data.
- are highly interpretable compared to more complex ML methods (e.g., neural networks).
- are very easy to explain to people as the logic of decision trees closely mirrors human decision-making.
- can be displayed graphically, and thus are easily interpreted even by a non-expert.

In a decision tree model, either categorical and continuous variables can be used as the outcome variable depending on whether we want classification trees (categorical outcomes) or regression trees (continuous outcomes). Decision trees are particularly useful when predictors interact well with the outcome variable (and with each other).

7.1.1 Regression trees

In regression trees, the following two steps will allow us to create a decision tree model:

1. We divide the prediction space (with several predictors) into distinct and non-overlapping regions, using a *top-down*, *greedy* approach – which is also known as *recursive binary splitting*. We begin splitting at the top of the tree and then go down by successively splitting the prediction space into two new

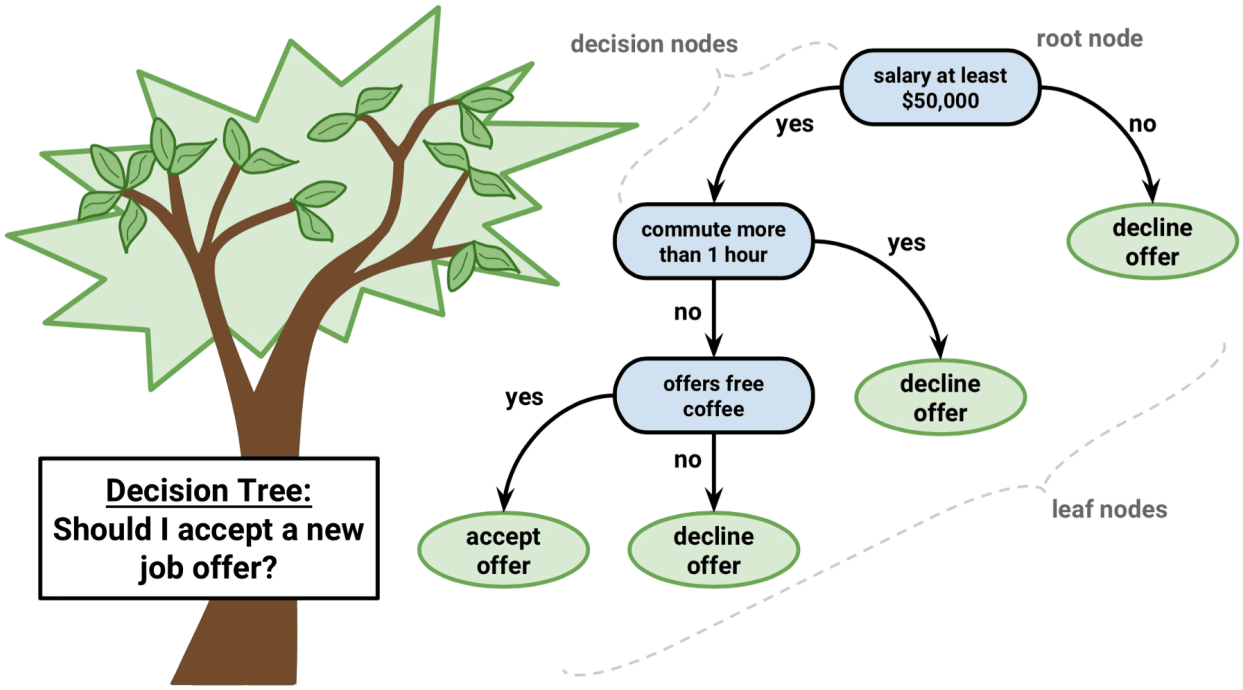


Figure 7.1: An example of decision tree approach

branches. This step is completed by dividing the prediction space into high-dimensional rectangles and minimizing the following equation:

$$RSS = \sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

where RSS is the residual sum of squares, y_i is the observed predicted variable for the observations $i = (1, 2, 3, \ldots, N)$ in the training data, j is the index for the j^{th} split, s is the cutpoint for a given predictor X_i, $\hat{y}_{R_1}$ is the mean response for the observations in the $R_1(j, s)$ region of the training data and $\hat{y}_{R_2}$ is the mean response for the observations in the $R_2(j, s)$ region of the training data.

2. Once all the regions $R_1, \ldots, R_J$ have been created, we predict the response for a given observation using the mean of the observations in the region of the training data to which that observation belongs.

7.1.2 Classification trees

A classification tree is very similar to a regression tree, except that the decision tree predicts a qualitative (i.e., categorical) variable rather than a quantitative (i.e., continuous and numerical) variable. The procedure for splitting the data in multiple branches is the same as the one we described for the regression tree above. The only difference is that instead of using the mean of the observations in the region of the training data, we assume that each observation belongs to the *mode* class (i.e., most commonly occurring class) of the observations in the region of the training data. Also, rather than minimizing RSS, we try to minimize the *classification error rate*, which is the fraction of the training observations in a given region that do not belong to the most common class:

$$E = 1 - max_k(\hat{p}_{mk})$$

where $\hat{p}_{mk}$ is the proportion of training observations in the m^{th} region that are from the k^{th} class. However, only classification error is **NOT** good enough to split decision trees. Therefore, there are two other indices for the same purpose:

1. **The Gini index**:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

where K represents the number of classes. This is essentially a measure of total variance across the K classes. A small Gini index indicates that a node contains predominantly observations from a single class.

2. **Entropy**:

$$Entropy = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

Like the Gini index, the entropy will also take on a small value if the m^{th} node is pure.

When building a classification tree, either the Gini index or the entropy is typically used to evaluate the quality of a particular split, as they are more sensitive to the changes in the splits than the classification error rate. Typically, the Gini index is better for minimizing misclassification, while the Entropy is better for exploratory analysis.

7.1.3 Pruning decision trees

Sometimes decision trees end up having many branches and nodes, yielding a model that overfits the training data and poorly fits the validation or test data. To eliminate this overfitting problem, we may prefer to have a smaller and more interpretable tree with fewer splits at the cost of a little bias. One strategy to achieve this is to grow a very large tree and then prune it back in order to obtain a *subtree*.

Given a subtree, we can estimate its error in the test or validation data. However, estimating the error for every possible subtree would be computationally too expensive. A more feasible way is to use *cost complexity pruning* by getting a sequence of trees indexed by a nonnegative tuning parameter α – which also known as the complexity parameter (cp). The cp parameter controls a trade-off between the subtree's complexity and its fit to the training data. As the cp parameter increases from zero, branches in the decision tree get pruned in a nested and predictable fashion. To determine the ideal value for the cp parameter, we can try different values of cp in a validation set or use cross-validation (e.g., K-fold approach). By checking the error (using either RSS, or Gini index, or Entropy depending on the prediction problem) for different sizes of decision trees, we can determine the ideal point to prune the tree.

7.2 Decision trees in R

In the following example, we will build a classification tree model, using the science scores from PISA 2015. Using a set of predictors in the **pisa** dataset, we will predict whether students are above or below the mean scale score for science. The average science score in PISA 2015 was 493 across all participating countries (see PISA 2015 Results in Focus for more details). Using this score as a cut-off value, we will first create a binary variable called `science_perf` where `science_perf`= High if a student's science score is equal or larger than 493; otherwise `science_perf`= Low.

```
pisa <- pisa[, science_perf := as.factor(ifelse(science >= 493, "High", "Low"))]
```

In addition, we will subset the students from the United States and Canada and choose some variables (rather than the entire set of variables) to make our example relatively simple and manageable in terms of time. We will use the following variables in our model:

Label	Description
WEALTH	Family wealth (WLE)
HEDRES	Home educational resources (WLE)
ENVAWARE	Environmental Awareness (WLE)
ICTRES	ICT Resources (WLE)
EPIST	Epistemological beliefs (WLE)
HOMEPOS	Home possessions (WLE)
ESCS	Index of economic, social and cultural status (WLE)
reading	Students' reading score in PISA 2015
math	Students' math score in PISA 2015

We call this new dataset `pisa_small`.

```
pisa_small <- subset(pisa, CNT %in% c("Canada", "United States"),
                     select = c(science_perf, WEALTH, HEDRES, ENVAWARE, ICTRES,
                                EPIST, HOMEPOS, ESCS, reading, math))
```

Before we begin the analysis, we need to install and load all the required packages.

```
decision_packages <- c("caret", "rpart", "rpart.plot", "randomForest", "modelr")
install.packages(decision_packages)

library("caret")
library("rpart")
library("rpart.plot")
library("randomForest")
library("modelr")

# Already installed packages that we will use
library("data.table")
library("dplyr")
library("ggplot2")
```

Next, we will split our dataset into a training dataset and a test dataset. We will train the decision tree on the training data and check its accuracy using the test data. In order to replicate the results later on, we need to set the seed – which will allow us to fix the randomization. Next, we remove the missing cases, save it as a new dataset, and then use `createDataPartition()` from the `caret` package to create an index to split the dataset as 70% to 30% using p = 0.7.

```
# Set the seed before splitting the data
set.seed(442019)

# We need to remove missing cases
pisa_nm <- na.omit(pisa_small)

# Split the data into training and test
index <- createDataPartition(pisa_nm$science_perf, p = 0.7, list = FALSE)
train_dat <- pisa_nm[index, ]
test_dat  <- pisa_nm[-index, ]
```

```
nrow(train_dat)
```

```
## [1] 16561
```

```
nrow(test_dat)
```

```
## [1] 7097
```

Alternatively, we could simply create the index using random number generation with `sample.int()`.

```
n <- nrow(pisa_nm)
index <- sample.int(n, size = round(0.7 * n))
```

To build a decision tree model, we will use the `rpart` function from the `rpart` package. In the function, there are several elements:

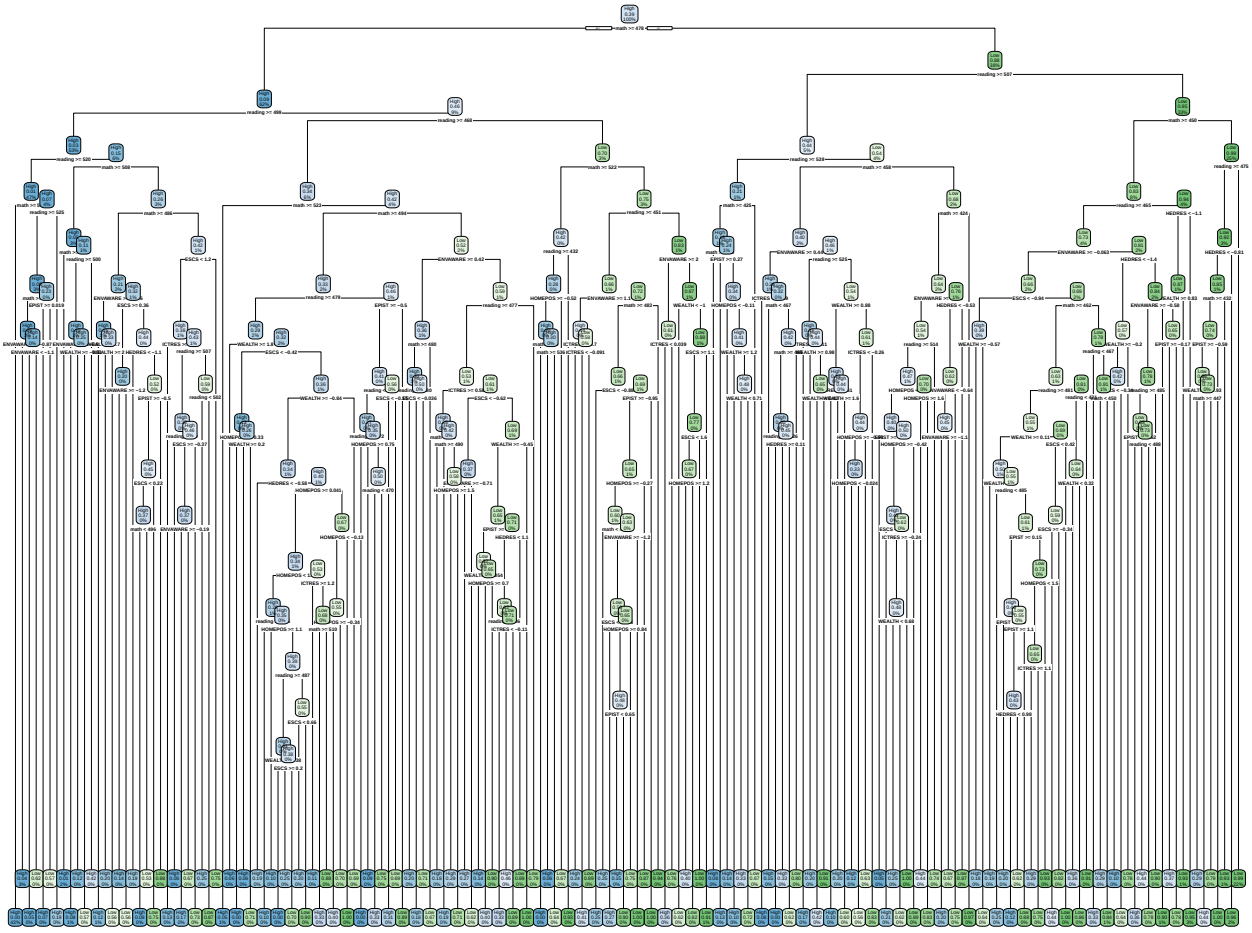
- `formula = science_perf ~ .` defines the dependent variable (i.e., science_perf) and the predictors (and `~` is the separator). Because we use `science_perf ~ .`, we use all variables in the dataset (except for science_perf) as our predictors. We could also write the same formula as `science_perf ~ math + reading + ESCS + ... + WEALTH` by specifying each variable individually.
- `data = train_dat` defines the dataset we are using for the analysis.
- `method = "class"` defines what type of decision tree we are building. `method = "class"` defines a classification tree and `method = "anova"` defines a regression tree.
- `control` is a list of control (i.e., tuning) elements for the decision tree algorithm. `minsplit` defines the minimum number of observations that must exist in a node (default = 20); `cp` is the complexity parameter to prune the subtrees that don't improve the model fit (default = 0.01, if `cp` = 0, then no pruning); `xval` is the number of cross-validations (default = 10, if `xval` = 0, then no cross validation).
- `parms` is a list of optional parameters for the splitting function.`anova` splitting (i.e., regression trees) has no parameters. For `class` splitting (i.e., classification tree), the most important option is the split index – which is either `"gini"` for the Gini index or `"information"` for the Entropy index. Splitting based on `information` can be slightly slower compared to the Gini index (see the vignette for more information).

We will start building our decision tree model `df_fit1` (standing for decision tree fit for model 1) with no pruning (i.e., `cp = 0`) and no cross-validation as we have a test dataset already (i.e., `xval = 0`). We will use the Gini index for the splitting.

```
dt_fit1 <- rpart(formula = science_perf ~ .,
                 data = train_dat,
                 method = "class",
                 control = rpart.control(minsplit = 20,
                                         cp = 0,
                                         xval = 0),
                parms = list(split = "gini"))
```

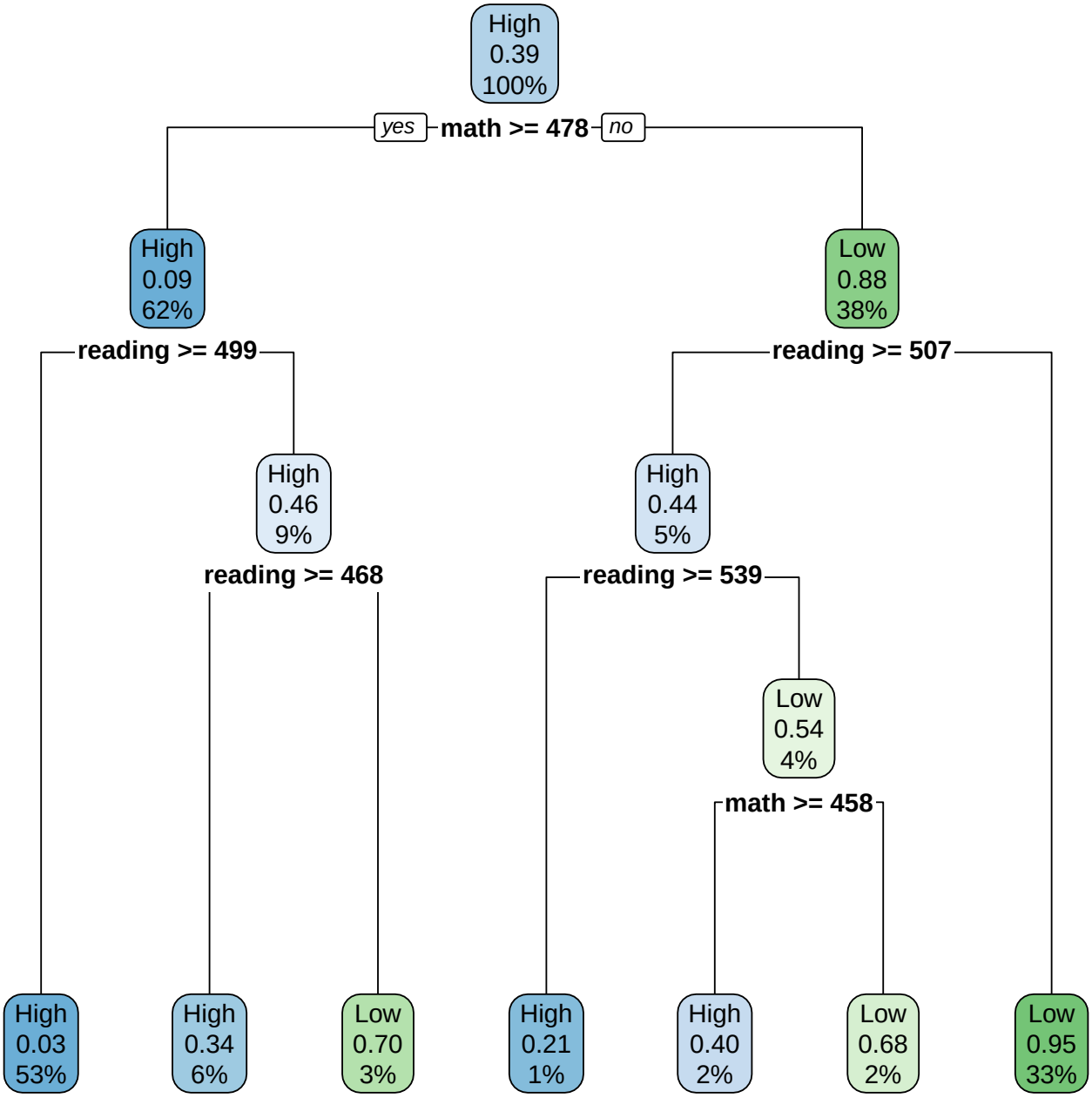
The estimated model is very likely to have too many nodes because we set `cp = 0`. Due to having many nodes, first we will examine the results graphically, before we attempt to print the output. Although the `rpart` package can draw decision tree plots, they are very basic. Therefore, we will use the `rpart.plot` function from the `rpart.plot` package to draw a nicer decision tree plot. Let's see the results graphically using the default settings of the `rpart.plot` function.

```
rpart.plot(dt_fit1)
```



How does the model look like? It is **NOT** very interpretable, isn't it? We definitely need to prune the trees; otherwise the model yields a very complex model with many nodes – which is very likely to overfit the data. In the following model, we use `cp = 0.005`. Remember that as we increase `cp`, the pruning for the model will also increase. The higher the cp value, the shorter the trees with possibly fewer predictors.

```
dt_fit2 <- rpart(formula = science_perf ~ .,
                 data = train_dat,
                 method = "class",
                 control = rpart.control(minsplit = 20,
                                         cp = 0.005,
                                         xval = 0),
                 parms = list(split = "gini"))

rpart.plot(dt_fit2)
```



We could also estimate the same model with the Entropy as the split criterion, `split = "information"`, and the results would be similar (not necessarily the tree itself, but its classification performance).

```
dt_fit2 <- rpart(formula = science_perf ~ .,
                 data = train_dat,
                 method = "class",
                 control = rpart.control(minsplit = 20,
                                         cp = 0.005,
                                         xval = 0),
                parms = list(split = "information"))
```

Now our model is less complex compared compared to the previous model. In the above decision tree plot, each node shows:

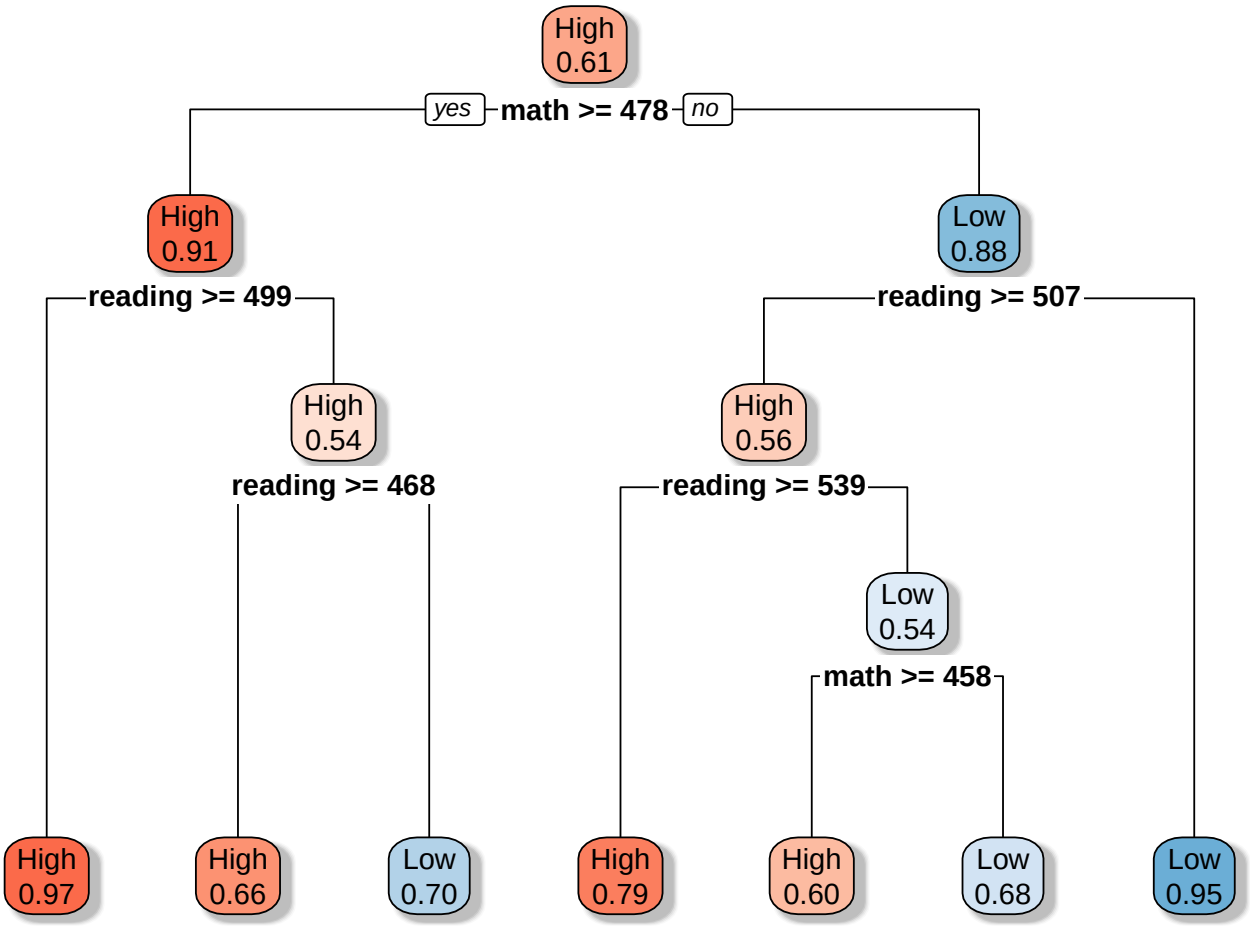
- the predicted class (High or low)

- the predicted probability of the second class (i.e., "Low")
- the percentage of observations in the node

Let's play with the colors to make the trees even more distinct. Also, we will adjust which values should be shown in the nodes, using `extra = 8` (see other possible options HERE). Each node in the new plot shows:

- the predicted class (High or low)
- the predicted probability of the fitted class

```
rpart.plot(dt_fit2, extra = 8, box.palette = "RdBu", shadow.col = "gray")
```



An alternative way to prune the model is to use the `prune()` function from the `rpart` package. In the following example, we will use our initial complex model `dt_fit1` and prune it.

```
dt_fit1_prune <- prune(dt_fit1, cp = 0.005)
rpart.plot(dt_fit1_prune, extra = 8, box.palette = "RdBu", shadow.col = "gray")
```

which would yield the same model that we estimated. Now let's print the output of our model using `printcp()`:

```
printcp(dt_fit2)
```

```
##
## Classification tree:
## rpart(formula = science_perf ~ ., data = train_dat, method = "class",
##     parms = list(split = "gini"), control = rpart.control(minsplit = 20,
##         cp = 0.005, xval = 0))
##
```

```
## Variables actually used in tree construction:
## [1] math    reading
##
## Root node error: 6461/16561 = 0.39013
##
## n= 16561
##
##          CP nsplit rel error
## 1 0.7461693      0   1.00000
## 2 0.0153227      1   0.25383
## 3 0.0147036      3   0.22319
## 4 0.0080483      4   0.20848
## 5 0.0050000      6   0.19239
```

In the output, `CP` refers to the complexity parameter, `nsplit` is the number of splits in the decision tree based on the complexity parameter, and `rel error` is the relative error (i.e., $1 - R^2$) of the solution. This is the error for predictions of the data that were used to estimate the model. The section of `Variables actually used in tree construction` shows which variables have been used in the final model. In our example, only math and reading have been used. What happened to the other variables?

In addition to `printcp()`, we can use `summary()` to print out more detailed results with all splits.

```
summary(dt_fit2)
```

We don't print the entire summary output here. Instead, we want to focus on a specific section in the output:

```
Variable importance
    math  reading ENVAWARE     ESCS    EPIST  HOMEPOS
      46       37        5        4        4        4
```

Similarly, `varImp()` from the `caret` package also gives us a similar output:

```
varImp(dt_fit2)
```

```
##              Overall
## ENVAWARE  582.079065
## EPIST     791.498637
## ESCS      427.818610
## HEDRES      5.287639
## HOMEPOS    17.914110
## math     5529.901785
## reading  5752.549285
## WEALTH      7.572725
## ICTRES      0.000000
```

Both of these show the importance of the variables for our estimated decision tree model. The larger the values are, the more crucial they are for the model. In our example, math and reading seem to be highly important for the decision tree model, whereas ICTRES is the least important variable. The variables that were not very important for the model are those that were not included in the final model. These variables are possibly have very low correlations with our outcome variable, `science_perf`.

We can use `rpart.rules` to print out the decision rules from the trees. By default, the output from this function shows the probability of the **second** class for each decision/split being made (i.e., the category "low" in our example) and what percent of the observations fall into this category.

```
rpart.rules(dt_fit2, cover = TRUE)
```

```
##  science_perf                                                  cover
##          0.03 when math >=        478 & reading >=        499     53%
```

```
##          0.21 when math <  478        & reading >=        539      1%
##          0.34 when math >=        478 & reading is 468 to 499      6%
##          0.40 when math is 458 to 478 & reading is 507 to 539      2%
##          0.68 when math <  458        & reading is 507 to 539      2%
##          0.70 when math >=        478 & reading <  468            3%
##          0.95 when math <  478        & reading <  507           33%
```

Furthermore, we need to check the classification accuracy of the estimated decision tree with the **test** data. Otherwise, it is hard to justify whether or not the estimated decision tree would work accurately for prediction. Below we estimate the predicted classes (either high or low) from the test data by applying the estimated model.First we obtain model predictions using `predict()` and then turn the results into a data frame called `dt_pred`.

```
dt_pred <- predict(dt_fit2, test_dat) %>%
  as.data.frame()

head(dt_pred)
```

```
##         High        Low
## 1 0.97465045 0.02534955
## 2 0.05406386 0.94593614
## 3 0.05406386 0.94593614
## 4 0.66243386 0.33756614
## 5 0.97465045 0.02534955
## 6 0.05406386 0.94593614
```

This dataset shows each observation's (i.e., students from the test data) probability of falling into either *high* or *low* categories based on the decision rules that we estimated. We will turn these probabilities into binary classifications, depending on whether or not they are >= 50%. Then, we will compare these estimates with the actual classes in the test data (i.e., `test_dat$science_perf`) in order to create a confusion matrix.

```
dt_pred <- mutate(dt_pred,
  science_perf = as.factor(ifelse(High >= 0.5, "High", "Low"))
  ) %>%
  select(science_perf)

confusionMatrix(dt_pred$science_perf, test_dat$science_perf)
```

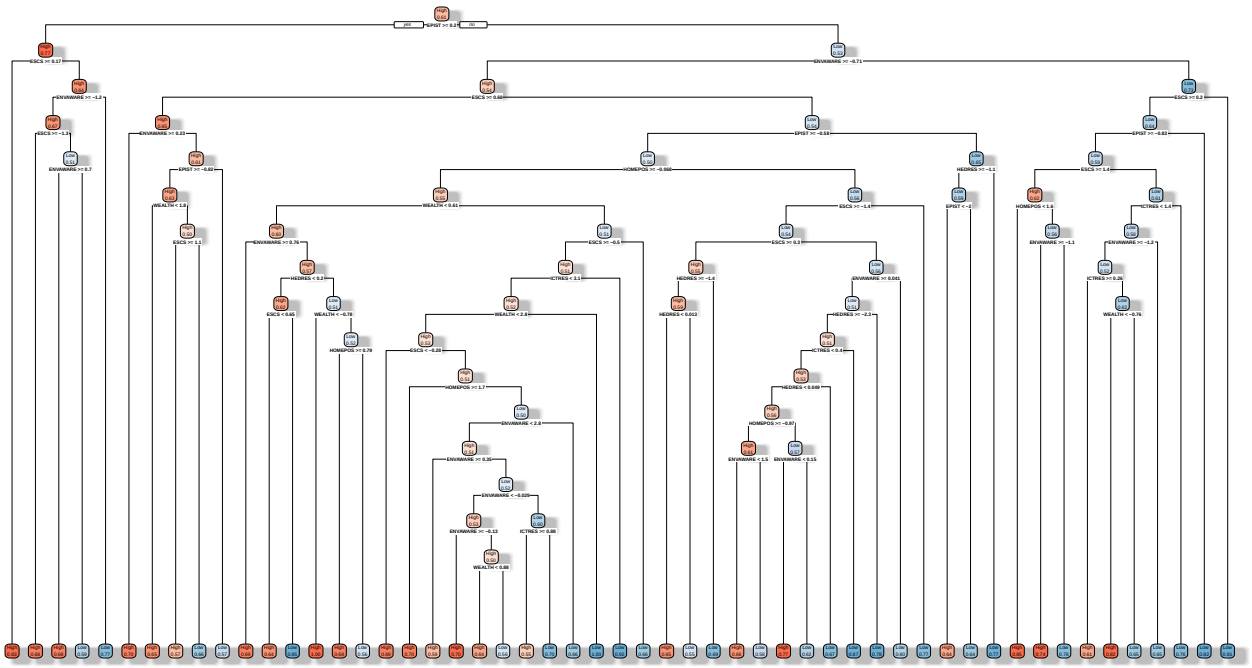
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction High  Low
##       High 4076  316
##       Low   252 2453
##
##                Accuracy : 0.92
##                  95% CI : (0.9134, 0.9262)
##     No Information Rate : 0.6098
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8311
##  Mcnemar's Test P-Value : 0.008207
##
##             Sensitivity : 0.9418
##             Specificity : 0.8859
##          Pos Pred Value : 0.9281
```

```
##          Neg Pred Value : 0.9068
##              Prevalence : 0.6098
##          Detection Rate : 0.5743
##    Detection Prevalence : 0.6189
##       Balanced Accuracy : 0.9138
##
##        'Positive' Class : High
##
```

The output shows that the overall accuracy is around 92%, sensitivit is 94%, and specificity is 89%. For only two variables, this is very good. However, sometimes we do not have predictors that are highly correlated with the outcome variables. In such cases, the model tuning might take much longer.

Let's assume that we did **NOT** have reading and math in our dataset. We still want to predict `science_perf` using the remaining variables.

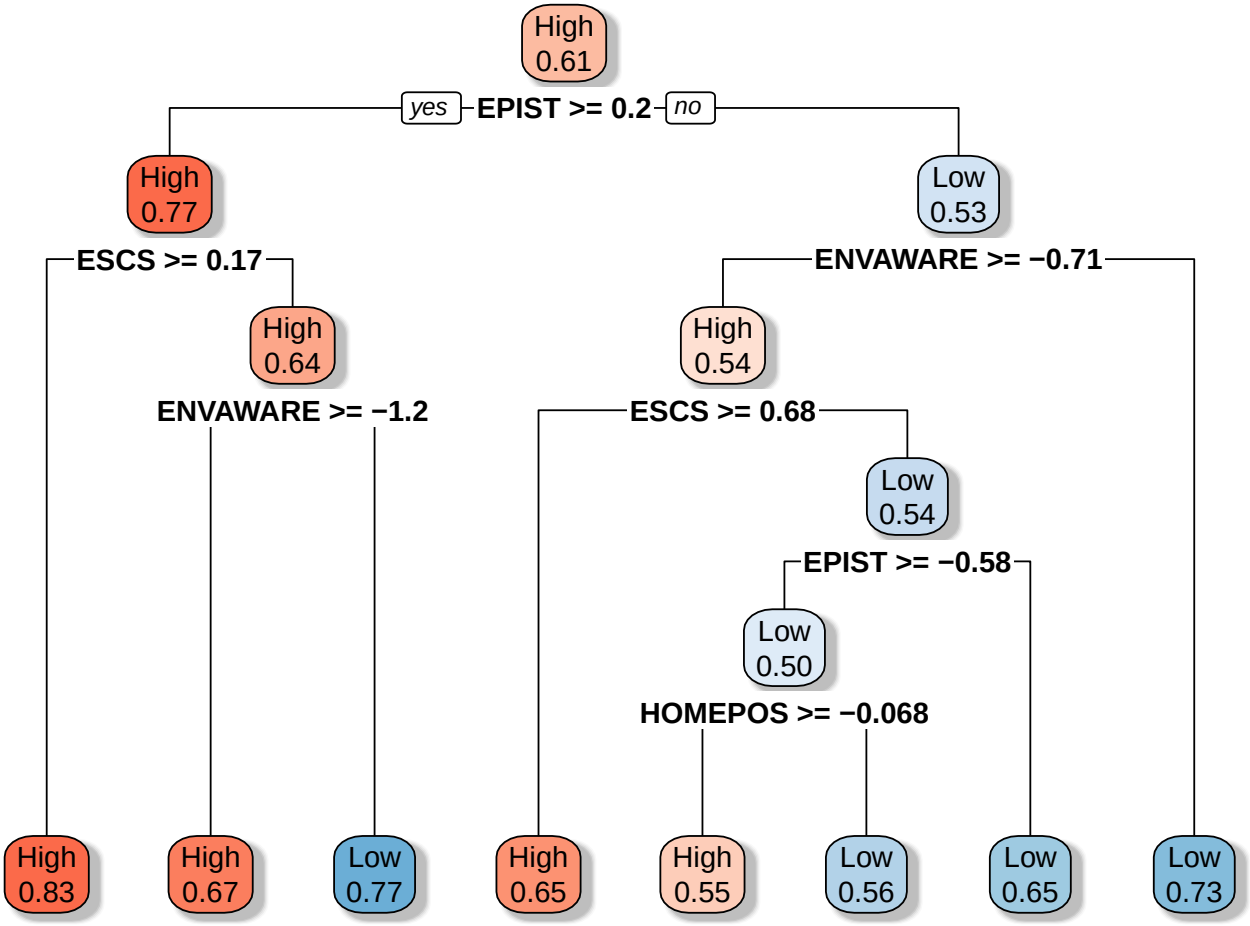
```
dt_fit3a <- rpart(formula = science_perf ~ WEALTH + HEDRES + ENVAWARE + ICTRES + EPIST +
                    HOMEPOS +ESCS,
                  data = train_dat,
                  method = "class",
                  control = rpart.control(minsplit = 20,
                                          cp = 0.001,
                                          xval = 0),
                 parms = list(split = "gini"))

rpart.plot(dt_fit3a, extra = 8, box.palette = "RdBu", shadow.col = "gray")
```



Now, let's change cp to 0.005.

```
dt_fit3b <- rpart(formula = science_perf ~ WEALTH + HEDRES + ENVAWARE + ICTRES + EPIST +
                    HOMEPOS + ESCS,
                  data = train_dat,
                  method = "class",
                  control = rpart.control(minsplit = 20,
                                          cp = 0.005,
```

```
                                        xval = 0),
               parms = list(split = "gini"))

rpart.plot(dt_fit3b, extra = 8, box.palette = "RdBu", shadow.col = "gray")
```



Since we also care about the accuracy, sensitivity, and specificity of these models, we can turn this experiment into a small function.

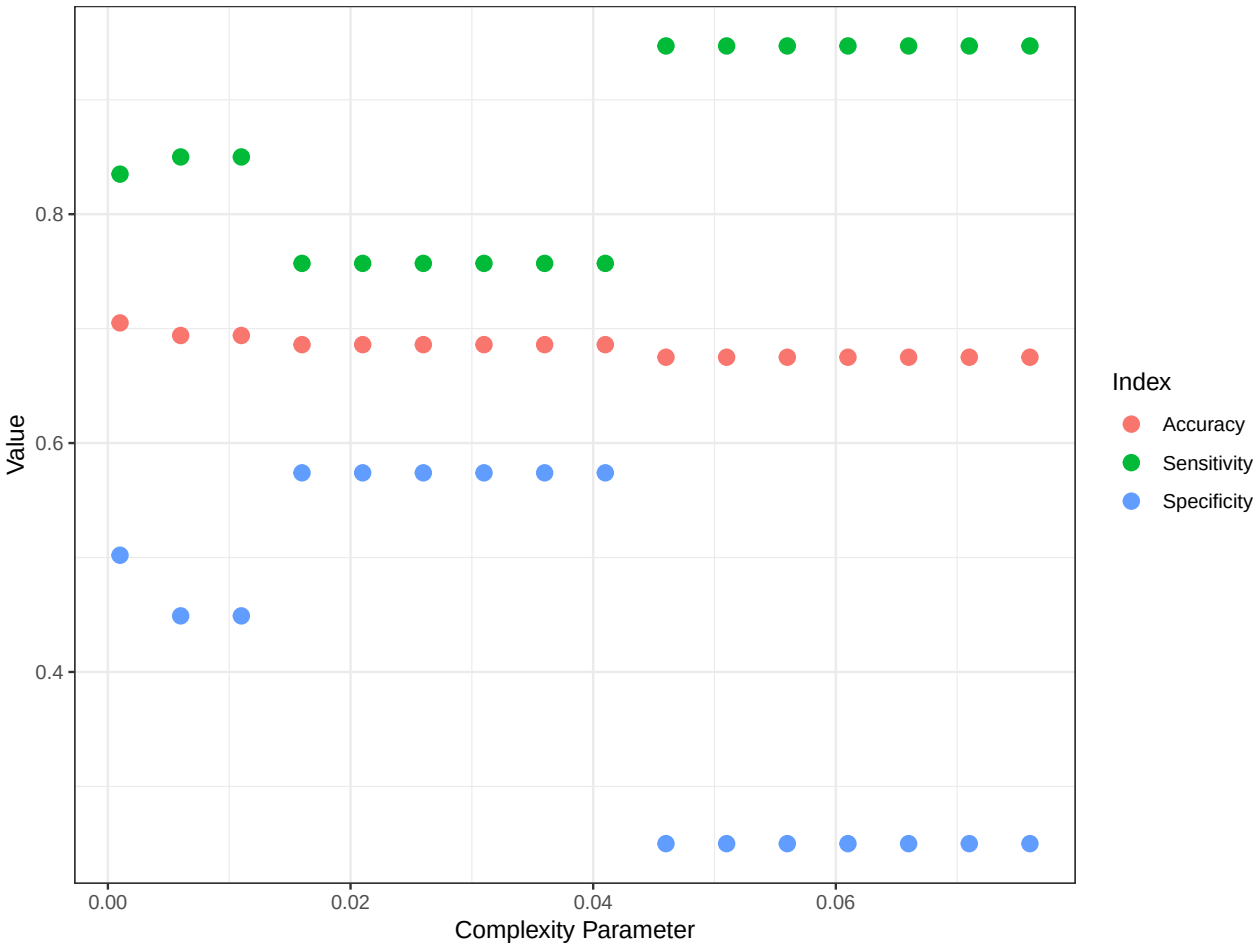
```
decision_check <- function(cp) {
  require("rpart")
  require("dplyr")

  dt <- rpart(formula = science_perf ~ WEALTH + HEDRES + ENVAWARE + ICTRES + EPIST +
                HOMEPOS + ESCS,
              data = train_dat,
              method = "class",
              control = rpart.control(minsplit = 20,
                                      cp = cp,
                                      xval = 0),
              parms = list(split = "gini"))

  dt_pred <- predict(dt, test_dat) %>%
    as.data.frame() %>%
    mutate(science_perf = as.factor(ifelse(High >= 0.5, "High", "Low"))) %>%
```

```
    select(science_perf)

  cm <- confusionMatrix(dt_pred$science_perf, test_dat$science_perf)

  results <- data.frame(cp = cp,
                        Accuracy = round(cm$overall[1], 3),
                        Sensitivity = round(cm$byClass[1], 3),
                        Specificity = round(cm$byClass[2], 3))

  return(results)
}

result <- NULL
for(i in seq(from=0.001, to=0.08, by = 0.005)) {
  result <- rbind(result, decision_check(cp = i))
}

result <- result[order(result$Accuracy, result$Sensitivity, result$Specificity),]
result
```

```
##               cp Accuracy Sensitivity Specificity
## Accuracy9  0.046    0.675       0.947       0.250
## Accuracy10 0.051    0.675       0.947       0.250
## Accuracy11 0.056    0.675       0.947       0.250
## Accuracy12 0.061    0.675       0.947       0.250
## Accuracy13 0.066    0.675       0.947       0.250
## Accuracy14 0.071    0.675       0.947       0.250
## Accuracy15 0.076    0.675       0.947       0.250
## Accuracy3  0.016    0.686       0.757       0.574
## Accuracy4  0.021    0.686       0.757       0.574
## Accuracy5  0.026    0.686       0.757       0.574
## Accuracy6  0.031    0.686       0.757       0.574
## Accuracy7  0.036    0.686       0.757       0.574
## Accuracy8  0.041    0.686       0.757       0.574
## Accuracy1  0.006    0.694       0.850       0.449
## Accuracy2  0.011    0.694       0.850       0.449
## Accuracy   0.001    0.705       0.835       0.502
```

We can also visulize the results using `ggplot2`. First, we wil transform the `result` dataset into a long format and then use this new dataset (called `result_long`) in `ggplot()`.

```
result_long <- melt(as.data.table(result),
                    id.vars = c("cp"),
                    measure = c("Accuracy", "Sensitivity", "Specificity"),
                    variable.name = "Index",
                    value.name = "Value")

ggplot(data = result_long,
       mapping = aes(x = cp, y = Value)) +
  geom_point(aes(color = Index), size = 3) +
  labs(x = "Complexity Parameter", y = "Value") +
  theme_bw()
```



In the plot, we see that there is a trade-off between sensitivity and specificity. Depending on the situation, we may prefer higher sensitivity (e.g., correctly identifying those who have "high" science scores) or higher specificity (e.g., correctly identifying those who have "low" science scores). For example, if we want to know who is performing poorly in science (so that we can design additional instructional materials), we may want the model to identify "low" performers more accurately.

7.2.1 Cross-validation

As you may remember, we set `xval = 0` in our decision tree models because we did not want to run any cross-validation samples. However, cross-validations (e.g., K-fold approach) are highly useful when we do not have a test or validation dataset, or our dataset is to small to split into training and test data. A typical way to use cross-validation in decision trees is to not specify a cp (i.e., complexity parameter) and perform cross validation. In the following example, we will assume that our dataset is not too big and thus we want to run 10 cross-validation samples (i.e., splits) as we build our decision tree model. Note that we use `cp = 0` this time.

```
dt_fit4 <- rpart(formula = science_perf ~ WEALTH + HEDRES + ENVAWARE + ICTRES +
                   EPIST + HOMEPOS + ESCS,
                 data = train_dat,
                 method = "class",
                 control = rpart.control(minsplit = 20,
                                         cp = 0,
                                         xval = 10),
```

```
                parms = list(split = "gini"))
```

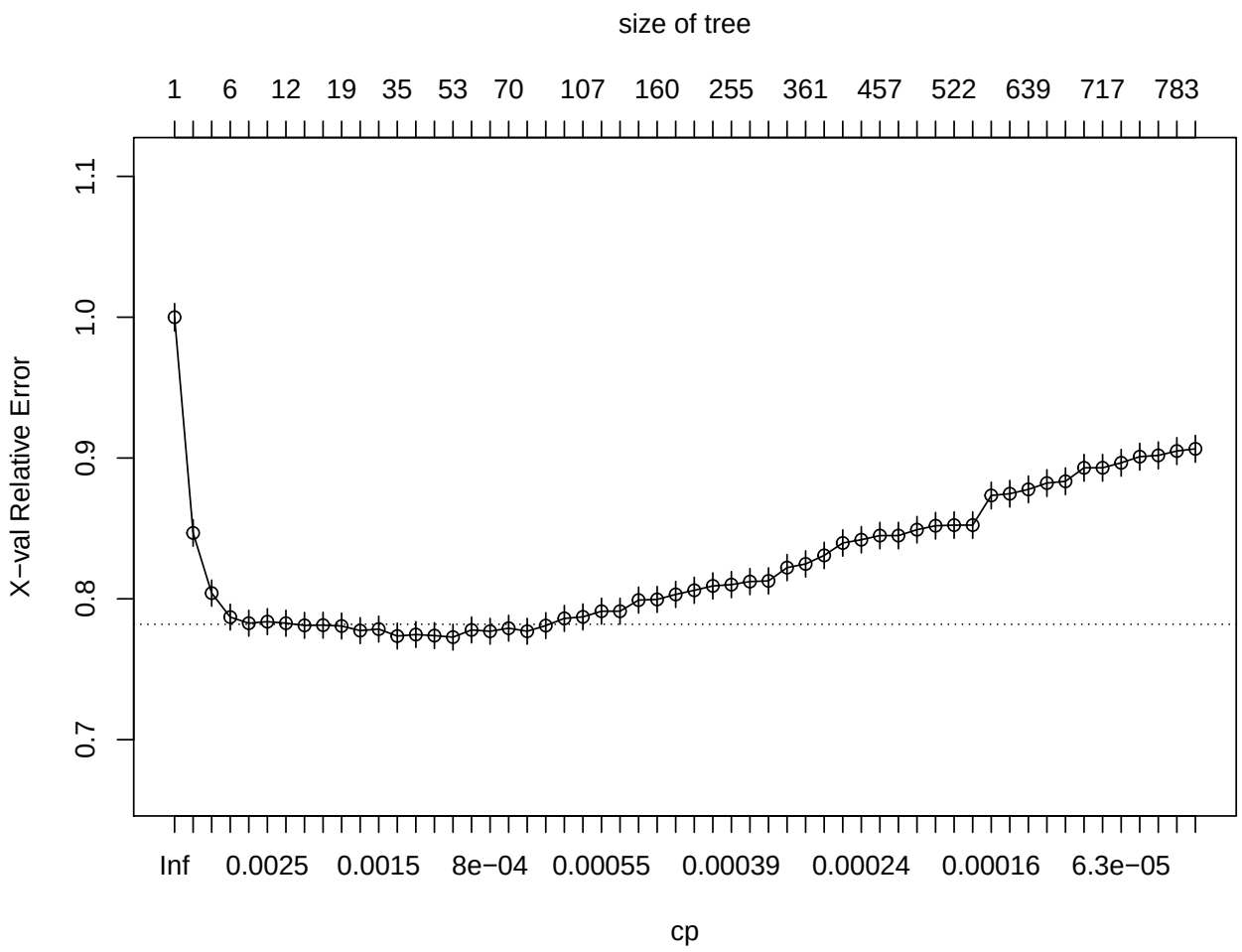
In the results, we can evaluate the cross-validated error (i.e., X-val Relative Error) and choose the complexity parameter that would give us an acceptable value. Then, we can use this cp value and prune the trees. We use `plotcp()` function to visualize the cross-validation results.

```
printcp(dt_fit4)
```

```
##
## Classification tree:
## rpart(formula = science_perf ~ WEALTH + HEDRES + ENVAWARE + ICTRES +
##     EPIST + HOMEPOS + ESCS, data = train_dat, method = "class",
##     parms = list(split = "gini"), control = rpart.control(minsplit = 20,
##         cp = 0, xval = 10))
##
## Variables actually used in tree construction:
## [1] ENVAWARE EPIST    ESCS     HEDRES   HOMEPOS  ICTRES   WEALTH
##
## Root node error: 6461/16561 = 0.39013
##
## n= 16561
##
##            CP nsplit rel error  xerror      xstd
## 1  7.7233e-02      0   1.00000 1.00000 0.0097156
## 2  4.3646e-02      2   0.84553 0.84677 0.0093682
## 3  1.1918e-02      3   0.80189 0.80406 0.0092417
## 4  5.6493e-03      5   0.77805 0.78703 0.0091875
## 5  2.6312e-03      7   0.76675 0.78270 0.0091733
## 6  2.3216e-03      9   0.76149 0.78378 0.0091769
## 7  1.9347e-03     11   0.75685 0.78270 0.0091733
## 8  1.8573e-03     13   0.75298 0.78115 0.0091683
## 9  1.7025e-03     16   0.74741 0.78130 0.0091688
## 10 1.5477e-03     18   0.74400 0.78068 0.0091667
## 11 1.5220e-03     24   0.73425 0.77743 0.0091560
## 12 1.3930e-03     30   0.72512 0.77852 0.0091596
## 13 1.2382e-03     34   0.71955 0.77356 0.0091430
## 14 1.0834e-03     35   0.71831 0.77465 0.0091467
## 15 1.0525e-03     47   0.70392 0.77387 0.0091441
## 16 8.5126e-04     52   0.69865 0.77279 0.0091404
## 17 8.2547e-04     54   0.69695 0.77790 0.0091575
## 18 7.7387e-04     57   0.69447 0.77697 0.0091544
## 19 7.2228e-04     69   0.68488 0.77914 0.0091616
## 20 6.7069e-04     72   0.68271 0.77697 0.0091544
## 21 6.1910e-04     84   0.67466 0.78099 0.0091677
## 22 5.8041e-04    101   0.66398 0.78610 0.0091845
## 23 5.6751e-04    106   0.66104 0.78718 0.0091880
## 24 5.4171e-04    123   0.65036 0.79121 0.0092010
## 25 5.1592e-04    148   0.63612 0.79121 0.0092010
## 26 4.9012e-04    153   0.63287 0.79910 0.0092262
## 27 4.6432e-04    159   0.62993 0.79957 0.0092277
## 28 4.3337e-04    198   0.61059 0.80313 0.0092388
## 29 4.1273e-04    211   0.60331 0.80607 0.0092480
## 30 4.0241e-04    231   0.59403 0.80916 0.0092576
## 31 3.8694e-04    254   0.58180 0.81009 0.0092604
```

```
## 32 3.6114e-04    275   0.57282 0.81226 0.0092671
## 33 3.3166e-04    298   0.56369 0.81272 0.0092685
## 34 3.0955e-04    310   0.55905 0.82216 0.0092970
## 35 2.7086e-04    360   0.54341 0.82479 0.0093048
## 36 2.5796e-04    380   0.53707 0.83083 0.0093226
## 37 2.4764e-04    399   0.53134 0.83965 0.0093481
## 38 2.3216e-04    411   0.52825 0.84197 0.0093547
## 39 2.2111e-04    456   0.51602 0.84492 0.0093630
## 40 2.1668e-04    467   0.51323 0.84492 0.0093630
## 41 2.0637e-04    495   0.50317 0.84909 0.0093747
## 42 1.9347e-04    507   0.50070 0.85188 0.0093824
## 43 1.8573e-04    521   0.49760 0.85234 0.0093837
## 44 1.7197e-04    529   0.49574 0.85234 0.0093837
## 45 1.5477e-04    538   0.49420 0.87339 0.0094403
## 46 1.2898e-04    632   0.47810 0.87463 0.0094435
## 47 1.1608e-04    638   0.47733 0.87773 0.0094515
## 48 1.0318e-04    646   0.47640 0.88222 0.0094630
## 49 9.2865e-05    667   0.47423 0.88345 0.0094662
## 50 7.7387e-05    672   0.47377 0.89305 0.0094902
## 51 6.8789e-05    716   0.47036 0.89305 0.0094902
## 52 5.8041e-05    725   0.46974 0.89661 0.0094990
## 53 5.1592e-05    740   0.46881 0.90094 0.0095095
## 54 3.8694e-05    770   0.46727 0.90187 0.0095117
## 55 2.2111e-05    782   0.46680 0.90497 0.0095192
## 56 0.0000e+00    796   0.46618 0.90652 0.0095229
```

```
plotcp(dt_fit4)
```



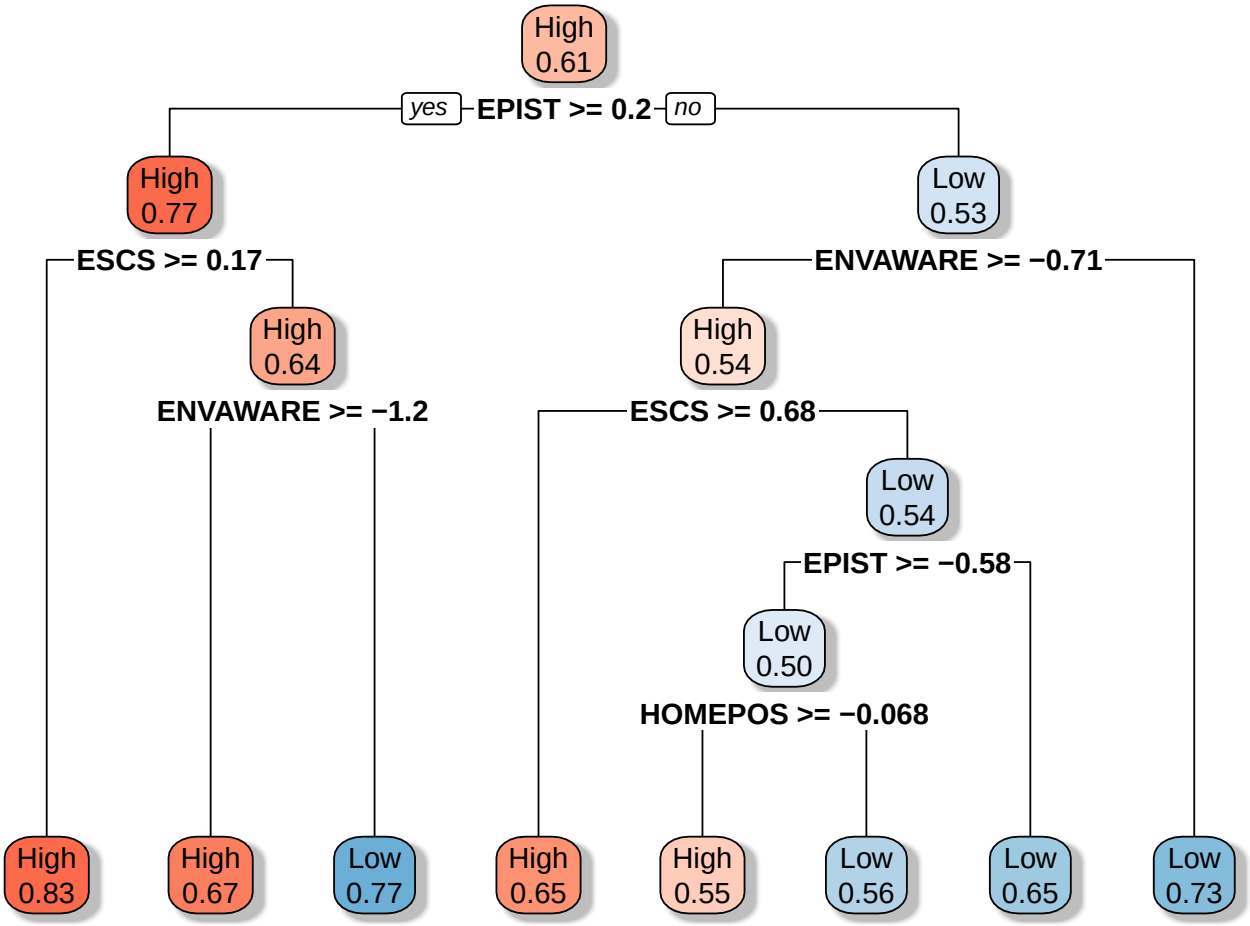
Next, we can modify our model as follows:

```
dt_fit5 <- rpart(formula = science_perf ~ WEALTH + HEDRES + ENVAWARE + ICTRES +
                   EPIST + HOMEPOS + ESCS,
                 data = train_dat,
                 method = "class",
                 control = rpart.control(minsplit = 20,
                                         cp = 0.0039,
                                         xval = 0),
                parms = list(split = "gini"))

printcp(dt_fit5)
```

```
##
## Classification tree:
## rpart(formula = science_perf ~ WEALTH + HEDRES + ENVAWARE + ICTRES +
##     EPIST + HOMEPOS + ESCS, data = train_dat, method = "class",
##     parms = list(split = "gini"), control = rpart.control(minsplit = 20,
##         cp = 0.0039, xval = 0))
##
## Variables actually used in tree construction:
## [1] ENVAWARE EPIST    ESCS     HOMEPOS
##
## Root node error: 6461/16561 = 0.39013
```

```
##
## n= 16561
##
##          CP nsplit rel error
## 1 0.0772326      0   1.00000
## 2 0.0436465      2   0.84553
## 3 0.0119177      3   0.80189
## 4 0.0056493      5   0.77805
## 5 0.0039000      7   0.76675
```

```
rpart.plot(dt_fit5, extra = 8, box.palette = "RdBu", shadow.col = "gray")
```



Lastly, for the sake of brevity, we demonstrate a short regression tree example below where we predict math scores (a continuous variable) using the same set of variables. This time we use `method = "anova"` in the `rpart()` function to estimate a regression tree.

Let's begin with cross-validation and check how R^2 changes depending on the number of splits.

```
rt_fit1 <- rpart(formula = math ~ WEALTH + HEDRES + ENVAWARE +
                   ICTRES + EPIST + HOMEPOS + ESCS,
                 data = train_dat,
                 method = "anova",
                 control = rpart.control(minsplit = 20,
                                         cp = 0.001,
                                         xval = 10),
                 parms = list(split = "gini"))
```

```
printcp(rt_fit1)
```

```
##
## Regression tree:
## rpart(formula = math ~ WEALTH + HEDRES + ENVAWARE + ICTRES +
##     EPIST + HOMEPOS + ESCS, data = train_dat, method = "anova",
##     parms = list(split = "gini"), control = rpart.control(minsplit = 20,
##         cp = 0.001, xval = 10))
##
## Variables actually used in tree construction:
## [1] ENVAWARE EPIST    ESCS     HEDRES   WEALTH
##
## Root node error: 104625310/16561 = 6317.6
##
## n= 16561
##
##           CP nsplit rel error  xerror      xstd
## 1  0.1121549      0   1.00000 1.00014 0.0100818
## 2  0.0382061      1   0.88785 0.88819 0.0091919
## 3  0.0353050      2   0.84964 0.85964 0.0089652
## 4  0.0166934      3   0.81433 0.81758 0.0085677
## 5  0.0078301      4   0.79764 0.80105 0.0084293
## 6  0.0070306      5   0.78981 0.79455 0.0083594
## 7  0.0063780      6   0.78278 0.78877 0.0083343
## 8  0.0040553      7   0.77640 0.78437 0.0083043
## 9  0.0033408      8   0.77235 0.78091 0.0082947
## 10 0.0030034      9   0.76901 0.77942 0.0082741
## 11 0.0028744     10   0.76600 0.77759 0.0082582
## 12 0.0024670     11   0.76313 0.77469 0.0082327
## 13 0.0021466     12   0.76066 0.77141 0.0082005
## 14 0.0021460     13   0.75851 0.77185 0.0082082
## 15 0.0019919     14   0.75637 0.77166 0.0082192
## 16 0.0018103     15   0.75438 0.76997 0.0082037
## 17 0.0017435     17   0.75076 0.76856 0.0081993
## 18 0.0017275     18   0.74901 0.76836 0.0081954
## 19 0.0016846     19   0.74728 0.76787 0.0081916
## 20 0.0016463     20   0.74560 0.76770 0.0081872
## 21 0.0015453     21   0.74395 0.76759 0.0081813
## 22 0.0013210     22   0.74241 0.76538 0.0081718
## 23 0.0012386     23   0.74109 0.76290 0.0081676
## 24 0.0012244     25   0.73861 0.76170 0.0081598
## 25 0.0011261     27   0.73616 0.75927 0.0081399
## 26 0.0011075     28   0.73504 0.75895 0.0081377
## 27 0.0011003     29   0.73393 0.75948 0.0081432
## 28 0.0010564     30   0.73283 0.75924 0.0081413
## 29 0.0010195     31   0.73177 0.75704 0.0081125
## 30 0.0010000     32   0.73075 0.75675 0.0081169
```

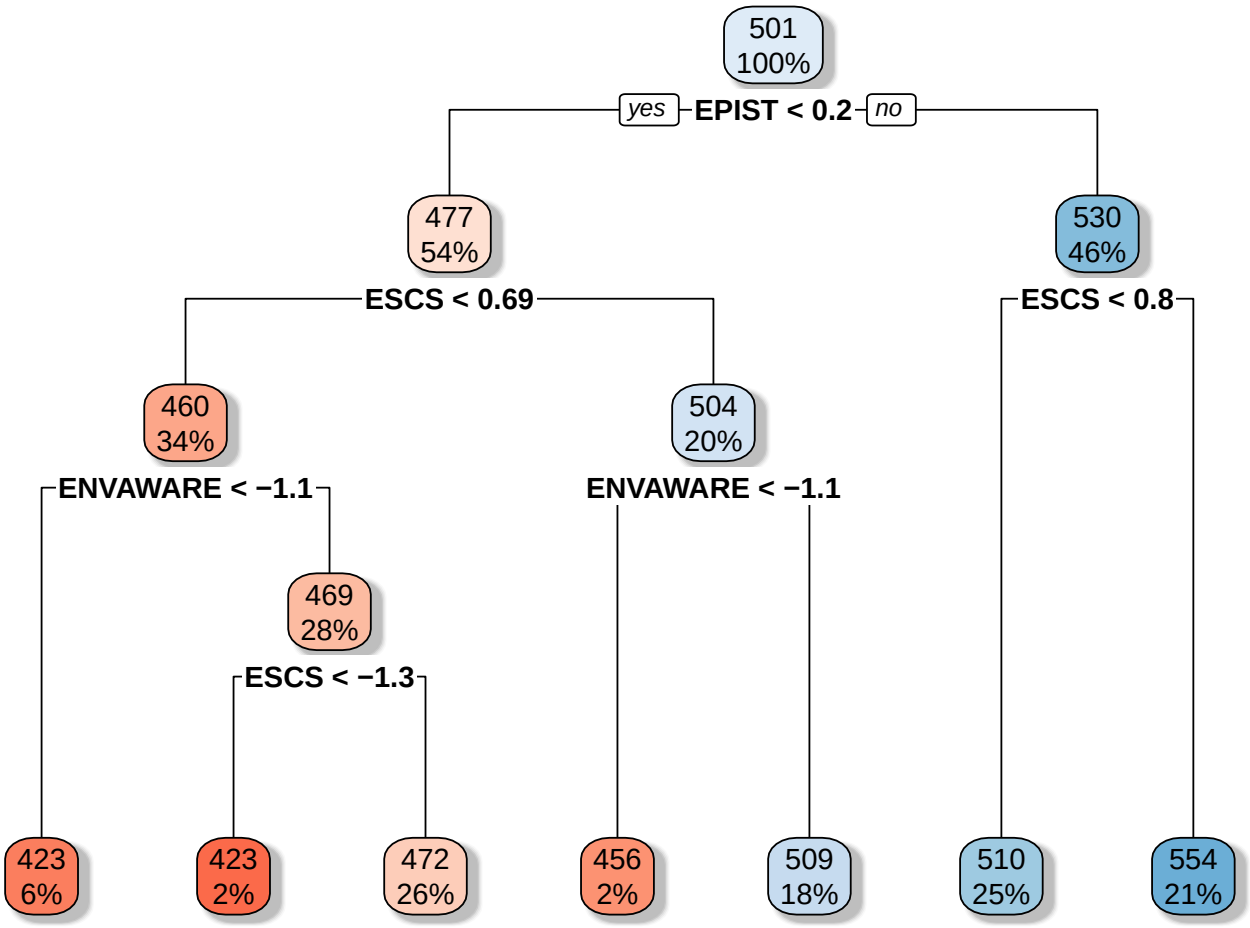
Then, we can adjust our model based on the suggestions from the previous plot. Note that we use `extra = 100` in the `rpart.plot()` function to show percentages (*Note*: `rpart.plot` has different *extra* options depending on whether it is a classification or regression tree).

```
rt_fit2 <- rpart(formula = math ~ WEALTH + HEDRES + ENVAWARE +
                   ICTRES + EPIST + HOMEPOS + ESCS,
```

```
                 data = train_dat,
                 method = "anova",
                 control = rpart.control(minsplit = 20,
                                         cp = 0.007,
                                         xval = 0),
                parms = list(split = "gini"))

printcp(rt_fit2)
```

```
##
## Regression tree:
## rpart(formula = math ~ WEALTH + HEDRES + ENVAWARE + ICTRES +
##     EPIST + HOMEPOS + ESCS, data = train_dat, method = "anova",
##     parms = list(split = "gini"), control = rpart.control(minsplit = 20,
##         cp = 0.007, xval = 0))
##
## Variables actually used in tree construction:
## [1] ENVAWARE EPIST    ESCS
##
## Root node error: 104625310/16561 = 6317.6
##
## n= 16561
##
##          CP nsplit rel error
## 1 0.1121549      0   1.00000
## 2 0.0382061      1   0.88785
## 3 0.0353050      2   0.84964
## 4 0.0166934      3   0.81433
## 5 0.0078301      4   0.79764
## 6 0.0070306      5   0.78981
## 7 0.0070000      6   0.78278
```

```
rpart.plot(rt_fit2, extra = 100, box.palette = "RdBu", shadow.col = "gray")
```



To evaluate the model accuracy, we cannot use the classification-based indices anymore because we built a regression tree, not a classification tree. Two useful measures that we can for evaluating regression trees are the mean absolute error (mae) and the root mean square error (rmse). The `modelr` package has several functions – such as `mae()` and `rmse()` – to evaluate regression-based models. Using the training and (more importantly) test data, we can evaluate the accuracy of the decision tree model that we estimated above.

```
# Training data
mae(model = rt_fit2, data = train_dat)
```

```
## [1] 56.48637
```

```
rmse(model = rt_fit2, data = train_dat)
```

```
## [1] 70.3226
```

```
# Test data
mae(model = rt_fit2, data = test_dat)
```

```
## [1] 56.65823
```

```
rmse(model = rt_fit2, data = test_dat)
```

```
## [1] 70.41532
```

We seem to have slightly less error with the training data than the test data. Is this finding suprising to you?

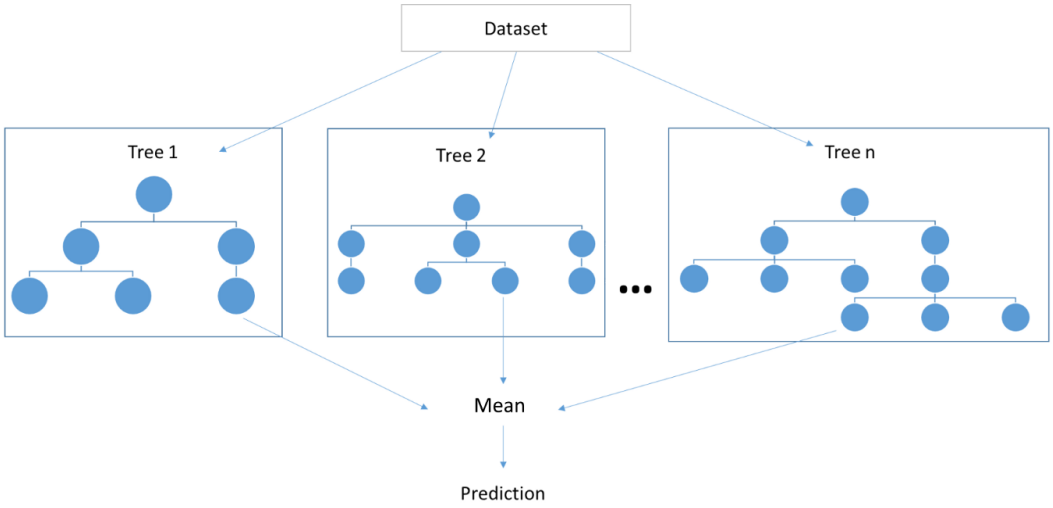


Figure 7.2: An example of random forests approach

7.3 Random Forests

Decision trees can sometimes be non-robust because a small change in the data may cause a significant change in the final estimated tree. Therefore, whenever a decision tree approach is not completely stable, an alternative method – such as **random forests** – can be more suitable for supervised ML applications. Unlike the decision tree approach where there is a single solution from the same sample, random forest builds multiple decision trees by splitting the data into multiple sub-samples and merges them together to get a more accurate and stable prediction.

The underlying mechanism of random forests is very similar to that of decision trees. However, random forests first build lots of bushy trees and then average them to reduce the overall variance. Figure 7.2 shows how a random forest would look like with three trees.

Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature (i.e., predictor) while splitting a node, it searches for the best feature among a random subset of features. That is, only a random subset of the features is taken into consideration by the algorithm for splitting a node. This results in a wide diversity that generally results in a better model. For example, if there is a strong predictor among a set of predictors, a decision tree would typically rely on this particular predictor to make predictions and build trees. However, random forests force each split to consider only a set of the predictors – which would result in trees that utilize not only the strong predictor but also other predictors that are moderately correlated with the outcome variable.

Random forest has nearly the same tuning parameters as a decision tree. Also, like decision trees, random forests can be used for both classification and regression problems. However, there are some differences between the two approaches. Unlike in decision trees, it is easier to control and prevent overfitting in random forests. This is because random forests create random subsets of the features and build much smaller trees using these subsets. Afterwards, it combines the subtrees. It should be noted that this procedure makes random forests computationally slower, depending on how many trees random forest builds. Therefore, it may not be effective for *real-time* predictions.

The random forest algorithm is used in a lot of different fields, like banking, stock market, medicine, and e-commerce. For example, random forests can be used to detect customers who will use the bank's services more frequently than others and repay their debt in time. It can also used to detect fraud customers who want to scam the bank. In educational testing, we can use random forests to analyze a student's assessment history (e.g., test scores, response times, demographic variables, grade level, and so on) to identify whether

the student has any learning difficulties. Similarly, we can use examinee-related variables, test scores, and test administration date to identify whether an examinee is likely to re-take the test (e.g., TOEFL or GRE) in the future.

7.4 Random forests in R

In R, `randomForest` and `caret` packages can be used to apply the random forest algorithm to classification and regression problems. The use of the `randomForest()` function is similar to that of `rpart()`. The main elements that we need to define are:

- **formula**: A regression-like formula defining the dependent variable and the predictors – it is the same as the one for `rpart()`.
- **data**: The dataset that we use to train the model.
- **importance**: If TRUE, then importance of the predictors is assessed in the model.
- **ntree**: Number of trees to grow in the model; we often start with a large number and then reduce it as we adjust the model based on the results. A large number for **ntree** can significantly increase the estimation time for the model.

There are also other elements that we can change depending on whether it is a classification or regression model (see `?randomForest` for more details). In the following example, we will focus on the same classification problem that we used before for decision trees. We initially set `ntree = 1000` to get 1000 trees in total but we will evaluate whether we need all of these trees to have an accurate model.

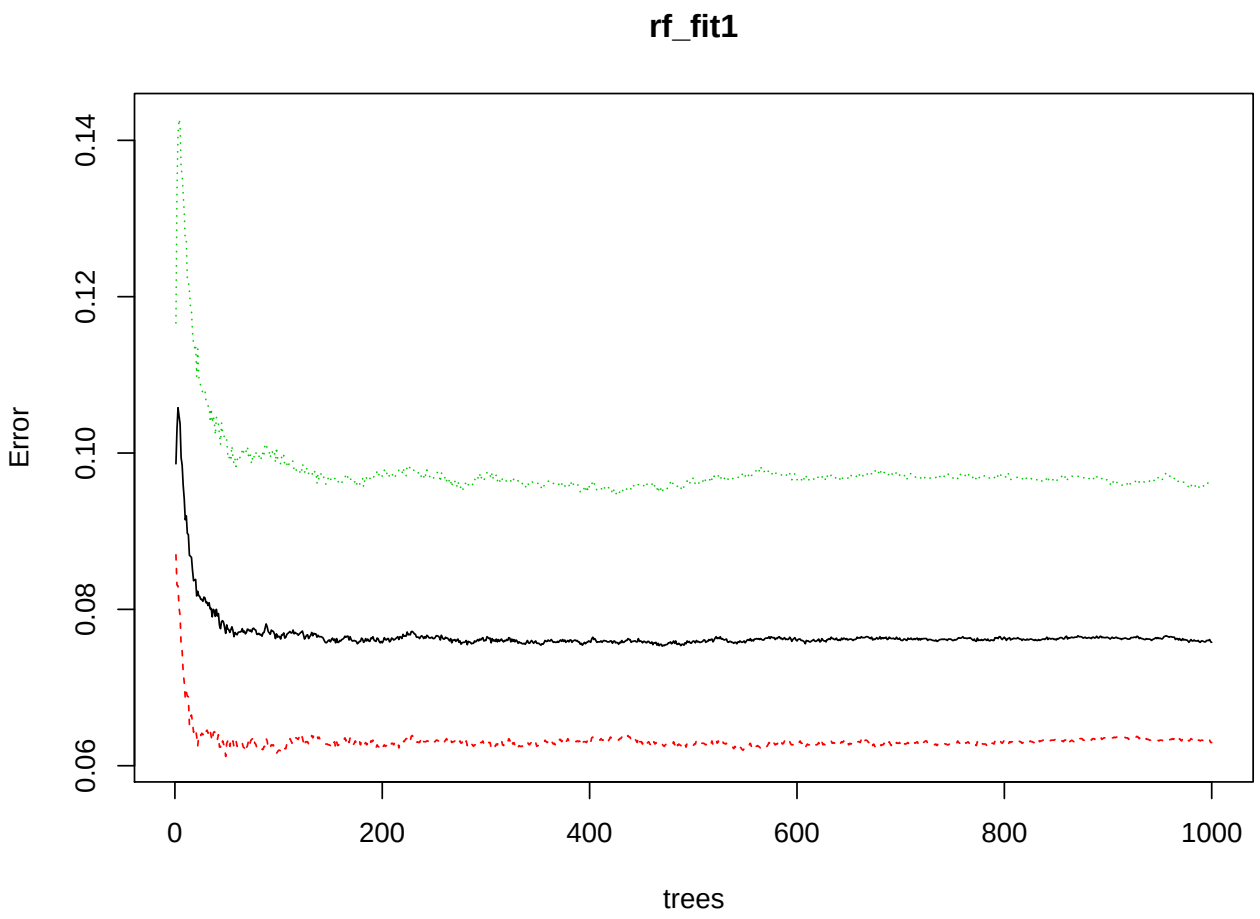
```
library("randomForest")
library("caret")

rf_fit1 <- randomForest(formula = science_perf ~ .,
                        data = train_dat,
                        importance = TRUE, ntree = 1000)

print(rf_fit1)
```

```
##
## Call:
##  randomForest(formula = science_perf ~ ., data = train_dat, importance = TRUE,      ntree = 1000)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 7.58%
## Confusion matrix:
##      High  Low class.error
## High 9464  636   0.0629703
## Low   619 5842   0.0958056
```

In the output, we see the confusion matrix along with classification error and out-of-bag (OOB) error. OBB is a method of measuring the prediction error of random forests, finding the mean prediction error on each training sample, using only the trees that did not have in their bootstrap sample. The results show that the overall OBB error is around 7.6%, while the classification error is 6% for the *high* category and around 10% for the *low* category.

Next, by checking the level error across the number of trees, we can determine the ideal number of trees for our model.

```
plot(rf_fit1)
```



The plot shows that the error level does not go down any further after roughly 50 trees. So, we can run our model again by using `ntree = 50` this time.

```
rf_fit2 <- randomForest(formula = science_perf ~ .,
                        data = train_dat,
                        importance = TRUE, ntree = 50)

print(rf_fit2)
```

```
##
## Call:
##  randomForest(formula = science_perf ~ ., data = train_dat, importance = TRUE,      ntree = 50)
##                Type of random forest: classification
##                      Number of trees: 50
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 7.95%
## Confusion matrix:
##      High  Low class.error
## High 9459  641  0.06346535
## Low   675 5786  0.10447299
```

We can see the overall accuracy of model (92.12%) as follows:

```
sum(diag(rf_fit2$confusion)) / nrow(train_dat)
```

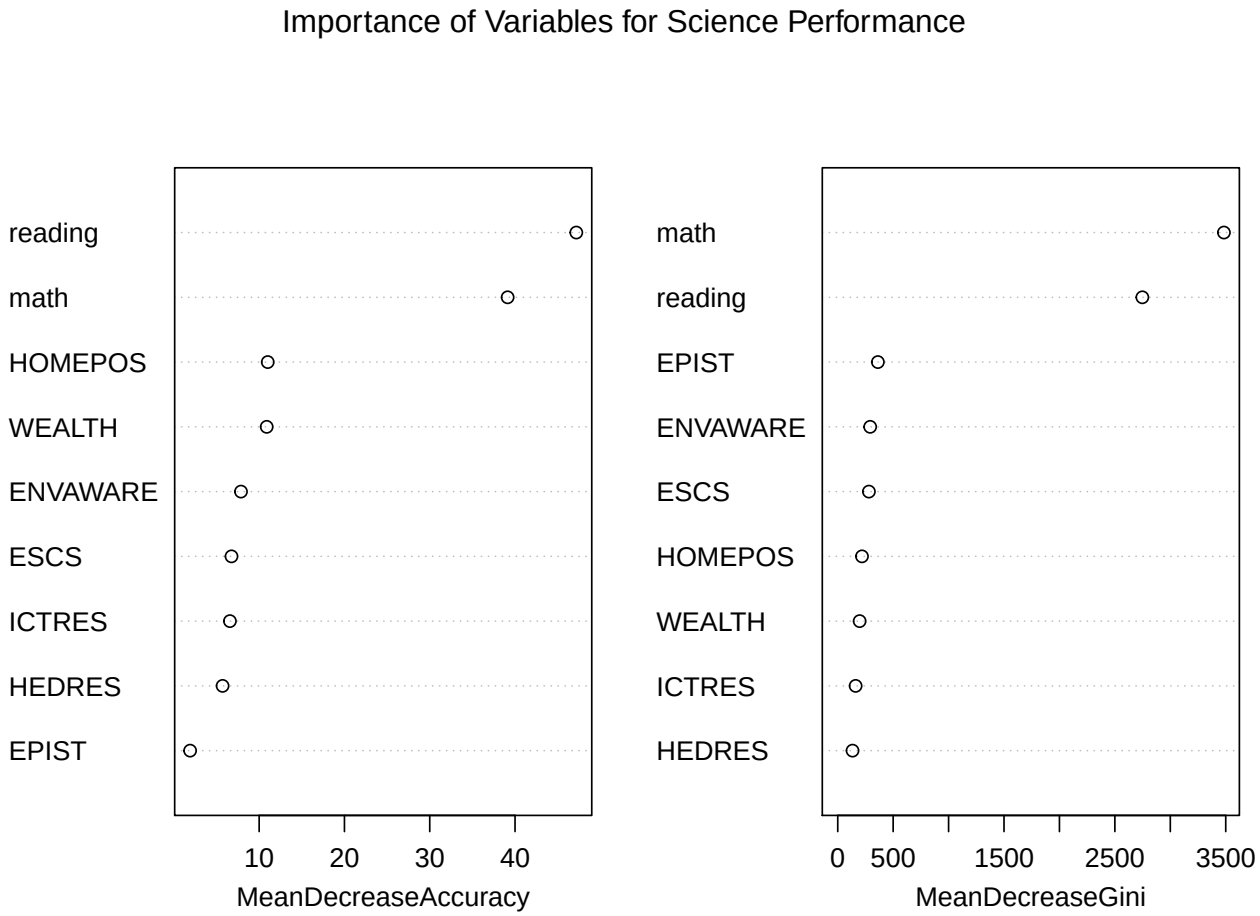
```
## [1] 0.9205362
```

As we did for the decision trees, we can check the importance of the predictors in the model, using `importance()` and `varImpPlot()`. With `importance()`, we will first import the importance measures, turn it into a data.frame, save the row names as predictor names, and finally sort the data by MeanDecreaseGini (or, you can also see the basic output using only `importance(rf_fit2)`)

```
importance(rf_fit2) %>%
  as.data.frame() %>%
  mutate(Predictors = row.names(.)) %>%
  arrange(desc(MeanDecreaseGini))
```

```
##        High       Low MeanDecreaseAccuracy MeanDecreaseGini Predictors
## 1 28.659419 33.636045            39.132181        3483.8309       math
## 2 36.009283 34.864208            47.182695        2747.9847    reading
## 3  1.737624  1.235376             1.906881         362.2179      EPIST
## 4  4.234494  6.218419             7.870379         292.6858   ENVAWARE
## 5  5.395840  3.214590             6.759172         281.1615      ESCS
## 6  6.820185  6.218776            11.009217         218.7035    HOMEPOS
## 7  6.795979  9.105067            10.887967         197.7090     WEALTH
## 8  5.246056  3.811507             6.574882         161.3102     ICTRES
## 9  7.454470  1.509708             5.713732         133.4999     HEDRES
```

```
varImpPlot(rf_fit2,
           main = "Importance of Variables for Science Performance")
```



The output shows different importance measures for the predictors that we used in the model. `MeanDecreaseAccuracy` and `MeanDecreaseGini` represent the overall classification error rate (or, mean

squared error for regression) and the total decrease in node impurities from splitting on the variable, averaged over all trees. In the output, math and reading are the two predictors that seem to influence the model performance substantially, whereas EPIST and HEDRES are the least important variables. `varImpPlot()` presents the same information visually.

Next, we check the confusion matrix to see the accuracy, sensitivity, and specificity of our model.

```
rf_pred <- predict(rf_fit2, test_dat) %>%
  as.data.frame() %>%
  mutate(science_perf = as.factor(`.`)) %>%
  select(science_perf)

confusionMatrix(rf_pred$science_perf, test_dat$science_perf)
```

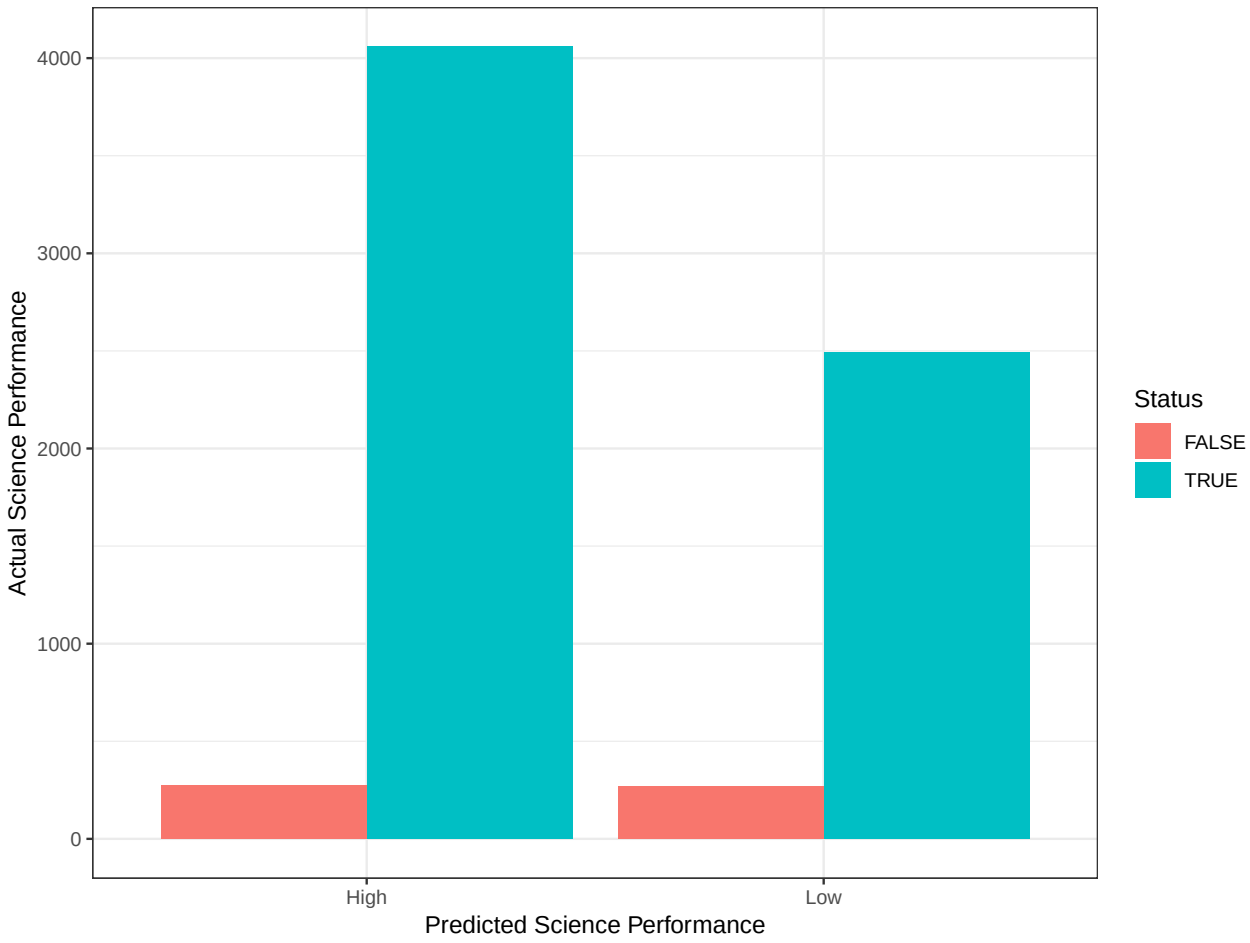
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction High  Low
##       High 4058  274
##       Low   270 2495
##
##                Accuracy : 0.9233
##                  95% CI : (0.9169, 0.9294)
##     No Information Rate : 0.6098
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8389
##  Mcnemar's Test P-Value : 0.8977
##
##             Sensitivity : 0.9376
##             Specificity : 0.9010
##          Pos Pred Value : 0.9367
##          Neg Pred Value : 0.9024
##              Prevalence : 0.6098
##          Detection Rate : 0.5718
##    Detection Prevalence : 0.6104
##       Balanced Accuracy : 0.9193
##
##        'Positive' Class : High
##
```

The results show that the accuracy is quite high (92%). Similarly, sensitivity and specificity are also very high. This is not necessarily surprising because we already knew that the math and reading scores are highly correlated with the science performance. Also, our decision tree model yielded very similar results.

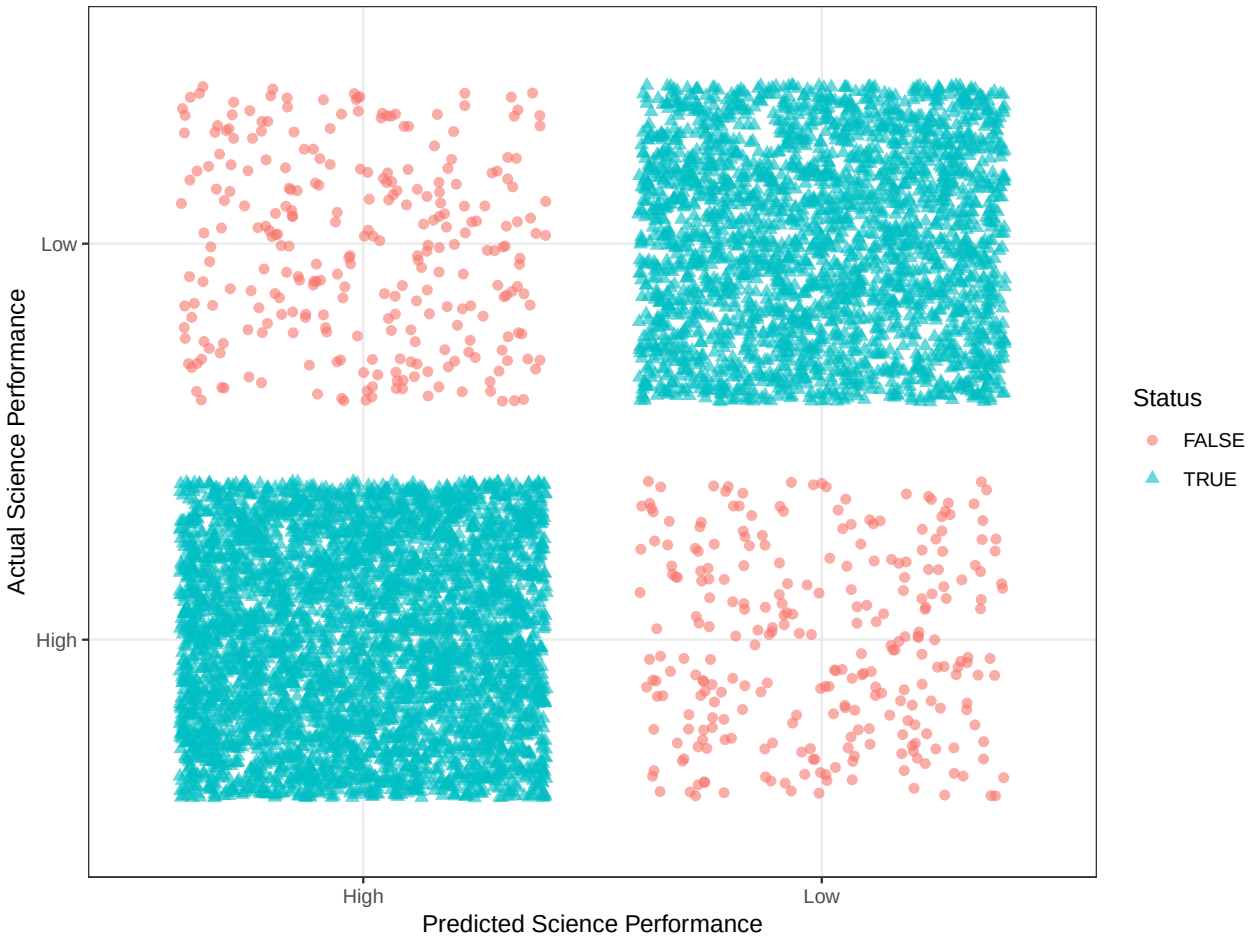
Finally, let's visualize the classification results using `ggplot2`. First, we will create a new dataset called `rf_class` with the predicted and actual classifications (from the test data) based on the random forest model. Then, we will visualize the correct and incorrect classifications using a bar chart and a point plot with jittering.

```
rf_class <- data.frame(actual = test_dat$science_perf,
                       predicted = rf_pred$science_perf) %>%
  mutate(Status = ifelse(actual == predicted, TRUE, FALSE))

ggplot(data = rf_class,
       mapping = aes(x = predicted, fill = Status)) +
  geom_bar(position = "dodge") +
```

```
  labs(x = "Predicted Science Performance",
       y = "Actual Science Performance") +
  theme_bw()
```




```
ggplot(data = rf_class,
       mapping = aes(x = predicted, y = actual,
                     color = Status, shape = Status)) +
  geom_jitter(size = 2, alpha = 0.6) +
  labs(x = "Predicted Science Performance",
       y = "Actual Science Performance") +
  theme_bw()
```



Like decision trees, random forests can also be used for cross-validation, using the package `rfUtilities` that utilizes the objects returned from the `randomForest()` function. Below we show how cross-validation would work for random forests (output is not shown). Using the `randomForest` object that we estimated earlier (i.e.,, `rf_fit2`), we can run cross validations as follows:

```
install.packages("rfUtilities")
library("rfUtilities")

rf_fit2_cv <- rf.crossValidation(
  x = rf_fit2,
  xdata = train_dat,
  p=0.10, # Proportion of data to test (the rest is training)
  n=10,   # Number of cross validation samples
  ntree = 50)


# Plot cross validation verses model producers accuracy
par(mfrow=c(1,2))
plot(rf_fit2_cv, type = "cv", main = "CV producers accuracy")
plot(rf_fit2_cv, type = "model", main = "Model producers accuracy")
par(mfrow=c(1,1))

# Plot cross validation verses model oob
par(mfrow=c(1,2))
```

```
plot(rf_fit2_cv, type = "cv", stat = "oob", main = "CV oob error")
plot(rf_fit2_cv, type = "model", stat = "oob", main = "Model oob error")
par(mfrow=c(1,1))
```

Chapter 8

Supervised Machine Learning - Part II

8.1 Support Vector Machines

The support vector machine (SVM) is a family of related techniques developed in the 80s in computer science. They can be used in either a classification or a regression framework, but are principally known for/applied to classification (of which they are considered one of the best classification techniques because of their flexibility). Following James et al. (2013), we will make the distinction here between maximal margin classifiers (basically a support vector classifier with a cost parameter of 0 and a separating hyperplane), support vector classifiers (or an SVM with a linear kernel), and support vector machines (which employ non-linear kernels).

8.1.1 Maximal Margin Classifier

8.1.1.1 Hyperplane

The concept of a hyperplane is a critical concept in SVM, therefore, we need to understand what exactly a hyperplane is to understand SVM. A **hyperplane** is a subspace whose dimension is one less than that of the ambient space. Specifically, in a *p*-dimensional space, a **hyperplane** is a flat affline subspace of dimensional *p - 1*, where affline refers to the fact that the subspace need not pass through the origin.

We define a hyperplane as

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$

Where $X_1, X_2, ..., X_p$ are predictors (or *features*). Therefore, for any observation of $X = (X_1, X_2, \ldots, X_p)^T$ that *satisfies* the above equation, the observation falls directly onto the hyperplane. However, a value of X does not need to fall onto the hyperplane, but could fall on either side of the hyperplane such that either

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p > 0$$

or

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p < 0$$

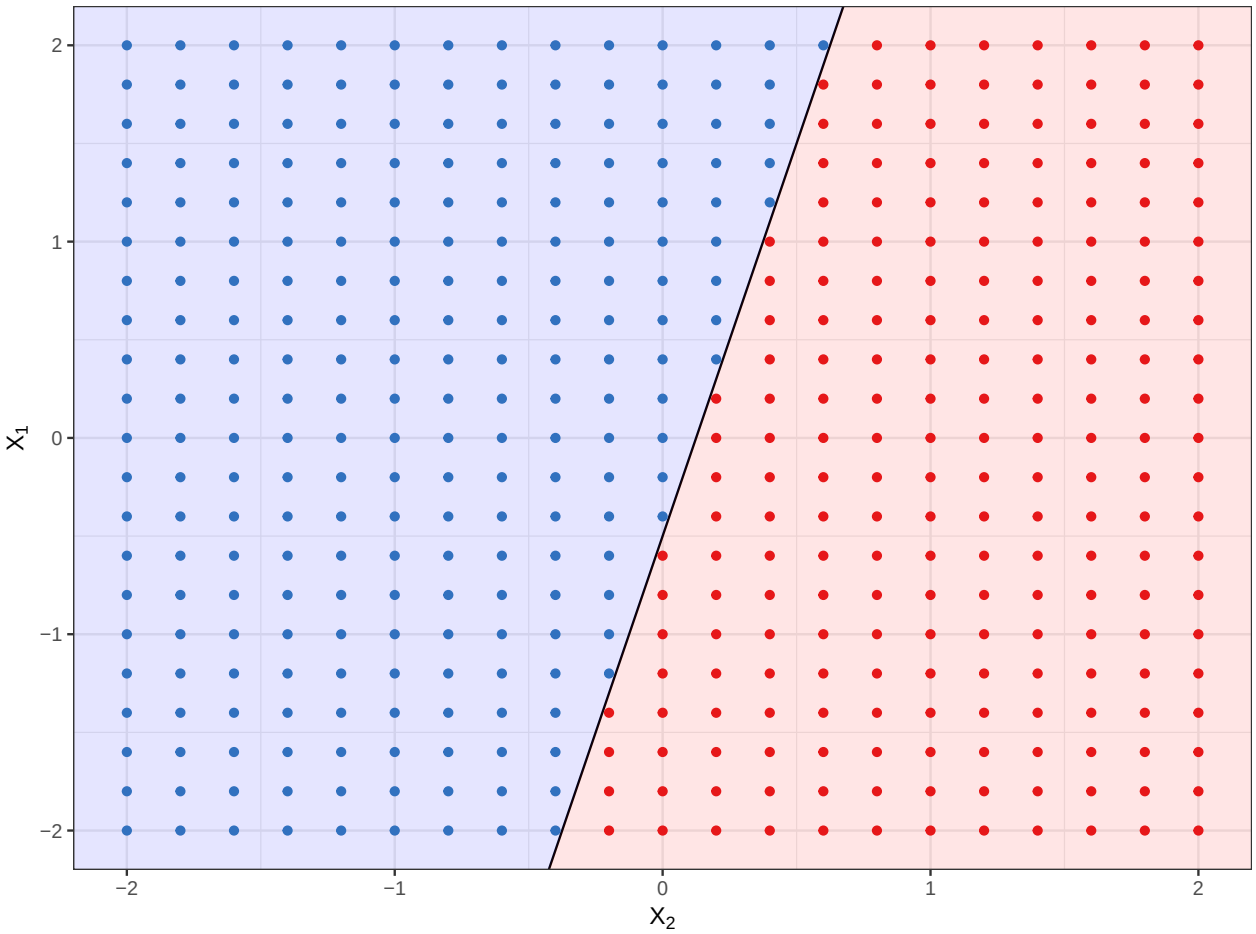


Figure 8.1: The hyperplane, $.5 + 1X_1 + -4X_2 = 0$, is black line, the red points occur in the region where $.5 + 1X_1 + -4X_2 > 0$, while the blue points occur in the region where $.5 + 1X_1 + -4X_2 < 0$.

occurs. In that situation, the value of X lies on one of the two sides of the hyperplane and the hyperplane acts to split the p-dimensional space into two halves.

Figure 8.1 shows the hyperplane $.5 + 1X_1 + -4X_2 = 0$. If we plug a value of X_1 and X_2 into this equation, we know based on the sign alone if the points falls on one side of the hyperplane or if it falls directly onto the hyperplane. In Figure 8.1 all the points in the red region will have negative signs (i.e., if we plug in the values of X_1 and X_2 into the above equation the sign will be negative), while all the points in the blue region would be positive, whereas any points that would have no sign are represented by the black line (the hyperplane).

Wee can apply this idea of a hyperplane to classifying observations. We learned earlier how it important it is when applying machine learning techniques to split our data into training and testing data sets to avoid overfitting. We can split our $n + m$ by p matrix of observations into an n by p $\mathbf{X}$ matrix of training observations, which fall into one of two classes for $Y = y_1, .., y_n$ where $Y_i \in -1, 1$ and an m by p matrix $\mathbf{X}^*$ of testing observations. Using just the training data, our goal is develop a model that will correctly classify our testing data using just a hyperplane and we will do this by creating a **separating hyperplane** (a hyperplane that will separate our classes).

Let's assume we have the training data in Figure 8.2 and that the blue points correspond to one class (labelled as $y = 1$) and the red points correspond to the other class ($y = -1$). The separating hyper plane has the property that:

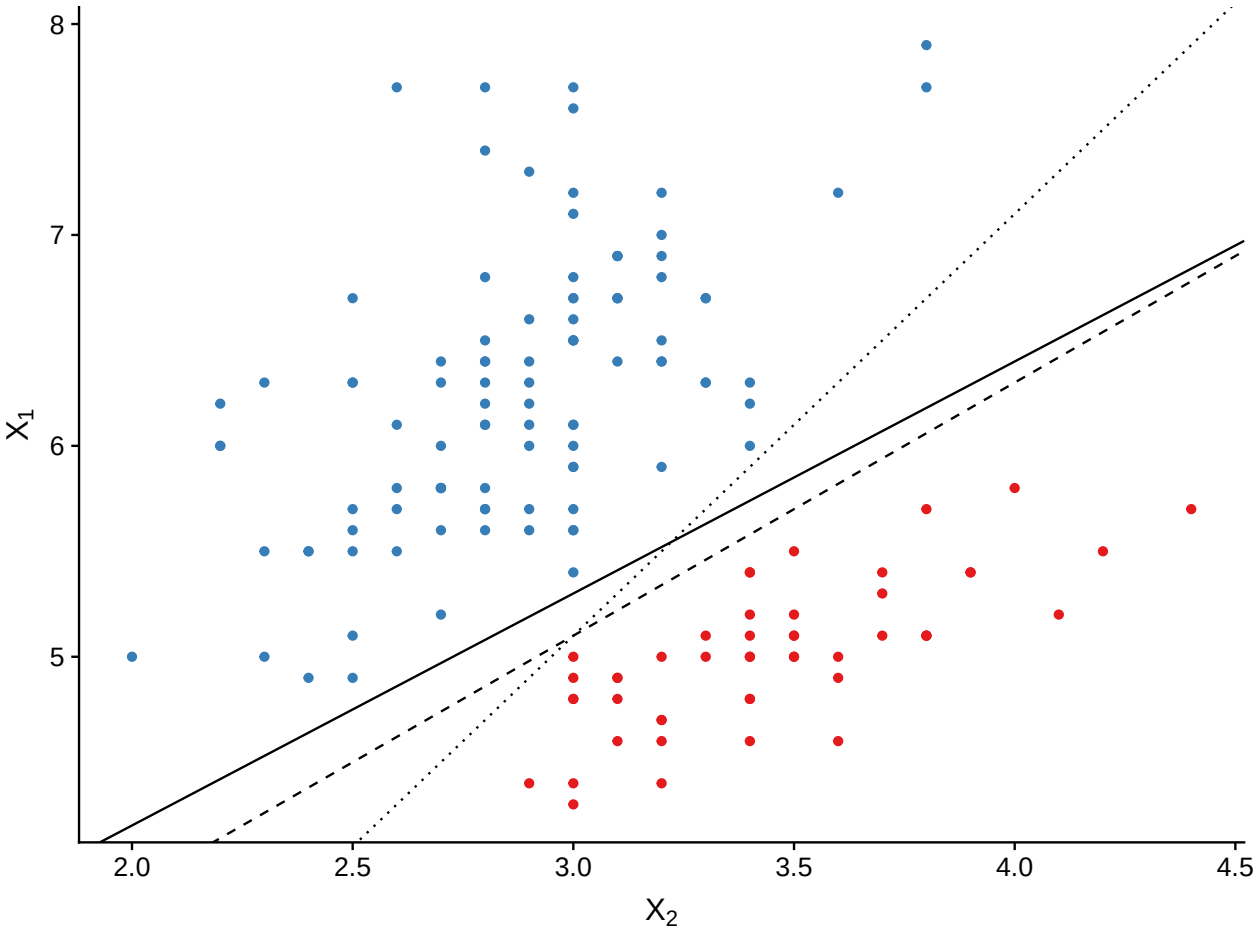


Figure 8.2: Candidate hyperplanes to separate the two classes.

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{p1} > 0 \quad \text{if} \quad y_i = 1$$

and

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{p1} < 0 \quad \text{if} \quad y_i = -1$$

Or more succintly,

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{p1}) > 0$$

Ideally, we would create a hyperplane that perfectly separates the classes based on X_1 and X_2. However, as Figure 8.2 makes clear, we can create many separating hyperplanes of which 3 of these are shown. In fact, it's often the case that an infinite number of separating hyperplanes could be created when the classes are perfectly separable. What we need to do is to develop some kind of a criterion for selecting one of the many separating hyperplanes.

For any given hyperplane, we have two pieces of information available for each observation: 1) the side of the hyperplane it lies on (represented by its sign) and 2) the distance it is from the hyperplane. The natural criterion for selecting a separating hyperplane is to **maximize the distance** it is from from the training observations. Therefore, we compute the distance that each training observation is from a candidate hyperplane. The minimal such distance from the observation to the hyperplane is known as the **margin**. Then we will select the hyperplane with the largest margin (the **maximal margin hyperplane**) and classify observations based on which side of this hyperplane they fall (**maximal margin classifier**). The hope is

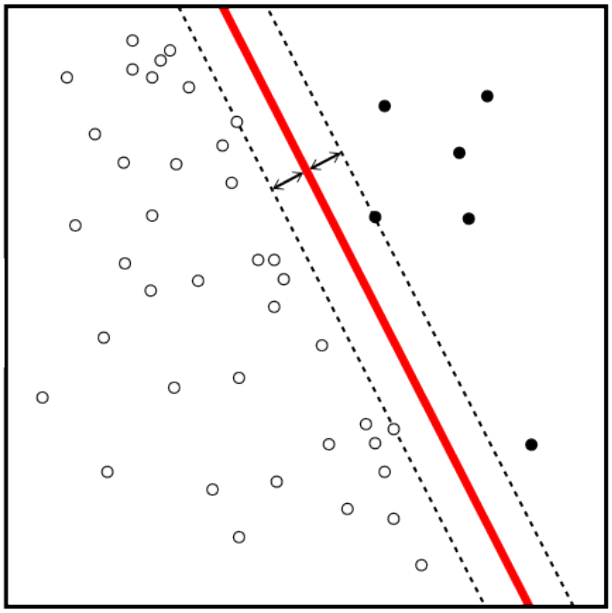


Figure 8.3: Maximal margin hyperplane. Source: https://tinyurl.com/y493pww8

that a classifier with a large margin on the training data will also have a large margin on the test observations and subsequently classify well.

Figure 8.3 depicts a maximal margin classifier. The red line corresponds to the maximal margin hyperplane and the distance between one of the dotted lines and the black line is the **margin**. The black and white points along the boundary of the margin are the **support vectors**. It is clear in Figure 8.3 that the maximal margin hyperplane depends only on these two support vectors. If they are moved, the maximal margin hyperplane moves, however, if any other observations are moved they would have no effect on this hyperplane *unless* they crossed the boundary of the margin.

The problem in practice is that a separating hyperplane usually doesn't exist. Even if a separating hyperplane existed, we may not want to use the maximal margin hyperplane as it would perfectly classify all of the observations and may be too sensitive to individual observations and subsequently overfitting.

Figure 8.4 from James, et al. (2013) clearly illustrates this problem. The left figure shows the maximal margin hyperplane (solid) in a completely separable solution. The figure on the right shows that when a new observation is introduced that the maximal margin hyperplane (solid) shifts rather dramatically relative to its original location (dashed).

8.1.2 Support Vector Classifier

Our hope for a hyperplane is that it would be relatively insensitive to individual observations, while still classifying training observations well. That is, we would like to have what is termed a **soft margin classifier** or a **support vector classifier**. Essentially, we are willing to allow some observations to be on the incorrect side of the margin (classified correctly) or even the incorrect side of the hyperplane (incorrectly classified) if our classifier, overall, performs well.

We do this by introducing a tuning parameter, C, which determines the number and the severity of violations to the margin/hyperplane we are willing to tolerate. As C increases, our **tolerance** for violations will increase and subsequently our margin will widen. C, thus, represents a **bias-variance tradeoff**, when C is small bias should be low, but variance will likely be high, whereas when C is large, bias is likely high but our variance is typically small. C will be selected, optimally, through cross-validation (as we'll see later).

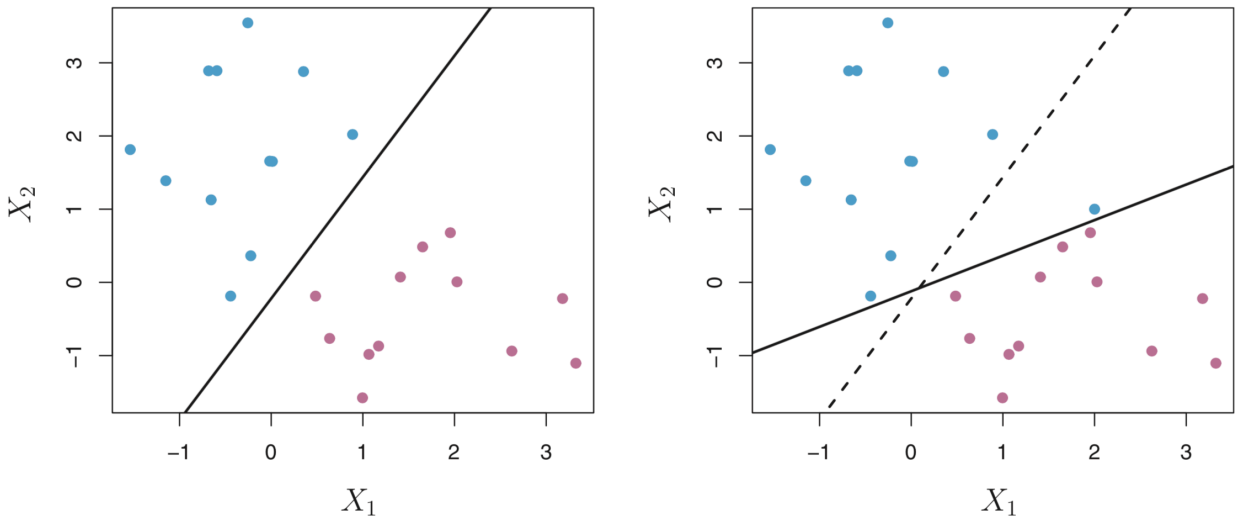


Figure 8.4: The impact of adding one observations to the maximal margin hyperplane from James et al. (2013).

The observations that lie on the margin or violate the margin are the only ones that will affect the hyperplane and the classifier (similar to the maximal margin classifier). These observations are the **support vectors** and only they will affect the support vector classifier. When C is large, there will be many support vectors, whereas when C is small, the number of support vectors will be less.

Because the support vector classifier depends on only the on the support vectors (which could be very few) this means they are quite **robust to the observations that are far** from the hyperplane. This makes this technique similar to logistic regression.

8.1.2.1 Example

In our example, we'll try and classify whether someone scores at or above the mean on the science scale we created earlier. To do support vector classifiers (and SVMs) in R, we'll use the `e1071` package (though the `caret` package could be used, too).

```
# check if e1071 is installed
# if not, install it
if (!("e1071" %in% installed.packages()[,"Package"])) {
  install.packages("e1071")
  library("e1071")
} else {
  library("e1071")
}
```

The `svm` function in the `e1071` package requires that the outcome variable is a factor. So, we'll do a mean split (at the OECD mean of 493) on the `science` scale and convert it to a factor.

```
pisa[, sci_class := as.factor(ifelse(science >= 493, 1, -1))]
```

While, I'm coding this variable as 1 and -1 to be consistent with the notation above, it doesn't matter to the `svm` function. The only thing the `svm` function needs to perform classification and not regression is that the outcome is a factor. If the outcome has just two values, a 1 and -1, but is not a factor, `svm` will perform regression.

We will use the following variables in our model:

Label	Description
WEALTH	Family wealth (WLE)
HEDRES	Home educational resources (WLE)
ENVAWARE	Environmental Awareness (WLE)
ICTRES	ICT Resources (WLE)
EPIST	Epistemological beliefs (WLE)
HOMEPOS	Home possessions (WLE)
ESCS	Index of economic, social and cultural status (WLE)
reading	Reading score
math	Math score

We'll subset the variables to make it easier and in order for the model fitting to be performed in a reasonable amount of time in R, we'll just subset the United States and Canada.

```
pisa_sub <- subset(pisa, CNT %in% c("Canada", "United States"), select = c(sci_class, WEALTH, HEDRES, EI
```

To fit a support vector classier, we use the `svm` function. Before we get started, let's divide the data set into a training and a testing data set. We will use a 66/33 split, though other splits could be used (e.g., 50/50).

```
# set a random seed
set.seed(442019)

# svm uses listwise deletion, so we should just drop
# the observations now
pisa_m <- na.omit(pisa_sub)

# select the rows that will go into the training data set.
train <- sample(1:nrow(pisa_m), 2/3 * nrow(pisa_m))

# subset the data based on the rows that were selected to be in training data set.
train_dat <- pisa_m[train, ]
test_dat <- pisa_m[-train, ]
```

To perform support vector classification, we pass the `svm` function the `kernel = "linear"` argument. We also need to specify our tolerance, which is represented by the `cost` argument. The `cost` parameter is essentially the inverse of the tolerance parameter, C, described above. When the `cost` value is low, the tolerance is high (i.e., the margin is wide and there are lots of support vectors) and when the `cost` value is high, the tolerance is low (i.e., narrower margin). By default `cost = 1` and we will tune this parameter via cross-validation momentarily. For now, we'll just fit the model.

```
svc_fit <- svm(sci_class ~., data = train_dat, kernel = "linear")
```

We can obtain basic information about our model using the `summary` function.

```
summary(svc_fit)
```

```
##
## Call:
## svm(formula = sci_class ~ ., data = train_dat, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.1111111
```

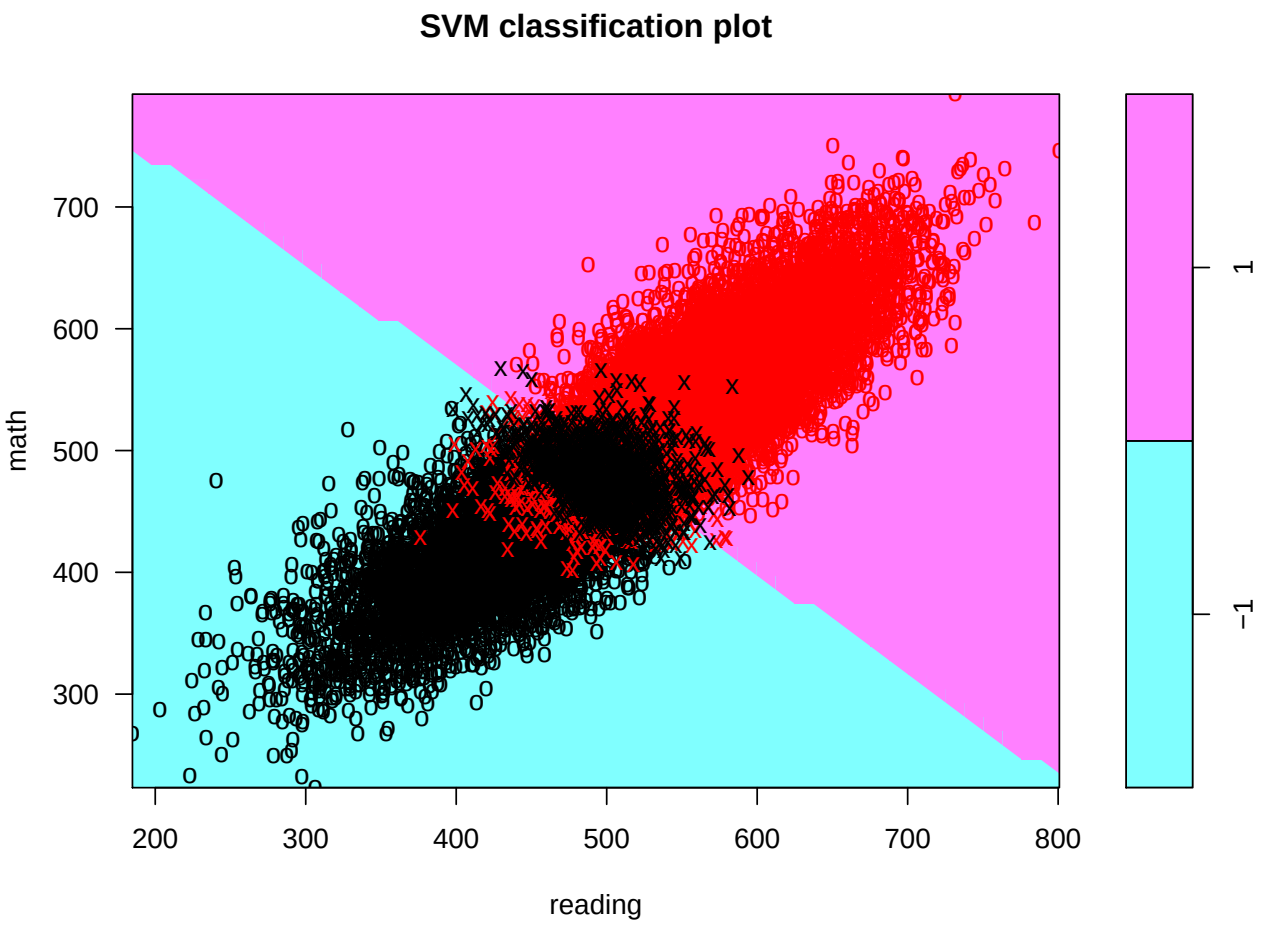


Figure 8.5: Support vector classifier plot for all the training data.

```
##
## Number of Support Vectors:  2782
##
##  ( 1390 1392 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

We see there are 2782 support vectors: 1390 in class -1 and 1392 in class 1. We can also plot our model but we need to specific the two features we want to plot (because our model has nine feature). Let's look at the model with `math` on the y-axis and `reading` on the x-axis.

```
plot(svc_fit, data = train_dat, math ~ reading)
```

In this figure, the red points correspond to observations that belong to class 1 (below the mean on science), while the black points correspond to observations that belong to class -1 (at/above the mean on science); the Xs are the support vectors, while the Os are the non-support vector observations; the upper triangle (purple) are for class 1, while the lower triangle (blue) is for class -1. While the decision boundary looks jagged, it's just an artifact of the way it's drawn with this function. We can see that many observations are misclassified (i.e., some red points are in the lower triangle and some black points are in the upper triangle).

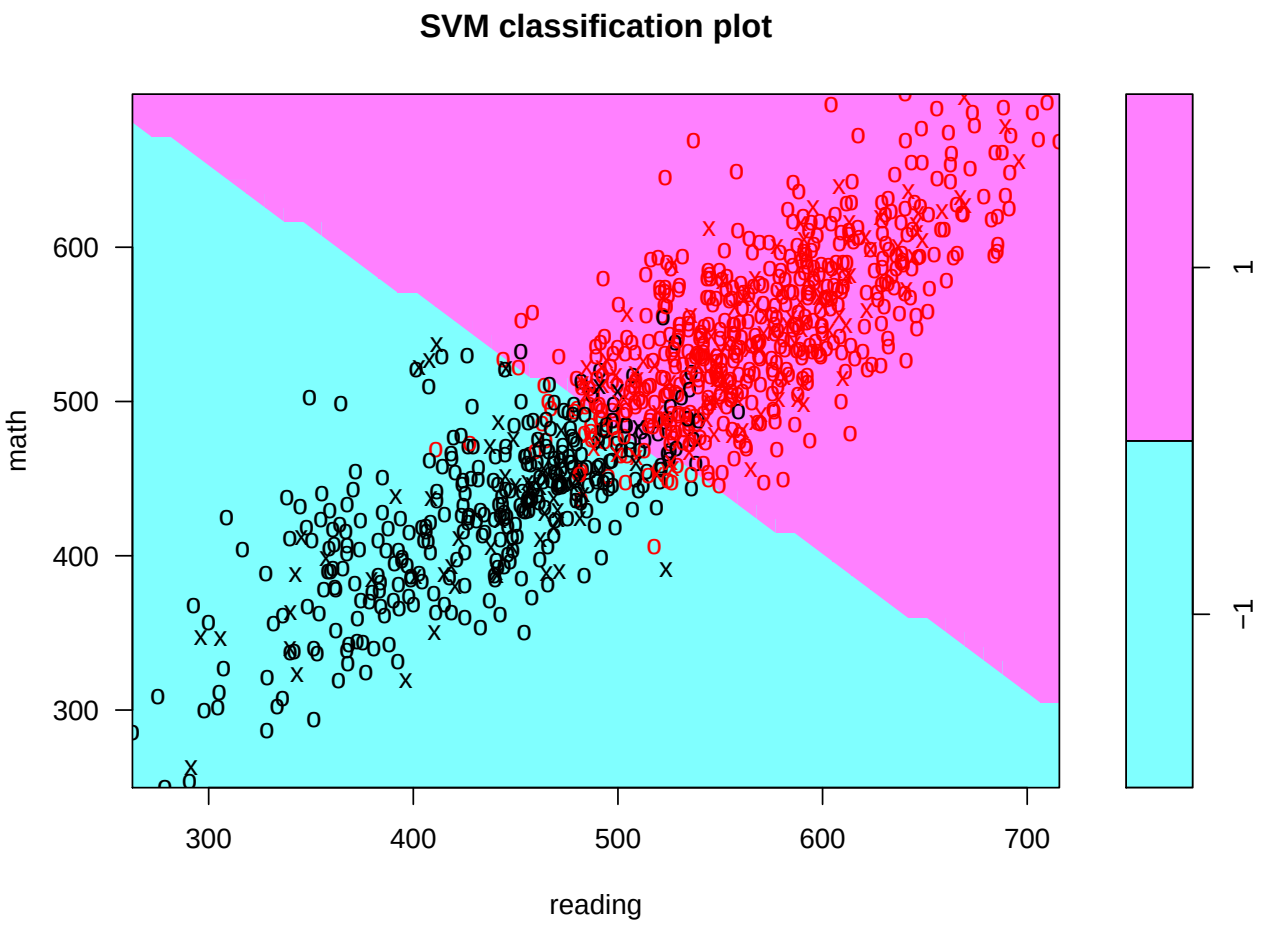


Figure 8.6: Support vector classifier plot for all a random subsample (n = 1000) of training observations.

However, there are a lot of observations shown in this figure and it is difficult to discern the nature of the misclassification.

As was discussed in the section on data visualization, with this many points on a figure it is difficult to evaluate patterns, not to mention that the figure is extremely slow to render. Therefore, let's take a random sample of 1,000 observations to get a better sense of our classifier. This is shown in Figure 8.6.

```
set.seed(1)
ran_obs <- sample(1:nrow(train_dat), 1000)
plot(svc_fit, data = train_dat[ran_obs, ], math ~ reading)
```

Notice that few points are crossing the hyperplane (i.e., are misclassified). This looks like the classier is doing pretty good.

Initially when we fit the support vector classifier we used the default cost parameter, but we really should select this parameter through tuning via cross-validation as we might be able to do an even better job at classifying. The `e1071` package includes a `tune` function which makes this easy and automatic. It performs the tuning via 10-folds cross-validation by default, which is probably a fine tradeoff (see James, et al. 2013 for a comparison of k-folds vs. leave one out cross-validation). We need to provide the `tune` function with a range of cost values (which again corresponds to our tolerance to violate the margin and hyperplane).

```
tune_svc <- tune(svm, sci_class ~., data = train_dat,
                 kernel="linear",
                 ranges = list(cost = c(.01, .1, 1, 5, 10)))
```

On my Macbook Pro (2.6 GHz Intel Core i7 and 16 GB RAM) it takes approximately 2 minutes run this. Without doing this subsetting, it will take quite a bit longer to do.

We can view the cross-validation errors by using the `summary` function on this object.

```
summary(tune_svc)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.07316727
##
## - Detailed performance results:
##    cost      error  dispersion
## 1  0.01 0.07342096 0.005135708
## 2  0.10 0.07354766 0.004985649
## 3  1.00 0.07316727 0.004952085
## 4  5.00 0.07329406 0.004879146
## 5 10.00 0.07335747 0.004887063
```

And then select the best model and view it.

```
best_svc <- tune_svc$best.model
summary(best_svc)
```

```
##
## Call:
## best.tune(method = svm, train.x = sci_class ~ ., data = train_dat,
##     ranges = list(cost = c(0.01, 0.1, 1, 5, 10)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.1111111
##
## Number of Support Vectors:  2782
##
##  ( 1390 1392 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

Next, we write a function to evaluate our classifier that has one argument that takes a confusion matrix.

```
#' Evaluate classifier
#'
```

```
#' Evaluates a classifier (e.g. SVM, logistic regression)
#' @param tab a confusion matrix
eval_classifier <- function(tab, print = F){
  n <- sum(tab)
  TP <- tab[2,2]
  FN <- tab[2,1]
  FP <- tab[1,2]
  TN <- tab[1,1]
  classify.rate <- (TP + TN) / n
  TP.rate <- TP / (TP + FN)
  TN.rate <- TN / (TN + FP)
  object <- data.frame(accuracy = classify.rate,
                       sensitivity = TP.rate,
                       specificity = TN.rate)
  object
}
```

The confusion matrix is just a list of all possible outcomes (true positives, true negatives, false positives, and false negatives). A confusion matrix for our `best_svc` can be created by:

```
# to create a confusion matrix this order is important!
# observed values first and predict values second!
svc_cm_train <- table(train_dat$sci_class,
                      predict(best_svc))
svc_cm_train
```

```
##
##        -1    1
##   -1 5563  606
##   1   550 9053
```

The top-left are the true negatives, the bottom-left are the false negatives, the top-right are the false positives, and the bottom-right are the true positives. We can request the accuracy (the % of observations that were correctly classified), the sensitivity (the % of observations that were in class 1 that were correctly identified), and specificity (the % of observations that were in class -1 that were correctly identified) using the `eval_classifier` function.

```
eval_classifier(svc_cm_train)
```

```
##    accuracy sensitivity specificity
## 1 0.9267056   0.9427262   0.9017669
```

Performance is pretty good overall. We see that class -1 (specificity) isn't classified as well as class 1 (sensitivity). These statistics are likely overly optimistic as we are evaluating our model using the training data (the same data that we used to build our model). How well does the model perform on the testing data?

```
svc_cm_test <- table(test_dat$sci_class,
                     predict(best_svc, newdata = test_dat))
svc_cm_test
```

```
##
##        -1    1
##   -1 2780  281
##   1   278 4547
```

```
eval_classifier(svc_cm_test)
```

```
##    accuracy sensitivity specificity
## 1 0.9291149   0.9423834   0.9081999
```

Still impressively high! This is a very good classifier indeed. This is likely because `math` and `reading` are so highly correlated with `science` scores.

We can extract the coefficients from our model that make up our decision boundary.

```
beta0 <- best_svc$rho
beta <-  drop(t(best_svc$coefs) %*% as.matrix(train_dat[best_svc$index, -1]))
beta0
```

```
## [1] -1.220883
```

```
beta
```

```
##       WEALTH       HEDRES     ENVAWARE       ICTRES        EPIST
##   0.04398688  -0.24398165   0.36167882  -0.09803825   0.04652237
##      HOMEPOS         ESCS      reading         math
##   0.22005477  -0.15065808 188.02960807 196.93421586
```

With more complicated SVMs with non-linear kernels, coefficients don't make any sense and generally are of little interest with applying these models.

8.1.2.2 Comparison to logistic regression

Support vector classifiers are quite similar to logistic regression. This has to do with them having similar loss functions (the functions used to estimate the parameters). In situations where the classes are well separated, SVM (more generally), tend to do better than logistic regression and when they are not well separated, logistic regression tends to do better (James, et al., 2013).

Let's compare logistic regression to the support vector classier. We'll begin by fitting the model

```
lr_fit <- glm(sci_class ~. , data = train_dat, family = "binomial")
```

and then viewing the coefficients.

```
coef(lr_fit)
```

```
##  (Intercept)       WEALTH       HEDRES     ENVAWARE       ICTRES
## -41.82682653   0.11666541  -0.26667828   0.30159987  -0.13594566
##        EPIST      HOMEPOS         ESCS      reading         math
##   0.05053261   0.20699211  -0.24568642   0.03917470   0.04651408
```

How does it do relative to our best support vector classifier on the training and the testing data sets? For the training data set

```
lr_cm_train <- table(train_dat$sci_class,
                     round(predict(lr_fit, type = "response")))
lr_cm_train
```

```
##
##         0    1
##   -1 5567  602
##   1   541 9062
```

```
eval_classifier(lr_cm_train)
```

```
##    accuracy sensitivity specificity
## 1 0.9275298   0.9436634   0.9024153
```

and then for the testing data set.

```
lr_cm_test <- table(test_dat$sci_class,
                    round(predict(lr_fit, newdata = test_dat, type = "response")))
lr_cm_test
```

```
##
##         0    1
##   -1 2780  281
##   1   275 4550
```

```
eval_classifier(lr_cm_test)
```

```
##    accuracy sensitivity specificity
## 1 0.9294953   0.9430052   0.9081999
```

Equivalent out to the hundredths place. Either model would be fine here.

8.1.2.3 Using Apache Spark for machine learning

Apache Spark is also capable of running support vector classifiers. It does this using the `ml_linear_svc` function. The amazing thing about this is that you can use it to run the entire data set (i.e., there is no need to subset out a portion of the countries). If we tried to do this with the `e1071` package it would be very impractical and take forever, but with Apache Spark it is feasible and reasonably quick (just a few minutes).

We'll again use the `sparklyr` package to interface with Spark and use the `dplyr` package to simplify interacting with Spark.

```
library(sparklyr)
library(dplyr)
```

We first need to establish a connection with Spark and then copy a subsetted PISA data set to Spark.

```
sc <- spark_connect(master = "local")
spark_sub <- subset(pisa,
                    select = c(sci_class, WEALTH, HEDRES, ENVAWARE, ICTRES,
                               EPIST, HOMEPOS, ESCS, reading, math))
spark_sub <- na.omit(spark_sub) # can't handle missing data
pisa_tbl <- copy_to(sc, spark_sub, overwrite = TRUE)
```

Now, we'll let Spark partition the data into a training and a test data set.

```
partition <- pisa_tbl %>%
  sdf_partition(training = 2/3, test = 1/3, seed = 442019)
pisa_training <- partition$training
pisa_test <- partition$test
```

We are ready to run the classifier in Spark. Unlike the `svm` function, the tolerance parameter is called `reg_param`. This parameter should be optimally selected like it was for `svm`. By default the tolerance is 1e-06.

```
svc_spark <- pisa_training %>%
  ml_linear_svc(sci_class ~ .)
```

We then use the `ml_predict` function to predict the classes.

```
svc_pred <- ml_predict(svc_spark, pisa_training) %>%
  select(sci_class, predicted_label) %>%
  collect()
```

Then print the confusion matrix and the criteria that we've been using to evaluate our models.

```
table(svc_pred)
```

```
##          predicted_label
## sci_class     -1      1
##        -1 145111  12353
##        1   10753 121967
```

```
eval_classifier(table(svc_pred))
```

```
##    accuracy sensitivity specificity
## 1 0.9203747   0.9189798   0.9215503
```

Again, this is really good. How does it look on the testing data?

```
svc_pred_test <- ml_predict(svc_spark, pisa_test) %>%
  select(sci_class, predicted_label) %>%
  collect()
```

```
table(svc_pred_test)
```

```
##          predicted_label
## sci_class    -1     1
##        -1 72577  6199
##        1   5438 60953
```

```
eval_classifier(table(svc_pred_test))
```

```
##    accuracy sensitivity specificity
## 1 0.9198372   0.9180913   0.9213085
```

Pretty impressive. We can also Apache Spark to fit logistic regression using the `ml_logistic_regression` function.

```
spark_lr <- pisa_training %>%
  ml_logistic_regression(sci_class ~ .)
```

And view the performance on the training and test data sets.

```
## Training data
svc_pred_lr <- ml_predict(spark_lr, pisa_training) %>%
  select(sci_class, predicted_label) %>%
  collect()
table(svc_pred_lr)
```

```
##          predicted_label
## sci_class     -1      1
##        -1 146217  11247
##        1   11133 121587
```

```
eval_classifier(table(svc_pred_lr))
```

```
##    accuracy sensitivity specificity
## 1 0.9228765   0.9161166   0.9285742
```

```
## Test data
svc_pred_test_lr <- ml_predict(spark_lr, pisa_test) %>%
  select(sci_class, predicted_label) %>%
  collect()
table(svc_pred_test_lr)
```

```
##          predicted_label
## sci_class    -1     1
##        -1 73098  5678
##        1   5646 60745
```

```
eval_classifier(table(svc_pred_test_lr))
```

```
##    accuracy sensitivity specificity
## 1 0.9219933   0.9149584   0.9279222
```

We could also use the logistic regression in R as it's pretty quick even with this large of a data set (in fact, it's slightly quicker).

Finally, it is quite common to evaluate these models using AUC. We can let Apache Spark do this for the test data sets.

```
# extract predictions
pred_svc <- ml_predict(svc_spark, pisa_test)
pred_lr <- ml_predict(spark_lr, pisa_test)

ml_binary_classification_evaluator(pred_svc)
```

```
## [1] 0.9795075
```

```
ml_binary_classification_evaluator(pred_lr)
```

```
## [1] 0.9805051
```

We want these values as close to 1 as a possible. These values are all quite large and corroborate that these are both good classifiers.

8.1.3 Support Vector Machine

SVM is an extension of support vector classifiers using **kernels** that allow for a non-linear boundary between the classes. Without getting into the weeds, to solve a support vector classifier problem all you need to know is the inner products of the observations. Assuming that x_i and x_i' are two observations and p is the number of predictors (features), their inner product is defined as:

$$\langle x_i, x_i' \rangle = \begin{bmatrix} x_{i1} x_{i2} \dots x_{ip} \end{bmatrix} \begin{bmatrix} x_{i1}' \\ x_{i2}' \\ \vdots \\ x_{ip}' \end{bmatrix} = x_{i1}x_{i1}' + x_{i2}x_{i2}' + \dots x_{ip}x_{ip}'$$

More succinctly, $\langle x_i, x_i' \rangle = \sum_{i=1}^{p} x_{ij}x_{ij}'$. We can replace the inner product with a more general form, $K(x_i, x_i')$, where K is a kernel (a function that quantifies the similarity of two observations). When,

$$K(x_i, x_i') = \sum_{i=1}^{p} x_{ij}x_{ij}$$

We have the linear kernel and this is the support vector classifier. However, we can use a more flexible kernel. Such as:

$$K(x_i, x_i') = (1 + \sum_{i=1}^{p} x_{ij}x_{ij})^d$$

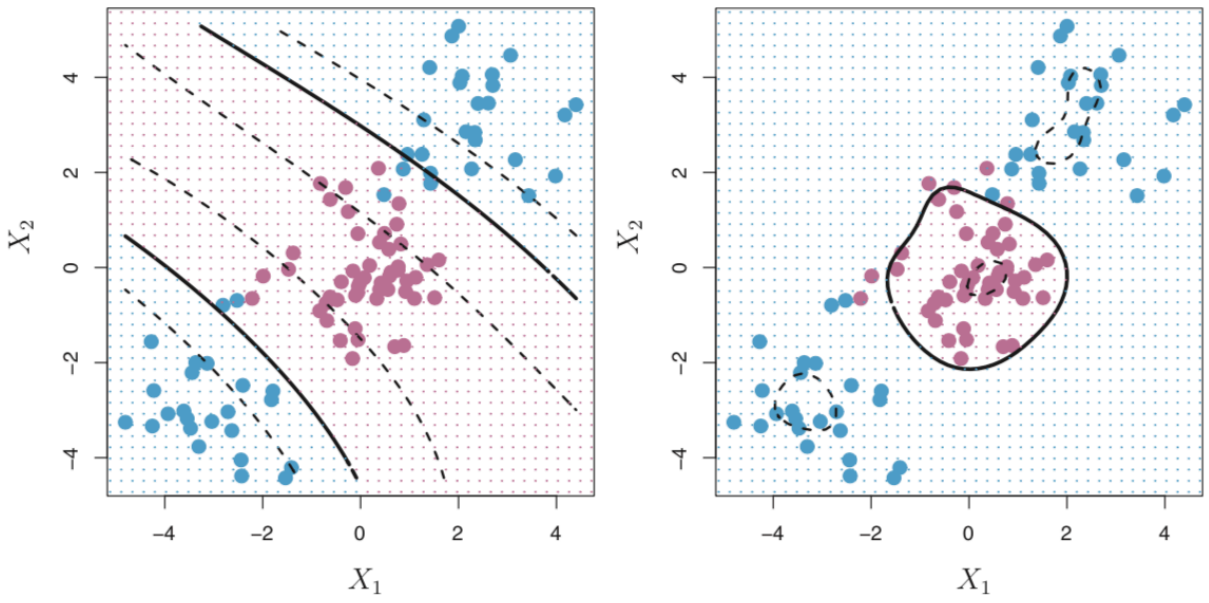


Figure 8.7: Non-linear decision boundary with a polynomial kernel (left) and radial kernel (right) from James et al., 2013.

which is known as a **polynomial kernel** of degree d and when $d > 1$ we have much more flexible decision boundary than we do for support vector classifiers (when $d = 1$ we are back to the support vector classifier).

Another very common kernel is the **radial kernel**, which is given by:

$$K(x_i, x_i') = \exp\left(-\gamma \sum_{i=1}^{p} (x_{ij} - x_{ij})^2\right)$$

where γ is a positive constant. Note, both d and γ are selected via tuning and cross-validation.

Both of these kernels are worth considering when the decision boundary is non-linear. Figure 8.7 from James, et al. (2013) gives an example of a non-linear boundary. We see that the classes are not linearly separated and if we tried to use a linear decision boundary, we would end up with a very poor classifier. Therefore, we need to use a more flexible kernel. In both cases, we should expect that an SVM would greatly outperform both a support vector classifier and logistic regression.

8.1.3.1 Examples

We will continue trying to build the best classifier of whether someone scored in the upper or lower half on the science scale and again use the `svm` function in the `e1071` package. For brevity, we'll consider only the radial kernel. By default gamma is set to 1. We'll explicitly set it to 1 below and cost to 1.

```
svm_fit <- svm(sci_class ~., data = train_dat,
               cost = 1,
               gamma = 1,
               kernel = "radial")
```

Again, we can request some basic information about our model.

```
summary(svm_fit)
```

```
##
## Call:
## svm(formula = sci_class ~ ., data = train_dat, cost = 1, gamma = 1,
##     kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##
## Number of Support Vectors:  6988
##
##  ( 3676 3312 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

This time we see we have 6988 support vectors, 3676 in class -1 and 3312 in class 1. Quite a bit more support vectors than the support vector classifier. Lets visually inspect this model by plotting it against the math and reading features on the same subset of test takers (Figure 8.8.

```
plot(svm_fit, data = train_dat[ran_obs, ], math ~ reading)
```

We see that the decision boundary is now clearly no longer linear and we again see decent classification. Before we investigate the fit of the model, we should tune it.

```
tune_svm <- tune(svm, sci_class ~., data = train_dat,
                 kernel = "radial",
                 ranges = list(cost = c(.01, .1, 1, 5, 10),
                               gamma = c(0.5, 1, 2, 3, 4)))
```

We can see which model was selected

```
summary(tune_svm)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   0.1   0.5
##
## - best performance: 0.07583144
##
## - Detailed performance results:
##    cost gamma      error  dispersion
## 1  0.01   0.5 0.17670590 0.010347886
```

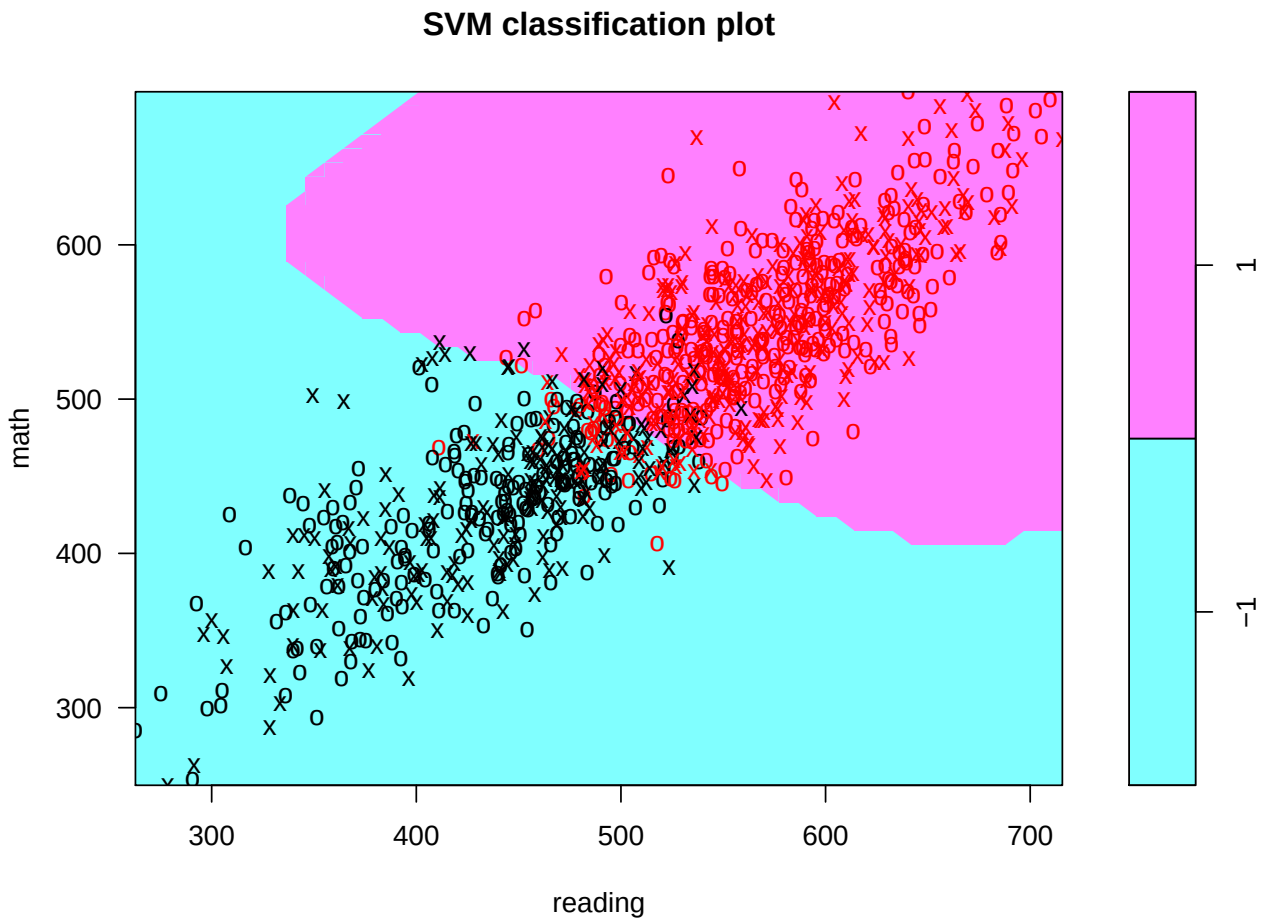


Figure 8.8: Support vector classifier plot for all a random subsample (n = 1000) of training observations.

```
## 2   0.10   0.5 0.07583144 0.008233504
## 3   1.00   0.5 0.07754307 0.009223528
## 4   5.00   0.5 0.08375680 0.008098257
## 5  10.00   0.5 0.08718054 0.008157393
## 6   0.01   1.0 0.36425229 0.011955406
## 7   0.10   1.0 0.13162657 0.007210614
## 8   1.00   1.0 0.08242504 0.008590571
## 9   5.00   1.0 0.09402859 0.009848512
## 10 10.00   1.0 0.10074908 0.007984562
## 11  0.01   2.0 0.39113500 0.011126599
## 12  0.10   2.0 0.31409966 0.010609909
## 13  1.00   2.0 0.11469785 0.006880824
## 14  5.00   2.0 0.12363760 0.006591525
## 15 10.00   2.0 0.12465198 0.006523243
## 16  0.01   3.0 0.39113500 0.011126599
## 17  0.10   3.0 0.38257549 0.012981991
## 18  1.00   3.0 0.17562831 0.007488136
## 19  5.00   3.0 0.17277475 0.006502038
## 20 10.00   3.0 0.17309173 0.006145790
## 21  0.01   4.0 0.39113500 0.011126599
## 22  0.10   4.0 0.39107163 0.011072976
## 23  1.00   4.0 0.23960159 0.011545434
## 24  5.00   4.0 0.22641364 0.008709051
## 25 10.00   4.0 0.22641360 0.008779341
```

And then select the best model and view it.

```
best_svm <- tune_svm$best.model
summary(best_svm)
```

```
##
## Call:
## best.tune(method = svm, train.x = sci_class ~ ., data = train_dat,
##     ranges = list(cost = c(0.01, 0.1, 1, 5, 10), gamma = c(0.5,
##         1, 2, 3, 4)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.1
##       gamma:  0.5
##
## Number of Support Vectors:  6138
##
##  ( 3095 3043 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

Finally, we see how well this predicts on both the training observations

```
svm_cm_train <- table(train_dat$sci_class,
                      predict(best_svm))
svm_cm_train
```

```
##
##        -1    1
##   -1 5620  549
##   1   519 9084
```

```
eval_classifier(svm_cm_train)
```

```
##    accuracy sensitivity specificity
## 1 0.9322851   0.9459544   0.9110066
```

and finally the testing observations.

```
svm_cm_test <- table(test_dat$sci_class,
                      predict(best_svm, newdata = test_dat))
svm_cm_test
```

```
##
##        -1    1
##   -1 2781  280
##   1   291 4534
```

```
eval_classifier(svm_cm_test)
```

```
##    accuracy sensitivity specificity
## 1 0.9275932   0.9396891   0.9085266
```

Performance is very comparable to the support vector classifier and logistic regression implying there isn't much gain from the use of non-linear decision boundary.

8.1.4 Lab

For the lab, we'll try to build the best classifier for the "Do you expect your child will go into a ?" item. Using the following variables (and any variables that you think might be relevant in the codebook) and **data for just Mexico**, try and build the best classifier. Do the following steps:

1. Split the data into a training and a testing data set. Rather than using a 66/33 split, try a 50/50 or a 75/25 split.
2. Fit a decision tree **or** random forest

- Prune your model and plot your model (if using decision trees)
- Determine the ideal number of trees (if using random forests)

3. Fit a support vector machine

- Consider different kernels (e.g., linear and radial)
- Visually inspect your model by plotting it against a few features. Create a few different plots.
- Tune the parameters.
  - How many support vectors do you have?
  - Did you notice much difference in the error rates?
  - Does your model have a high tolerance?
- (OPTIONAL): When fitting the support vector classifier, you could try and fit it using Apache Spark
  - If you do this, use the `ml_binary_classification_evaluator` function to calculate AUC.

4. Run a logistic regression

- Examine the coefficients table

5. Evaluate the fit of your models using the `eval_classifier` function on the testing data.

- Which model(s) fits the best? Can you improve it?

6. Record your accuracy, sensitivity, and specificity for all the models (decision tree or random forest and SVM) to share.

The following table contains the list of variables you could consider (this were introduced earlier):

Label	Description
DISCLISCI	Disciplinary climate in science classes (WLE)
TEACHSUP	Teacher support in a science classes of students choice (WLE)
IBTEACH	Inquiry-based science teaching an learning practices (WLE)
TDTEACH	Teacher-directed science instruction (WLE)
ENVAWARE	Environmental Awareness (WLE)
JOYSCIE	Enjoyment of science (WLE)
INTBRSCI	Interest in broad science topics (WLE)
INSTSCIE	Instrumental motivation (WLE)
SCIEEFF	Science self-efficacy (WLE)
EPIST	Epistemological beliefs (WLE)
SCIEACT	Index science activities (WLE)
BSMJ	Student's expected occupational status (SEI)
MISCED	Mother's Education (ISCED)
FISCED	Father's Education (ISCED)
OUTHOURS	Out-of-School Study Time per week (Sum)
SMINS	Learning time (minutes per week) -
TMINS	Learning time (minutes per week) - in total
BELONG	Subjective well-being: Sense of Belonging to School (WLE)
ANXTEST	Personality: Test Anxiety (WLE)
MOTIVAT	Student Attitudes, Preferences and Self-related beliefs: Achieving motivation (WLE)
COOPERATE	Collaboration and teamwork dispositions: Enjoy cooperation (WLE)
PERFEED	Perceived Feedback (WLE)
unfairteacher	Teacher Fairness (Sum)
HEDRES	Home educational resources (WLE)
HOMEPOS	Home possessions (WLE)
ICTRES	ICT Resources (WLE)
WEALTH	Family wealth (WLE)
ESCS	Index of economic, social and cultural status (WLE)
math	Students' math scores
reading	Students' reading scores

Chapter 9

Unsupervised machine learning

9.1 Clustering

Clustering is a broad set of techniques for finding subgroups of observations within a data set. When we cluster observations, we want observations in the same group to be similar and observations in different groups to be dissimilar. Because there isn't a response variable, this is an unsupervised method, which implies that it seeks to find relationships between the observations without being trained by a response variable. Clustering allows us to identify which observations are alike, and potentially categorize them therein.

9.2 Distance Measures

The classification of observations into groups requires some methods for computing the distance or the (dis)similarity between each pair of observations. The result of this computation is known as a dissimilarity or distance matrix. There are many methods to calculate this distance information; the choice of distance measures is a critical step in clustering. It defines how the similarity of two elements (x, y) is calculated and it will influence the shape of the clusters.

The choice of distance measures is a critical step in clustering. It defines how the similarity of two elements (x, y) is calculated and it will influence the shape of the clusters. The classical methods for distance measures are Euclidean and Manhattan distances, which are defined as follow:

Euclidean distance:

$d_{euc}(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$

Manhattan distance:

$d_{man}(x,y) = \sum_{i=1}^{n} |(x_i - y_i)|$

where x and y are two vectors of length n.

The choice of distance measures is very important, as it has a strong influence on the clustering results. For most common clustering software, the default distance measure is the Euclidean distance. However, depending on the type of the data and the research questions, other dissimilarity measures might be preferred and you should be aware of the options.

9.3 K-means clustering

K-means clustering is the most commonly used unsupervised machine learning algorithm for partitioning a given data set into a set of k groups (i.e., k clusters), where k represents the number of groups pre-specified by the researcher. It classifies objects in multiple groups (i.e., clusters), such that objects within the same cluster are as similar as possible (i.e., high intra-class similarity), whereas objects from different clusters are as dissimilar as possible (i.e., low inter-class similarity). In K-means clustering, each cluster is represented by its center (i.e, centroid) which corresponds to the mean of points assigned to the cluster.

There are several k-means algorithms available. The standard algorithm is the Hartigan-Wong algorithm (1979), which defines the total within-cluster variation as the sum of squared distances Euclidean distances between items and the corresponding centroid:

$W(C_k) = \sum_{x_i \in C_k} (x_i - \mu_k)^2$

where:

X_i *is a data point belonging to the cluster* C_k μ_k is the mean value of the points assigned to the cluster C_k

Each observation is assigned to a given cluster such that the sum of squares (SS) distance of the observation to their assigned cluster centers is minimized.

We can define the total within-cluster variation as follows:

$SS_{total.within} = \sum_{k=1}^{k} W(C_k) = \sum_{k=1}^{k} \sum_{x_i \in C_k} (x_i - \mu_k)^2$

The total *within-cluster* sum of square measures the compactness (i.e., goodness) of the clustering and we want it to be as small as possible.

Generally, K-means algorithm can be used as follows:

1. Specify the number of clusters (K) to be created.
2. Select randomly k objects from the data set as the initial cluster centers or means
3. Assign each observation to their closest centroid, based on the Euclidean distance between the object and the centroid
4. For each of the k clusters, update the cluster centroid by calculating the new mean values of all the data points in the cluster. The centroid of a K^{th} cluster is a vector of length p containing the means of all variables for the observations in the k^{th} cluster; p is the number of variables.
5. Iteratively minimize the total within sum of square (see Eq. 4) by iterating steps 3 and 4 until the cluster assignments stop changing or the maximum number of iterations is reached.

9.4 K-means clustering in R

To be completed later…

Check out the following website for some examples in R: https://uc-r.github.io/kmeans_clustering

Chapter 10

Summary

10.1 Topics covered

10.1.1 Exploratory data analysis

1. Data Wrangling
   - Subsetting, creating variables, reshaping, and summarizing
   - `data.table`
   - `dplyr`
   - `sparklyr` and Apache Spark
2. Data Visualization
   - Static visualizations
   - `ggplot2`
   - `ggplot2` add-ons `GGally`, `ggExtra`, and `ggalluvial`
   - `cowplot` as an additional `ggplot2` theme
   - Interactive visualizations
   - `plotly`

10.1.2 Supervised learning

3. Decision trees
   - Classification and regression trees
   - `rpart`
   - `rpart.plot`
4. Random forests
   - Random forests for classification and regression
   - `randomForest`
5. Model building and evaluation
   - `modelr`
   - `caret`
6. Support Vector Machines
   - Maximal margin classifers
   - Support vector classifiers
   - Support vector machines with polynomial and radial kernels
   - Logistic regression
   - Tuning and evaluating the models
   - `e1071`

- `sparklyr`

10.2 Methods we didn't cover

1. Regression
   - Penalized regression
     - ridge, lasso, elastic net
     - `glmnet`
   - Principal components and partial least squares (a supervised version of PC) regression
     - `pls`
   - Non-linear regression
     - Polynomials, splines (smoothing splines), generalized additive models
     - `splines`
     - `gam`
2. K-nearest-neighbors (KNN)
   - `caret`
   - `class`
3. Unsupervised learning
   - K-means clustering
   - Hierarchical clustering
   - `cluster`
   - `factoextra`