Class 2 - CS Fundamentals

Most important skill - knowing what to Google and how to work from other people's code!

"Does anyone ever get good at R or do they just get good at googling how to do things in R" https://twitter.com/mousquemere/status/1125522375141883907

StackOverflow will be your new best friend!

Something I saw on Twitter and now use daily - RStudio Projects! We'll set one up now. Here are some instructions - https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects

Functions

"In programming, a named section of a program that performs a specific task. In this sense, a function is a type of procedure or routine."

Functions usually return something. I use functions when I find myself running the same piece of code multiple times, often with different data. I then put that code into a function and pass it the data I want it to use. For example, if I want to make the same set of plots for two different products, I can stick my code into a function and just pass it the name of the product.

```
## super simple example
Add2ToIt <- function(number){
  return(number + 2)
}

Add2ToIt(7)
```

```
## [1] 9
```

```
## a function that returns the difference between two numbers we give it
PrintDifference <- function(num1, num2){
  diff <- num2 - num1
  return(paste("The difference between those two numbers is ", diff))
}

PrintDifference(1, 8)
```

```
## [1] "The difference between those two numbers is  7"
```

```
## all together - let's make a function that reverses whatever string you give it!
```

Libraries (aka packages)

"Most programming languages come with a prewritten set of functions that are kept in a library."

Things you can do in R without loading additional libraries are called the base functions. Loading a library just means having access to its functions. The first time you use it, you have to install it (just run `install.packages("NAME")`). We will go over one of my favorite libraries, `dplyr` (pronounced duh-PLY-er) and `ggplot2`.

```
### have to install libraries once before using them
#install.packages("dplyr")
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

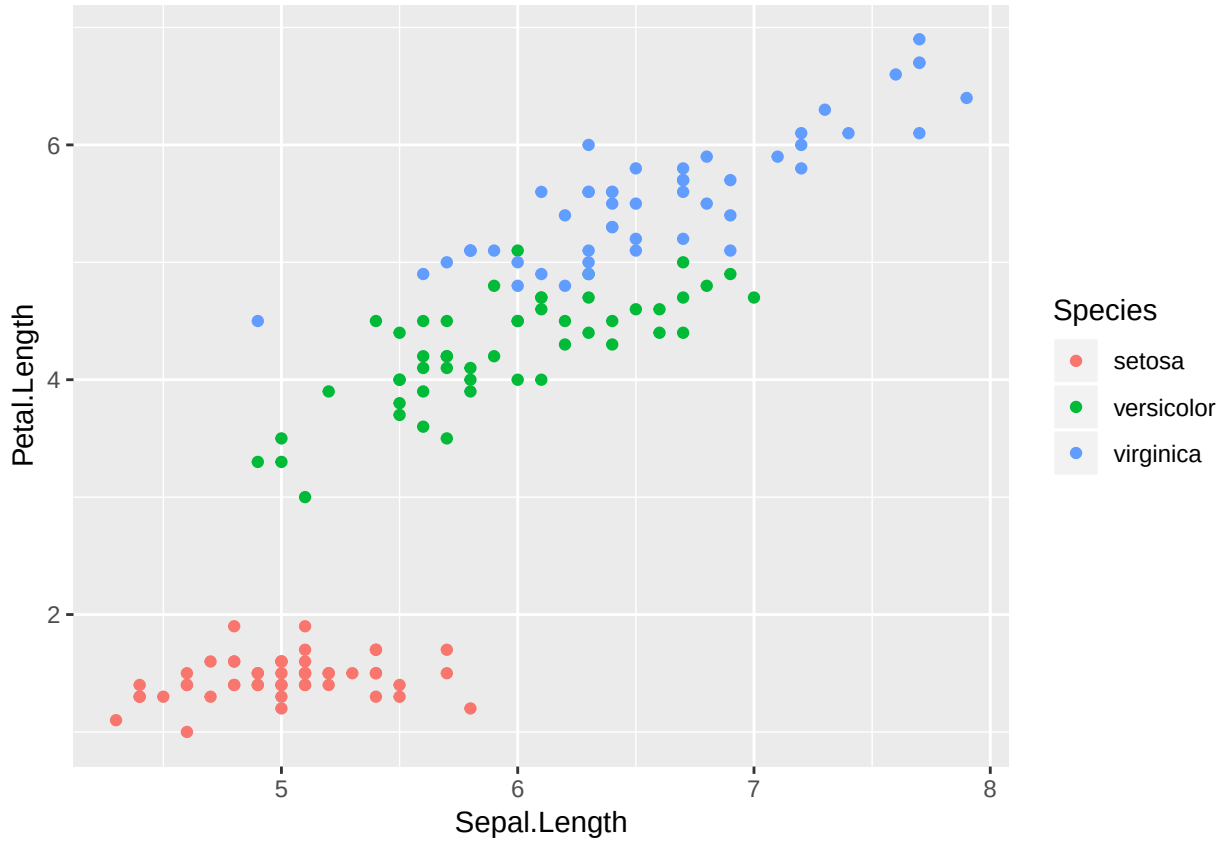
```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

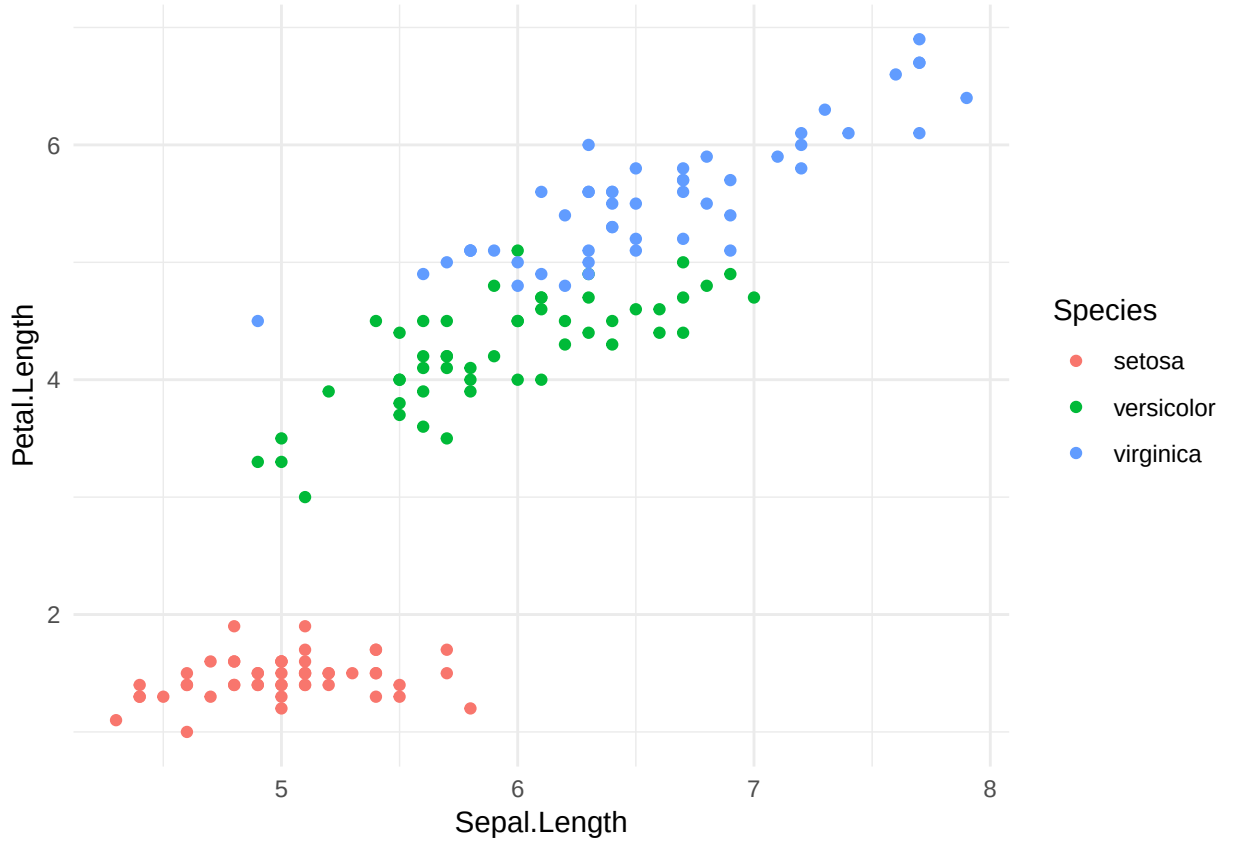
```
data <- iris

data %>%
  group_by(Species) %>%
  summarise(sepal_mean_yeehaw = mean(Sepal.Length))
```

```
## # A tibble: 3 x 2
##   Species    sepal_mean_yeehaw
##   <fct>                  <dbl>
## 1 setosa                  5.01
## 2 versicolor              5.94
## 3 virginica               6.59
```

```
small_petal <- data %>%
  filter(Petal.Length <= mean(Petal.Length))


##--- Let's try plotting things! ---##
#install.packages("ggplot2")
library(ggplot2)

## what do you see?
ggplot(data, aes(x = Sepal.Length, y = Petal.Length, color = Species)) +
  geom_point()
```




```
## what if we wanted to present this? how could we make it prettier?
ggplot(data, aes(x = Sepal.Length, y = Petal.Length, color = Species)) +
  geom_point() +
  theme_minimal()
```




```
## All together - what else could we change? how do we do that? let's google!
```

Conditionals

Making new columns is essential! This is where you take your questions and change the data around to help you answer them. For example, I might take number of past tours and make a new column called "AnyPastTours" with just 1 (yes) or 0 (no).

Making new column example taken from http://www.talkstats.com/threads/adding-a-new-column-in-r-data-frame-with-valu 30924

```
## what is we want the species names to be capitalized? how can we replace them all at once?
data$Species <- as.character(data$Species) # what happens if we don't run this first?
data$Species[data$Species == "setosa"] <- "Setosa"

## what if we want to make a new column? and set it conditionally? let's use more if/then statements!
summary(data$Sepal.Length)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.300   5.100   5.800   5.843   6.400   7.900
```

```
summary(data$Sepal.Width)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   2.800   3.000   3.057   3.300   4.400
```

```r
###------------- Two ways to conditionally set a new column! Use the one that you like best ----------
### way 1 - if/else statements
data$SepalSize <- ifelse(data$Sepal.Length < 5.8 & data$Sepal.Width < 3, "Small",
                  ifelse(data$Sepal.Length > 5.8 & data$Sepal.Width > 3, "Big",
                    "Average"))

## way 2 - conditionally setting of rows
data$SepalSize <- "Average"
data$SepalSize[data$Sepal.Length < 5.8 & data$Sepal.Width < 3] <- "Small"
data$SepalSize[data$Sepal.Length > 5.8 & data$Sepal.Width > 3] <- "Big"

## now - getting averages using our new column!
data %>%
  group_by(SepalSize) %>%
  summarise_if(is.numeric, mean)
```

```
## # A tibble: 3 x 5
##   SepalSize Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <chr>            <dbl>       <dbl>        <dbl>       <dbl>
## 1 Average           5.74        3.11         3.37        1.01
## 2 Big               6.70        3.28         5.38        2.00
## 3 Small             5.34        2.56         3.73        1.19
```

```r
data %>%
  group_by(Species) %>%
  count(SepalSize) # not that useful :(
```

```
## # A tibble: 8 x 3
## # Groups:   Species [3]
##   Species    SepalSize     n
##   <chr>      <chr>     <int>
## 1 Setosa     Average      48
## 2 Setosa     Small         2
## 3 versicolor Average      25
## 4 versicolor Big           8
## 5 versicolor Small        17
## 6 virginica  Average      30
## 7 virginica  Big          17
## 8 virginica  Small         3
```

```r
data %>%
  group_by(Species, SepalSize) %>%
  summarise (n = n()) %>%
  mutate(freq = n / sum(n)) # more useful :)
```

```
## # A tibble: 8 x 4
## # Groups:   Species [3]
##   Species    SepalSize     n  freq
##   <chr>      <chr>     <int> <dbl>
## 1 Setosa     Average      48  0.96
## 2 Setosa     Small         2  0.04
## 3 versicolor Average      25  0.5
```

```
## 4 versicolor Big           8  0.16
## 5 versicolor Small        17  0.34
## 6 virginica  Average      30  0.6
## 7 virginica  Big          17  0.34
## 8 virginica  Small         3  0.06
```

```
## Advanced - this is how we would also impute missing data, which is good for regressions
```

Other fundamentals that aren't as essential (optional)

These are things in any data scientist's toolkit. You may not need to use all of them for a project, but they're important to have at the ready to make your life easier!

Loops

```
## super simple example
for(i in 1:5){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```
## Optional - make a loop that sqaures a list of numbers and stores the results in a new list
```

If/Else statements

Usually nested within a loop, so let's do that here

```
my_list <- c(-1, 50, -5, 10, 12)

## for each element in the list, we want to print it if its positive, otherwise print "negative"
for(i in 1:length(my_list)){
  if(my_list[i] > 0) {
    print(my_list[i])
  }

  else {
    print("negative")
  }
}
```

```
## [1] "negative"
## [1] 50
## [1] "negative"
## [1] 10
## [1] 12
```

For the remaining classes - what is tidy data?

You'll need to bring your own data to the next class. It needs to be "tidy" - what does this mean?

- Each "observation" has its own row (e.g. each row in `iris` is a flower)
- Each variable has its own column

If your data set does not look like this, manipulate it in Excel until it does. You may have to combine columns together or merge duplictous rows. If you have questions on if your data is tidy or not, ask me!

Here is a handy paper on how to organize your Excel spreadsheets - https://www.tandfonline.com/doi/abs/10.1080/00031305.2017.1375989

Here is some further reading about what "messy" data looks like - https://ramnathv.github.io/pycon2014-r/explore/tidy.html or https://www.michaelchimenti.com/2014/08/what-is-tidy-data/

Tip & Tricks

- Variable names and column names should always be a continuous string (i.e. no spaces) and cannot start with a number.
- Here are some RStudio cheatsheets on different packages - https://www.rstudio.com/resources/cheatsheets/
- Here are some RStudio shortcuts - https://appsilon.com/r-studio-shortcuts-and-tips/
- Here's a twitter account that has nothing but RStudio tricks - https://twitter.com/rstudiotips
- We'll see more later, but coding is fun and so are the people that make packages, like ones that set color palettes for you based on Game of Thrones (https://cran.r-project.org/web/packages/gameofthrones/gameofthrones.pdf) or Harry Potter (https://cran.r-project.org/web/packages/harrypotter/harrypotter.pdf)
- For those with SQL access - you can connect directly from R! Here's what you can do:
  - install the `odbc` library and load it
  - run this `con <- dbConnect(odbc::odbc(), .connection_string = "Driver={ODBC Driver 13 for SQL Server};driver={SQL Server};server=toursbi\\bi;database=martSales;trusted_connectio`
  - Go up to "Connections" in the Enviroment pane and press "Connect"
  - Make a new code chunk with SQL (press the drop-down arrow) make the header look like `{sql connection=con, output.var = "NAME_OF_NEW_TABLE"}`
  - You can copy and paste any SQL script here. Just ensure you remove all cases of `(NOLOCK)` and have these at the beginning (as two separate lines):
    * `SET ANSI_WARNINGS OFF`
    * `SET NOCOUNT ON`