HAD 5744: Quantitative Methods I
Dr. Hoagland

Assignment 0: Introduction to R

This "assignment" is an opportunity to try the R statistical package and to start to learn some of its behaviors and options. It will not be graded and does not constitute part of the official workload of HAD 5744. Hopefully, however, it helps you clarify where your coding skills lie and what strengths you are bringing into the course!

Feel free to reach out to Dr. Hoagland (alexander.hoagland@utoronto.ca) with any questions about the assignment. Also, feel free to reach out to each other on Quercus or during orientation to talk about the assignment – we have a lot of group work in this course to encourage the diffusion of helpful ideas and skills.

The first page includes information about (a) setting up R and RStudio and (b) integrating GitHub Copilot into your workflow. I highly recommend that each student have this **ready to go before the first lecture.**

The rest of the "assignment" follows on additional pages.

HAD 5744: Quantitative Methods I
Dr. Hoagland

Setting up R

You do this in two parts: first, by installing R – this is the software that your computer will run under the hood every time you run a command.

Downloading and Installing R

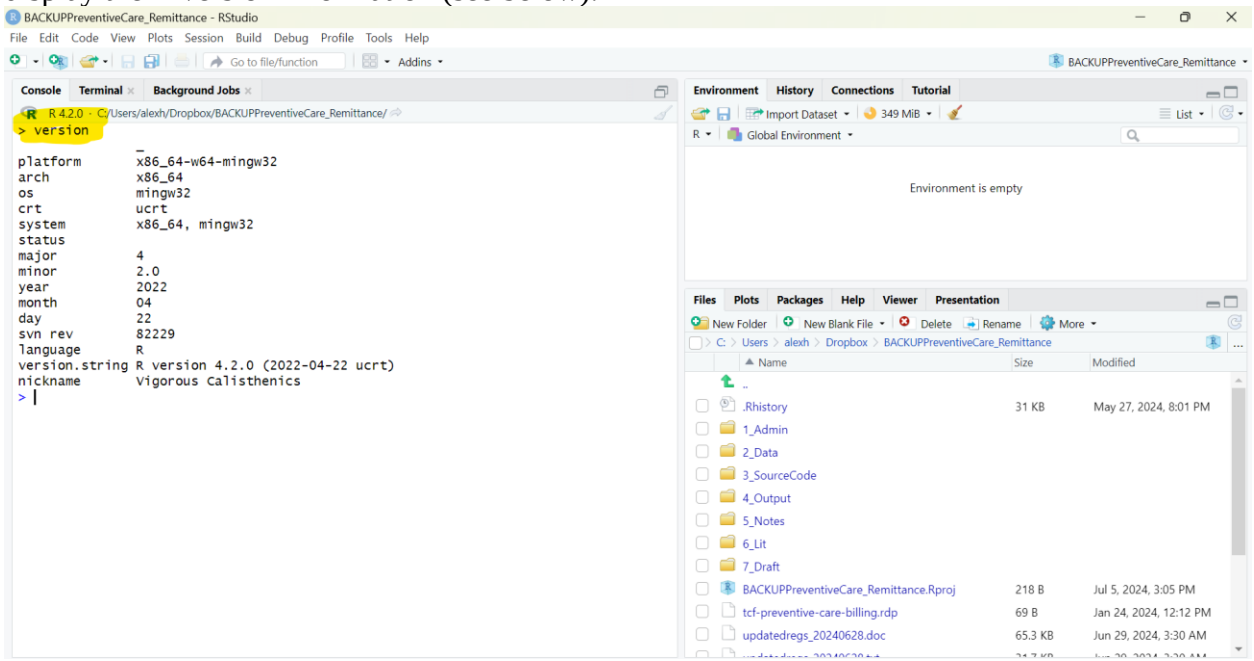
1. Visit the R Project's official website: https://cran.r-project.org/
2. Click "Download R" for your operating system
3. Follow the installation instructions. During this stage, you may be asked to select something called a "CRAN mirror." It's not important what this is, but you should pick one that looks geographically proximate to where you currently are. For example, if you are in Toronto, I recommend picking the MBNI University of Michigan mirror (https://repo.miserver.it.umich.edu/cran/). Doing so will make your installation a few seconds faster, at least; otherwise, it shouldn't matter too much.
4. You do not need to open R once it is installed (see below).

Once this is done, you install RStudio; this is the environment that you use to look at data, write and execute code, and create figures/slides/papers. It will run R for you in the background.

Downloading and Installing RStudio

1. Go to the RStudio official website: https://posit.co/download/rstudio-desktop/. Note that it has a big button with a #1 on it that says "1: Install R." You've already done that!
2. Use the big button "2: Install RStudio" to select the appropriate RStudio version for your operating system.
3. Follow the installation instructions.

Once you are done, open RStudio. On first launch, RStudio will automatically detect your R installation, so no other work is needed there. If you want, you can verify that R is properly installed in RStudio by typing `version` in the Console pane and pressing Enter. This should display the R version information (see below).



You are now ready to use R and RStudio for your statistical and data analysis tasks!

HAD 5744: Quantitative Methods I
Dr. Hoagland

Setting up and Using GitHub Copilot

GitHub Copilot is a fantastic AI resource that can help you separate out the mechanics of what you're doing (the writing the code part) from the actual research (the coming up with and evaluating models part). GitHub Copilot can integrate seamlessly into RStudio and is free to use for students!

Installation Prerequisites:

1. Make sure you have a (free!) GitHub account. You can sign up here: https://GitHub.com/join.
2. Apply for the GitHub Student Developer Pack here: https://education.GitHub.com/pack. You have to provide a picture of your student ID or some other proof you're a student, and approval can take a few days, so the sooner the better!

Step 1: Accessing GitHub Copilot. Visit the GitHub Copilot page (once signed in): https://GitHub.com/features/copilot. Follow the instructions to enable Copilot for your account.

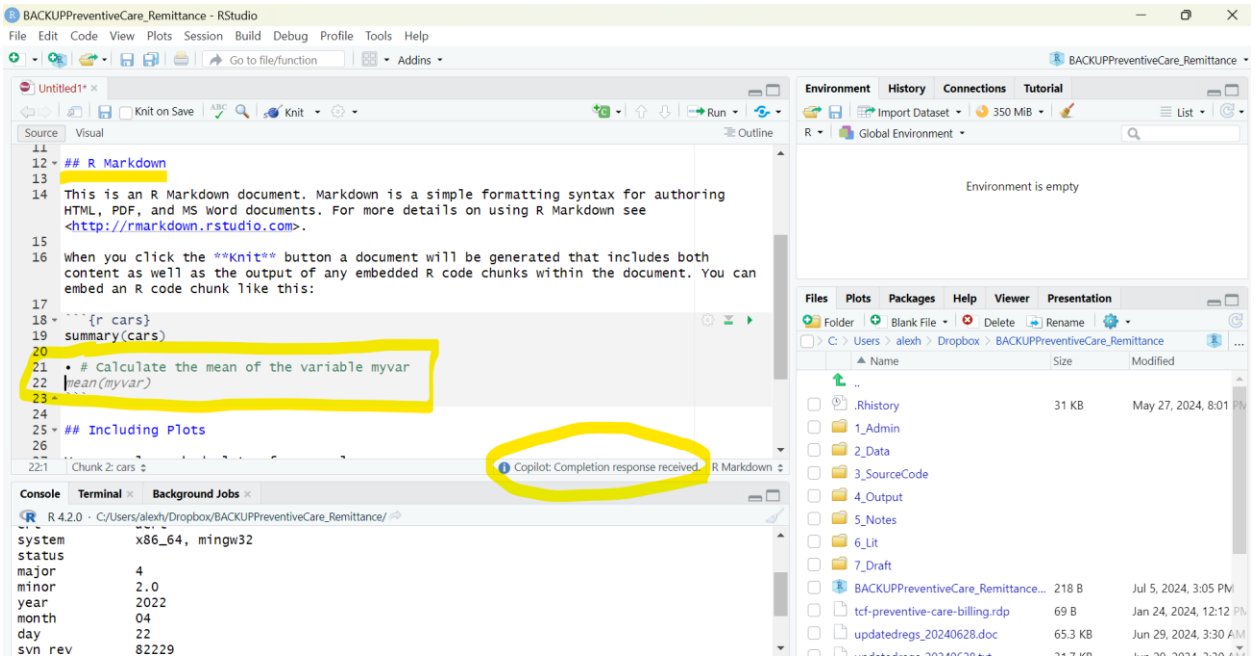
Step 2: Integrating GitHub with RStudio

1. After completing the above, open RStudio and go to `Tools` > `Global Options`
2. At the very bottom left, click the "Copilot" button with the GitHub cat icon
3. Check the "Enable GitHub Copilot" button. You will need to sign into your GitHub account again and give permission for communication between RStudio and GitHub.
4. That's it!

Using GitHub Copilot

To use Copilot, you need to be writing your code in a code file, not just in the console. There are two types of files: .R files for only R code, and .Rmd files (called R Markdown) to include code and text. We like to use the latter in this class.

- Create a new .Rmd file (`File` > `New File` > `R Markdown`, and save it (preferably in a separate folder for this class) as "Assignment0.Rmd."
- In an .Rmd file, the white areas are for text (just like this Word document). The gray areas are called "code chunks" and are the pieces of code that will be evaluated once you run ("compile") the file.
- A weird, persnickety note about code chunks: they all have to be named, and each name must be unique. We'll discuss this more in class, but if your file isn't compiling, there's a good chance this is why!
- Go to one of these code chunks and start typing the following (including the comment hashtag):
- `# Calculate the mean of the variable myvar`
- Then hit enter, and you should see something close to the following:



- If you hit the "Tab" key, the suggested code will be added to your file. Hooray!

HAD 5744: Quantitative Methods I
Dr. Hoagland

"Assignment" Questions

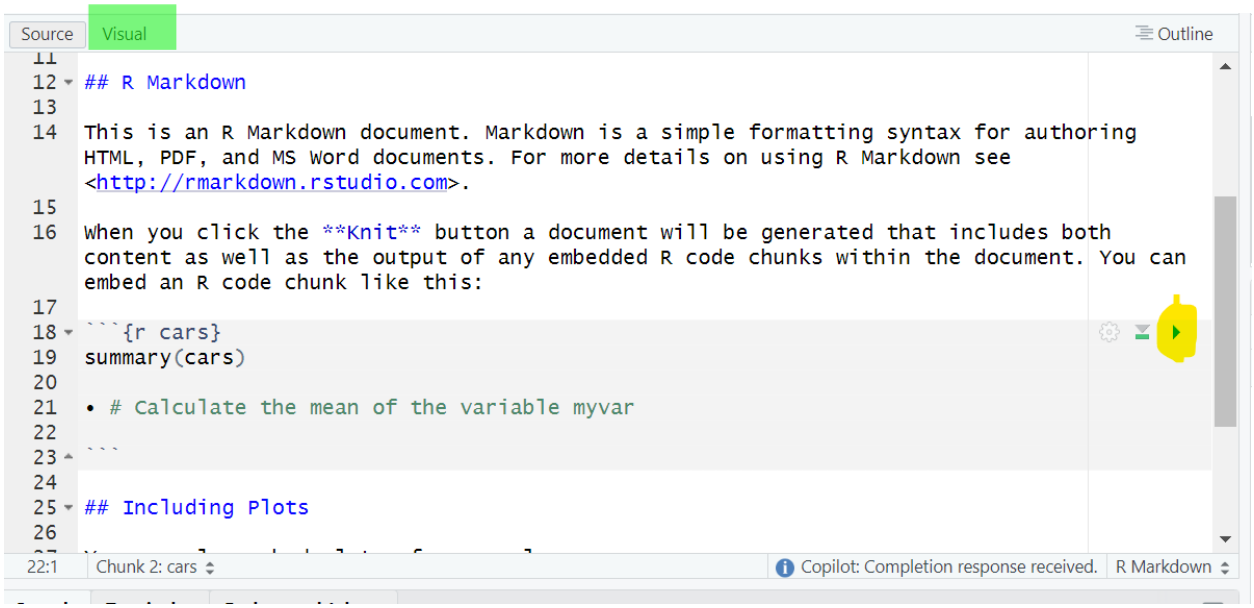
Text like this will be general instructions. Complete this in your Assignment0.Rmd file, and experiment with integrating code (in the gray chunks) and text (in the white boxes).

```
Text like this will be my commands to R, the R prompt is a "greater than"
sign (>).

Text like this will be output from R in my examples.
```

Note: to run code that you've typed into your .Rmd file, use the green triangles to run a chunk of code. The output will come out just below the code chunk:

Throughout, we will take a "script-based approach" to coding, where we write code directly into .Rmd files or the console to produce output. There are some "menu-based" options you can use for things like importing/exporting data and figures. If you're interested, you can read more here (and on other pages in this guide):



I recommend using the "visual" option in your .Rmd files (highlighted above in green). It makes it much easier to see what's plan text and what's code, as well as to format your text in a way that you might be used to (e.g., like a Word doc).

Variable Assignment.

There are 4 panels in RStudio, including:

1. The Console (bottom left; this is like your scratch paper; we have already played around with this)
2. The code file (top left; this is where we will create the assignment)
3. The environment, in the top right
4. Help files and outputs, in the bottom right

Let's look at the Environment tab. This is where you store objects and load data. To define an object, you use the left arrow (< and then a hypen -) like so:

```
> n <- 15
```

You can type the name of any object to look at that object.

```
> n
[1] 15
```

Variables must start with a letter, but may also contain numbers and periods. R is case sensitive.

```
> N <- 26.42
> N
[1] 26.42
> n
[1] 15
```

Other data types are available. You do not need to declare these; they will be assigned automatically.

```
> name <- "Mike"  # Character data
> name
[1] "Mike"

> q1 <- TRUE            # Logical data
> q1
[1] TRUE

> q2 <- F
> q2
[1] FALSE
```

Now for the bottom right panel. This is where your figures will show up once you make them, but you can also access help files here. Just use a question mark followed by the command you want help with:

```
> ?ls
```

Simple calculation

R may be used for simple calculation, using the standard arithmetic symbols +, -, *, /, as well as parentheses and ^ (exponentiation).

```
> a <- 12+14
> a
[1] 26
> 3*5
[1] 15
> (20-4)/2
[1] 8
> 7^2
```

```
[1] 49
```

Standard mathematical functions are available.

```
> exp(2)
[1] 7.389056
> log(10)   # Natural log
[1] 2.302585
> log10(10) # Base 10
[1] 1
> log2(64)  # Base 2
[1] 6
> pi
[1] 3.141593
> cos(pi)
[1] -1
> sqrt(100)
[1] 10
```

Problem 1: Use R as a calculator to compute the following values. Present and discuss your answers, as needed, in the text before or after your code chunk. (Do this for every problem from now on.)

(a) 27(38-17)

(b) $\ln(14^7)$

(c) $\sqrt{\frac{436}{12}}$

2. Vectors

Vectors may be created using the c command, separating your elements with commas.

```
> a <- c(1, 7, 32, 16)
> a
[1]  1  7 32 16
```

Sequences of integers may be created using a colon (:).

```
> b <- 1:10
> b
 [1]  1  2  3  4  5  6  7  8  9 10

> c <- 20:15
> c
[1] 20 19 18 17 16 15
```

Other regular vectors may be created using the seq (sequence) and rep (repeat) commands.

```
> d <- seq(1, 5, by=0.5)
> d
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0

> e <- seq(0, 10, length=5)
> e
[1]  0.0  2.5  5.0  7.5 10.0

> f <- rep(0, 5)
> f
[1] 0 0 0 0 0

> g <- rep(1:3, 4)
> g
 [1] 1 2 3 1 2 3 1 2 3 1 2 3

> h <- rep(4:6, 1:3)
> h
[1] 4 5 5 6 6 6
```

Random vectors can be created with a set of functions that start with r, such as rnorm (normal) or runif (uniform).

```
> x <- rnorm(5)   # Standard normal random variables
> x
[1] -1.4086632  0.3085322  0.3081487  0.2317044 -0.6424644

> y <- rnorm(7, 10, 3)  # Normal r.v.s with mu = 10, sigma = 3
> y
[1] 10.407509 13.000935  8.438786  8.892890 12.022136  9.817101  9.330355

> z <- runif(10)  # Uniform(0, 1) random variables
> z
 [1] 0.925665659 0.786650785 0.417698083 0.619715904 0.768478685 0.676038428
 [7] 0.050055548 0.727041628 0.008758944 0.956625536
```

If a vector is passed to an arithmetic calculation, it will be computed element-by-element.

```
> c(1, 2, 3) + c(4, 5, 6)
[1] 5 7 9
```

If the vectors involved are of different lengths, the shorter one will be repeated until it is the same length as the longer.

```
> c(1, 2, 3, 4) + c(10, 20)
[1] 11 22 13 24

> c(1, 2, 3) + c(10, 20)
[1] 11 22 13
Warning message:
longer object length
        is not a multiple of shorter object length in: c(1, 2, 3) + c(10, 20)
```

Basic mathematical functions will apply element-by-element.

```
> sqrt(c(100, 225, 400))
```

```
[1] 10 15 20
```

To select subsets of a vector, use square brackets ([ ]).

```
> d
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
> d[3]
[1] 2
> d[5:7]
[1] 3.0 3.5 4.0
```

A logical vector in the brackets will return the TRUE elements.

```
> d > 2.8
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> d[d > 2.8]
[1] 3.0 3.5 4.0 4.5 5.0
```

The number of elements in a vector can be found with the length command.

```
> length(d)
[1] 9
> length(d[d > 2.8])
[1] 5
```

Note: if you are ever confused about the structure of an object you're working with in R, I recommend you check out the str() function (for "structure"). This function gives output specific to the type of object you're working with, including data types, length, objects contained inside regression output, etc. You can read more about it here: https://www.r-bloggers.com/2023/08/exploring-rs-versatile-str-function-unraveling-your-data-with-ease/amp/

Problem 2: Create the following vectors in R.

a = (5, 10, 15, 20, ..., 160)

b = (87, 86, 85, ..., 56)

Use vector arithmetic to multiply these vectors and call the result d. Select subsets of d to identify the following.

(a) What are the 19th, 20th, and 21st elements of d?

(b) What are all of the elements of d which are less than 2000?

(c) How many elements of d are greater than 6000?

Simple statistics

There are a variety of mathematical and statistical summaries which can be computed from a vector.

```
> 1:4
[1] 1 2 3 4

> sum(1:4)
[1] 10
```

```
> prod(1:4)       # product
[1] 24

> max(1:10)
[1] 10
> min(1:10)
[1] 1
> range(1:10)
[1]  1 10

> X <- rnorm(10)
> X
 [1]  0.2993040 -1.1337012 -0.9095197 -0.7406619 -1.1783715  0.7052832
 [7]  0.4288495 -0.8321391  1.1202479 -0.9507774

> mean(X)
[1] -0.3191486

> sort(X)
 [1] -1.1783715 -1.1337012 -0.9507774 -0.9095197 -0.8321391 -0.7406619
 [7]  0.2993040  0.4288495  0.7052832  1.1202479

> median(X)
[1] -0.7864005

> var(X)
[1] 0.739266

> sd(X)
[1] 0.8598058
```

Problem 3: Using d from problem 2, use R to compute the following statistics of d:

(a) sum
(b) median
(c) standard deviation

Matrices

Matrices can be created with the matrix command, specifying all elements (column-by-column) as well as the number of rows and number of columns.

```
> A <- matrix(1:12, nr=3, nc=4)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

You may also specify the rows (or columns) as vectors, and then combine them into a matrix using the rbind (cbind) command.

```
> a <- c(1,2,3)
> a
[1] 1 2 3
```

```
> b <- c(10, 20, 30)
> b
[1] 10 20 30
> c <- c(100, 200, 300)
> c
[1] 100 200 300
> d <- c(1000, 2000, 3000)
> d
[1] 1000 2000 3000

> B <- rbind(a, b, c, d)
> B
  [,1] [,2] [,3]
a    1    2    3
b   10   20   30
c  100  200  300
d 1000 2000 3000

> C <- cbind(a, b, c, d)
> C
     a  b   c    d
[1,] 1 10 100 1000
[2,] 2 20 200 2000
[3,] 3 30 300 3000
```

To select a subset of a matrix, use the square brackets and specify rows before the comma, and columns after.

```
> C[1:2,]
     a  b   c    d
[1,] 1 10 100 1000
[2,] 2 20 200 2000


> C[,c(1,3)]
     a   c
[1,] 1 100
[2,] 2 200
[3,] 3 300

> C[1:2,c(1,3)]
     a   c
[1,] 1 100
[2,] 2 200
```

Matrix multiplication is performed with the operator %*%. Remember that order matters!

```
> B%*%C
      a      b       c       d
a    14    140    1400 1.4e+04
b   140   1400   14000 1.4e+05
c  1400  14000  140000 1.4e+06
d 14000 140000 1400000 1.4e+07

> C%*%B
        [,1]    [,2]    [,3]
```

```
[1,] 1010101 2020202 3030303
[2,] 2020202 4040404 6060606
[3,] 3030303 6060606 9090909
```

You may apply a summary function to the rows or columns of a matrix using the apply function.

```
> C
     a  b   c    d
[1,] 1 10 100 1000
[2,] 2 20 200 2000
[3,] 3 30 300 3000

> sum(C)
[1] 6666

> apply(C, 1, sum)      # sums of rows
[1] 1111 2222 3333

> apply(C, 2, sum)      # sums of columns
   a    b    c    d
   6   60  600 6000
```

Problem 4: Use R to create the following two matrices and do the indicated matrix multiplication.

$$\begin{bmatrix} 7 & 9 & 12 \\ 2 & 4 & 13 \end{bmatrix} \times \begin{bmatrix} 1 & 7 & 12 & 19 \\ 2 & 8 & 13 & 20 \\ 3 & 9 & 14 & 21 \end{bmatrix}$$

What is the resulting matrix?

Data

We will not work as directly with matrices typically, but will prefer instead to work with data. This is stored in R as a "data frame" object.

You can load your data from a variety of sources, including .csv, .dta, .xlsx., and others. You will need the help of various packages to do so. This is a place where Copilot and Google can help!

Like any object, you store the data frame in your Environment tab with a name. So if you have a .xlsx file, you would write something like this:

```
> install.packages("readxl")
> library(readxl)
> mydata <- read_xlsx("mydata.xlsx")
```

The first line installs the package you want to use – you only need to run this once on your computer (and don't forget the quotation marks). Then, each time you open RStudio, you need to "call" the package (think about this like inviting the vampire into your house, but in a nice way). That's done on the second line. Then, the third line uses the command "read_xlsx" from the readxl package to load your dataset.

The data is stored as a complex object called a data frame. The data frame views rows as cases and columns as variables. All elements in a column must be the same mode, but different columns may be different modes.

```
> grades.df <- data.frame(Name, Test1, Test2)
> grades.df
   Name Test1 Test2
1   Bob    80    40
2  Bill    95    87
3 Betty    92    90
```

Summary functions applied to a data frame will be applied to each column.

```
> mean(grades.df)
    Name    Test1    Test2
      NA 89.00000 72.33333
Warning message:
argument is not numeric or logical: returning NA in: mean.default(X[[1]], ...)

> mean(grades.df[,2:3])
   Test1    Test2
89.00000 72.33333
```

Note: as similar as matrices and data frames appear, R considers them to be quite different. Many functions will work on one or the other, but not both. You can convert from one to the other using as.matrix or as.data.frame.

```
> C.df <- data.frame(a,b,c,d)
> C.df
  a  b   c    d
1 1 10 100 1000
2 2 20 200 2000
3 3 30 300 3000

> C.df%*%B
Error in C.df %*% B : requires numeric matrix/vector arguments
> as.matrix(C.df)%*%B
     [,1]    [,2]    [,3]
1 1010101 2020202 3030303
2 2020202 4040404 6060606
3 3030303 6060606 9090909

> C
     a  b   c    d
[1,] 1 10 100 1000
[2,] 2 20 200 2000
[3,] 3 30 300 3000

> mean(C)
[1] 555.5

> mean(as.data.frame(C))
   a    b    c    d
   2   20  200 2000
```

R also has some sample datasets that are easy to install and use without downloading source data files (that is, like an Excel spreadsheet). For example:

Problem 5: Install the package NHSRDatasets, which includes sample datasets from the NHS. Load the library and then load the ons_mortality data into your environment. What are the means and standard deviations for the numeric variables in this dataset? What are the levels of the categorical variables? Remember you can use ?ons_mortality for a data dictionary.

Tidyverse for data manipulation

The key way we manipulate data is through the tidyverse package. This is the main thing we will be learning about and using in conjunction with our quantitative methods, so we will cover this a lot in class.

```
> install.packages("tidyverse")
> library(tidyverse)
```

You will essentially want to load this every time you start a new R project or .Rmd file. Tidyverse has an especially handy set of commands to help you manipulate data to create tables, figures, etc. We will cover a lot of this in class, but as an example, let's work through how to use tidyverse to summarize data by groups.

Tidyverse uses what's called a pipe operator to flow from the starting point (your data) to an end point (your output). The pipe operator looks like this: %>%. If we want to get total count by category for example, we use the pipe operator to go from (1) the ungrouped data to (2) grouped data and then to (3) the summarized dataset, like this:

```
> mydata %>% group_by(category_1) %>% summarize(total_counts = sum(counts))
```

You can store this as its own object (say, "summarized_data") just as you would assign any object in R (with your left arrow).

Problem 6: Use the tidyverse notation to take your NHSRDataset::ons_mortality data to find the average number of deaths by week_no. You may find it helpful to add a step in here that uses the command "filter(category_1 == "Total deaths")" in order to ignore some extraneous information. Can you report this in a nice table?

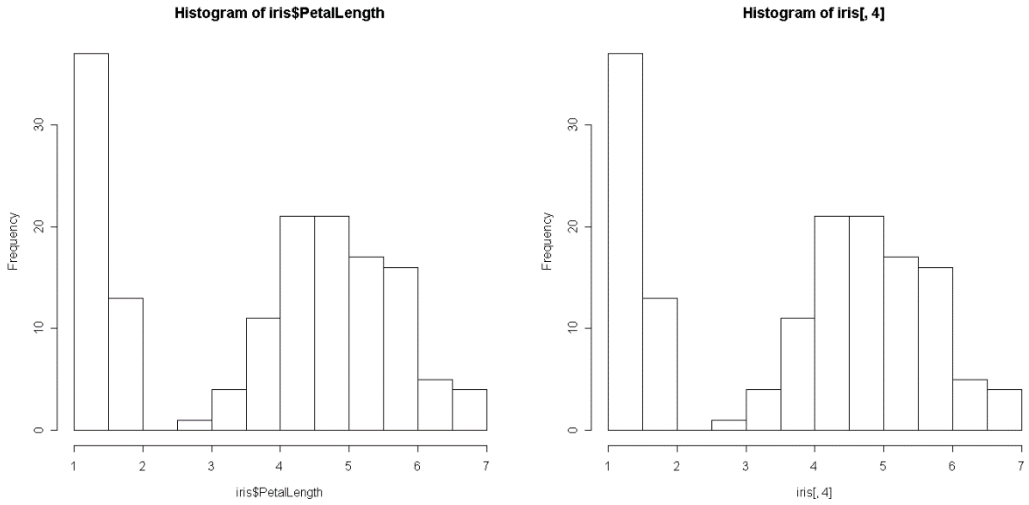
Graphics

R has two main ways to produce figures: base R, which is easy but ugly and not very customizable; and ggplot, which is a subset of the tidyverse. Ggplot has a lot of flexibility and so we will use it a lot more in our contexts.

Base R graphics look like this:

```
> hist(iris$PetalLength)
>
```

```
> hist(iris[,4])# alternative specification
```



For ggplot, there are infinite possibilities. Here is a quick crash course (we will cover this many more times in class!):

The basic structure of a `ggplot` call is:

```R
ggplot(data = <DATA>, mapping = aes(<MAPPINGS>)) + <GEOM_FUNCTION>()
```

Where we have:
- `data`: The data frame containing your data.
- `mapping`: Aesthetic mappings describing how variables in the data are mapped to visual properties (axes, colors, sizes, etc.).
- `aes()`: Function to specify aesthetic mappings.
- `<GEOM_FUNCTION>`: Functions that create the actual plot layers (e.g., `geom_point()` for scatter plots, `geom_line()` for line plots). Common functions include:
    - `geom_point()`: Scatter plots
    - `geom_line()`: Line plots
    - `geom_bar()`: Bar charts
    - `geom_histogram()`: Histograms
    - `geom_boxplot()`: Boxplots
    - `geom_smooth()`: Smoothed conditional means

A basic plot would look like this:

```R
# Load example data
data(mpg)

# Create a scatter plot
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point()
```

This code creates a scatter plot of engine displacement (`displ`) vs. highway miles per gallon (`hwy`) from the `mpg` dataset. Look for it in the bottom right corner of your RStudio page, or just below your code chunk in the top left depending on how you run your code.

You can add more layers:

```R
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(method = "lm")
```

This adds a linear regression line (`geom_smooth(method = "lm")`) to the scatter plot.
Or additional aesthetics like colors:

```R
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = class)) +
  geom_point()
```

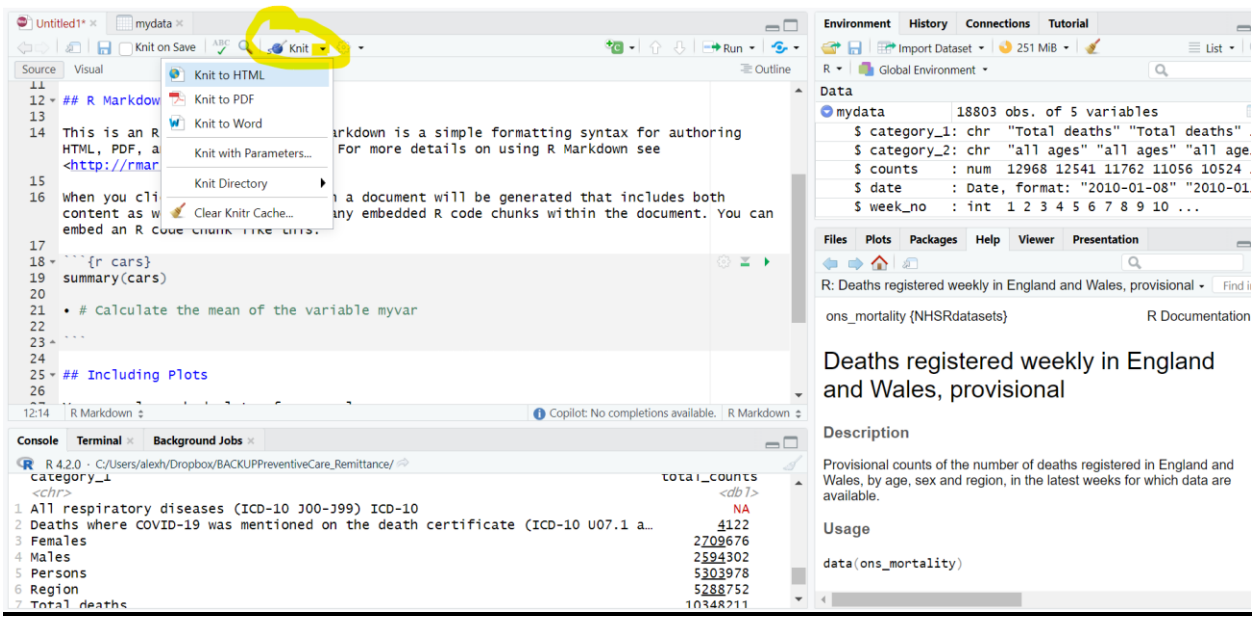
This colors the points based on the `class` variable.

You'll get lots more practice at this! But for now:

Problem 7: Using your loaded data, make a scatterplot showing the relationship between week_no and (the average number of) deaths where COVID-19 was mentioned on the death certificate

Compiling your file

Once you have written up all the code chunks (in the gray areas of your .Rmd file) and any text that you want to describe those outputs (in the white areas of the file), you are ready to combine them into a single document. To do this, you use the "Knit" button near the top of RStudio:



Most of the time, it will be most convenient to click the arrow next to the ball of yarn and make sure that you are selecting "Knit to Word." Once you click this, your .Rmd file will **run all code in your document** and then produce a Word file with the outputs of your code and the text you wrote to describe it. Pretty cool!

Note: in order to knit, your R software will start entirely from scratch. This means that **every line of code you needed to complete the analysis must be in your .Rmd file**. Frequently, we switch to the console to play around or test/debug something we're trying to do. If you do this, you have to make sure your code ends up back in the .Rmd file or your file will not knit.

Problem 8: Compile your .Rmd file with everything you've done so far into a Word document. If you want to, send it to me! There are some additional problems below that you can incorporate if you want to try them out.

Additional problems if you want to try them:

1. **Creating your own data frame.** Create a data frame `df` with two columns: `age` (a sequence from 21 to 30) and `income` (randomly generated numbers between 40000 and 60000).
    a. Display the summary statistics for the data frame `df`.
    b. Calculate the correlation between `age` and `income`.
2. **Importing and Exporting Data.** Import a CSV file containing data on housing prices into R (look around for your favorite online or ask Chat GPT).
    a. Display the first 6 rows of the imported data.
    b. Save the data frame `df` from Problem 2 as a CSV file named `df_output.csv`.
3. **Descriptive Statistics and Visualizations** Using the `housing_prices` data (this is example data included in R):
    a. Calculate the mean, median, and standard deviation of the `price` column.

b. Create a histogram of the `price` column using ggplot.
c. Create a boxplot of the `price` column by a categorical variable (e.g., `neighborhood`) using ggplot.