Section 1
Introduction to R

1.1 – Getting Started With R and RStudio

General introduction

R is a software package for statistical computing and data analysis. It is also a programming language, based on the now ancient S language. What makes R easier to use is that it is an interpreted language, and not a compiled language like C++. This is important for interactive data analysis, as you do not have to compile a fully functional "program" in one go, and can run small lines of code at a time.

R is freely distributed software (www.r-project.org) with contributions from developers from around the world. It is one of the main software for statistical computing. We say that R is also an "environment" because it contains a lot of pre-programmed functions (packages) that the user can call and use for various analysis.

Getting started

As mentioned previously, you can download the software free from www.r-project.org. This will give you the basic R package with an interactive console. It is highly recommended that you download RStudio (https://posit.co/downloads/) as well, which gives you a much cleaner and easier interface to work with R.

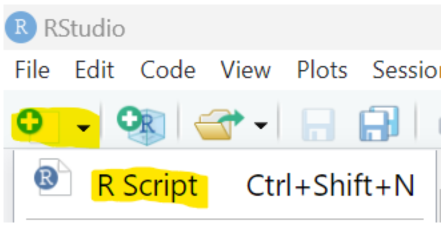
If you would like to run RStudio on a tablet or other device, you may want to try the Posit Cloud (https://posit.cloud/) service. This gives you access to RStudio remotely, which may be useful for using RStudio in class if you don't have or don't want to bring a Windows/Mac laptop to class with you. The course's lecture notes will frequently have R code integrated into the notes. You can download R script files from Canvas which have all of the R code from the lecture in one file. This allows you to quickly run the code from the lectures.

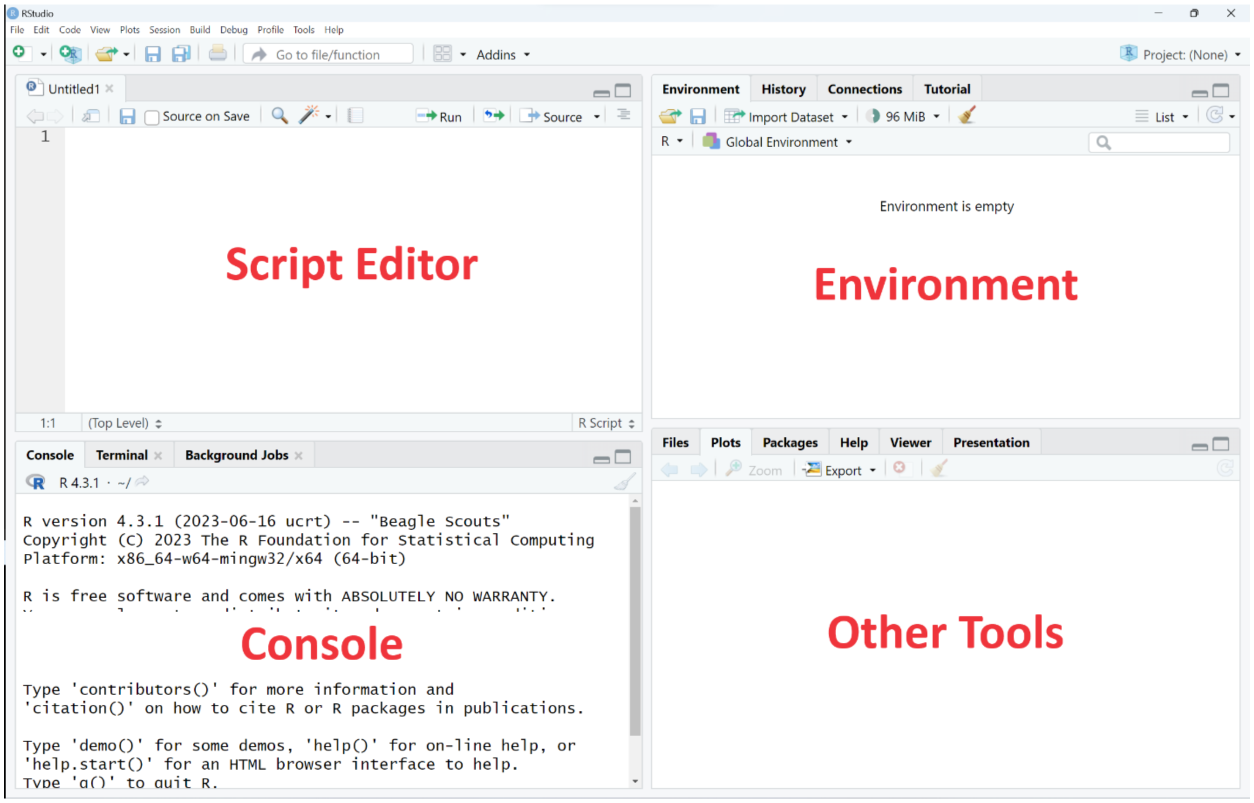
There are some minor things we will do in this class that you won't be able to do in the cloud version, so I still recommend installing the local, non-cloud version if you have a Windows/Mac computer. RStudio is also available on all campus computers if you would prefer to work on class assignments that way too!

For more instructions on downloading and setting up R/RStudio, check Canvas for an announcement with a tutorial video.

RStudio layout

When you first open RStudio, I would recommend creating an R file to type your code into. You can do this by clicking on the new document icon in the upper left corner and then choosing an "R script" as shown to the right. Once you do this, your RStudio will be divided up into four windows: your script editor, console, environment, and other utilities, identified in the image on the next page.





```
R version 4.3.1 (2023-06-16 ucrt) -- "Beagle Scouts"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.

Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

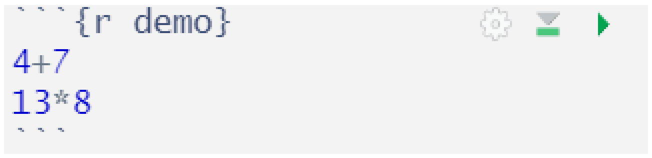
Here is an explanation of each of these sections:

- **Script editor:** This is where you can freely work on typing up your own R code, loading up someone else's script file to try their R code, or save your R code to share with others. It's a good sandbox to try out typing out R code without needing to run it right away.
- **Console**: This is where the magic happens. You run all of your R code in the console, and the text output of your code will display here. If you want to run code in the console, you typically would find the single line of code in the script editor that you want to run, and click the **"Run"** button (shortcut: Ctrl/Cmd+Enter) at the top of your script editor. You can also type in R code directly into the console, but it is difficult to save your code this way.
  - **Important**: when you run code from your script editor, it will only run one line at a time, based on where your cursor is. You can run multiple lines at once by highlighting the code you want to run and then clicking run.
- **Environment**: Any objects or variables you work with will be listed and displayed here. This helps keeps track of any variables you've defined, any data sets you've loaded into R, or any statistical models you have built.
- **Other tools**: The last window is used for a variety of things, including:
  - **Files**: You can use this window to pick a folder to view data files and load them in R.
  - **Plots**: If you run R code that creates a plot, it displays here.
  - **Packages**: Some R functionality is not included in the "base" installation of R, and so we often install extra packages for more functionality. This can be done here.
  - **Help**: R has pretty good help documentation for most built-in functions, and if you load help files, they will display here.

R scripts and R markdown
For each section in this book, there is an accompanying R script and R markdown file that both contain all of the R Code used in each section. R script files are what we already opened – they're basically just text files that let you run code line-by-line interactively in R. For writing up your own R code quickly and doing quick data analysis in R yourself, this is how I would recommend working with R.

R markdown is made for presentation of R code, output, and written text. When you make an R markdown document, it "knits" your work into a Word, PDF, or HTML document that puts all of these elements together in a nice, readable document. For this class, I'll be mostly sharing the documents for you to run chunks of code within the R markdown file for demonstration purposes. You can choose to write up your code in R markdown too if you would like to as well! Chunks of code are written with the following syntax:

~~~
```{r name of chunk}
Code goes here
```
~~~



~~~
```{r demo}
4+7
13*8
```
~~~

When you write up your code this way, you will see a few buttons appear in your R markdown file, including a play-like button. Clicking the play button will run the code in that chunk. The button to the left of that will run the code, while also running all chunks that came before. This is useful if you just opened the file and want to see how a chunk in the middle of the document works – but that chunk may be dependent on other chunks in the document above it.

We'll see more on how this works as we learn some basic R coding in the next section!

1.2 – Coding in RStudio

Try out some code!
Let's try out using R by loading up the R script for this section and running some R code!

```
4+7
13*8
hist(mtcars$mpg)
```

In RStudio, move your cursor to the first line of code that calculates 4 + 7. After you click on that line, click the run button at the top of the script editor window, and you'll see the output in the console. Your cursor will automatically move to the next line so that you can keep clicking run to see the output of each successive line of code. The final line of code doesn't produce output in the console, instead, it creates a plot of some data built into R.

Variables
An important concept in computer programming is the concept of **variable**. A variable in computer science is a name given to some storage location for later reference. In more practical terms, it is a binding between a symbol and a value. Some helpful tips for using variables:

- A variable cannot start with a digit or underscore symbol.
- Variable naming is case sensitive.

R supports many different modes of variables: integer, numeric, logical, character and complex. Unlike many other languages, you do not need to declare the type of variable you are using! R is like Python in the sense that you can be very 'loose' when using variables.

```
w = 9L                #integer
x = 9                 #numeric
y = -2.3336           #numeric
z = "stat 200"        #character

substr(z,1,4)         #gives the 1st-4th character of the
                       variable z
substr(y,1,4)         #???
```

Vectors
A vector is a collection of objects all of the same data type or mode. If need be, R will coerce your input to enforce that rule.

```
x1=c(1, 2, 3, 4, 5)   #numeric
x2=c(1:5)             #integer, similar to x1
x3=c(5,-3,FALSE)      #mixes numeric and logical
x4=c(1.2,'a')         #mixes numeric and character
x5=rnorm(10,0,1)      #10 randomly generated data values from a
                       standard normal distribution
x6=x5>1               #logical vector based on how x5 was randomly
                       generated.
```

We use brackets `[]` to access the elements of a vector. Thus, `x[1]` is the first element of x, etc...

```
x1[1]                 #first element of vector x1
x1[4]                 #fourth element of vector x1
x1[6]                 #sixth element of vector x1
```

You can also use vectors of numbers to create a vector that is a subset of the larger vector.

```
x1[1:3]               #first three elements of x1
x1[x1 > 3]            #all elements of x1 that are greater than 3
```

Vectors are useful for typing in a small amount of data. If you had the data to the right on starting salaries for full time data analysts in the USA (thousands of dollars), we could enter it into R as shown below:

Salary
72
91
80
50
90
50
55
60
100
67

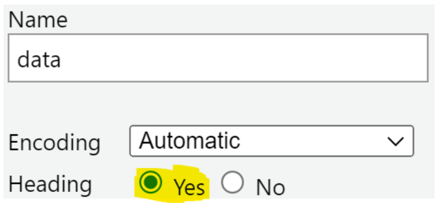
```
salary = c(72, 91, 80, 50, 90, 50, 55, 60, 100, 67)
```

We can pass vectors through many different functions in R to get useful information.

```
mean(salary)          #mean/average of salaries
sd(salary)            #standard deviation of salaries
median(salary)        #median of salaries
summary(salary)       #collection of summary statistics
                       for salaries
```

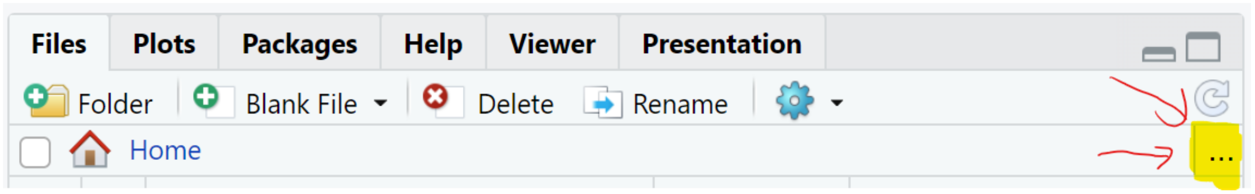
1.3 – Analyzing Data Sets in RStudio

Loading data


Name: data
Encoding: Automatic
Heading: [✓ Yes] No

While you can write code to load data into RStudio, there are also some easy ways to load data without using code. One way is to use the dialogs **Import Dataset -> From Text (base)**, which will open a dialog for you to find a data file to load into R. If you load data this way, it will by default not consider any "header" rows in the data set. If the first row of the data contains names for the data in that column, change this setting to "Yes" as shown above!

Another way we can load files in is through the "Files" tab in our lower right window. If you create a folder on your computer where you store data files, you can have this window list all of the files in that folder in this window directly. Just click the button with the three dots to change which folder is displayed here. Files from that folder won't display in the pop-up window, but once you select the folder, they will appear in the Files tab.



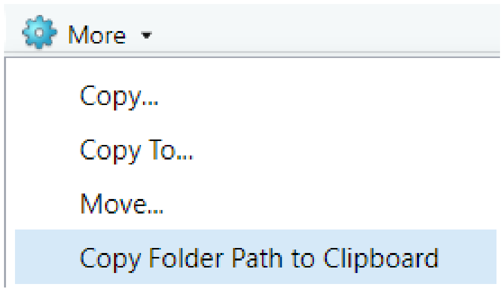
Once you do that, you can simply load up files into R by clicking on the file in this window, previewing to make sure it looks good, and then you're all set!

What kind of files can we load into R? While many types of data can be loaded in, the easiest format is called CSV or "comma separated values." This is a popular format for storing files in a simplistic, spreadsheet-like format. If you already have data in a spreadsheet (Google Docs, Excel), it is possible to save those files as CSV too. While it is possible to load Excel files into R, we typically avoid this because they are structured and encoded inefficiently for this purpose and thus take more time to load into R.

Loading data into R markdown

Note that loading data into an R markdown file takes a bit more effort. If you load in a data set to R in this way, the R markdown file won't know to look for that specific file, as it was not loaded into the markdown file itself. At the beginning of each R markdown file in line 10, I will insert the following code for you:

```
#knitr::opts_knit$set(root.dir = "")
```



This code tells the file where to look for data files on your computer. The # symbol at the beginning is the comment character, which makes this line of code not run in case you don't need to load in a data file to that R markdown file. If you do use this line, remove the comment character! Inside the quotes at the end, you need to give the file path to where your files are being stored to R. If you've used the file window to locate the folder with your files already, you can click on the "More" menu and then "Copy Folder Path to Clipboard" as shown on the right. Then you just have to paste this file path inside the quotes above. Once you've done this, as long as the data file you want to work with is in this folder, you can load in data set to the variable name `data.file` with the following code:

```
data.file = read.csv("datafile.csv")
```

You can also use the read.csv function to load data files in when using an R script too – but it's usually simpler to use the point-and-click methods shown previously.

Working with data
Let's try loading the **amazonbooks.csv** file from Canvas into RStudio now! In RStudio, clicking on a data set from the upper right of the right will open a preview window of the data. This preview window also typically opens when you load the data set. This allows us to easily see the different variables listed by their names. To access just one column of a data set, use a dollar sign ($) followed by the name of the column. We can then use this column though many different functions:

```
amazonbooks$Price #Price vector from books data set
mean(amazonbooks$Price)
sd(amazonbooks$Price)
median(amazonbooks$Price)
summary(amazonbooks$Price)
```

For the non-numerical variables in our data set, we might be interested in finding proportions. To do this, it would be helpful to identify how many items there are of each outcome. One way we can do this is with a `table` function.

```
table(amazonbooks$Binding)
```

We can see that this gives us the total number of hardcover (H) and paperback (P) books respectively. Because this table output works like a vector, we could directly calculate the proportion of paperback books using the R code below:

```
table(amazonbooks$Binding)[2]/length(amazonbooks$Binding)
```

The "[2]" makes sure we are taking the second count from the above output for the paperback books, and then the length function will just tell us the total number of books, regardless of binding type. Dividing these two numbers gets us the proportion we desire!

We might also be interested in obtaining the mean price of paperback books only as well. This might be easily done if we create a new data frame in R that is a subset of the original data frame. Using the `subset` function in R, we'll call this new subsetted data `paperbackbooks`.

```
paperbackbooks = subset(amazonbooks, Binding == "P")
```

This function takes the original data set as the first argument, and then asks for a condition for subsetting in the second argument. There's two things to note for the condition: first, two equals signs are used to check for equality, as one equals sign is already used for defining a variable. Also, non-numeric outcomes need to be put in quotes ("P") to distinguish them from the names of variables. From here, we can compute the mean just as we did before, but with the new data frame.

```
mean(paperbackbooks$Price)
```

There are many different types of conditions you can use for subsetting too! This condition could be based on other variable types too – maybe you want to make an expensive books data set? (`Price > 40`) Or maybe you're interested in big books? (`Pages > 500`) Old books? (`Year < 1990`).

For making subsets, knowing the functions of the following comparators will be helpful:

`==` _______________ `>=` _______________ `<=` _______________

`!=` _______________ `|` _______________ `&` _______________

Getting help
If you don't remember how to use a particular function, R has extensive documentation built into any function you can run in it! To access help on how to use a particular function, try putting a question mark before the function's name and run that in the console. This will open the help tab in the bottom right window of RStudio with the appropriate help file for that function.

```
?subset
?summary
?mean
```

1.4 – Additional Practice

Example: A study on bears collected many measurements on their size and other attributes. This data can be found in the **bears.csv** file on Canvas.

- `Sex`
- `Age` (years)
- Head length (in): `Head.L`
- Head width (in): `Head.W`
- Neck girth (in): `Neck.G`
- Chest girth (in): `Chest.G`
- `Weight` (lbs)

Use RStudio to find the following:

- The mean weight of a bear.
- The mean head length of a bear.
- The standard deviation for the head length of a bear.
- The proportion of female bears.
- The mean weight of male bears.
- The mean weight of female bears.
- The mean weight of bears with a neck girth of more than 20.