Lab 1: Kick Off

Get familiar with R Markdown

Markdown is a streamlined markup language designed to convert plain text into formatted text. Figure 1 gives you some examples and you can explore more details from this useful Markdown Guide.

Figure 1: Examples of markdown

R Markdown is a tool that enhances Markdown by adding *R code* into the mix. It's like writing a report and doing your data analysis at the same time. You can write in plain text, spice it up with Markdown formatting, and insert R code. When you're ready, R Markdown will run the code and include the results in your document.

To create a R Markdown document, we can click "New File" in RStudio, and choose "R Markdown" file (see Figure 2). You can fill up some basic information, such as the title of the document, author's name, the date the document has been generated. The default output of the file will be in HTML format.

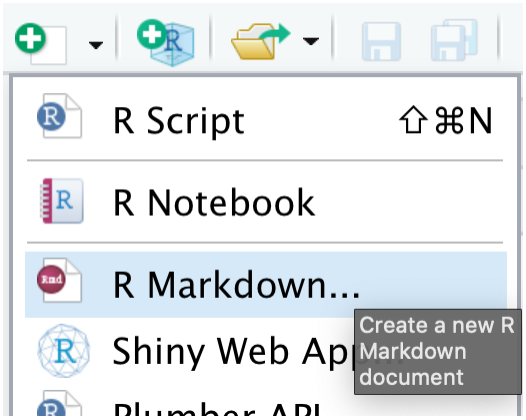
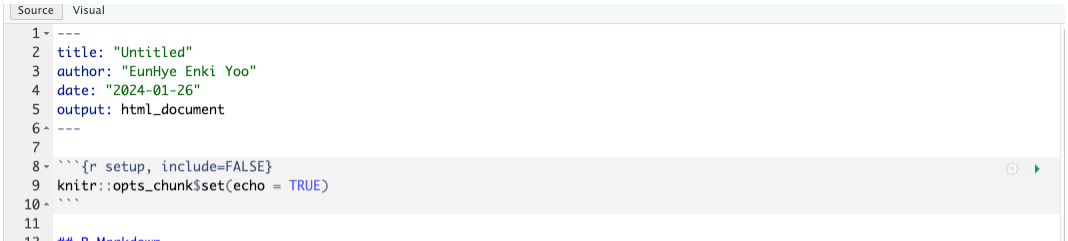


Figure 2: Create R Markdown file

Once you click the "OK" button, you will see a R Markdown interface as shown in Figure 3, which includes three basic parts - YAML metadata, Code chunks (to write R code), and text. When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. Below are some examples for using R code to do some analysis and visualization and more details on using R Markdown can be found HERE.



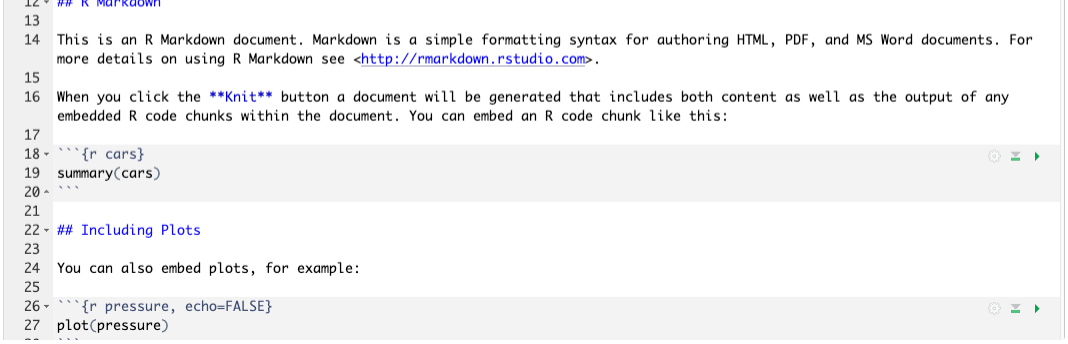


Figure 3: R markdown interface

Embed an R code chunk

```
# load libraries
library(tidyverse) # an collection of R packages designed for data science
```

```
# load diamonds data table
data(diamonds)
# check the first 6 lines of the loaded diamonds table
head(diamonds)
#> # A tibble: 6 × 10
#>   carat cut       color clarity depth table price     x     y     z
#>   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
#> 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
#> 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
#> 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
#> 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
#> 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
#> 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
# compute basic statistical summary of variables within the diamonds table
summary(diamonds)
#>      carat               cut        color        clarity
#>  Min.   :0.2000   Fair      : 1610   D: 6775   SI1    :13065
#>  1st Qu.:0.4000   Good      : 4906   E: 9797   VS2    :12258
#>  Median :0.7000   Very Good:12082   F: 9542   SI2    : 9194
#>  Mean   :0.7979   Premium   :13791   G:11292   VS1    : 8171
#>  3rd Qu.:1.0400   Ideal     :21551   H: 8304   VVS2   : 5066
#>  Max.   :5.0100                     I: 5422   VVS1   : 3655
#>                                     J: 2808   (Other): 2531
#>      depth           table           price             x
#>  Min.   :43.00   Min.   :43.00   Min.   :  326   Min.   : 0.000
#>  1st Qu.:61.00   1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710
#>  Median :61.80   Median :57.00   Median : 2401   Median : 5.700
#>  Mean   :61.75   Mean   :57.46   Mean   : 3933   Mean   : 5.731
#>  3rd Qu.:62.50   3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540
#>  Max.   :79.00   Max.   :95.00   Max.   :18823   Max.   :10.740
#>
#>        y                z
#>  Min.   : 0.000   Min.   : 0.000
#>  1st Qu.: 4.720   1st Qu.: 2.910
#>  Median : 5.710   Median : 3.530
#>  Mean   : 5.735   Mean   : 3.539
```

```
#>  3rd Qu.: 6.540   3rd Qu.: 4.040
#>  Max.   :58.900   Max.   :31.800
#>
```

Including Plots

You can also embed plots, for example:

```
# draw a histogram of variable "price" from the diamonds table
hist(diamonds$price)
```

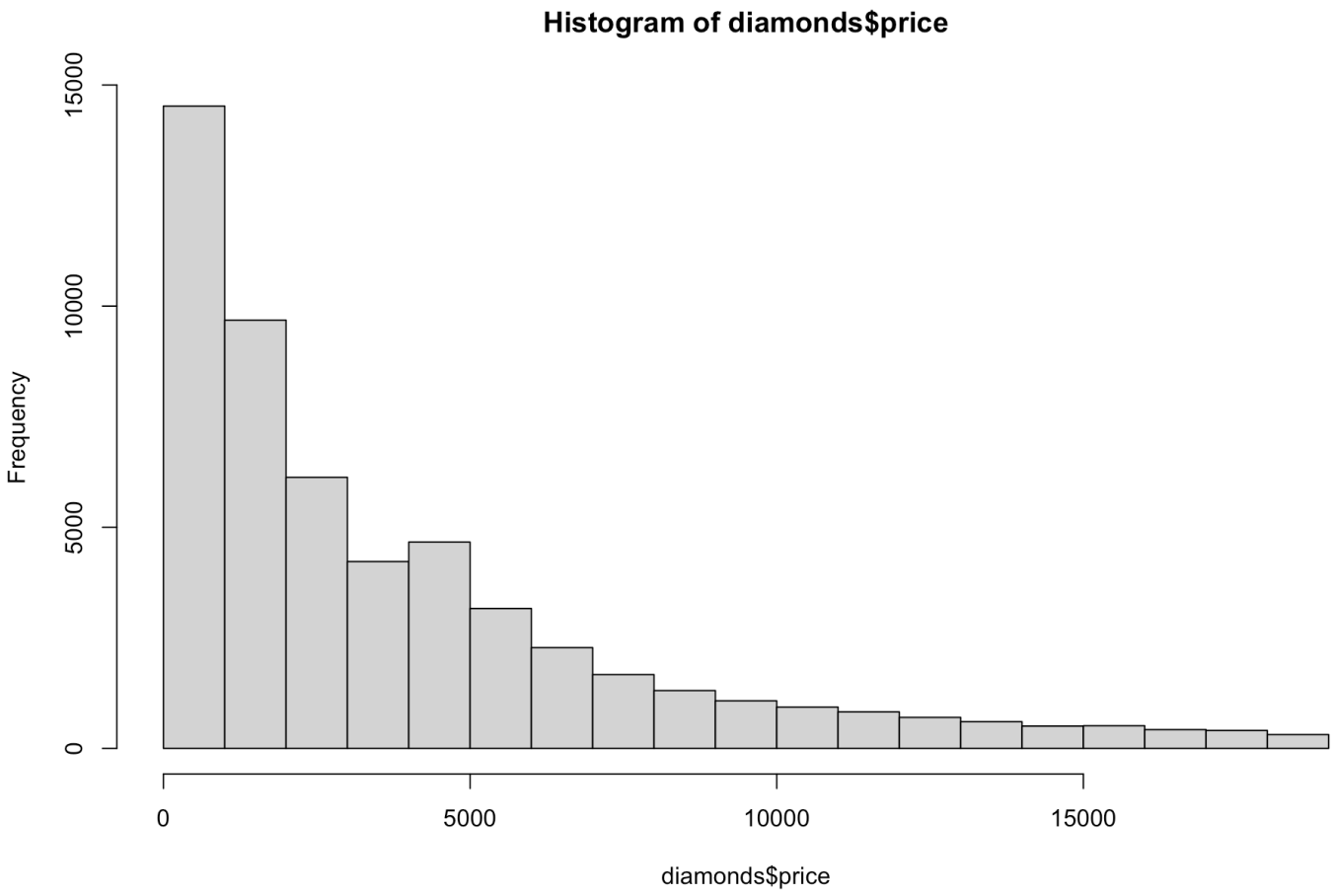


Figure 4: Histogram of diamonds$price

Note that sometimes, it may be easier for you to work directly in an R script as practice. To do this, create a new R Script at the top of your RStudio window by choosing “R Script”.

In this course, we will be using R Markdown for two purposes:

1. Lab instructions will be given in `html` printouts created by R Markdown
2. You will write your lab reports using **R Markdown**

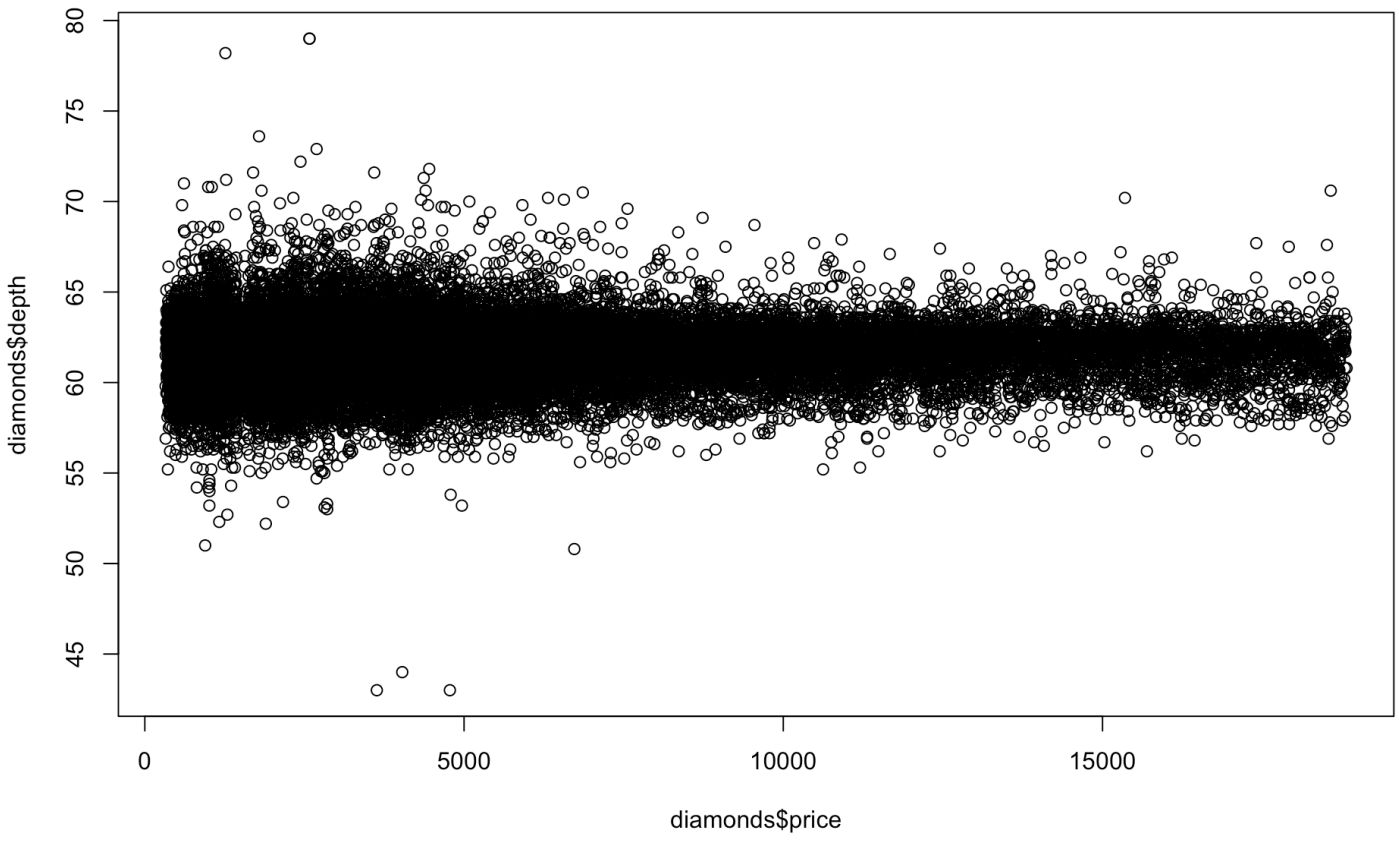
> **Tip**
>
> The R Markdown code used to create the html printouts will be provided for reference.

Let’s get started!

R coding

R is a programming language that is very useful for statistics. R's base package includes very useful statistical and data visualization functions like:

```
# Correlations: measure the correlation between variables price and depth
cor.test(diamonds$price, diamonds$depth)
#>
#>  Pearson's product-moment correlation
#>
#> data:  diamonds$price and diamonds$depth
#> t = -2.473, df = 53938, p-value = 0.0134
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#>  -0.019084756 -0.002208537
#> sample estimates:
#>        cor
#> -0.0106474
#scatter plot: create a scatter plot of variables price and depth
plot(diamonds$price, diamonds$depth)
```



Scatter plot of variables price and depth

For now, we will do a light-hands on intro to statistics in R

Question 1: Dataset

What do we need to do statistics? For starters, we need some data. So let's make some! R has the ability to randomly generate data using a simple function `rnorm()`!

`rnorm()`, if you can guess, generates **r**andom data that follows a **norm**al distribution (or bell shaped-curve), but more on that later. A **function** in r is followed by a set of parentheses (i.e., `()`). The parentheses should include any number of user defined arguments.

In the case of `rnorm()`, there are three basic arguments:

- `n`: indicates the number of observations we want
- `mean`: indicates the average value of the observations
- `sd`: indicates the standard deviation of the observations

For example, if we want to generate a dataset with **1,000 samples**, with an **average of 300**, and a **standard deviation 50**, we can achieve by using the code below with the three arguments mentioned earlier. More specifically, as we generate samples randomly, which means everytime we will get different results. So in order to generate the exact same data each time we run `rnorm()`, we can use `set.seed()` before calling `rnorm()` function. Then, we use the `rnorm()` function to generate 1,000 samples/observations with `mean = 300` (i.e., average of 300) and `sd = 50` (i.e., standard deviation 50). Moreover, as we don't want to have to rerun this function every time we want to access our data, so we use `<-` or equivalently `=` to store our data to a named vector, `sampled_data` in this case. The comment key (indicated by a hashtag) in code chunk is commonly used to easily describe the purpose of the code and any algorithms used to accomplish the purpose. The comment key tells r to not run whatever follows it on the same line.

```
# for reproducibility
set.seed(211)
# generate samples with rnorm function and
# store the samples in a variable called: sampled_data
sampled_data <- rnorm(n = 1000, mean = 300, sd = 50)
```

Instead of printing out all 1,000 sampled data, we can check the first parts of the samples by using `head()` function with the name of our vector `sampled_data` (see below). We can then view the first few observations of our data:

```
# look the first 6 sampled data
head(sampled_data)
#> [1] 263.8432 393.8224 268.6892 202.1015 343.5566 255.3600
```

> **Note**
>
> Note the function `set.seed()`. Calling this function prior to generating our random data ensures that we can generate the exact same data each time we run this code! This will allow us to get the same answers when we calculate some statistics.

Does anyone see any problems with our data so far? For starters, I find the number of decimals annoying, so we can reduce these decimals using the `round()` function. The `round()` has the argument of `digits`, which tells the function the number of digits to round to. Let's use it with 2 digits and see what we get.

```
# for reproducibility
set.seed(211)
# we first create the same samples, and then round the number of digits to 2
sampled_data_rounded <- rnorm(n = 1000, mean = 300, sd = 50) |>
```

```
  round(digits = 2)
# print out the first part of the samples
head(sampled_data_rounded)
#> [1] 263.84 393.82 268.69 202.10 343.56 255.36
```

> **Note**
>
> Data analysis often involves many steps, we use pipe operator (i.e., `|>`) to emphasize a sequence of actions. It takes the output of one function and passes it into another function as an argument. This allows us to link a sequence of analysis steps. Take `rnorm(n = 1000, mean = 300, sd = 50) |> round(digits = 2)` as an example, we first randomly generate 1,000 samples, then round the sample values to 2 digits.

For your answer to question 1, generate the *SAME* data set that I did above, and print the first few observations using the `head()` function. To do this, you will have to include all of the required lines of code in the R chunk signified by the three tick marks (i.e., 1,000 samples with an average of 300 and a standard deviation of 50). (34 points)

Question 2: Descriptive statistics

Now that we have our dataset, we want to examine and describe it. We discussed some of these ways in class including finding the measures of central tendency (mean and median) and measures of variability (minimum, maximum, variance and standard deviation).

All of these measures can be calculated automatically using their own functions in R. The functions are as follows:

- Mean: `mean()`
- Median: `median()`
- Minimum: `min()`
- Maximum: `max()`
- Variance: `var()`
- Standard Deviation: `sd()`

In order to use these functions, all you have to do is put your dataset's name in between the parentheses, just as we did for the `head()` function.

For your answer to question 2, print the mean, median, minimum, maximum, variance, and standard deviation of the generated dataset in Question 1 (33 points).

Question 3: Data visualization

Descriptive statistics are not complete without data visualization! The type of data visualization (graph/plot) used depends on the kind of data you have. We discussed four general types of data in class. These types are:

- Quantitative discrete
- Quantitative continuous
- Qualitative ordinal
- Qualitative nominal

For your answer to question 3, there are three sub-questions to answer.

Q3a: What type of data do you have? (11 points)

You can write your answer in your Q3a code chunk using the comment key (indicated by a hashtag).

Here's an example:

```
# Our data is 'REPLACE YOUR ANSWER'.
```

After determining what type of data we have, we then need to determine what type of data visualization to use. Two methods discussed in class are the histogram and boxplot. The histogram can be created using the function `hist()`, while the boxplot can be created using the function `boxplot()`.

For your answer to this part, please plot both of these plots with your data.

Q3b: Display a histogram for your data. (11 points)

Q3C: Display a boxplot for your data. (11 points)

After you plot both of these plots, you are all done!

Your can create your own Rmarkdown to complete the Lab 1 homework, or you can use the blank answer Rmarkdown template (download from UBLearns). Make sure to successfully **knit** your document and submit the **html** on UBLearns under Lab 1. You will likely have to compress your document into a **.zip** file to submit to UBLearns. Reach out to the TA or myself if you are unable to do so.