HOW TO RUN ESSENTIAL ANALYSES IN R

EDITION ONE

For First Year Undergraduate Students With Limited Background in Programming

NAREN D. SELVARATNAM

"At times overwhelming, but deeply satisfying programming language for statistical computing"

How to Run Essential Analyses in R Studio: For First Year Undergraduate Students with No Background in Programming

First Edition

Naren D. Selvaratnam
B.A. (Psych)., M.Sc. (Psych)., M.S. (Ed LDRS)., Ph.D. (Ed MGMT) (R)

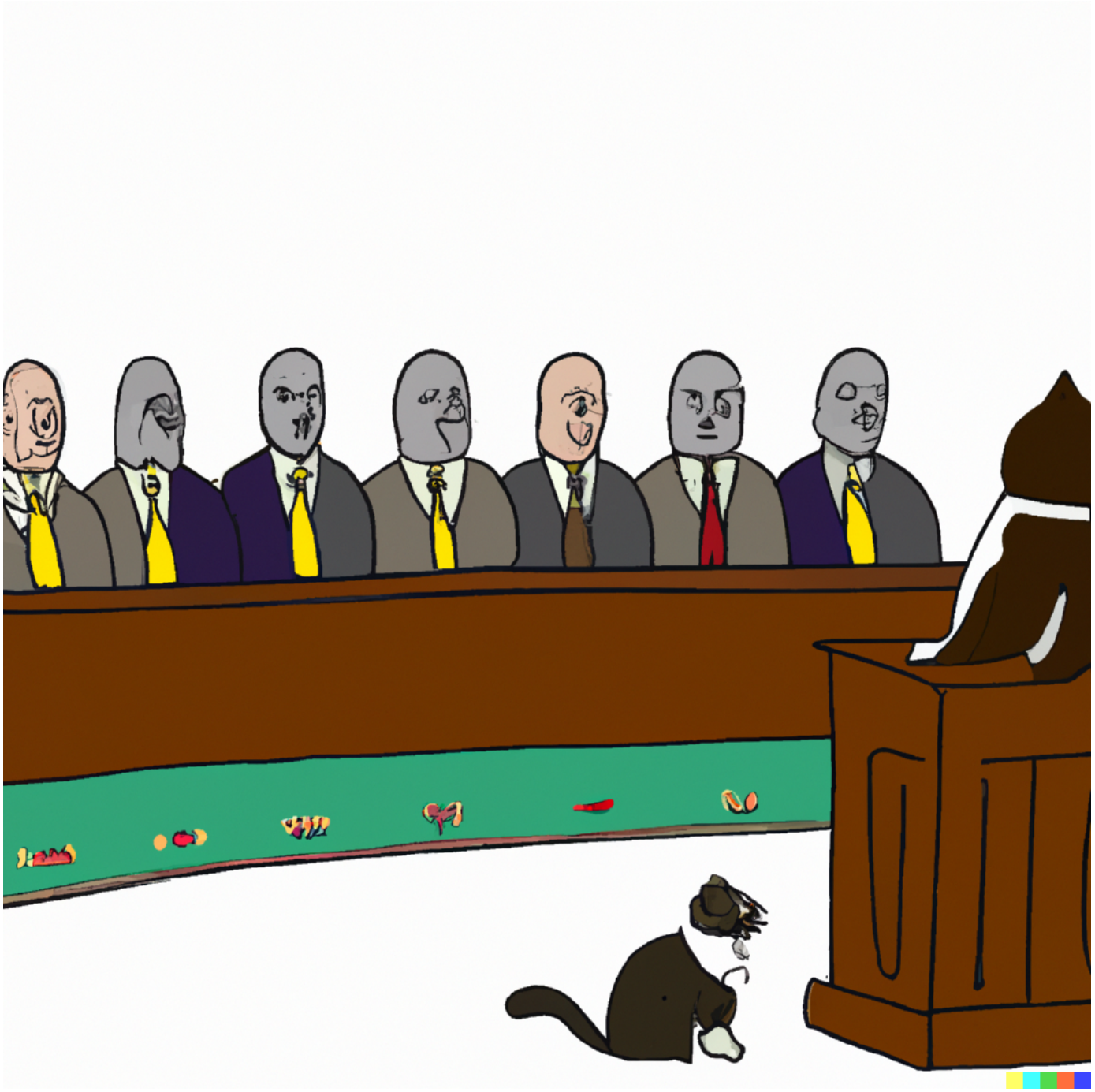
Eagan Journal of Contemporary Research (EJCR) is a peer-reviewed open-access e-journal introduced in the year 2021 to disseminate scientific knowledge in Sri Lanka with a specific focus on Psychology, Education, and Public Health disciplines. To further enhance the interest in scholarly work, the following short guide to R is written. This guidebook is the first issue of many to come.

Website: www.eaganjournal.com, Email: eaganjournal@gmail.com

EJCR is published by Deep Haven Counseling (DHC) (Pvt) Ltd.
Address: 117/2/39, Horahena, Kottawa, Pannipitiya

This e-book is distributed free of charge.

Contents



To explore the unknown frontiers of Science

1.0 Introduction

R is one of my go-to software for data analysis. Since *SPSS* does not offer some high-end analysis for us on a *Macintosh* operating system, I was always on the lookout for a better alternative. I have used *JASP* and *JAMOVI* as well. However, the degree to which it could assist me with various forms of statistical modeling is minimal. Unlike *SPSS*, *R* is more flexible and offers quite a lot of customization for researchers. At first, I was intimidated by *R*, but, slowly, I managed to learn the basics to help me run most of the analysis required to teach my undergraduate students. I am still learning *R* and the different packages it offers (more on packages later).

Essentially, *R* is an open-source software. First, you should install *R* and then *R Studio*. *R* will offer you a unique experience if you, like most of us, are used to running statistical analyses with a few clicks and interpreting ready-made outputs on SPSS. I personally love running analyses in *R Studio*. But, if you are new to *R*, you should learn the *R* language. However, there is no reason to worry as this guide will help you learn the basics of *R language* with an engaging psychological study and a simple set of variables to play around with.

I am by no means a programming expert. I am more or less like you with this software. I just have been using this for an extended period of time. As a result, I thought I should share what I have learned with you so that you can also effectively master *R*. In that manner, consider this book as a concise narration of a researcher who has no background in programming towards a similar group of researchers.

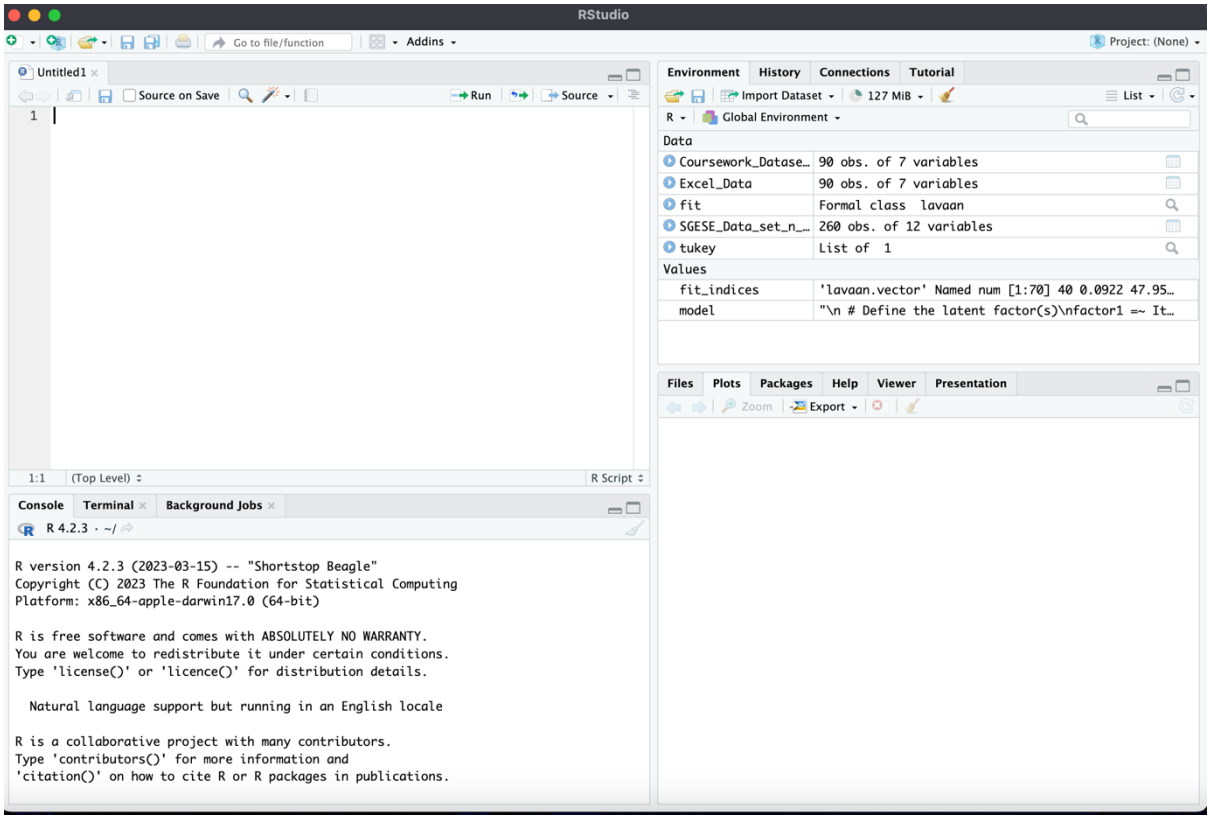


Figure 1: R Console

Now, before continuing this discussion further, let's get an idea about how *R* the software looks like when it is fully installed. Take a moment to carefully look at *Figure 1*. Unlike *SPSS*, here you will notice 4 windows. On the left, you notice two separate spaces. The upper gives your data view, and the lower is the place where you give *R* commands (or scripts). Similarly, you have two windows to the right in which the upper displays your variables and the lower shows the visual output of any analysis you conduct. Since you have just now learned some information about *R* console, you must be wondering how to install this software on your computer or laptop. I have a MacBook Air, and I am now explaining how I installed it on my laptop. First, you should go to Cran.r-project.org. Once you are on this website, you can download *R*. Make sure to select the *R* that best fits your *Mac* or *PC*. Upon installing *R*, you should then install *R Studio*. What you see in *Figure 1* is *R Studio* when it is installed and opened.

Sample Study

To make the best of this study guide, I encourage you to download the Excel File named *'Cleaned_Data'* to your laptop. This is a dataset I developed with my first year first semester psychology students at Sri Lanka Institute of Information Technology *(SLIIT)*. We collected data from 59 students to examine the relationship between 'generalized self-efficacy' and 'general anxiety'. Self-efficacy was measured by the generalized self-efficacy scale *(GSES)* and anxiety was measured by general anxiety disorder 7 *(GAD-7)*. We entered all our data into *Microsoft Excel* and did some data cleaning. This final dataset is after all the cleaning and we have retained some demographic variables as well to further assist you with the analyses to come.

Variable	**Level of measurement**
Gender	Nominal
Age	Ratio
Year of study	Ordinal
GSES Q1	Ordinal
GSES Q2	Ordinal
GSES Q3	Ordinal
GSES Q4	Ordinal
GSES Q5	Ordinal
GSES Q6	Ordinal
GSES Q7	Ordinal
GSES Q8	Ordinal
GSES Q9	Ordinal
GSES Q10	Ordinal
GSES Total	Ratio
GAD7 Q1	Ordinal
GAD7 Q2	Ordinal
GAD7 Q3	Ordinal
GAD7 Q4	Ordinal
GAD7 Q5	Ordinal
GAD7 Q6	Ordinal
GAD7 Q7	Ordinal
GAD7 Total	Ratio

Table 1: Sample study variables

2.0 Understanding & Installing Packages in R Studio

Okay, since we now have a good understanding of how to install *R*. Let's take a moment to further understand the nature in which *R Studio* works. *R* is an open-source software and it enables researchers to modify and develop their own packages to run analyses specific to certain academic fields. For instance, although many in our field of psychology lack a preference for statistics, some of us feel excited working with larger datasets, especially to understand latent structures of psychological constructs. A latent structure can only be measured using a set of complicated processes of statistics. Most psychology post-graduate programs cover the 'classical test theory' *(CTT)* procedures. However, 'item response theory' *(IRT)* and other 'latent trait modeling' *(LTM)* methods are hardly discussed. Most of the time, they are limited to individuals who are enrolled in psychometrics programs. Most of my colleagues hate all of these statistical concepts. I understand their feelings. But, once you start understanding research methods and statistics, a whole new dimension of reality opens up for students. I know it did for me. Now, despite the benefits statistics bring to our lives, the biggest problem we have as of now is the difficulty of obtaining affordable software for data analysis. Most software is expensive; but, it turns out that *R* is an affordable alternative.

In the *R Studio*, to run anything from basic descriptive statistics to complicated *LTM*, we need to install different packages. These packages are available to be downloaded from the 'packages' tab on the *R Studio*. The image given below shows where to locate this tab.

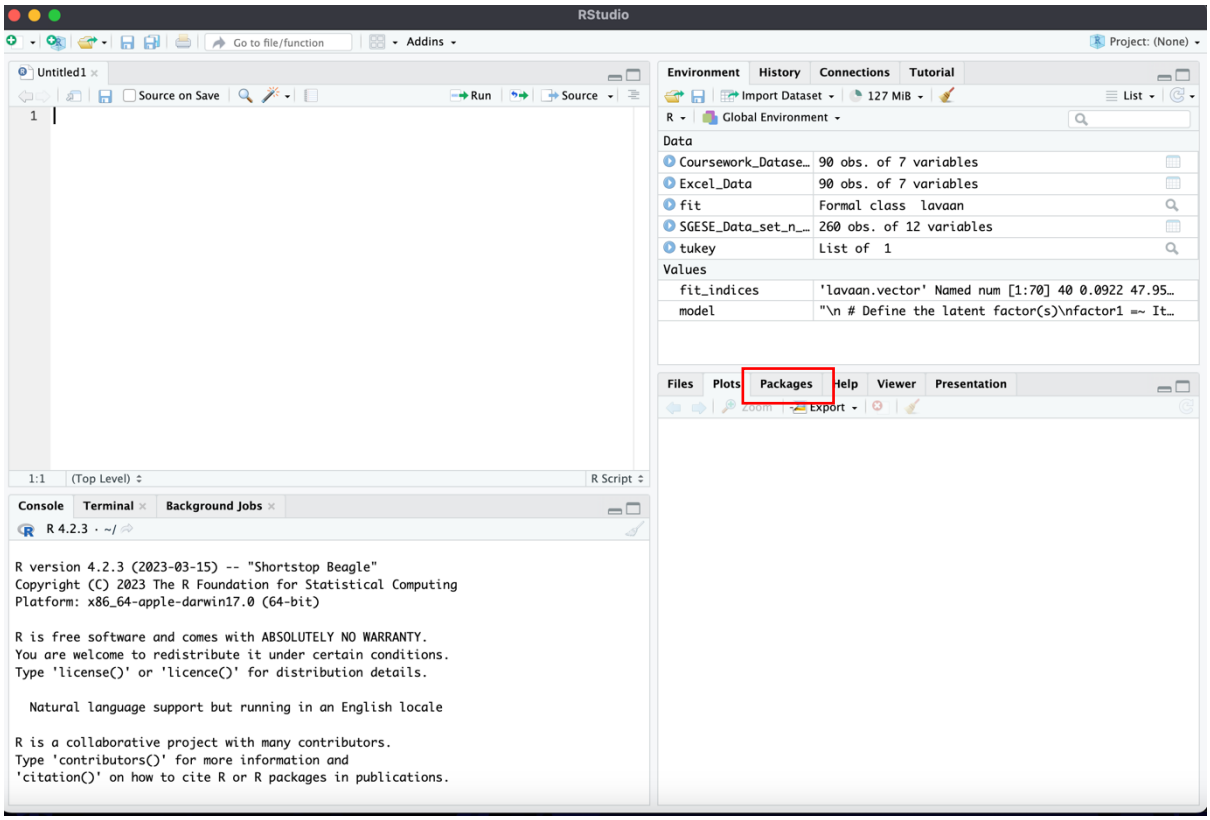


Figure 2: Locating packages in R

Alternatively, I can enter a script to load a specific *R package* as well. In the proceeding lines, I am going to show you how to do this. Please note that *R scripts* are always colored in

blue in this report while *R output* is colored in green. Now, to run some descriptive statistics, I am going to first install a package named 'pacman.' So, I am going to enter the below given *R* script.

```
install.packages ("pacman")
```

It might not work if you just copy and paste. So, it would be better if you could write this yourself on the lower window to your left. When you enter *R* commands, take good note of the inverted commas. If you miss it here, it will not run. Once you type and press enter, this package will be installed. If it runs smoothly, you should get an output like this on your lower left window.

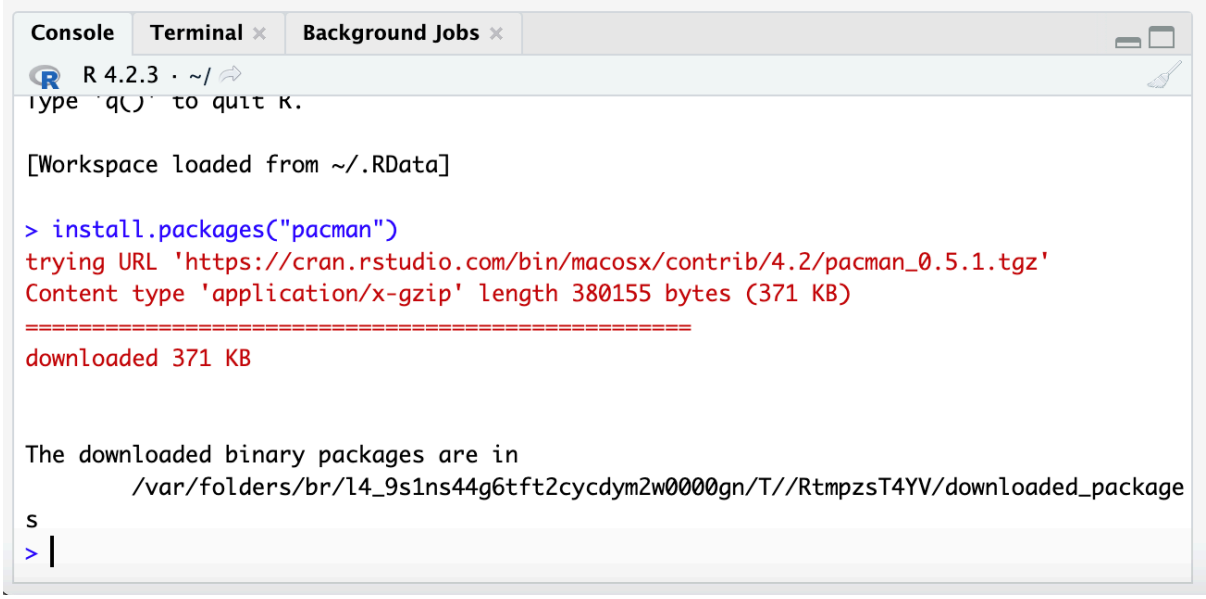


Figure 3: Running scripts in R

Once this process is completed, you should load the installed package to use it for analyses. This process has to be done each time you open *R* for analyses. To load, add the below-given lines to the same lower left box.

```
library(pacman)
```

Upon writing the above, hit enter, add the below-given script as well, and hit enter. Sometimes, you might have to manually write them if it does not seem to work.

```
pacman::p_load(pacman, dplyr, GGally, ggplot2, ggthemes, ggvis, httr, lubridate, plotly, rio, rmarkdown, shiny, stringer, tidyr)
```

Given below is the breakdown of the above code.

```
pacman::p_load(
  pacman,     # The 'pacman' package itself (required to use 'p_load')
  dplyr,      # Data manipulation package
  GGally,     # Extension of 'ggplot2' for exploratory data analysis
  ggplot2,    # Data visualization package
  ggthemes,   # Additional themes for 'ggplot2' graphics
```

```
  ggvis,      # Interactive data visualization package
  httr,       # Package for working with HTTP requests
  lubridate,  # Package for working with dates and times
  plotly,     # Interactive web-based plotting package
  rio,        # Data import/export package
  rmarkdown,  # Package for creating dynamic documents
  shiny,      # Web application framework
  stringer,   # String manipulation package
  tidyr       # Data tidying package
)
```

The above code entails multiple packages commonly used for psychology research. A combination of packages is usually used for statistical analyses. In this book, we will be using some of the packages and their functions, but not all of these. However, once you have completed this book, you will have the confidence to run more analyses using other packages as well.

3.0 Useful R Packages for Psychology Undergraduates

Here is a list of useful *R packages* for psychology students. Here I have done a brief introduction for each package. Based on your preference, you may go ahead and explore others as well. Always remember, the first step is to get familiarized with the general ways and means of *R*. So, let's take a moment to review the listed packages.

a. **base:** Provides fundamental functions for basic statistical procedures
b. **dplyr:** This is an essential package for data manipulation. dplyr provides a variety of functions including filter(), select(), and group_by() for efficiently analyze large datasets. Data scientists call this process *data wrangling.*
c. **tidyr:** Helps in *tidying* data.
d. **ggplot2:** Helps visualize data and offers researchers a greater degree of customization.
e. **psych:** Offers descriptive statistics, factor analysis, reliability analysis, etc. required for psychological testing and measurement.
f. **lme4:** For fitting linear and nonlinear mixed-effects models.
g. **effsize:** Helps calculate and interpret effect sizes of statistical tests.
h. **psychTools:** Provides various functions for psychological testing, including item analysis, test scaling, and scoring.
i. **foreign:** Interprets datasets form *SPSS*, *SAS*, etc.
j. **ggstatsplot:** An extension to ggplot2
k. **ltm:** Helps run item response theory models such as Rasch model, graded response model, etc. that is commonly used in psychometrics. Helps in investigating latent traits.
l. **nortest:** Helps conduct normality tests
m. **MBESS:** Useful for effect size calculations.
n. **jtools:** Assists in summarizing regression models.
o. **pwr:** Helps with power and sample size calculations.
p. **ez:** Makes the analysis of factorial arrangements simpler.
q. **lavaan:** Used in structural equation modeling and latent trait modeling

These packages cover a wide range of statistical analyses commonly used in psychology research, from data manipulation and visualization to hypothesis testing and basic forms of statistical modeling. I did not include some of the complicated packages as it does not fit the scope of this book. Since most students learn basic concepts of validity and reliability, I included 'ltm' for the exact purpose of introducing students to the diverse and engaging field of psychometrics. That being said, the best way to utilize this book is to have good research methods textbooks to go through as you learn the R programming language. I have listed two books you could find in both Sri Lanka and internationally for your reference. Both are excellent books.

Azam, S. M. F., Yajid, M. S. A., Tham, J., Hamid, J. A., Khatibi, A., Johar, M. G. M. Arrifin,I. A. (2021). *Research methodology: Building research skills*, McGraw Hill Education (Malaysia) Sdn Bhd, Kuala Lumpur: Malaysia
Gravetter, F. J., & Wallnau, L. B. (2017). *Statistics for the Behavioral Sciences, Tenth Edition*. Boston, USA: Cengage Learning.

My YouTube Research Playlist

To further help students learn the fundamental concepts of quantitative research methodology. I strongly advise you to review these videos so that you can begin the proceeding statistical analyses with confidence.

1. Research for beginners: The scientific method
2. Research questions & first impressions
3. Measurements in quantitative research
4. Correlations (a step by step guide)
5. Correlations part II & Chi-Square test
6. Introduction to qualitative research
7. Qualitative research part II
8. Shapiro-Wilk, ANOVA, & Kruskal Wallis on R Studio (simple instructional video)
9. T-Test & ANOVA manual computation
10. How to write a simple draft research proposal (for absolute beginners)
11. 15 lessons I learned about research paper publications
12. How to write a results section?

All of the above videos are available in my YouTube channel, Naren D. Selvaratnam

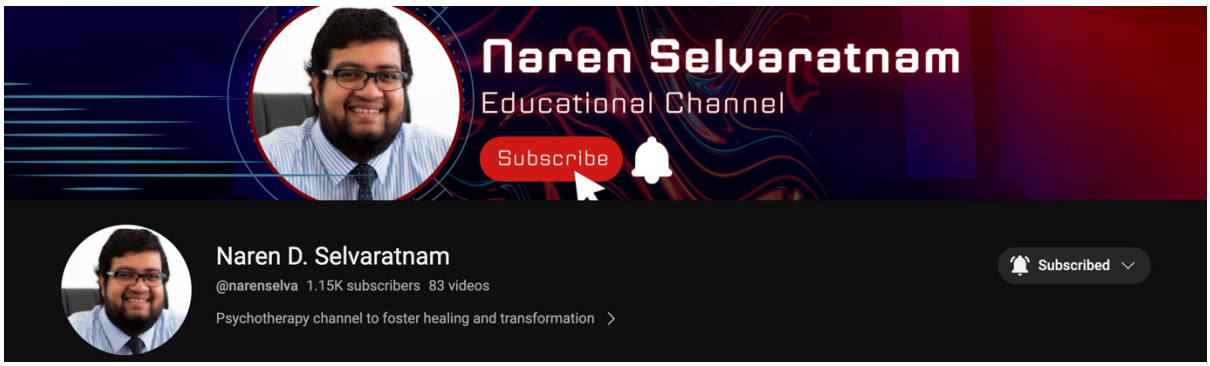


Figure 4: YouTube channel information

4.0 Loading Excel Data to R Studio

Now that you have loaded some of the packages. Let's learn how to load Excel data to *R*. In one of the previous sections, I introduced the variables we have. If you cannot recall, please take a moment to review them again. The collected data was cleaned and saved to a file named 'Cleaned_Data.' Let's import this dataset to *R Studio* now. Please follow the following steps.

Click '**File'** and then go to **'Import Dataset'** and then select **'From Excel.'** You will get a pop-up window as shown in *figure 4*.

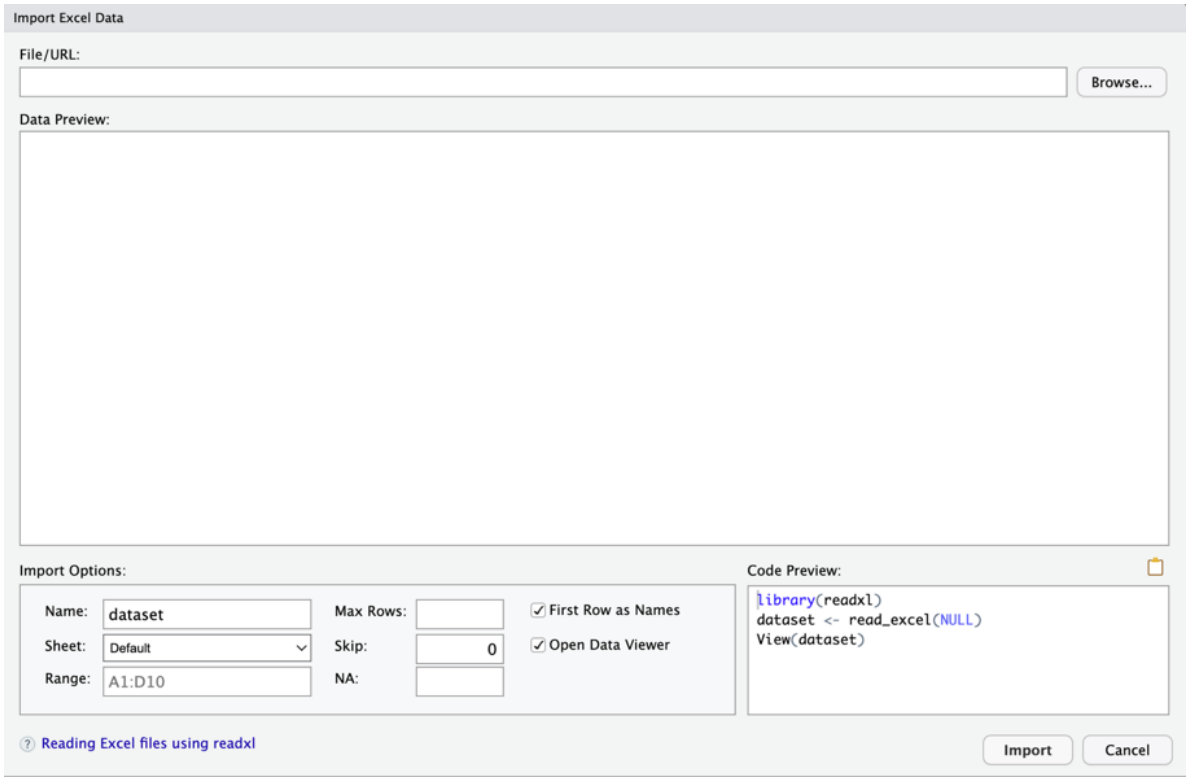


Figure 5: Loading data

Click on the **'Browse'** and select your Excel file from your *PC* or *Mac*. The Excel sheet should contain data added appropriately. Remove unnecessary variables and calculations from your *Excel* to properly load it on *R Studio*. We call this process data cleaning. Let's say you collected data through a Google form. It is a good practice to see whether you have participants who have missed questions, completed the survey in an extremely short period of time, those who have unusual answer combinations, etc. This is a slow process; but, we should spend some time to understand the kind of data we have in our dataset. For instance, in the present dataset, there were participants with extremely high efficacy and high anxiety. This is an unusual combination since anxiety precipitates in the absence of self-beliefs. In that sense, quantitative researchers should have an in-depth understanding of the theoretical basis of constructs utilized in the study. I will explain more about constructs and their measurements in a later edition under latent trait modeling. For now, we will focus on fundamentals. So, load only the Excel data that you have cleaned. Name your dataset as well. For this example, I am going to name my data set "Cleaned_Data." Once you import your

data, your console should look like this *(figure 6)*. You will notice that my data now appears on the left upper window.

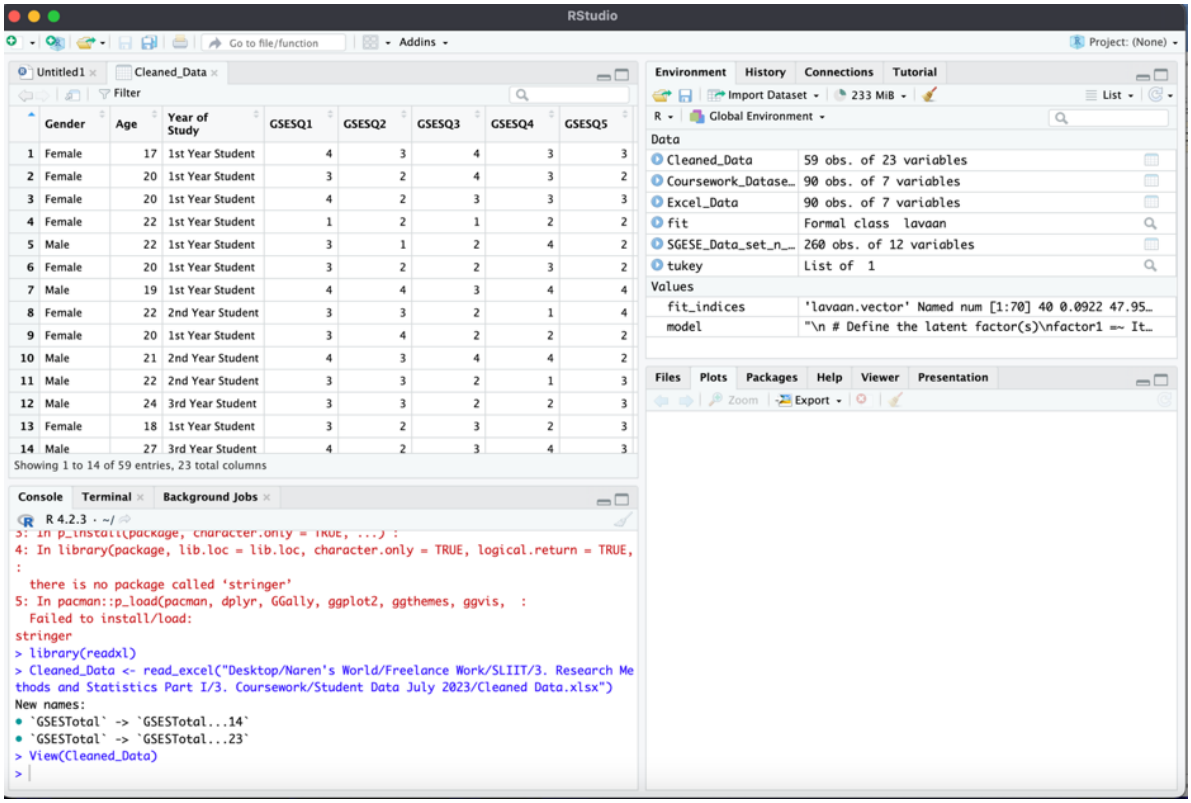


Figure 6: Excel data loaded

Take a moment to browse through your excel data. Scan whether you have all your variables. If you have a trackpad, simply by swiping it to left and right, you can observe the full dataset. Once the dataset is loaded, you are all set to commence using *R* commands.

5.0 Defining Variables in R Studio

Upon loading data to *R Studio*, the next step for us is to define our variables. Usually, this process is done for categorical variables. Once you define these categorical variables, it is easy for us to proceed with any other analyses to come. But, before we go any further, let's try to comprehend the logic behind defining variables in *R studio*. Similar to many other programming languages variables have to be defined due to several crucial reasons. I have listed some of them down below.

- **Memory Allocation:** When you define your variables, the system remembers them. Usually, when we define, we allocate a new name and this newly defined variable can be accessed at any time during your analysis session. Such allocation of memory is important for efficient data manipulation and analysis.

- **Data Integrity:** Defining variables helps researchers to further specify their data type (e.g., numeric, character, logical) and initial values of the variable identified for defining. This process boosts the integrity of our data.

- **Debugging and Error Handling:** Having properly defined variables makes it easier for the researcher to debug if there are any errors in the codes written.

- **Readability and Documentation:** Properly defined variables are readable and help researchers efficiently handle analyses. Being able to read a code clearly is a bigger advantage than having defined variables. You may also consider this as a process of documentation. In this book, even I am doing some form of documentation for you. In that sense, my codes will be mostly self-explanatory once you go through the first few of them.

- **Scope:** Variables in R have a scope that helps researchers determine how to access and modify these variables. In this edition of the book, I have not included this aspect. To understand the concept of variable scope, it would be better to know the very basics of R. Once you finish this book, you will be more confident to understand the rest.

Now that we know the benefits of defining variables in *R Studio*, see an example in *R* of defining variables from the dataset we have loaded already. Since I have 2 categorical demographic variables, I am going to first define them. Let's start with 'Gender.' The below-given code has to be entered into the *R console*.

```
> RGender <- table (Cleaned_Data$Gender)
```

In this above code, 'RGender' is the new variable I created using the existing variable 'Gender' in the dataset 'Cleaned_Data' which is already loaded to *R Studio*. Hereafter, for most of my analyses, instead of 'Gender', I will be using 'RGender.' I will point out any exceptions to this rule. If you run the above code, it commands *R* to create a table of frequencies for values of the 'Gender' variable in a data frame called 'Cleaned_Data.' Let's try to further break down this code for us to further understand its elements.

- **Cleaned_Data$Gender:** In this extracted portion of the above code, the data frame is 'Cleaned_Data.' '$Gender' is used to extract the values from the 'Gender' column of the dataset named 'Cleaned_Data.' The 'Gender' column should contain categorical (nominal) data such as "Male," "Female," etc. In the loaded dataset, data of the "Gender" variable are entered using words instead of numbers.
- '$Gender' uses the '$' operator to extract a specific column from the data frame identified. In this case, the 'Gender' column is extracted from the data frame.

Untitled1 | Cleaned_Data

Filter

	Gender	...2	Age	Year of Study	GSESQ1	GSESQ2	GSESQ3	GSESQ4	GSES
1	Female	2	17	1st Year Student	4	3	4	3	
2	Female	2	20	1st Year Student	3	2	4	3	
3	Female	2	20	1st Year Student	4	2	3	3	
4	Female	2	22	1st Year Student	1	2	1	2	
5	Male	2	22	1st Year Student	3	1	2	4	
6	Female	2	20	1st Year Student	3	2	2	3	
7	Male	1	19	1st Year Student	4	4	3	4	
8	Female	2	22	2nd Year Student	3	3	2	1	
9	Female	2	20	1st Year Student	3	4	2	2	
10	Male	1	21	2nd Year Student	4	3	4	4	
11	Male	1	22	2nd Year Student	3	3	2	1	
12	Male	1	24	3rd Year Student	3	3	2	2	
13	Female	2	18	1st Year Student	3	2	3	2	
14	Male	1	27	3rd Year Student	4	2	3	4	

Showing 1 to 14 of 59 entries, 24 total columns

Figure 7: Loaded Excel data

- **table():** The 'table()' function in *R* is used to create the frequency table. It takes a vector as input and gives us back a table that displays the frequency (count) of each unique value in that vector. In this case, the vector is the values in the 'Gender' column of 'Cleaned_Data.'
    - **What is a vector?** A vector is a 'one-dimensional data structure.' In *R*, a vector is a fundamental data structure that can hold a sequence of values of the same data type. In our example, the vector holds categorical values that represent two types of 'Gender', such as 'Male' and 'Female.' This is a one-dimensional data structure (aka *1D arrays*). Refer to the section on terms for a concise explanation of *1D arrays*.
- The 'Gender' column contains data as displayed in *Figure 7*. Thus, the resulting 'RGender' table might look like this.

```
Gender
Female   Male   Other
     2      3       1
```

- In the above example, the count for **Male** is **3**, **Female** is **2**, and for **Other** is **1**. These kinds of frequency tables are excellent for summarizing categorical variables and to understand the nature of the collected data.

Since our 'Year of Study' variable is also categorical, we can repeat the same process for 'Year of Study' as well. The *R* command you should enter in the console is given below.

```
YoS <- table(Cleaned_Data$`Year of Study`)
```

Once these are entered, you will notice, your right topmost square will now show new variables entered.

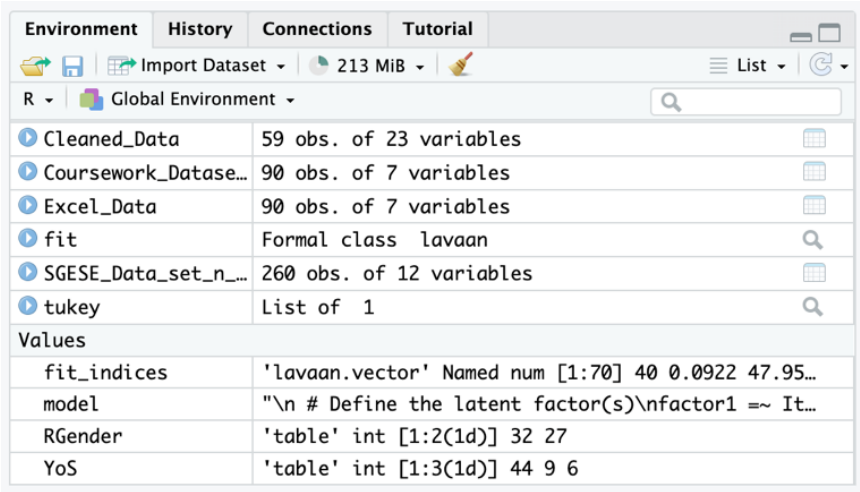


Figure 8: Newly created variables

With this lesson, we have completed most of the basic lessons required to run simple statistical procedures. In the proceeding sections, I will take you through descriptive statistics, normality testing, and correlations. There are a lot of statistical procedures that could be introduced in this book, but, to make this a feasible and entertaining read, I included the most essential ones for a first-year student.

6.0 Descriptive Statistics on R

Something I have noticed often in student coursework is the limited emphasis they place on descriptive statistics. Oftentimes, students attempt to test hypotheses. Accordingly, they conduct inferential statistical procedures as opposed to descriptive analyses. Although inferential statistical measures are essential for testing statistical significance, it is important students spend substantial time examining other trends within their data. This objective of unraveling trends within data can be achieved by running effective descriptive analyses. As a result, in this section, I am introducing some useful *R* commands to facilitate descriptive analyses.

Working with Categorical Data

Now let's do some descriptive analysis. I have colored codes in blue and output in green. Let's first obtain some summaries of the categorical variables we have already defined. This can be easily obtained by typing the code given below in the *R console*. You may have noticed that I use *R Studio* and *R Console* interchangeably. Make sure to check the difference between these two before proceeding further. Check **section 10** for definitions. Here I have given both the input and the output.

```
> summary(RGender)
```

```
Number of cases in table: 59
Number of factors: 1
```

summary() is a generic function in *R* that provides summary statistics and information about the structure of an object. You can obtain detailed information by giving the *R* command, **str()** displayed below. Here also, I have given both the input (code) and the output for your reference. Please make sure to try these rights now so that you know how to enter these codes and obtain output. If you get error messages, you should try to type in the code. Usually, when you type the name of your defined variable, *R Studio* gives a pop-up list of variables, and you can choose the appropriate variable from the list.

```
> str(RGender)
```

```
Female   Male
    32     27
```

Now, we have some understanding of the nature of our categorical variable. Our total sample is n = 59. We have 32 female participants and 27 male participants. We can display this categorical variable and its frequencies by typing in the below-given *R* command. This helps us to generate a bar chart. Since our data is categorical, for visualization purposes, a bar chart would be ideal.

```
> barplot(RGender)
```

When you enter this command, the below-presented image will appear on the bottom right window of R Studio.

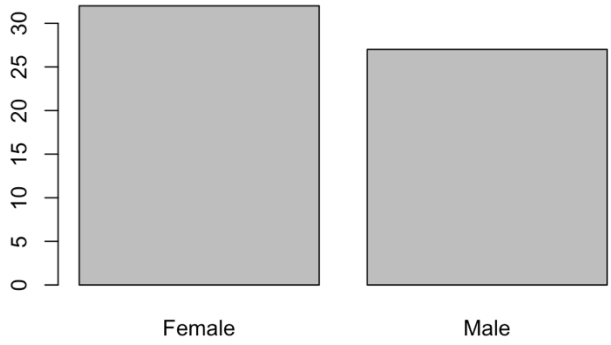


Figure 9: Gender visualization

Unlike SPSS, R Studio provides a greater degree of flexibility, so, I am going to add a new title and change the color of the above bar plot. Now, instead of just '> barplot(RGender)' I am going to add a couple more components to the above code. Given below is my new code.

```
> barplot(RGender, col = "white", main = "Gender Frequencies")
```

In the above, after typing RGender, I have included a comma, and have then typed '**col**' which is a function we can enter to add a color to our figure. Here, I have added "**white**" within inverted commas, and I have written the full word in lowercase. I have then added another comma and have written "**main**" which stands for the main title. After adding this, I ran my new code, and here is my new output. What do you think?

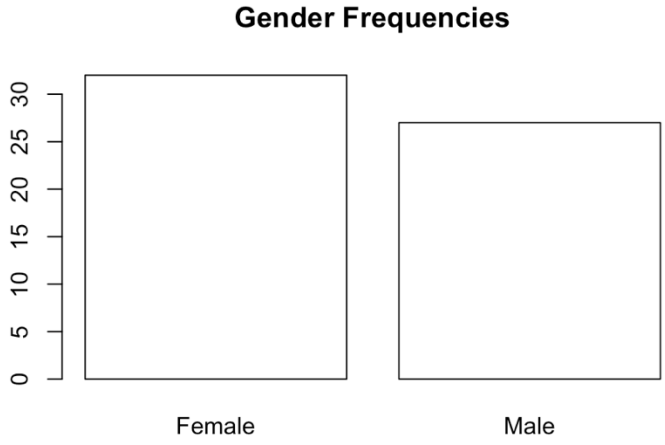


Figure 10: Gender visualization, new elements added to the visualization

R Studio is a great platform for developing high-quality visualizations. You may simply right-click the image and select 'save image as' to go ahead and save your generated bar plot in a format (i.e., jpeg, png, etc.) that you think fits. So, if you think research is your calling, it is not a bad decision to master these skills right away to be good at descriptive analysis.

Now that we know how to do some descriptive statistics, let's check our second categorical variable 'Year of Study' as well. I have presented the codes and the findings below from our dataset.

```
> summary(YoS)
Number of cases in table: 59
Number of factors: 1

> str(YoS)
 'table' int [1:3(1d)] 44 9 6
 - attr(*, "dimnames")=List of 1
  ..$ : chr [1:3] "1st Year Student" "2nd Year Student" "3rd Year Student"
```

Unlike the output of RGender, the **str()** this time has given some more details about our data. Let's decode this information one by one so that we can make sense of our descriptive data.

- **'table':** This indicates the researcher the data type of the object. In this example, it is a table that is commonly used in R for storing frequency or contingency tables.

- **int:** This lets the researcher know that he/she is working with integers (whole numbers).

- **[1:3(1d)]:** In this component, '1d' tells the researcher that this data has one dimension **(1d)** and consists of 3 distinct values **(1:3)**. This means that the table is a one-dimensional vector with three values.

- **44 9 6:** These are the frequencies associated with the categories or levels. Although a vector displays the same type of data, in our example there are 3 distinct values. Each value has a specific frequency. In this analysis, 44 students are first-year students, 9 students are second-year students, and 6 are 3rd year students.

- **attr(*, "dimnames")=List of 1:** In this bit of the output, "**dimanames**" refers to dimension names which are often used to label the levels or categories in the table. In some sense, this information provides metadata about our object. The "**List of 1**" indicates the '**dimnames**' attribute contains a list *(vector)* with one element. In R, a list is a data structure that can hold multiple values of different types. However, since we have previously defined our variable (YoS), in this case, the list, which now is a vector contains one element which is used to store the names of the dimensions.

- **..$: chr [1:3]** "1st Year Student" "2nd Year Student" "3rd Year Student": These are the dimension names, and they are of character data type **(chr).** They correspond to the labels for each count in the table. In this case, they represent the categories or levels: "1st Year Student," "2nd Year Student," and "3rd Year Student."

As described above, if one intends, one can obtain an in-depth insight into a specific variable in a dataset. This is useful to determine further analyses that are required to do for future analyses.

Very similar to what we did earlier, we can also generate a bar chart for this categorical variable (YoS) by the following *R* command.

```
> barplot(YoS)
```

The resulting bar chart for the above command is presented below.

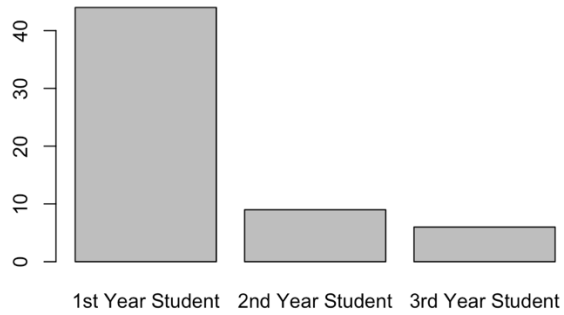


Figure 11: YoS visualization

Similar to what we did with the RGender variable, this barplot can also be made more meaningful by adding a couple more components to the code. Here is an example with the output. Just to make things interesting, this time, I changed the color of the barplot.

```
barplot(YoS, col = "lightblue", main = "Year of Study")
```

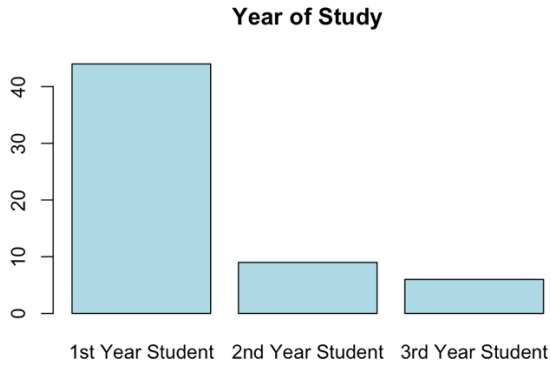


Figure 12: YoS visualization with new colors

With what we have done in the above analyses, I hope now you have some understanding of analyzing categorical variables and making sense of them. Now, from this, let's move to discuss further analysis to do with our continuous data.

Working with Continuous Data

Considering what we have examined in our categorical data, I think we have a world to explore here with our continuous data. To enable my work with continuous data, we can load our packages.

```
> pacman::p_load(pacman, dplyr, psych)
```

If you want to install a specific package, let's say **'psych'** then you can add this code.

```
install.packages ("psych)
load(psych)
```

Now, the first thing I am going to do is to obtain some summary statistics about my continuous variable in the 'Cleaned_Data' named 'Age.' For that, I am going to add this code.

```
summary_stats <- describe(Cleaned_Data$Age)
> print(summary_stats)
```

Before proceeding further, let's try to decipher the above code. I have listed the main components of the code and their explanations.

- **describe(Cleaned_Data$Age):** Here the describe() is used with the argument "Cleaned_Data$Age." To understand this first start with "Cleaned_Data$Age." The $ says the variable 'Age' of our data file named "Cleaned_Data." This code essentially describes the "Age" variable of the dataset named "Cleaned_Data" simply put.

- **summary_stats <- describe(...):** Once the results are obtained for **describe(Cleaned_Data$Age),** store the results in a variable called "summary_stats." In R Studio **<-** operator is used to assign the results of a calculation or function to a variable. If you recall, we used the same operator when we were defining variables as well.

- Accordingly, we enter the first code, summary_stats <- describe(Cleaned_Data$Age)

- **print(summary_stats):** After the first code is entered, print() provides the results stored in "summary_stats." I have given the output below. Let's see whether we can make sense of the obtained results.

```
   vars  n  mean  sd  median  trimmed  mad  min  max  range  skew  kurtosis  se
X1    1  59  21   1.64  21      20.96   1.48  17   27    10    0.6   1.86     0.21
>
```

Unlike the output of categorical variables, here for this continuous variable, we have quite a lot of information. Let's try to understand what each of these columns tells us about our variable, 'Age.'

- **vars:** variable name, X1.

- **n:** sample size, or the number of observations. The variable X1 has 59 observations. So, one could say there are 59 individual informants associated with this variable.

- **mean:** Here, the mean is the average of the variable. In this case, the mean age is 21 years for the sample of 59.

- **sd:** Standard deviation, which is the variability of data, given at 1.48.

- **median:** The middle value of the data, which is also 21.

- **trimmed:** Trimmed mean is the average calculated upon removing a specific percentage of extreme values from both ends of the data.

- **mad:** The "mad" column represents the median absolute deviation, a measure of data variability that is less sensitive to extreme values than the standard deviation.

- **min:** Minimum value of the variable. In this analysis, it is 17.

- **max:** Similar to the minimum of 17, the maximum is 27. This means the age varies between 17 and 27 years.

- **range:** Max – Min (27 – 17) = 10.

- **skew:** Symmetry of the data distribution. Usually, if these values range between -2 and +2, and close to zero, the distribution is mostly likely normal. This can visually be observed using histograms (discussed in later sections). In this analysis, skewness is 0.6 which is mostly likely to be normal, but there is a slight right skew.

- **kurtosis:** Kurtosis measures the "tailedness" of the data distribution. A positive kurtosis indicates heavy tails (more extreme values), while a negative kurtosis indicates light tails. For "X1," the kurtosis is 1.86, suggesting slightly heavier tails than a normal distribution.

- **se:** Standard error of the mean which estimates a variability of 0.21 for the sample mean.

If you look closely at our output, it gives a wide range of descriptive information about central tendency (mean, median, mode), variability (standard deviation), and the nature of our distribution (skewness, kurtosis). Information about variability and the shape of the distribution is extremely useful in most types of inferential statistics as demonstrating normality is an assumption in the parametric strain of inferential statistics.

In my original Excel dataset, I have created a total variable for both **GSES** and **GAD7**. I did this in Excel itself to avoid unnecessary hassle. I performed the same commands I ran for **Age** to understand the nature of the **GSES** variable. This is also a continuous variable.

```
GSES_Summary <- describe(Cleaned_Data$GSESTotal...15)
> print(GSES_Summary)

     vars  n  mean   sd median trimmed  mad min max range skew kurtosis   se
X1      1 59 29.12 3.61     29    29.1 2.97  20  38    18 0.01     0.08 0.47
>
```

Can you try to interpret the above data? Now, unlike the Age variable, if you look at skewness and kurtosis values, both of them indicate a greater degree of normality. So, this data is most likely to be normally distributed. We can visually observe normality and also test using a normality test like *Kolmogorov-Smirnov* or *Shapiro-Wilk* test. Similar to running analyses for one continuous variable at a time, we can further do statistical analyses by dividing the **GSES** total by a categorical variable like **Gender**. Here is the sample code and the resulting output. Here, take note that I have used my original '**Gender**' variable instead of the newly defined '**RGender**.'

```
> mean_by_gender <- Cleaned_Data %>%
+   group_by(Gender) %>%
+   summarise(Mean_GSESTotal = mean(GSESTotal...15, na.rm = TRUE))
```

Once you enter the above, then enter the final line.

```
> print(mean_by_gender)
```

Here is your output. Take a moment to observe the output. It provides the mean of **GSES** based on the levels of our categorical variable, **Gender**.

```
# A tibble: 2 × 2
  Gender Mean_GSESTotal
  <chr>           <dbl>
1 Female           28.4
2 Male             30.0
>
```

Now that we have completed this analysis, let's try to further decode this *R* command.

- **mean_by_gender <-** ...: Here, we are creating a new variable called "mean_by_gender" to store the results of our calculation. <- indicates the results are assigned to this variable.

- **Cleaned_Data %>%..:** Here, %>% is a pipe operator. Usually, we use this in R to chain together a series of statistical operations. What this code says is to start with the 'Cleaned_Data' and then perform the next set of operations.

- **group_by(Gender) %>%** ...: Here, we're grouping the data by the "Gender" variable. So, we are separating the dataset into two groups because Gender has two distinct types of values one representing Males and the other representing Females.

- **summarise(Mean_GSESTotal = mean(GSESTotal...15, na.rm = TRUE)):** Within each Gender group, now we want to calculate a summary statistic which is GSES mean. Please note the variable that contains the composite score of GSES in my uploaded dataset is named GSESTotal…15. The function summarise() is to create this summary. Essentially, this says "For each group, calculate the average of the 'GSESTotal...15' variable and call it 'Mean_GSESTotal'."

- **mean(GSESTotal...15, na.rm = TRUE):** This part calculates the mean (average) of the "GSESTotal...15" variable within each group. The na.rm = TRUE part tells R to ignore any missing values (NA) when calculating the mean.

So, in simple terms, this code is grouping the "**Cleaned_Data**" by gender, and for each group (e.g., males and females), it calculates the average of the "**GSESTotal...15**" variable. The results are stored in a new variable called "**mean_by_gender**," which will contain the average values for each gender group. This can help us understand how the "**GSESTotal...15**" variable differs between different genders in the dataset. You can add further functions to the same code. For instance, for the above analysis, I added another command to generate standard deviation as well. Given below is the input and the output.

```
summary_by_gender <- Cleaned_Data %>%
+   group_by(Gender) %>%
+   summarise(
+     Mean_GSESTotal = mean(GSESTotal...15, na.rm = TRUE),
+     SD_GSESTotal = sd(GSESTotal...15, na.rm = TRUE)
+   )
> print(summary_by_gender)

# A tibble: 2 × 3
  Gender Mean_GSESTotal SD_GSESTotal
  <chr>          <dbl>       <dbl>
1 Female          28.4        3.45
2 Male            30.0        3.67
>
```

I think now with these commands we can carry out quite a lot of descriptive analysis. For instance, we can see whether the **GSES** score varies based on YoS and we can perform similar operations to **GAD7** as well. It is practically not possible to come up with brand-new commands for newer analyses if we are new to the *R* language. But, by following the above steps, we can utilize our understanding to generate newer codes through **Chat GPT**. Although we discourage students from using *Chat GPT* to write essays, it is a powerful tool when used for learning programming languages. However, make sure you give appropriate commands. Sometimes, the scripts of *Chat GPT* do not work on *R*. You should always indicate the package you are using, and at times you need to have a conversation with it to obtain the correct code. However, it is still a beneficial tool. Since I am new to programming, I generated all the above commands from *Chat GPT*. With time, you will learn the language, so, until that time, we can use *Chat GPT*'s assistance.

Histograms

Now that we have done some descriptive analyses on our continuous variables, let's draw a histogram for our **GSESTotal** variable. You may enter the below-given command. Please note that direct copying and pasting might not work, especially since I have a unique name for my GSESTotal variable. Instead of GSESTotal, it has named the same variable GSESTotal…15 when loading the *Excel* sheet to *R*. If you manually type them, *R* will prompt the variable list for you to select to complete the command.

```
> hist(Cleaned_Data$GSESTotal...15)
```

When you enter the above code, you should get a histogram like the one displayed in figure 10.

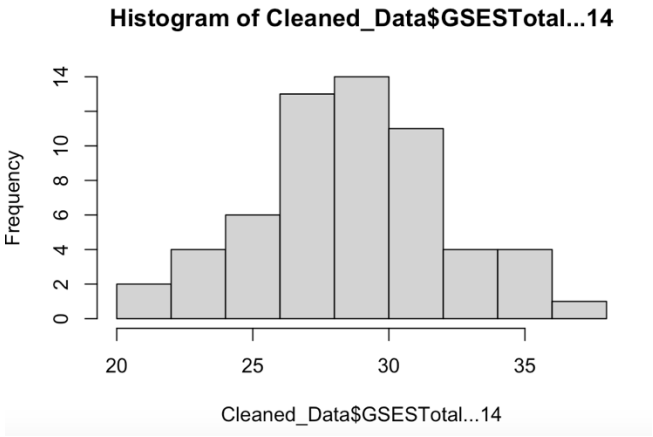


Figure 13: Histogram

A histogram is a good way to observe the distribution of data for a continuous variable. Further, this is a great method to observe the normality of the distribution. As per the descriptive statistics we obtained earlier, both skewness scores and kurtosis scores are closer to zero. So, we know our data is normally distributed. We can observe the same visually here. Just like how we added new commands to our bar charts, in this example also, we can add new labels. Take a moment to observe the below-given code and the resulting output.

```
> hist(Cleaned_Data$GSESTotal...15, col = "lightblue", main = "Histogram with Normality Curve")
```

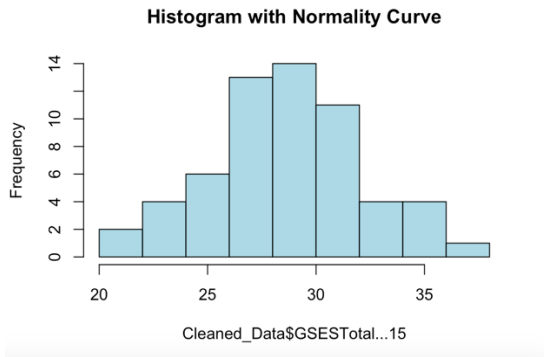


Figure 14: Histogram with new colors added

Unlike the previous histogram, in this new histogram, we have changed the title and also the color as well. If you want to add a new **x label**, you can simply add **xlab = "GSES**" to the last bit of the code. Thus, your new code will look like this. Similarly, the title is represented by **main = "Histogram with Normal Curve."**

```
hist(Cleaned_Data$GSESTotal...15, col = "lightblue", main = "Histogram with Normal Curve", xlab = "GSES")
```

This code will make sure you have a much better histogram with a proper title, color, and labels. Let's make this histogram white to make this *APA*-friendly.

```
hist(Cleaned_Data$GSESTotal...15, col = "white", main = "Histogram with Normality Curve", xlab = "General Self Efficacy")
```

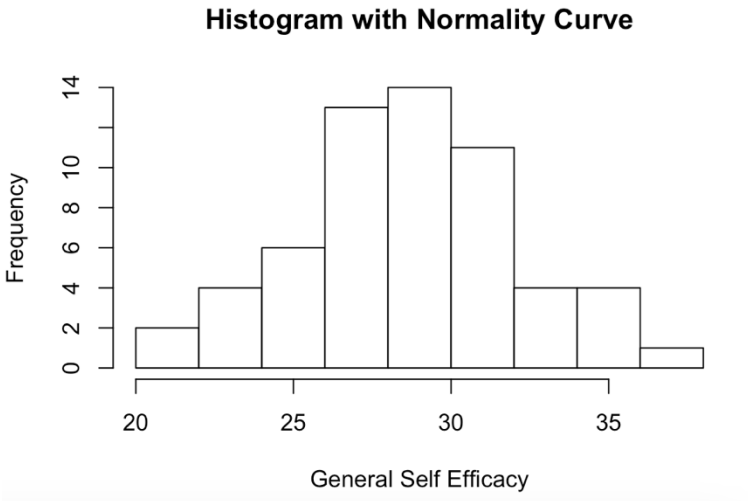


Figure 15: APA formatted histogram

In this example, I have not included the normality curve although I have named the label so. In the proceeding section, we will be discussing how to inspect the normality of a variable visually using a histogram and a normality curve.

7.0 Normality Testing on R

You will notice, similar to running a bar chart, I just now ran a histogram for the GSES Total variable. Here, again, I have given the code and the output.

```
> hist(Cleaned_Data$GSESTotal...15, main = "Normality Testing for GSES", xlab =
"GSES")
```

Look at my output now.

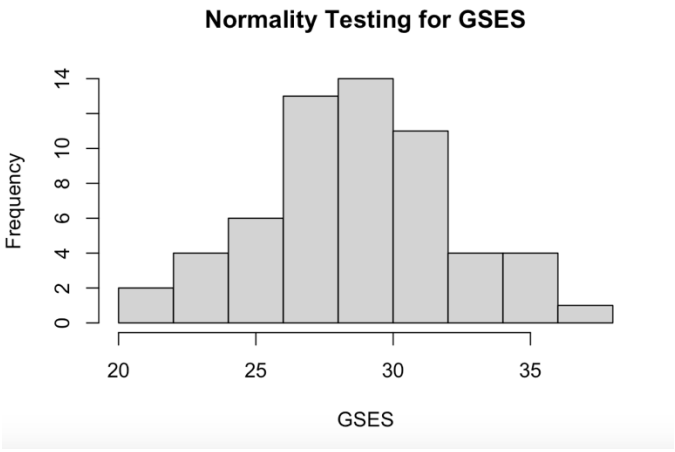


Figure 16: Histogram header and x labels included

Although generating a histogram is an easier job, developing the normality curve is a bit of trouble. This has 7 steps. Even if you use ChatGPT to generate your commands, it still takes multiple attempts to get them right. Here I have demonstrated how I did it. You can follow this step by step. If by any chance, you get error messages in one of the steps, a different system has to be used. However, with the dataset I have used, assuming you followed all the steps in this book as it is, this, R script should run well. Let's take a moment to review the steps associated with obtaining a histogram with a normality curve.

Histogram with normality curve

Step 1: Create the histogram

```
hist(Cleaned_Data$GSESTotal...15, col = "lightgray", main = "Histogram with Normal
Curve", xlab = "General Self Efficacy")
```

Step 2: Calculate mean and standard deviation of the data. The operator **<-** should immediately indicate what we are doing here. We are assigning results to new variables.

```
mean_value <- mean(Cleaned_Data$GSESTotal...15)
sd_value <- sd(Cleaned_Data$GSESTotal...15)
```

Step 3: Create a sequence of **x** values for the curve

```
x <- seq(min(Cleaned_Data$GSESTotal...15), max(Cleaned_Data$GSESTotal...15), length =
100)
```

Step 4: Calculate the corresponding y values using **dnorm()** for the normal distribution.
Previously defined variables of step 2and 3 are included here in this code. Take note.
y <- dnorm(x, mean = mean_value, sd = sd_value)

Step 5: Determine the maximum count in the histogram
max_count <- max(hist(Cleaned_Data$GSESTotal...15, plot = FALSE)$counts)

Step 6: Scale the y values to fit the histogram, considering the maximum count
y <- y * max_count / max(y)

Step 7: Add the normal distribution curve to the histogram
lines(x, y, col = "red", lwd = 2)

If all your commands run smooth, this is your output.

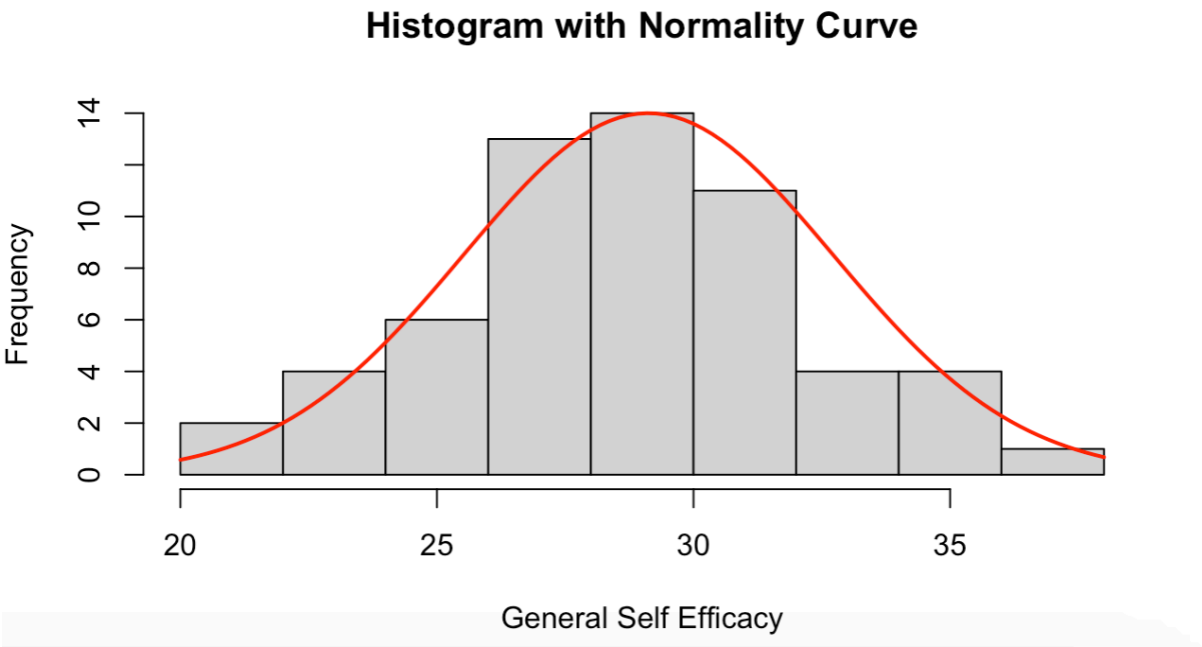


Figure 17: Histogram with normality curve

Now, you probably must already have felt that this script is way too much work to get the normality curve. So, let's try to understand the logic here.

Let's try to understand the logic of the earlier code.

Step 1: Create the histogram
hist(Cleaned_Data$GSESTotal...15, col = "lightgray", main = "Histogram with Normal Curve", xlab = "General Self Efficacy")

This is an easier step. We just created a histogram like we did in the previous section of this book. To make our histogram look better, we have given a color, title, and x label as well.

Step 2: Calculate the mean and standard deviation of the data
```
mean_value <- mean(Cleaned_Data$GSESTotal...15)
sd_value <- sd(Cleaned_Data$GSESTotal...15)
```

This step is also self-explanatory. We have found the mean and standard deviation of GSESTotal…15 variable and have assigned the results to two new variables named mean_value and sd_value.

Step 3: Create a sequence of **x** values for the curve
```
x <- seq(min(Cleaned_Data$GSESTotal...15), max(Cleaned_Data$GSESTotal...15), length = 100)
```

This step is very new to us. To draw a smooth curve, we need a bunch of x values that cover the range of our data. Think of this like creating a line of points on a graph. Now, you might wonder whether this length can be changed. Usually, a larger length offers a finely spaced sequence as opposed to a shorter length.

Step 4: Calculate the corresponding y values using **dnorm()** for the normal distribution
```
y <- dnorm(x, mean = mean_value, sd = sd_value)
```

In this step, we use **dnorm()** function to calculate y values for a normal curve. This curve helps us to see if our data looks like a bell-shaped curve.

Step 5: Determine the maximum count in the histogram
```
max_count <- max(hist(Cleaned_Data$GSESTotal...15, plot = FALSE)$counts)
```

In this step, we figure out how tall the tallest bar in our histogram is. This helps us make sure our curve fits nicely on top of the histogram.

Step 6: Scale the y values to fit the histogram, considering the maximum count
```
y <- y * max_count / max(y)
```

In this step, we adjust the height of our curve so it matches the histogram. It's like stretching or squeezing the curve to make it fit nicely on top of the bars.

Step 7: Add the normal distribution curve to the histogram
```
lines(x, y, col = "red", lwd = 2)
```

In this final step, we draw a red colored curve on top of the histogram. This is the curve that helps us to see if our data looks like a bell-shaped curve. The col = ‘red’ gives the color and 1wd = 2 gives the thickness of it.

Phoof!! I do not know about you guys, I already feel like a programmer.

Shapiro-Wilk for Normality Testing

Now that we know how to visually observe normality using histograms, let's see whether we could do the same using a test dedicated to investigating normality. There are two commonly used tests for this. One is the *Kolmogorov-Smirnov* test and the other is the *Shapiro-Wilk* test. Here, I have chosen *Shapiro Wilk* as this is a test ideal to be used when you have smaller samples. Also, *Shapiro Wilk* is a very sensitive test making it detect even slight deviations from normality. Finding normality is a simple process compared to what we did to generate normality curves. I have given both the code and the output for **GSES**.

```
> shapiro.test(Cleaned_Data$GSESTotal...14)

        Shapiro-Wilk normality test

data:  Cleaned_Data$GSESTotal...14
W = 0.98463, p-value = 0.6611
```

Well, can you interpret? Our variable is normally distributed as per the *Shapiro-Wilk* test. The same was confirmed during our visual inspection of normality using histograms. Usually, we encourage students to observe normality using multiple methods. You can observe normality descriptively through skewness/kurtosis scores, visually through histograms, *q-q plots*, and *p-p plots*, and lastly through statistical tests such as *Shapiro-Wilk*. In this study, **GSES** is normally distributed as per all of these three.

Let's do the same for our **GAD7** Total variable as well. I first decided to go for my descriptive analysis. I added the following code, and here is my output.

```
GAD_Score <- describe(Cleaned_Data$GAD7Total)
> print(GAD_Score)

   vars  n  mean   sd  median  trimmed  mad  min  max  range  skew  kurtosis   se
X1    1  59  7.53  3.61     7     7.37  2.97    0   23     23   1.1     3.89.  0.47
>
```

Based on the above findings, our data indicates both a noticeable skew and heavy tails indicating extreme values. Now, my next option is to go for a visual inspection to further investigate this. So, I decided to generate a histogram. Given below is my R command.

```
> hist(Cleaned_Data$GAD7Total, main = "Normality Testing for GAD", xlab = "GAD")
```

The output is given on the next page (figure 9). Needless to say, the histogram looks very skewed. With the information I already have obtained using descriptive measures and visual inspection methods, I am pretty sure Shapiro-Wilk will say this variable is not normally distributed. But, let's check nevertheless.

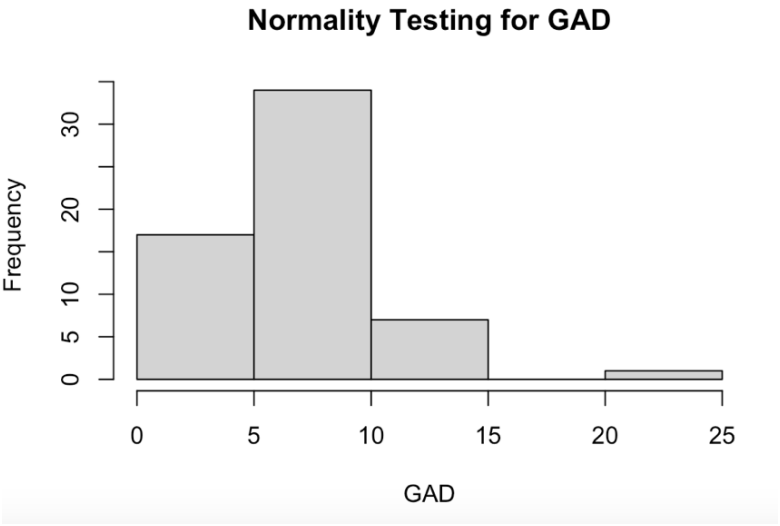


Figure 18: Histogram for GAD

So, this is the code for Shapiro Wilk and the output. Can you interpret it this time?

```
> shapiro.test(Cleaned_Data$GAD7Total)

        Shapiro-Wilk normality test

data:  Cleaned_Data$GAD7Total
W = 0.91862, p-value = 0.0007543
```

Similar to the previous two methods, Shapiro-Wilk also confirms GAD7 to be not normally distributed. Thus, in our study, one of our variables is normally distributed, and one is not. Now, I have more decision-making to do before proceeding with inferential statistical measures.

I further decided to go ahead and inspect the histogram based on gender categories. This requires a different code. I have given it below with the output. Can you see whether you could try to interpret the code?

```
hist(Cleaned_Data$GAD7Total[Cleaned_Data$Gender=="Male"], main = "Histogram for Males", col = "lightblue", xlab = "GAD7")
```

I performed the same for females by slightly changing my code. I have given my code below.

```
hist(Cleaned_Data$GAD7Total[Cleaned_Data$Gender=="Female"], main = "Histogram for Females", col = "lightblue", xlab = "GAD7")
```

Both the outputs of the above codes are demonstrated on the next page. Based on the obtained results, either men have no anxiety, or women tend to be more truthful in their answers. There is a clear gender-related difference here. Such differences can further be observed through inferential statistical measures such as T-Tests. However, that is not the objective of this book. I will make sure to include them in a later version of this book. For now, I am satisfied with these.

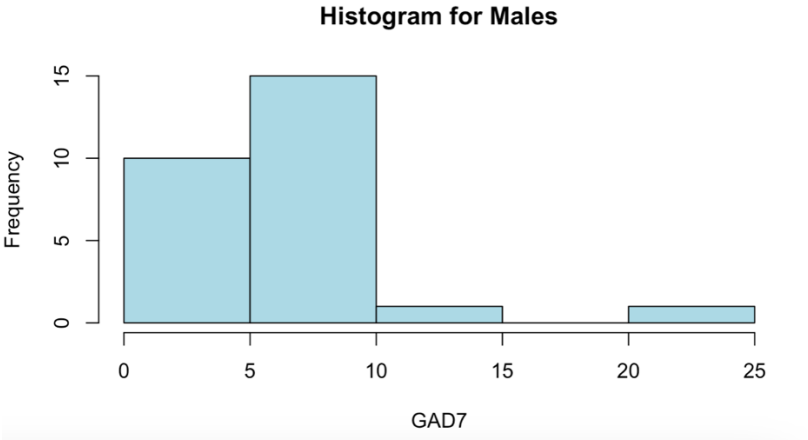


Figure 19: Histogram for GAD7 (Males)

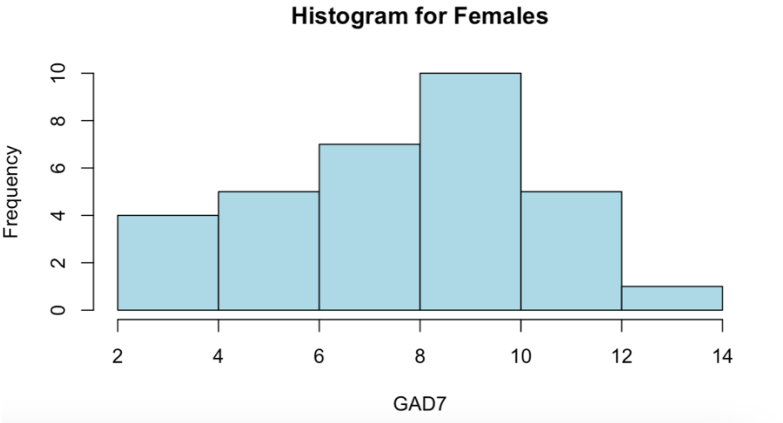


Figure 20: Histogram for GAD7 (Females)

I think the lesson on normality is the biggest lesson here. However, I think now all of you have a good understanding of normality testing. Now, take a moment to recall the study I have included in section 2. Our intention was to investigate the relationship between general self-efficacy and anxiety. Let's start with that, shall we?

8.0 Running a Correlational Analysis on R

We have slowly approached the final lesson in this short book. If you recall our study from section 2, we are trying to find a relationship between self-efficacy and general anxiety. First, let's develop our hypothesis.

H_1: Self-efficacy and anxiety share a statistically significant relationship

Since this book is intended for first-semester students, I decided to investigate this hypothesis through a correlational analysis. Now that we know one of our variables has violated the assumption of normality, I am thinking of using Spearman correlation to investigate this relationship. Surprisingly running a correlation test is far easier compared to generating histograms and normality curves. Here is the code and the output.

```
> cor.test(Cleaned_Data$GSESTotal...15, Cleaned_Data$GAD7Total, method = "spearman")

        Spearman's rank correlation rho

data:  Cleaned_Data$GSESTotal...15 and Cleaned_Data$GAD7Total
S = 49420, p-value = 0.000425
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
-0.4441752
```

I am not going to explain the code as it is self-explanatory. By now we have learned multiple functions and operators. So, by now you should be able to dissect these codes and understand them. Based on the output, self-efficacy, and anxiety share a statistically significant negative relationship, r_s = -.444, p = 0.000425.

To further make things interesting, I also ran a correlational analysis separately for men and women. Take note of how I have changed my R commands. First I have given the analysis for men, and then women.

Analysis for men

```
> cor.test(Cleaned_Data$GSESTotal...15[Cleaned_Data$Gender == "Male"],
Cleaned_Data$GAD7Total[Cleaned_Data$Gender == "Male"], method = "spearman")

        Spearman's rank correlation rho

data:  Cleaned_Data$GSESTotal...15[Cleaned_Data$Gender == "Male"] and
Cleaned_Data$GAD7Total[Cleaned_Data$Gender == "Male"]
S = 4863.5, p-value = 0.01042
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
-0.4845797
```

Based on the output, self-efficacy, and anxiety for males share a statistically significant negative relationship, r_s = -.4845797, p = 0.01042.

Analysis for women

```
> cor.test(Cleaned_Data$GSESTotal...15[Cleaned_Data$Gender == "Female"],
Cleaned_Data$GAD7Total[Cleaned_Data$Gender == "Female"], method = "spearman")

        Spearman's rank correlation rho

data:  Cleaned_Data$GSESTotal...15[Cleaned_Data$Gender == "Female"] and
Cleaned_Data$GAD7Total[Cleaned_Data$Gender == "Female"]
S = 7528.9, p-value = 0.03197
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
-0.3799251
```

Based on the output, self-efficacy, and anxiety for males share a statistically significant negative relationship, r_s = -.3799251, p = 0.03197. It is interesting to note that females have a lower correlational coefficient compared to men. This could be due to the fact that women have more spreaded scores for anxiety as opposed to men (*Figure 19 and Figure 20*).

Drawing the Scatter Plot and the Best-Fit Line

To further complete the analysis, I also drew the scatter plot and the best-fit line. I have given the input and output below for your reference. A negative correlation is clear.

```
> plot(Cleaned_Data$GSESTotal...15, Cleaned_Data$GAD7Total,
+      xlab = "GSESTotal...15", ylab = "GAD7Total",
+      main = "Scatter Plot")
> fit <- lm(Cleaned_Data$GAD7Total ~ Cleaned_Data$GSESTotal...15)
> abline(fit, col = "blue")
>
```

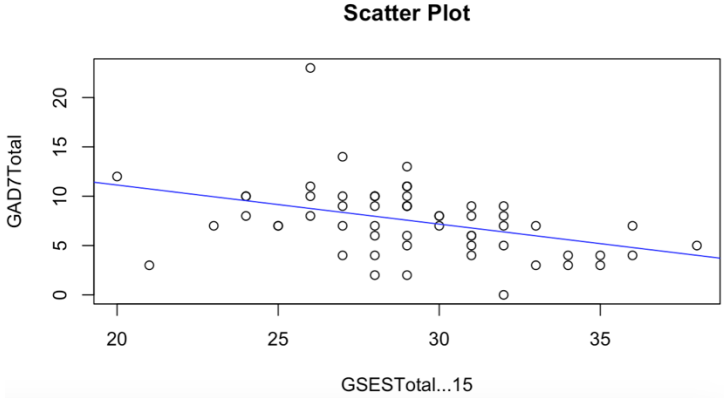


Figure 21: Scatter plot and best-fit line for GSES and GAD7

9.0 Concluding Remarks

Wow, what a relief!!!!

We must all be proud of ourselves. We have now mastered the basics of a new programming language. It is my understanding that this book helps you see *R* as a valuable software for statistical analyses. I wrote this book to help my first-year students develop key skills in *R* Studio to help them run simple studies that require nothing more than a correlation. One might argue that all of these can easily be done using *SPSS*. That is definitely correct, however, *SPSS*'s functionality is far simpler compared to the greater benefits *R* offers. In that sense, learning this will help students, especially during their graduate programs.

Another bigger problem I always felt using *SPSS* was the readymade nature of *SPSS* output. Compared to that, *R* offers me great flexibility and customization. In that sense, *R* is an excellent alternative everyone should master. I also believe that psychology graduates should possess marketable skills when they enter the job market. I think knowing a little bit of programming for statistical computing opens up a new set of avenues to succeed in life. That being said, I should also mention that I started my career as a junior statistical analyst for a USA-based strategy consulting firm.

This book is a good read if you are totally new to *R* Studio. However, keep in mind that learnings from this book is limited for testing associations. For now, I complete this book with correlations. Soon, in the future, I will publish a subsequent edition of this book with elaborations for parametric and non-parametric group difference testing.

10.0 Definitions

1D array:
A 1D array is a linear collection of data elements, such as a list of numbers or characters in a given row.

2D array*:
A 2D array is a two-dimensional structure that stores data in rows and columns, forming a grid or table, commonly used for tables of data.

Categorical variables:
Categorical variables represent data that can be divided into distinct categories or groups, variables like gender, colors, type of vehicle brand, and yes/no responses are all categorical data. We also call yes/no type of answers dichotomous responses.

Constructs:
Constructs refer to theoretical concepts or ideas that researchers aim to measure or study. They are often abstract and may involve multiple variables. One has to develop an operational definition to convert it to a variable so that it can be measured.

Contingency table*:
A contingency table is used to display the frequency of two or more categorical variables' combinations, helping to analyze relationships between them.

Continuous variables*:
Continuous variables are those that can take any real value within a range, such as height or weight, and can have an infinite number of possible values.

Data wrangling*:
Data wrangling involves the process of cleaning, transforming, and organizing data to make it suitable for analysis.

Data tidying*:
Data tidying refers to the practice of structuring datasets to make them more organized, consistent, and ready for analysis using specific conventions.

Debugging*:
Debugging is the process of identifying and fixing errors or issues in computer code or scripts, ensuring they run correctly.

Effect size*:
Effect size measures the strength of a relationship or the magnitude of an effect in statistical analysis, often used in hypothesis testing.

General Linear Model*:
The General Linear Model (GLM) is a statistical framework used to analyze relationships between variables, including linear regression, ANOVA, and ANCOVA.

Generalized Linear Model (GLM)*:
The Generalized Linear Model (GLM) extends the GLM to handle a broader range of data types and distributions, such as binomial or Poisson.

Item Response Theory:
Item Response Theory (IRT) is a statistical model used in educational and psychological testing to evaluate the difficulty and discrimination of test items. Rasch model, graded response model, etc. are commonly used statistical methods in IRT.

Latent traits*:
Latent traits are unobservable characteristics or attributes that can influence or underlie observed behaviors or responses.

Latent Trait Modeling (LTM)*:
Latent Trait Modeling (LTM) is a statistical approach used to estimate latent traits and their relationships with observed variables.

Metadata*:
Metadata is data that describes other data, providing information about the content, structure, and context of datasets.

R Console*:
The R Console is an interface in *R* Studio where you can enter *R* commands and see their output, making it a hub for working with R.

R Studio*:
R Studio is an integrated development environment (IDE) for *R* that provides tools and features for data analysis, visualization, and code development.

SPSS:
Statistical Package for Social Sciences, the go-to statistical software for psychology students. I guess I do not even need to give a description.

SPSS Amos:
An extension to SPSS for researchers to conduct confirmatory factor analysis and structural equation modeling. This helps researchers understand complex relationships between multiple variables.

Statistical significance:
Statistical significance indicates whether an observed effect in data is likely to be real or just due to chance, often determined through hypothesis testing (this gives you the *p.* value).

Structural Equation Modeling (SEM)*:
Structural Equation Modeling (SEM) is a statistical technique used to analyze complex relationships among variables, often involving latent constructs.

The definitions marked with '*' are generated from ChatGPT.

NAREN D. SELVARATNAM

Naren is an accomplished academic and researcher with over 8 years of teaching experience and expertise in psychology, education, and management. Notably, Naren has played varied roles in successfully managing, developing, and/or instructing in franchised psychology and management programs of prestigious European institutions including Coventry University, Northampton University, Liverpool John Moores University, Vern Polytechnic, and London Metropolitan University.

As a PhD-trained researcher, Naren boasts a notable record of research publications including Scopus-Indexed articles that delve into macro-social aspects of Sri Lankan politics and governance. His research focal points encompass educational policy, psychotherapy, psychometrics, and psycholinguistics. Proficient in various statistical tools including SPSS, SPSS AMOS, JASP, JAMOVI, and R, Naren also possesses advanced skills in interpretative phenomenology for both psychological and sociological research.

Beyond academia, Naren is a dynamic entrepreneur with a diversified business portfolio spanning education, healthcare, and real estate. He is a distinguished academic and researcher with a solid educational foundation from the United States and Malaysia where he earned four degrees including a Ph.D. at near completion.