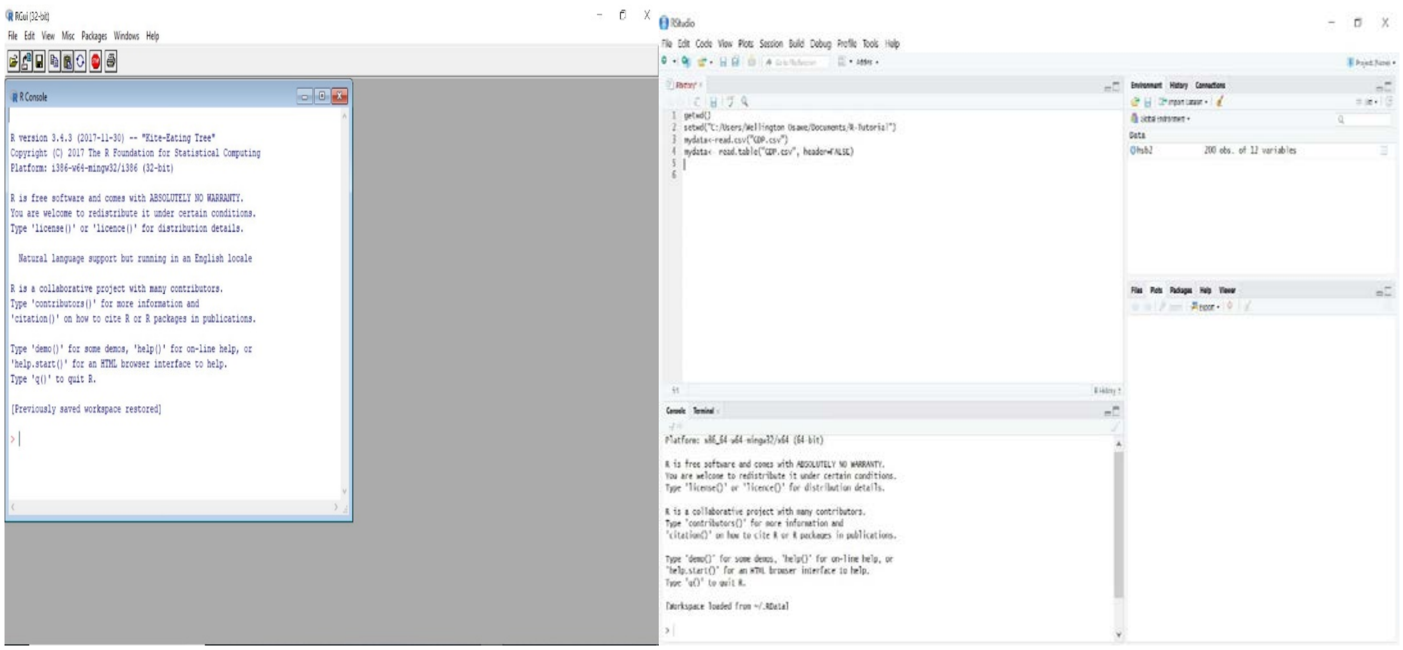
Nigeria Agricultural Policy Project (NAPP) April 2019

R-TRAINING MODULE


RGui (32-bit)
File Edit View Misc Packages Windows Help
R Console
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Environment History Connections
Files Plots Packages Help Viewer
Console Terminal

```
R version 3.4.3 (2017-11-30) -- "Kite-Eating Tree"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
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.

[Previously saved workspace restored]

>
```

```
getwd()
setwd("C:/Users/Wellington Osawe/Documents/R-Tutorial")
mydata<-read.csv("GDP.csv")
mydata<- read.table("GDP.csv", header=FALSE)
```

```
Data
hsb2        200 obs. of 12 variables
```

```
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
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.

[Workspace loaded from ~/.RData]

>
```

By

Dr. Osayanmon Wellington Osawe, FTF NAPP Project Scholar
Mrs. Blessing Iveren Agada, FTF NAPP Project Scholar
Supported by Dr. Michael Olabisi, Michigan State University

Table of Contents

SECTION ONE ... 3

1. Manual Overview ... 3
2. Introduction to R – Downloading R and R-Studo ... 4
3. Basic R Structure, Commands d Other Useful Resources ... 4
4. Overview of R-Packages ... 7
5. Data Management I – Reading and Writing Data in and out of R ... 8
6. Data Management II – Data Manipulation in R ... 11
7. [CLASS EXERCISE A – Stacking /Appending & Merging] ... 12

SECTION TWO ... 19

8. Descriptive Statistics: Statistical summaries and Data Visualization ... 20
9. Graphics in R – ggplot2 ... 24
10. Overview of Basic Empirical Models in R: Linear Regression ... 25
11. Class Exercises, Group Assignments and Presentation ... 28
12. Help ... 29

This training manual is made possible by the generous support of the American people through the United States Agency for International Development (USAID) under the Feed the Future initiative. The contents are the responsibility of the study authors and do not necessarily reflect the views of USAID or the United States Government

Published by the Department of Agricultural, Food, and Resource Economics, Michigan State University, Justin S. Morrill Hall of Agriculture, 446 West Circle Dr., Room 202, East Lansing,

SECTION ONE

1. Manual Overview

This is an introductory training/workshop in R programming language. It is intended to familiarize you with the basic functionalities in R and to spur your interest in this powerful open source analytical software. It is free and easily downloadable online. It has downloadable packages that are specific to different analytical techniques and models. R is a statistical program widely used in variety of discipline and is particularly, a powerful matrix-oriented programming language that allows you to create user-defined functions for execution. R also has built-in functions and libraries. These libraries and functions are also extensible, thereby allowing users to manipulate existing ones for their unique procedures.

2. Introduction to R – Downloading R and R-Studio

It is very easy to download R software because it is publicly available and a simple Google search will provide links to download the software. R and R-Studio are basically the same. The only difference being in the graphic user interface and aesthetics they each provide. R-Studio offers a rather user-friendly environment that provides more flexibility in terms of aesthetics and interactive-fluidity of the software. The choice of either one does not in any way have any influence on functionality or results obtained while using R. It is a matter of choice. For this workshop however, we will be using R-Studio.

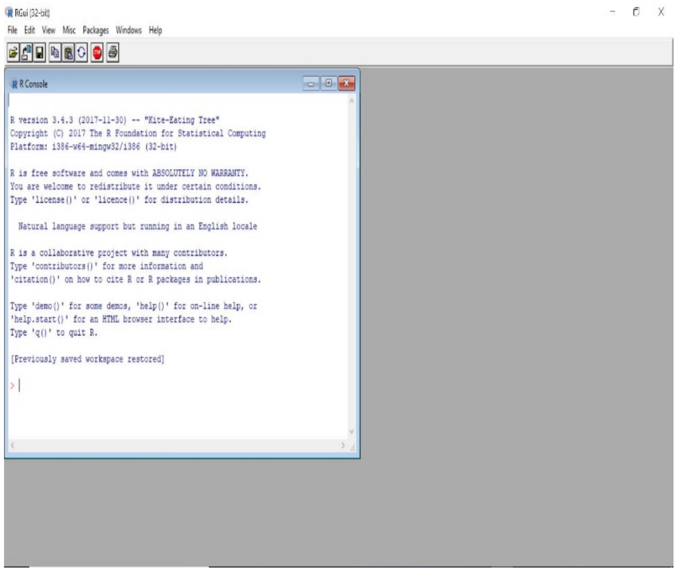
For Microsoft Windows machines, R can be downloaded through the link below:

a. https://cran.r-project.org/bin/windows/base/

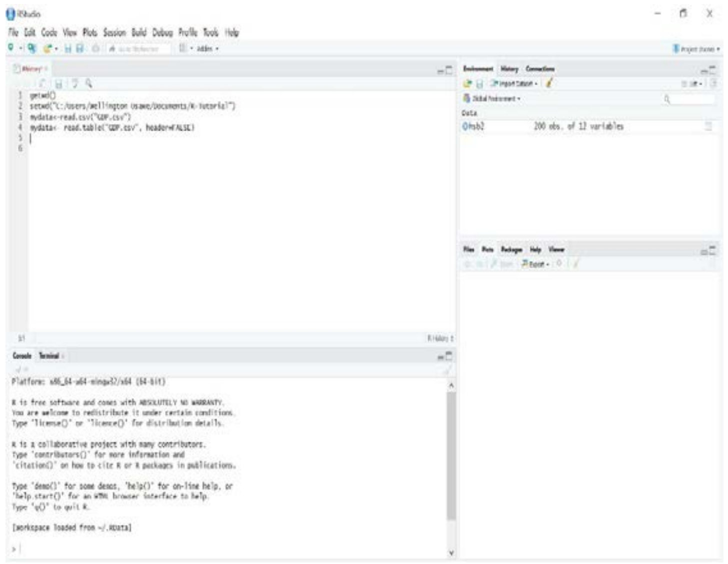
To download R-Studio, simply click on the link below and follow the instructions to download the software based on your computer and operating system:

b. https://www.rstudio.com/products/rstudio/download/#download

3. Basic R Structure, Commands d Other



```
R version 3.4.3 (2017-11-30) -- "Kite-Eating Tree"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
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.

[Previously saved workspace restored]

>
```


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Addins
Environment History Connections
Import Dataset
Global Environment
Files Plots Packages Help Viewer
Export
Console Terminal

```
getwd()
setwd("C:/Users/Wellington Usawe/Documents/R-tutorial")
mydata<-read.csv("GDP.csv")
mydata<- read.table("GDP.csv", header=FALSE)
```

```
Data
hsb2          200 obs. of 12 variables
```

```
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
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.

[Workspace loaded from ~/.RData]

>
```

Useful Resources

R, as an object-oriented programming language, works within the framework of 'object-assignment'. By this, it means, in using the software, you can create an object (in form of a store-house), assign functions to those objects and run any particular statistical analysis on those objects with results displayed on screen.

According to Peter Dalgaard (2008), *Introductory Statistics with R, 2nd edition,* the basic interaction mode in R is mainly that of expression evaluation. This means that the R-user simply impute the command or expression while the machine evaluates and prints out the result. All expressions follow a known syntax that is defined or programmed as a function to be executed.

- **Vectors/Scalars**

  Vectors are a fundamental concept in R, as many functions operate on and return vectors, so it is best to master these as soon as possible. For the technically inclined, in R, a (numeric) vector is an object consisting of a one-dimensional array of scalars (scalars do not have dimension).

  c. > rep (2,5)

  d. [1] 2 2 2 2 2

  The function rep above returns a vector of 2 repeated 5 times. You can get more information on what 'rep' is by typing '?rep' in the command prompt in R. More information on how to explore the Help function in R will be covered in subsequent sections.

  - > ?rep

  You can assign any object (including vectors) using the assignment operator '<-' or '=', and combine vectors and scalars with the 'c' function. "c" means to concatenate or link together.

  - a<- rep (2,5)
  - b<- 1:5

- c<- c(a,b)
- c

  [1] 2 2 2 2 2 1 2 3 4 5

- a+b

  [1] 3 4 5 6 7

Data files in R could be stored either as a "data.table" or as "data.frame". The difference between the two format is that, with data.table, a reference set of vectors are created within the software that allows the analyst to easily make reference to particular column at any given point in time. However, with data.frame, data is stored jointly as a set of column with no particular reference. The implication of this is that, when a current object contains data file in data.frame, you can only assess a particular column by having to call the name of the data frame with a dollar sign ($) and then the name of the column to be referenced. For example:

- > function(dataname$sex)

Note: R stores data.table both as a data.table and a data.frame allowing the user to apply functions that are applicable to both formats only when the data file is stored as a data.table. Therefore, all data.table are also data.frame but every data.frame is not a data.table. A data.frame can be converted to a data.table.

4. Overview of R-Packages

R allows user-defined programmes that enable different analysis to be conducted. In this regard, for certain analysis to be conducted that are not within the R default programme, users have created packages that can be run in R and therefore allows other users who have called these packages to execute functions related to these packages. A typical example is graphical analysis in R. R is widely regarded for the very beautiful and journal-oriented graphics that is associated with it. In order to use these graphical packages, the user must have installed the requisite package (e.g. ggplot) and then instruct R to recognize this package during the analysis. To instruct R to recognize the package that the user needs to use, the function, ***library()*** needs to be called. This function allows R to call into memory, a particular package that had been installed earlier. An example of this follows next.

To install packages in R, say ggplot and instruct R to recognize this package for the purpose of initiating an analysis, you can follow the syntax below at the command prompt:

- > install.packages ("ggplot")

  After the installation, you have to "call" this new package into R anytime an analysis that is related to the package is to be executed by following the syntax below:

- > library(ggplot2)

The same procedure applies for all packages that needs to be installed in R once the user knowns the name of the package.

[EXERCISE WITH CLASS DEMONSTRATION HERE]

5. Data Management I – Reading and Writing Data in and out of R

By default, R can read any file with the extension, csv (comma-separated), txt (text format) or other flat-form data file using the function 'read.table' or 'read.csv'. However, for some other data types such as those with extension '.sav', '.dta' and so on, required packages have to be installed and loaded in the library before R can conveniently read them. The read.table is one of the most frequently used functions for reading in data in R. The syntax can take the form below:

- > read.table ("the path to the filename") or
- > read.table ("the path to the filename", sep=",", header = TRUE) or
- Abc> read.table ("the path to the filename", sep=",", header = TRUE) #This assigns the data file an object named "Abc".

A part from this, just like every other statistical software around, data can be manually read into R from the R prompt/console. The subsections under Vectors and Scalars is a form of manual reading of new data into R. Take another example:

- A = 1:10 #This creates a vector of scalars from 1 to 10
- B = rnorm(10) # This creates a vector of 10 pseudo-random numbers
- x = c(2, 4, 6)
- y = c("ab", "bc", "cd")
- z = c("FALSE", "TRUE", "FALSE")
- xyz = data.frame(x, y, z)
- str(xyz)
- You can use the "colnames" function to change the name of each column or variable name using:
  - > colnames(xyz) = c("newname", "newname", "newname")

    - *Note that the above is only useful when the data file has few columns to be changed and the function applies to all columns in the data.*
    - *To change the name for just one of the columns particularly if it is a large dataset, simply use the syntax below:*
        - *names(xyz)[1]="newname"   #This will change only column 1*
- [ ] - This is a sharp bracket. Anything before the coma is the row and after is the column
    - *e.g. [a, 1:4] -------a is row value and 1:4 is the column description*

You will notice that in identifying numbers, we simply input the numbers without parenthesis. However, for characters, the parenthesis had to be placed so that R recognized them as characters and not numbers. As indicated earlier, the function 'c' means to concatenate. It allows the numbers or characters in the bracket to be linked together as one.

- Writing Data to a file in R

It is a common thing when working with a dataset in any statistical software to save your data for future use. This can easily be done in R. However, it is always a recommended that before saving your current in-use dataset, be sure to know your current working directory so it can be easier to find your information for later use.

- ✓ Use can use the following function to find out your current working directory:
    - getwd()
- ✓ To set your working directory to a preferred choice, use the function:
    - setwd("the path") e.g. setwd("C:/Users/Myname/Desktop/R-Training")
- ✓ Data frames can then be easily written to a file using the function:
    - Write.table(name of dataframe, file = "name of file.txt or .csv", quote = F)

        - E.g. write.table(xyz, file = “exercise1.txt”, quote = F)
        - You can remove the row index:
            - write.table(xyz, file = “exercise1.txt”, quote = F, row.names=F)
    - ✓ Individual objects created in R can also be saved in R using the function: *save()*
        - save(x, y, z, file = “filename.rData”)
        - You can recall these objects using the function: load(“filename.rData”)

If you have a lot of objects that you intend to save to a file in one run, you can save all objects in your workspace using the function: save.image()

[EXERCISE/CLASS WORK ON READING AND WRITING DATA IN R HERE]

- ➢ There are over 100 datasets that are part of the R package following the installation. These datasets can easily be loaded and worked with. To see a list of the datasets that are available, you can use the function:
    - > data ()
    - Take a look at any of the datasets by typing the name of the dataset inside the bracket in the function above. R automatically loads the dataset and you are free to examine the dataset.
    - Provide some basic summary statistics for the dataset named “cars”
    - The help function can be a useful function to get information about the dataset. Type: help(cars)
    - You can also make a simple plot of the key variables in the dataset (we will discuss more on graphics in subsequent sections)

6. Data Management II – Data Manipulation in R

Now that you are familiar with how to read data and understand how the R interface works, this section will use an exercise to teach you how to manipulate your data in R.

e. Merging; Appending; Subsetting etc

There are several ways that datasets can be manipulated in R. Sometimes the analyst has several data frames that needs to be combined. This can take the form of "merging" or "appending". At other times, the analyst may just be interested in trying to isolate a section of the datasets, this is called "subsetting". To illustrate the principle behind these data manipulation techniques and how they can be carried out in R, we will use an exercise.

2. Combining and Merging

There are generally two basic ways in which any data frame can be combined in R. First, the data frames can be stacked on top of each other (append). Second, they could be placed side-by-side (in which case, you simply merge the datasets). In doing this, R can allow the merging process to be done based on common variables or ensuring matched columns and so on following different syntaxes.

7. [CLASS EXERCISE A – Stacking /Appending & Merging]

To illustrate the above data manipulation process, do the following":

Step 1: rbind

Following your knowledge so far, create a data frame called "Me" having the following columns features:

a. A=integers from 1 to 5 [A=1:5]
b. B=5 random numbers from a normal distribution using rnorm[1] [B=rnorm(5)]
c. C=characters with Female, Male, Female, Male, Male

   [C=c("Female", "Male", "Female", "Male", "Male")]
d. Concatenate all into one object of data frame called "Me"

   [Me=data.frame(A, B, C)]

Create another data frame called "To" having the following column features:

- A=integers from 10 to 14
- B=5 random numbers from a normal distribution
- C= characters with Female, Male, Female, Male, Male
- Concatenate all into one object of data frame called "To"

➢ Create a third data frame called "You" having the following column features:

- D=integers of 2,4,6,8,10
- F=characters with F, M, F, M, F
- Concatenate all into one object of data frame called "You"

[1] Occasionally you may want to set the seed in R such that the same random numbers are produced anytime you want to use the function. To do this for instance, you can use the function: set.seed(100)

- Using the function “rbind” – meaning row bind, append the data frames “Me” and “To” into a new object name called append:
    - append = rbind(Me, To)
    - Check to see the implication of the exercise by typing append on the command prompt
    - You will observe that the two data frames have been stacked on each other. This was possible because the columns in the two datasets matched (5 with the same column names)
    - Type class(append) to see if it is still a data.frame or data.table
- Note that rbind can also be used for vectors. Take the example below of using vectors to create a 2x5 matrices:
    - lagos=rbind(1:5, 6:10) or
    - abuja=rbind(rnorm(5), runif(5))
    - check the class

Step 2: cbind

- Using the function “cbind” – meaning column bind, combine the data frames “Me” and “You” into a new object name called “meyou”:
- meyou = cbind(Me, You)
    - Check to see the implication of the exercise by typing merge on the command prompt
    - Now, try the following syntax:
        - metoyou=cbind(rbind(me,to),you)
    - You will observe that two data frames have been combined. First by appending “me” and “to” which means having a new data frame with 10 rows and combing this

new data frame with "you" data frame with just 5 rows. You will notice that even though they have different dimensions, the function still worked. This means that the function does not require that the columns to be the same or with the same row height. Although there is a caveat to this (we will see this later).

  - Type class(append) to see if it is still a data.frame or data.table

- Caveat: While cbind worked even when Me, To and You data frames were not exactly identical, as in the case above, it may not work if the number of rows of the shorter data frame does not evenly divide into the number of rows of the longer datasets as obtainable in the example above. rbind of "me" and "to" data frames resulted into a data frame with 10 rows and "you" data frame has only 5 rows. This worked because "you" has rows that evenly divides into the combination of "me" and "to".
    - Try typing this syntax:
    - cbind(rbind(me,to), you[-1,])   #We have simply omitted one row
- *NOTE: It is noteworthy to state that using cbind and rbind does not produce a data.frame except of course, one of the objects that you are binding is a data.frame*
  - *k=1:5*
  - *kk=letters[1:5]*
  - *kkk=cbind(k,kk)*
  - *is.data.frame(kkk)*

Step 3: Merging

- Merging is used to combine two datasets that have at least one variable common to both datasets. In this instance, the function, merge() is used.
- In some cases, merging can be successful even when both datasets do not have a common matching column or variable. In this instance, a different syntax is used.

- The basic syntax for the function is:
    - Merge(a, b) where "a" and "b" are the datasets to be merged
  - The argument "by" can be included to reference the column(s) to merge by if the two datasets have at least one common variable or column name
  - The argument "by.a" and "by.b" may also be used if there is no common column between the two datasets to be merged
  - *Note: If the two datasets have at least one common column names, then you do not necessarily have to use the "by" argument*

[CLASS EXERCISE B]:

- Using your new knowledge of R, create two datasets following the instructions below:
- Create the following data frames:
  - >great <-data.frame(pid=c(3:10), z1=rnorm(8, 50,2))
  - >mind <-data.frame(pid=rep(1:5, each=2),

    z2=sample(letters, 10, replace=TRUE))
- Now, merge both datasets and name it "greatmind" using the following syntax:
  - greatmind=merge(great, mind)
  - *What do you notice?*
  - *You will see that z1 in data frame "great" is recycled so as to match with the values of the "pid" in data frame "mind". Further, you would have noticed that only rows with matching pid in both data frames are retained. This is called* ***"Inner Join"***
    - *If you are working with a data.table dataset, you can use:*
    - *greatmind=great[mind]*
    - *To set a common variable, use:*

            - *greatmind = setkey(great, pid)[setkey(mind, pid),]*
            - *Note: "great" and "mind" must be data.table for this to work*
        - *In data.table, to remove/delete a column, use the syntax:*
            - *Data[, var1:=NULL]*
            - *e.g. greatmind_pid = greatmind[, pid:=NULL]   #This removes column pid*
    - *To merge all the values in both datasets without reference to any matching, we can use the following argument:*
        - *greatmind2=merge(great, mind, all=TRUE)*
    - *The above syntax is called* ***"Outer Join"***
    - *However, if our objective is to preserve everything in the "great" data.frame and only what matches with it in the "mind" data.frame, we will only specify it in the syntax below:*
        - *greatmind3=merge(great, mind, all.x=TRUE)*
    - alternatively, if our objective is to preserve all values in "mind" and only keep what matches in "great", we follow the same syntax but specified as"
        - greatmind4=merge(great, mind, all.y=TRUE)
- Merging datasets with no matching column/variable names
    - In instances when our objective is to merge two datasets that do not have common column names, the argument "by.x" and "by.y" becomes very useful.
    - Merge the "Me" and "great" data frames for this exercise:
        - all=merge(Me, great, by.x= "A", by.y= "pid")
        - *what do you notice?*

[Reshape a dataset]

- Occasionally, our data may come either as “wide” or “long” format. A dataset with observations having values that occur several times in a ***single row*** is referred to as “wide”. Conversely, when the dataset has several values for a single observation in ***multiple rows***, it is referred to as “long”. Several functions such as those for performing basic estimations and creating graphs require datasets to be in a long format and knowing how to change from wide to long and vice versa, would prove quite significant.
- Example:
  - >wide = data.frame (Sname = c(“Ayoola”, “Ahmed”, “Njoku”),

    exam1 = c(55.5, 89.9, 35.0),

    exam2 = c(80.0, 90.0, 100.0))
- o >long = data.frame(Sname = rep(c(“Ayoola”, “Ahmed”, “Njoku”), each=2),

  exam = rep(1:2, 3), score = c(55.5, 80.0, 89.9, 90.0, 35.0, 100.0))
- To change from one format to the other, the package “reshape2” has to be installed and loaded:
  - install.packages(“reshape2”)
  - library(reshape2)
  - [To change from wide to long]: use the function *melt()*
    - long2=melt(wide, id.vars= “Sname”, measure.vars=c(“exam1”, “exam2”),

      variable.name= “exam”, value.name= “score”)
      - *Note that reshape also has a default “reshape” function in* R

[Subset a dataset]

Subsetting a dataset is one of the useful ways of selecting elements in an object or simply put, it can be used for breaking down huge datasets into smaller structures for analysis. If you are interested in certain portions or sections of a dataset, you can use the *subset()* function to select or exclude the elements in the dataset to reflect only the sections (rows and/or columns) that is of interest to you. With the R object-oriented capability, you are able to create several datasets placed in different objects within the same R interface while working. The basic syntax is:

- ➢ *subset(data, var1 > 100, select = c(var2, var3))*
- ➢ *subset(data, var1 ==1, select = -var2)*
- ➢ *subset(data, select = var1:var4)*
    - o *In a data.table, you can do:*
        - ▪ **Row Subset**
            - *subset.row = data[var1==1& var2=="v"]*
        - ▪ **Column Subset**
            - *subset.col = data[,list(var1, var2)]*
        - ▪ ***Subset a random sample of 20 observations from dataset, data***
            - *newsubset=data[sample(1:nrow(data), 20, replace=FALSE),]*

- **Class Exercises, Group Assignments and Presentation**

SECTION TWO

8. Descriptive Statistics: Statistical Summaries and Data Visualization

Introduction to simple descriptive statistics and graphics - R plot

9. Graphics in R – ggplot2

10. Overview of Basic Empirical Models in R

Estimating Linear Regressions in R

Estimating Anova in R

11. Class Exercises, Group Assignments and Presentations

12. R Help

8. Descriptive Statistics: Statistical summaries and Data Visualization

✓ Introduction to simple descriptive statistics and basic graphics

Descriptive statistics provide a very useful way to examine your datasets by providing important information on attributes of the dataset. Some of the commonly used attributes include mean, median, standard deviation, frequency and percentages. R has default packages loaded at installation with functions for most of these statistical attributes. More so, R default *plot()* function can also be used to create plots for visualizing the data and statistics without needing to load new packages. That said, there are also new packages that can be loaded and called into library for many of these attributes and much more. We will demonstrate some of them in this section.

- For this sub-section, we will be using the dataset "ChickWeight" loaded in R.
- Type ChickWeight in the command prompt
- Examine the dataset. *(Note: It may be wise to put the dataset in an object for easy reference)*
    - data<- ChickWeight
    - str(data)
    - head(data)
    - tail(data)
- Sum and Length #The function *sum* provide the sum of a variable and the *length* shows number of observations of that variable or dataset
    - sum(data$weight)
    - length(data$weight)
- Mean, Median, Standard deviation and Standard error of mean:
    - mean(data$weight) #Mean
    - median(data$weight) #Median
    - sd(data$weight) #Standard deviation

        - sd(data$weight) / sqrt(length(data$weight))   #Standard error
    - *You can use R programming capability to define a programme (function) that executes the estimates above using the following syntax::*
        - stdev <- function(x) {

```
        sumx = sum(data$weight)
        meanx = sumx/length(x)
        dev = (x-meanx)^2
        dev2 = sum(dev)
        var = dev2/(length(x)-1)
        return(sqrt(var))
    }
```

        - *Check to see if you obtain the same value for standard deviation following the function above and the default function in R (sd)*
- Using the package "psych" to produce summary estimates:
    - Install the package "psych", if not already installed and load it
    - Library(psych) and type:
        - describe(data$weight)   ##se indicate the standard error of the mean
        - headTail(data)   ##A function in "psych" that reports first few and last observations in your dataset
        - describe(data)   #Factor var in asterisks *
- Summary:
    - summary(data)
    - summary(data$Diet)
- Quantile:

- quantile(data$weight) #Full quantile percentages
- quantile(data$weight, 0.50) #50th percentile

2. Cross-tabulation for grouped data
   - This is useful for examining summary statistics for grouped data. In our example dataset, we could examine the summary statistics of variable 'weight' of chicks based on the factor variable 'Diet'.
   - To do this, use the *"Summarize()"* function in the package "FSA". You have to install the package first and load it in the library before applying the function. Note that the first letter, 'S' is upper case
     - Install.packages("FSA")
     - Library(FSA)
   - >Summarize(weight~Diet, data=data) #Shows mean of weight by Diet
   - >summary(data$Diet) #Provide counts for levels of nominal var
   - >CT=xtabs(~Diet, data=data) #Provide counts for levels of nominal var
   - >prop.table(CT) #proportion/percentage of the count values
     - For cross-tabulation (two or more factor variables):
     - Let us subset the current data to a new one for ease of description:
     - ab=subset(data, Chick<=30 & weight<=50)
     - >XT = xtabs(~Diet + Chick, data=ab)
     - >prop.table(XT, margin = 1)

✓ **Basic graphics**

Sometimes, it is useful to take a visual examination of some of the variables in your dataset before any meaningful analysis are carried out. R has default functions for visualizing your data. More sophisticated functions will be discussed in subsequent section using R's "ggplot2" package. But for now, let us examine some basic graphical functions:

- hist(data$weight)   #Historam of weight
- plot(aa$weight, col = "red")
- plot(aa$Diet, aa$weight, col = "green")

9. Graphics in R – ggplot2

R's ggplot2 is a graphical package for creating graphics. It provides very flexible structures for easy manipulation and added aesthetics to the usual visualization functions in the default R base. It was created by Hadley Wickham based on the so called *"The Grammar of Graphics"*.

To use ggplot, the package *"ggplot2"* has to be installed first and then loaded, using the *library(ggplot2)* function.

9. In this example, we will use our dataset on ChickWeight
    a. Library(ggplot2)
    b. ggplot(data, aes(x=Time, y=weight)) + geom_point()
    c. ggplot(data, aes(x=Time, y=weight, color=Diet)) + geom_smooth()
    d. ggplot(data, aes(x=Time, y=weight, color=Diet))+geom_point()+geom_smooth()
    e. gg = ggplot(data, aes(x=Time, y=weight, color=Diet)) + geom_point() +

        labs(title= "Scatterplot", x = "Growth rate", y = "Weight gain")

    - **Add a Theme:**
        - gg1=gg + theme(axis.text.x = element_text(size=15),

          axis.text.y=element_text(size=15)) #Increase text size of x & y
    - Facets – create individual plots for each factor variable
        - gg1 + facet_wrap(data$Diet)

    - ***Compare ggplot and R default for graphs:***
        - hist(data$weight)
        - ggplot(data, aes(x=weight)) + geom_histogram()

10. Overview of Basic Empirical Models in R: Linear Regression

In this section, we will be examining causal inferences following the assumptions underlying estimations using the ordinary least squares (OLS) regression model. We will not spend too much time to explain what the model does or does not do, but how objective is to simply introduce you to the basic syntax of running the ordinary least squares regression in R and some other functions that might be useful in your learning process. Some of the basic assumptions of the OLS regression model include the assumption that the underlying relationship of the model parameters are linear, the errors are independent and that the residuals are normally distributed with homoscedasticity. An understanding of these basic assumptions are useful for interpreting the estimates of OLS regression models.

- We will continue with our ChickWeight data for this section
- stat <- ChickWeight
- summary(stat)   #Summary of the data
- cor(stat$weight, stat$Time)   #Correlation between var weight and feeding time
- You can also do a simple scatterplot to visualize the data:
    - plot(stat$Time, stat$weight)

- **Regression model:**
    - reg.mod = lm(weight ~ Time, data = stat)   #regression model
    - summary(reg.mod)   #summary of the results
    - Now let us examine some other useful functions:
        - class(reg.mod)   #Shows the class of model
        - names(reg.mod)   #Provide information on the model features
        - confint(reg.mod)   #Confidence interval for your estimates
        - hist(residual(reg.mod)   #Histogram of residuals

    - You can plot the results of the residuals and predict values:
        - plot(reg.mod, which = c(1,2) or simply
        - plot(reg.mod)
        - *What can you say about the predicted and fitted values?*
- Fit another model by adding the variable Diet to your model as a predictor:
    - reg.mod2 = lm(weight ~ Time + Diet, data = stat)
    - compare the two models to see if indeed Diet predict weight over feeding duration using the *anova()* function:
        - anova(reg.mod, reg.mod2)
    - *Does this new model predict weight better than the previous?*
        - *coef(summary(reg.mod2)) #Check estimates*
        - *plot(reg.mod2, which = c(1,2))*
- **Interactions in LM:**
    - In its simplest terms, interactions in empirical analysis provides a useful way of examining how the relationship between an explanatory variable and the dependent variable depends on another explanatory variable
    - *For example: Does the relationship between feeding time and weight depend on the diet given?*
        - *reg.mod3 = lm(weight ~ Time*Diet, data = stat)*
        - *coef(summary(reg.mod3))*

- **Categorical/factor Variables in OLS Model:**
    - In doing this, it is a wise choice to make R understand that the variable of choice is a factor variable and as such the base value can be chosen by the analyst. Otherwise, R chooses a default base value.
    - Let us examine the var Diet:
        - str(stat$Diet)   #A factor with four levels/categories
        - stat$Diet = factor(stat$Diet)
        - reg.mod4 = lm(weight ~ Diet, data = stat)
        - coef(summary(reg.mod4))
        - anova(reg.mod4)   #Anova table
        - You can choose your specific base/reference group:
        - coef(summary(lm(weight ~ C(Diet, base=2), data=stat)))

- *So far, we have briefly covered the linear models in regression estimation. With several demerits in using LM and in particular, the challenge of assuming a linear structure for the underlying parameters and the fact that it can only fit models with continuous dependent variables, other useful models are available that can easily be fitted and estimated using R. For example, researchers interest in discrete dependent variables may want to employ the generalized linear models using the function glm(). Several helpful online documentations are available for those interested in this and other useful models in R*

11. Class Exercises, Group Assignments and Presentation

12. Help

R has very detailed help documentation file apart from the very large community of experts providing useful resources to help beginners and experts alike. R-bloggers (https://www.r-bloggers.com/) and stackoverflow (https://stackoverflow.com/) are useful starting point for online resources for help on R.

- ?? for general and basic ideas of the function
- ? for specific information about the function you need more ideas on

Extras:

One way to input data in R could be:

```
>Entry = ("
Names        Region  Participation
James        NC           12
Kelvin       SS           20
Jonah        SW           14
Ahmed        NE           06
Bruce        SS           16
")
> data = read.table(textConnection(Entry), header = TRUE)
```

Reference:

Dalgaard, P. (2008) *Introductory Statistics with* R, Second Edition. Springer Science, Newyork.

ggplot: https://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html