Lab 1: Popular Computer Software for Biostatistics and Introduction to Descriptive Statistics

Introduction and Objective:

There is a few popular software commonly used by most biostatisticians. These are the SAS, JMP, SPSS, Prism/GraphPad, and R. Of course, for many easy tasks, even Microsoft Excel can do the job perfectly. Among the software, R is free and nowadays is more popular than most of the other software aforementioned.

We have installed Microsoft Excel, R, SPSS, and JMP on all the computers in our lab. We do not have the license for SAS or Prism/GraphPad, so we did not install them. R and JMP are also installed on some of the computers at the Tech Center. If you are interested in JMP, you can try it over there. It was developed based on SAS, so you may get an idea about SAS based on JMP. Although we do not have Prism/GraphPad on all the computers, I do have a copy on my computer and can demonstrate how to use it.

In today's lab, I am going to introduce to you the basic statistical functions of the software. However, in future labs, we are going to focus on using only one of them, the R. There are two reasons for this. Firstly, R is representing the trend of biostatistics software; secondly, it is the only free software among all the software that has been mentioned. Besides you can use it here in the lab, you can download it to your own computer for free and practice it at home.

I. Prism/GraphPad
   Demonstrate how to use Prism/GraphPad to
   1) input and import data
   2) perform basic statistical analyses

II. Microsoft Excel
   Demonstrate how to use Excel to
   3) input and import data
   4) perform basic statistical analyses

None of the above will be part of the lab final exam.

III. R

1) R and XQuartz Installation
2) RStudio Installation

First, google search "R Project". You will get the webpage for the R project, where you follow the instructions to download and install the version of R compatible with your operating system. There are versions of R for Linux, Mac, and Windows, respectively. Unfortunately, there are no R versions for Chromebooks.

Once you downloaded and installed R, you may choose to update the XQuartz on your computer, if you are using an old Mac. It is not always necessary though. But if your Mac is very old, let's say it was purchased 7 or 8 years ago, you may need to download and install the newest version of XQuartz, which allows you to visualize the R results. There is a link to XQuartz on the website of the R Project.

Only after you successfully installed R, you can google search "RStudio" and then install the appropriate version of it on your computer. RStudio is an IDE (Integrated Development Environment) for R, which comes with a code editor and a debugger. If you have no idea of what an IDE is, you can just think it is like clothes you can put on a "naked" software to make the programming experience more enjoyable. You MUST install R first before you install RStudio on your computer, otherwise, it will not work.

In this lab notes/manual, except in the introduction parts, all the R command lines are in **bold**; all the instruction notes are following a # and in blue; all the R results directly follow the R codes/command lines and are in red.

1) Set Working Directory

The first thing after you start an R session is always to set the working directory. The working directory is a directory where all your work in R is directly processed at. In other words, it is where you store your files for R to retrieve from. Any files you created by R will also be saved to this working directory by default. For instance, in a future lab, we are going to read files into R, and then create files by using write.csv() or write.table(). All these files you are going to read into R must be in your working directory before you can import them into R. And all the files you generate are also saved to the working directory.

The working directory is like a working bench in a biomedical research lab. In the biomedical research lab, you have different cabinets for storing regular chemicals, flammable

chemicals, and corrosive chemicals. You may also have storage space for different equipment. When you perform a certain experiment, you may bring all the reagents and equipment from where you store them to your working bench. Also, any products you generated by these experiments/reactions are on the working bench before you move or store them elsewhere.

In PCs, if you create a folder named "R" in C drive (you can create it in the way that you are used to doing, but in the future, once you are getting familiar with R programming, you can type **dir.create("C:/R")** to have this task fulfilled), and you want to set this folder as your working directory, you can type:

setwd("C:/R")

The "setwd" is a function in R (in a few minutes, we will discuss what an R function is.)

The "C:/R" in the brackets is what the function acts on.

The C:/R is the path for the R folder (the designated working directory).

When you type this command line, make sure you do not omit the quotation marks. Otherwise, you will get an error message (Do not worry, even you get an error message. Every R programmer gets error messages. Keep it in mind. We learn from the mistakes we made.)

If you are using a Mac, the command line you type to set the working directory would be different. Firstly, you need to login into your computer, and then on the desktop, you create a folder and name it as "R". You are going to use this folder as your working directory.

Then in your R session, after a ">", you type **setwd("**

Now you need to find the path of the folder you just created and named as R. To find its path, you can click on this folder, and then go to "File"→"Get Info", drag alongside "Where", on the keyboard press command+C (plus means you have to press the key of command and the letter C at the same time). You can visualize the path by going to "Edit"→"Show Clipboard". In your R session, after **setwd("**, type this path or paste it by press command+V, and then type /R right after the path without leaving any space in between the path and /R. After that you type **")**, and press return/enter.

For instance, for the R folder I created on the desktop of my MAC laptop, the path is /Users/angsun/Desktop. After pasting this path in the R session, I type /R. Therefore, after a greater than (“>”), it should read as setwd(“/Users/angsun/Desktop/R”), and the entire command line should look like:

>setwd(“/Users/angsun/Desktop/R”)

Of course, if you type this command line in your R session, you cannot set your working directory, because the path for the R folder on the desktop of your computer is not the same as this one.

Using RStudio, you can also easily set the working directory without typing a command line but just clicking the mouse a couple of times. Here is how you do it. Go to the lower-right window/panel, locate and change the folder by clicking on “…” (around the middle of your far right side of your screen) to the folder you would designate as the working directory, then click “More” that is next to a symbol of gear, choose “Set As Working Directory”. Then you set the designated folder (in this case, the folder named R) as the working directory. You can also see in the command line to set the working directory automatically appears in the Console at the lower-left panel.

If you wish to find your current working directory in an R session, you can type:

getwd()

2) Object and Function in R

Now, let’s do some descriptive statistics using R.

We have a data set including ten numbers: 1,2,3,4,5,6,7,8,9,10

Please type the following command line in R (Do NOT copy from this PDF file and paste it in your R session. Do type it, as this is the way to help you learn, understand, and memorize this computer language. You should keep this in mind when you are using this lab manual.):

a<-c(1,2,3,4,5,6,7,8,9,10)

In this simple command line, “a” is an “object”, which locates before “<-“ (The combination of a “less than” and a “dash” forms an “arrow”. The object is where the arrow is

pointed at. You can also type **c(1,2,3,4,5,6,7,8,9,10)->a** . This gives you the same result as the little a is defined as an assemblage containing the same 10 numbers. However, I prefer to write the code as in the way that I showed your first.)

Once you tried the above codes and typed the 10 numbers in R, I can tell you there is a short command line that allows you to generate an assemblage containing the same 10 numbers. Try and check the following code:

a1<-c(1:10)

You can see "a1" contains exactly the same 10 numbers as in object "a". The colon in the brackets means "to".

An "object" is representing a data structure you create and define in R. It has attributes and the method to process the attributes. Depends on how you define it, an object can represent a dataset containing the data you would like to process, a function you defined by yourself, or combinations of functions and the attributes they act on. Here, this object "a" is defined as a dataset including 10 numbers.

An object is always before "<-". You can name the object in a way you like. However, the object cannot be a number or start with a number. For instance, "12" cannot be defined as an object; "1a" cannot be defined as an object, but "a12" can be defined as an object. Symbols cannot be used in an object either. Underlines can be used in naming an object but cannot start an object.

Challenge Question 1:

Which of the following can be an object in R? (Try to solve it by yourself before you look at the answer at the end of this note. If you are not sure about the answers, just try them in R. When you learn this coding language, just as when you learn any language, do not be afraid to try. You will learn from the mistakes you make.)

A. A1
B. 2B
C. AAA
D. A2B3c5f

E. c2
F. IDK_3f
G. IDK*
H. IDK*8
I. _IDK

After "<-", it is how the object is defined.

In this example, the lower case "c" right after the "<-" and before the "(", is a "function" of R. This function is for setting aggregate. It means the object includes (combines/contains) whatever is in between the brackets.

The R functions are case-sensitive. Therefore, if you write the capital letter "C" instead of the little "c", R will not recognize it as a function.

3) Simple Descriptive Statistics in R

R has many built-in functions for statistics. For example, we can easily find out the average/mean of the 10 numbers defined in the object "a".

mean(a)

[1] 5.5

You can also find the median, the sum, and the range of these 10 numbers:

median(a)

[1] 5.5

sum(a)

[1] 55

range(a)

[1] 1 10

For more descriptive statistics, you can also use R to find the minimum, the maximum, the variance, and the standard deviation:

min(a)

[1] 1

max(a)

[1] 10

var(a)

[1] 9.166667

sd(a)

[1] 3.02765

Challenge Question 2:

How can you use R to find out the sum of the mean and the minimum of the 10 numbers? (Try to solve it by yourself before you look at the answer at the end of this note.)

You can also easily use a function summary() to show a few descriptive statistics:

summary(a)

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.25    5.50    5.50    7.75   10.00
```

How can we find out how many numbers are in a data set? Of course, for "a" we already know there are 10 numbers. But assume we do not know that, then how can we find out the number of data values in "a"? There are two different ways we can solve this problem. The first one is a little bit complicated. But it is only "a little bit" complicated. Instead of using 1 function, you have to use 2 functions. Here is how we can achieve the goal:

b<-table(a)

sum(b)

[1] 10

What if you type b in your R session? What result do you get?

If you type **b**, you should get:

```
a
 1  2  3  4  5  6  7  8  9 10
 1  1  1  1  1  1  1  1  1  1
```

You can see, there are two rows of numbers. In the first row of the numbers from the result of using the table(), you get the actual unique numbers in object "a". The second row of the numbers are actually the frequencies of the corresponding numbers that appear in "a". Adding up the frequencies of the numbers will get the total number of the data values in the data set.

The second one is using the built-in R function: length()

length(a)

[1] 10

R can also allow you to perform calculations. Let's find the standard error of the mean (SEM) of "a". Since R does not have a built-in function to calculate the SEM of a data set, we have to perform some calculations using R. We are going to learn in a lecture that SEM=SD/$\sqrt{n}$, where "SD" is the standard deviation and "n" is the sample size. Thus, we can do the calculation using R:

n<-sum(b)

SEM<-sd(a)/sqrt(n)

SEM

[1] 0.9574271

We learned how to use function c() to define a data set. For a small dataset, of course, you can type all the data and apply function c(). However, for a large data set, it is very inconvenient, even if it is not impossible to type in all the data. In future labs, we will learn how to import and select data from data tables using read.csv() or read.table() and the data.frame().

Answer Keys to the Challenge Questions:

1. A, C, D, E, and F
2. sum(mean(a)+min(a))

Or b<-mean(a), c<-min(a), d<-b+c