**Week 1**

* Introduction to R
  + Why R?
  + Advantages
    - Free
    - Open source
    - Most used statistical language
    - Essential for university work in stat
  + Disadvantages
    - Free
    - Open source
  + <https://www.r-project.org/>
* Code
  + >: command prompt symbol
  + +: continuation symbol
* Instructions
  + Download R
  + Download RStudio
    - <https://www.rstudio.com/products/rstudio/download/>
* Work areas
  + Top left: editor to write code
  + Bottom left: console, same as in R GUI
  + Top right: workspace
  + Bottom right: files, plots, package, help
* Practice
  + Use r as a calculator
* Notes:
  + Vectors in R are rows or columns of numbers or text. In R we apply functions to vectors
  + Ex: lets say we have a list of numbers: {7 8 9 10}. Assign this list of numbers to a vector called x. Since the list is ordered we write this as 7:10 (read 7 through 10)
    - x <- 7:10
    - x
    - [1] 7 8 9 10
    - <-: assignment symbol
* Type shortcut ctrl L on keyboard to clear console
* Naming conventions
  + We used x in example
  + Letters digits and dot can all be used
  + Must not start with digit followed by digits
  + Must not start with digit followed by period
  + Names that start with period are special and must be avoided
  + Names are case sensitive
  + Descriptive names are best
* Vectors and Functions
  + A vector is a basic data structure in R
  + To construct a vector in R we can use a function
  + The c() function – the combine function
  + Ex: form a numeric vector {5, 6, 7, 8}
    - This is a numeric vector of length 4
      * > c(5, 6, 7, 8)
      * [1] 5 6 7 8
    - > x <- c(5, 6, 7, 8)
    - > x
    - [1] 5 6 7 8
* Vectors and the sum function
  + > z <- c(3:6)
  + > sum (z)
  + [1] 18
  + Ex: find the mean of z
    - > sum(z)/length(z)
    - [1] 4.5
  + Vectors in R are objects
* Syntax
  + functionName(arg1 = val1, arg2 = val2, …)
  + ex: seq() function returns sequences of numbers
  + > seq(2,5)
    - Sequence of numbers 2 to 5
  + [1] 2 3 4 5
  + Ex:
    - > x<-seq(2,5)
    - > x
    - [1] 2 3 4 5
  + Ex:
    - > (x<-seq(2,5))
    - [1] 2 3 4 5
    - The parentheses lessen the amount of code you have to write
    - They send the instruction
* Comments
  + Short comments can be placed after code preceded by two spaces, #, and then one space
  + Entire commented lines should begin with # and one space
  + Code should always contain appropriate comments
* Ex:
  + > objects() # list user-defined objects, x was the only user-defined object
  + [1] “x”
  + > rm(x) # removes x from the global environment
* Text Values. “Hello World”
  + Assign “Hello” to a variable x. Assign “World” to a variable y
    - > x<- “Hello”
    - > y<- “World”
  + Use c() the concatenate function to print “Hello” “World”
  + > w<- c(x, y)
  + > w
  + [1] “Hello” “World”
* Use the paste function
  + > paste (x, y)
  + [1] “Hello World”
* Functions that don’t need parameters
  + date() # returns today’s date and current time
* R Help
  + Type help() at the prompt or type ?
  + Ex:
    - help(paste)
  + ex:
    - ?paste
  + Remember R is case sensitive
  + Ex: ?paint # is there such a function? No, so R will return a message saying function does not exist
* Workspace
  + q() to quite RSTudio. You are asked:
  + save workspace image to ~/.RData? [y/n]:
  + yes
  + restart RStudio
  + [Workspace loaded from ~/.RData]
    - This is the location of your files
  + Working Directory
  + >getwd() # get working directory – find the folder you are currently working in
* Datasets
  + R datasets package is a library in R
* To access and view the dataset CO2, type:
  + >data(“CO2”) # indicates to R the dataset of interest.
  + >View(CO2) # view CO2
  + Uptake is a numeric vector of carbon dioxide uptake rates
  + > CO2$uptake # accesses particular varibles within a dataset, in this case the uptake variable is accessed with the $
* Basic data types and data structures
  + Logical (true, false)
  + Integer
  + Numeric (real or decimal)
  + Complex (1 + 0i, 1 + 4i)
  + Character (“a”, “swc”)
* Data structurs
  + Vector
  + Matrix
  + Data frame
  + Factors
  + Tables
* A vector is a common and basic data structure in R. vectors can be any of the 5 basic data
* s
  + With a pre defined length
    - X <- vector(length = 10)
  + With a length and a type
    - vector(“character”, length = 10)
    - vector(“numeric”, length = 10)
    - vector(“integer”, length = 10)
    - vector(“logical”, length = 10)
  + syntax
    - vector(class of object, length)
  + as we have already seen, we can also create vectors by using the c() function
    - x<-c(1, 2, 3)
      * x is a numeric vector
    - y<- c(TRUE, TRUE, FALSE, FALSE)
      * y is a logical vector
    - z<-c(“bob”, “ann”, “jon“, “ivy”)
      * z is a character vector
* examining vectors
  + >typeof(z) # returns the datatype of z
  + [1] “character”
  + >length(z) # returns the length of z
  + [1] 4
  + >str(z) # determines structure of z
  + chr [1:4] “bob” “ann” “jon” “ivy”
    - returns character, length, and values of vector
* create vectors as sequences of vnumbers
  + >series <- 1:10
  + str(series)
  + int [1:10] 1 2 3 4 5 6 7 8 9 10
  + >seq(10)
  + [1] 1 2 3 4 5 6 7 8 9 10
  + >seq(1, 10, by = 0.5)
  + [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5
  + [13] 7.0 7.5 8.0 8.5 9.0 9.5 10.0
* Sequence function
  + seq() some syntax
  + seq(from, to)
  + seq(from, to, by)
  + ex: seq(from)
    - seq(7)
    - [1] 1 2 3 4 5 6 7

**Week 2: Descriptive Statistics**

* Warning
  + Use the assignment operator “<-”, industry standard
  + Do not use the equal sign
  + > print(x<-1:3)
  + [1] 1 2 3
  + > x
  + [1] 1 2 3
  + To remove x from the global environment, “rm(x)”
  + If you use the = as the assignment operator:
  + > print(x=1:3)
  + [1] 1 2 3
  + > x
  + Error: object ‘x’ not found
  + > print(z = 1:5)
  + Error in print.default(z = 1:5) : argument “x” is missing, with no default
  + > mean(x=1:6)
  + [1] 3.5
  + > x
  + Error: object ‘x’ not found
* Statistics
  + Terms defined
    - Data are collections of observations
    - Statistics Is the science of data
    - Population: the complete collection of all individuals (scores, people, measurements) to be studied
    - Sample: group of members selected from a population
    - Parameter: a numerical measurement describing some characteristic of a population
    - Statistics: a numerical measurement describing some characteristic of a sample
    - Individuals are the objects described by a set of data
    - A variable is any characteristic of an individual
    - A variable can be either categorical (qualitative) or quantitative
      * Caution: not all numbers represent quantitative variables
      * Ex: zip codes – 90210
      * Phone numbers
    - How to tell if a numbers is categorical or or quantitative?
      * Can you do math with it and still have it make sense?
  + Charting categorical data: bar & pie charts
    - Bars do not touch because each one represents categorical data
    - A simple bar chart
      * #Define a vector with 4 values
      * > x<-c(1, 4, 6, 10)
      * #Graph x
      * > barplot(x)
    - A better looking chart syntax
    - barplot(H, xlab, ylab, main, names.arg, col, border)
      * H is a vector(or matrix) containing numeric values used in bar chart
      * xlab is the label for the x axis
      * ylab is the label for the y axis
      * main is the title of the bar chart
      * names.arg is a vector of the names appearing under each bar
      * col is used to assign color to the bars in the graph
      * > dogs <- c(1, 4, 6, 10)
      * > breed\_name <- c(“Alsatian”, “Collie”, “Pug”, “Beagle”)
      * > barplot(dogs,names.arg = breed\_name, xlab = “BREED”, ylab = “Number of Dogs”, col = “GREEN”, main = “DOG CHART”, border = “BLACK”)
    - pie chart in R
      * in R a pie chart is created using the pie() function which takes positive numbers as a vector input
      * syntax
      * pie(x, labels, radius, main, col, clockwise)
        + x is a vector containing the numeric values used in the pie chart
        + labels is used to give descriptions to the slices
        + radius the pie is drawn centered in a square box whose sides range from -1 to 1
        + main indicates the title of the chart
        + col indicates the color palette
        + clockwise is a logical value indicating if the slices are drawn clockwise or anti-clockwise
        + additional parameters are available
      * ex:
        + > x<- c(4, 5, 6, 1, 4)
        + labels <- c(“Comedy”, “Action”, “Romance”, “Drama”, “SciFi”)
        + > pie(x, labels, radius = 0.8, main = “Movies”, col = rainbow(length(x)), clockwise = TRUE)
  + Charting quantitative data: histograms and scatter/dot plots
    - histograms
      * use between 5 – 10 bins and refine histogram accordingly
      * don’t oversummarize or undersummarize
      * creating a histogram: set up a vector of numbers then use the histogram function
      * > x<- c(4, 5, 6, 1, 4)
      * > hist(z)
      * #the histogram function
    - interpreting histograms
      * when describing a quantitative variable, we look for the overall pattern and for striking deviations from that patter. We can describe the overall pattern of a historgram by its shape, center, spread
      * histogram with a line connecting each column is too detailed
      * histogram with a smoothed curve highlighting the overall pattern of the distribution
      * common distribution shapes
        + symmetric
        + left skew
        + right skew
      * other shapes
        + biomodal
      * outliers
        + lie outside the overall pattern of the distribution
        + try to explain outliers
        + usually due to data entry mistake

can delete if it is

* + - stemplots
      * has stems and leaves
      * 10’s place is the stem
      * 1’s place is the leaf part
      * original data: 9, 9, 22, 32, 33, 39, 42, 52, 58, 70

|  |  |
| --- | --- |
| Stem | leaf |
| 0 | 99 |
| 1 |  |
| 2 | 2 |
| 3 | 2399 |
|  |  |
|  |  |
|  |  |

* + - creating stemplot in R
      * > z<- c(09, 09, 22, 32, 33, 39, 39,…)
      * > stem(z)
      * #the decimal point is 1 digit to the right of the |

0 | 99

2 | 22399

* not what we wanted
* stem produces a stem and leaf plot of the values in x. the parameter scale can be used to expand the scale of the plot. A value of scale = 2 will cause the plot to be roughly twice as long as the default
* usage:
  + stem(x, scale = 1, width = 80, atom = 1e-08)
* arguments
  + x is a numeric vector
  + scale: this controls the plot length
  + width: the desired width of the plot
* > z<- c(09, 09, 22, 32, 33, 39, 39,…)
* > stem (z, scale = 2)
* the decimal point is 1 digit to the right of the |, this is what we wanted

0 | 99

1 |

2 | 2

3 | 2399

4 | 29

5 | 28

6 |

7 | 0

* stemplots vs histograms
  + stemplot preserves information
    - you can read off the original data
  + histogram cannot read off values so histogram loses information
    - does not preserve information

**Quantitative Data: numerical Descriptors**

* measure of center: mean and median
* measure of spread: range, quartiles
* the five number summary and boxplots
* IQR and outliers
* sample variance and sample standard deviation
* > (x <- seq(2,5))
* [1] 2 3 4 5
* > mean(x)
* [1] 3.5
* Review
  + find the mean uptake rate
  + >data(“CO2”)
  + >View(CO2)
  + >HowMuch <- CO2$uptake
  + >mean(HowMuch)
  + >[1] 27.2131
* measure of center: median
  + with an odd number of data points, takes the middle number
  + with an even number of data points, takes mean of two middle points
  + the middle value when data is sorted in ascending order
* find the mean uptake rate
  + >median(HowMuch)
  + [1] 28.3
* comparing the mean and the median
  + if symmetric graph: mean and median are equal
  + if left skew: median is greater than mean
  + if right skew: median is less than mean
  + because mean is affected so much by outliers, mean is not a robust measure of center
  + median is less affected extreme values and so we say the median is a robust measure of center
  + robust = resistant
* measures of spread: range
  + range = maximum value – minimum value
  + [1] 7.7 45.5
* measure of spread: quartiles
  + quartiles are called quantiles in R
  + “professor has had multiple strokes so far”
  + Q1 at 25%
  + Q2 at 50%
  + Q3 at 75%
  + IQR: interquartile range = Q3 – Q1
  + >IQR(HowMuch)
  + [1] 19.225
  + 5 number summary
    - minimum Q1 Q2 Q3 maximum
    - to find the 5 number summary in R use the quantile function
* box plots
  + the information containe din a 5 number summar is displayed in a boxplot
    - 1. line within box marks the median value
    - 2. solid lines extending from box reac to min and max values
  + >data(“CO2”)
  + > B <- CO2$conc
  + >boxplot(B)
* new example
  + >print(mtcars)
  + exercise: create side by side boxplots to compare vehicle weight for number of cylinders (4, 6, 8)
  + use the boxplot function
  + >boxplot(wt~cyl, data = mtcars, main= toupper(“compare Vehicle Weight to number of cylinders”), xlab = “Number of Cylinders”, ylab = “Weight”, col = “pink”)
  + >w <- c(-10, 1, 2, 3)
  + >boxplot(w, horizontal = TRUE)
  + >mean(w)
  + [1] -1
  + > median (w)
  + [1] 1.5
* use the summary function
  + >summary(HowMuch)
  + gives all the values in the 5 number summary plus the mean
* sample variance and sample standard deviation in R
  + find the variance and SD of co2 uptake
    - >var(HowMuch)
    - >sqrt(var(HowMuch))
    - >sd(HowMuch)
      * sqrt(var(HowMuch)) = sd(HowMuch)
* properties of SD
  + 1. measures spread about the mean
  + 2. can be equal to 0, denotes no spread which occurs when all the values are equal
  + 3. becomes larger when values are more spread out
  + 4. s has the same units as the original data
  + 5. not a robust or resistant measure
* levels of measurements
  + another way to classify data is to use levels of measurement
  + nominal level
    - nominal levels of measurement is characterized by data that consists of names, labels, or categories only, and the data cannot be arranged in a n ordering scheme (ex: low to high)
    - ex: survey responses (yes, no)
  + ordinal level
    - ordinal level of measurement involves data that can be arranged in some order, but differenes between data values either cannot be determined or are meaningless
    - ex: grades (A B C D F) but (B-A) is meaningless
  + interval level
    - interval level of measurement like the ordinal level, with the additional property that the difference between any two data values is meaningful, however, there is no natural zero starting point(where none of the quantity is present )
    - ex: years – 2000, 2001, 2002, 1776, 1492
  + ratio level
    - ratio level of measurement the interval level with the additional property that there is also a natural zero starting point (where zero indicates that none of the quantity is present); for values at this level, differences and ratios are meaningful
    - ex: price of textbooks
  + summary
    - nominal: categories only
    - ordinal: categories with some order
    - interval : differences but no natural starting point
    - ration: differences, ratios, and a natural starting point
    - **midterm question on this**
  + we will now look at nominal and ordinal data in R

**4/15/19**

* ordinal data in R
  + ordered() function for ordinal data
  + to create an ordered factor, we have two options
    - 1. use factor() with the additional argument ordered = TRUE
    - 2. use the ordered() function
* More on data types
  + scalar
    - single numbers
    - >4
  + vector
    - # numeric vector
    - a<-c(1, 2, 5.3 , 6, -2, 4)
    - #character vector
    - b<- c(“one”, “two”, “three”)
      * is.factor(b)
      * returns FALSE
    - #logical vector
    - c<- c(TRUE, TRUE, TRUE, FALSE, TRUE, FALSE)
      * can use the function
      * is.vector(c)
      * returns TRUE
    - to refer to the elements in a vector by their position, we use subscripts
      * #creates numeric vector
      * # numeric vector
      * a<-c(1, 2, 5.3 , 6, -2, 4)
      * #vieiw numeric vector
      * >a
      * #2nd and 4th elements of vector
      * >a[c(2,4)]
      * [1] 2 6
      * #extract sequence of 4th to 6th values
      * >a[c(4:6)]
      * [1] 6.0 -2.0 4.0
  + matrices
    - a matrix is a table
    - syntax: matrix\_name<-matrix(vector, nrow = r, ncol = c, byrow = FALSE, dimnames = list(char\_vector\_rownames, char\_vector\_colnames))
    - all columns must be of the same mode(numeric, character, logical)
    - they must also all have the same length
      * byrow = TRUE
        + indicates the matrix should be filled by rows
      * byrow = FALSE
        + this is the default: matrix is filled by columns
      * dimnames: optional labels for columns and rows
    - ex: Generate a 5x4 numeric matrix
      * > y <-matrix(1:20, nrow = 5, ncol = 4)
      * creates matrix
      * > y[,3] # 3rd column of the matrix
      * [1] 11 12 13 14 15
      * >y[2:4, 1:3] #. extracts rows and columns
    - ex:
      * > cells<-c(1, 26, 24, 68)
      * >rnames<-c(“R1”, “R2”)
      * >cnames<-c(“C1”, “C2”)
      * >mymatrix<-matrix(cells, nrow = 2, ncol = 2, byrow = TRUE, dimnames = list(rnames, cnames))
    - arrays
      * arrays of matrices
      * arrays are similar to matrices but have more than one matrix
      * an array is created using the array function
      * syntax
        + array(data = NA, dim = length(data), dimnames = NULL)
        + data has to be provided
        + dim has to be created
      * example: createa na array of two matrices, each with 2 rows and 3 columns, use the array function with dimension specified by using the following syntax dim(rows, columns, tables)

tables refers to number of matrices being created

* + - * + > yy <-array(1:12, dim = c(2,3,4))
        + > yy
      * ex: create an array of two 3.3 matrices, each with 3 rows and 3 columns. Use the array funcoin with dimension sspecified by using the following syntax dim (rows, columns, tables)
        + >vector1<- c(5, 9, 3)
        + >vector2<- c(10, 11, 12, 13, 14, 15)
        + >MyArray <- array(c(vector1, vector2), dim = c(3,3,2))
        + >print(MyArray) or View(MyArray)
        + putting it all together
        + is.array() checks if it as array
  + data frames
    - the following are characteristics of a dataframe
      * a dataframe is a table (or array-like structure) in which each column contains values of one variable and each row contains one set of values from each column
      * column names should be non-empty
      * the row names should be unique
      * the data stored in a dataframe can be of numeric, factor, or character type
      * each column should contain the same number of data items
    - syntax
      * data.frame( . . . , row.names = NULL, check.rows = FALSE, check.names = TRUE, fix.empty.names = TRUE, stringsAsFactors = default.stringsAsFactors())
      * can have columns with different modes (numeric, character)
      * ex:
        + >d <- c(2,3,4,6)
        + >e <- c("red", "white", "red", NA)
        + >f <- c(TRUE,TRUE,TRUE,FALSE)
        + >df1 <- data.frame(d,e,f)
        + >print(df1)
      * have we created a dataframe? how to check:
        + >is.data.frame()
      * add variable names
        + names(df1) <- c(“ID”, “Color”, “Passed”)
      * select certain columns
        + >df1[c(“ID”, “Passed”)]
      * extract values of ID
        + > df1$ID
  + lists
    - to create a list: list()
      * ex: create a list containing strings, numbers, anda logical value
      * > my\_list <-list(“PSTAT”, “MATH”, c(10), TRUE)
      * > print(my\_list)
    - accessing elements of a lsit
      * > my\_list[1] # returns the first element in the list
      * [[1]]
      * [1] “PSTAT”
* review
  + vectors: contain elements of the same mode
  + matrix: all one mode
  + array: an array of 2 or more matrices, all one mode
  + dataframe: different columns can have different modes
  + list:
* calculations across arrays
  + use the apply() function
  + syntax: apply(x, margin, fun)
    - x is an array
    - margin is a vector
    - fun is the function to be applied across the elements of the array
  + ex:
    - > d <- c(2,3,4,6)
    - > f <- c(5,6,10,12,45)
    - > new.array <- array(c(d,f),dim = c(3,3,2))
    - > print(new.array)
    - > results <-apply(new.array, c(1), sum)
    - >print(result)
      * c(1) indicates that the function applied over rows since parameter is 1
        + if parameter 2 then sum of each column
      * # sums up total of rows over all tables
* importing data
  + keyboard input example
  + #create a dataframe using keyboard input
    - 1. import dataset by first downloading it onto your local computer and then using the import dataset feature of RStudio
    - import the data programmatically by executing commands in the consolw window of RStudio
  + install some more packages:
    - tibble
    - readr
  + import a CSV text file
    - first check working directory using: getwd()
    - to set new directory, use: setwd()
    - >getwd()
  + example of CSV
    - on gauchospace
  + install tibble package
    - a tibble is a kind of dataframe
    - a tibble displays only the first 10 rows and all the columns that will fit on your screen
    - function: as.tibble(dataframe)
      * make dataframe look like a tibble
      * coerce dataframe into a tibble
      * not actually a tibble, still a dataframe, just makes it look like tibble for analysis
* importing data
  + write code directly
    - read.csv functions
    - > mydata <- read.csv(“filename.txt”)
    - If you use read.csv, creates a dataframe, not a tibble
  + save(DFWLAX, file = “PSTAT 10 F18/DATA/MyDFLAX.R”)
  + load(“PSTAT 10 F18/DATA/MyDFLAX.R”)