DATA 101 MIDTERM FUNCTIONS

Lecture 1

```
> options(digits=x) #Shows x digits in calculations.

> name_of_function <- function(x)(function_body) #Creating a function including a var x

> pie(example) #Displays a piechart of "example"

> barplot(example) #Displays a barplot of "example"

> max(random_vector) #Displays the maximum value stored in random_vector

> objects() #Show us the objects in our script

> ls() #Serves the same purpose as objects()

> runif(number_of_values, min = min_value, max = max_value)
#Simulates "number_of_values" values between min_value and max_value

>library(package_name) #To load an existing (installed) package

>install.packages("package_name") #To install a package that is not on the computer

> package_name::name_of_object
#To load just one object out of the package instead of the whole thing
#(package must be installed first)

> search() #Shows the loaded packages in the script
```

Lecture 2

```
> setwd("file_path") #Sets working directory to file_path

> dump("usefuldata", "useful.R") #Saves usefuldata object into useful.R file

> source("useful.R") #Retrieves userful.R

> dump(list = objects(), "all.R") #Saves all objects created into all.R

> save.image("temp.RData")
#To save all current workspace image information to a file without quitting
-------------------------------------------------------------------------------
> length(name_of_vector) #To find the number of elements in name_of_vector
```

```
> name_of_vector[location_of_element] #Extracts location_of_element from name_of_vector

> test <- c(v1,v2,v3) #The c() function collects v1,v2,v3 together
#into a vector called test, c() can also be used for concatenating vectors

>  rivers[c(2,4)] # extracts the 2nd and 4th elements from rivers:
#To extract more than one element using c() to create a vector of the elements we want
#To exclude values instead, simply add a negative  sign to c() to become -c()

>  rivers[rivers > 2000] #Extracts the elements that are above 2000 in the vector rivers

> which( rivers > 2000 ) #To find the index of these values

> sort(rivers, decreasing = TRUE) #Sorts rivers in decreasing order

> seq( start, end, step )
#Creates a sequence of numbers that starts at "start" and ends at "end" and jumps by "step"

> rep(num, number_of_times) #Repeats num for "number_of_times" times
#This can be used with a vector, ex: rep(c(3,4), 7) repeats 3 and 4 for 7 times

> rep(c(1,2,3), each = num) #Repeats 1 "num" times then 2 "num" times then 3 "num" times
-------------------------------------------------------------------------------------------
> sample(possible_outcomes, size = value, replace = TRUE) #Simulates based on this:
#possible_outcomes = a vector containing all the acceptable outcomes
#size = how many times we want the simulation to happen
#replace = repeated values? then TRUE, want each simulation to be unique? choose FALSE

> table(above_simulation)
#Creates a table containing possible outcomes and how many times they occured in simulation
#Can be used for other things, not just simulations
-------------------------------------------------------------------------------------------
> substr(x, start, stop) #x is a vector of character strings
#start and stop signify when to start and end the substr() function

> paste(sample_vector, "test") #concatenates "test" next to the vector sample_vector

> paste(sample_vector, "test", sep = "seperator")
#The sep parameter controls what goes between components being pasted together

> paste0(sample_vector, "test", "s") #Shorthand way to set sep = "s"

> paste(sample_vector, "test", collapse = " and ")
#The collapse parameter collapses the resulting vector into a single string and the
# " and " is what is seperating each component of the old vector
-------------------------------------------------------------------------------------------
> test <- factor(c("apple","banana")) #Creates a factor containing apple and banana

> as.integers(test) #prints 1 for apple and 2 for banana

> levels(test) #Prints the levels of the factor
-------------------------------------------------------------------------------------------
> xy <- data.frame(x,y) #Constructs dataframe from vectors x and y existing in workspace

> xynew <- data.frame(x,y, new = testv) #Adds a column called new and contains testv
```

```
> nrow(test_df) #Returns number of rows in test_df dataframe

> ncol(test_df) #Returns number of columns in test_df dataframe

> dim(test_df) #Returns both columns and rows of test_df

> str(test_df) #Gets a summary of test_df (works with any object not just dataframes)

> test_df[1,2] #Extracts value at first row second column

> test_df$Column1 #Used to access only Column1 in dataframe test_df

> with(test_df, Column1) #Does same thing as above function, no need for $ here

> mean(test_df$Column1) #Finds the average of values in Column1 of test_df

> head(test_df) #Shows the first few rows of test_df

> tails(test_df) #Shows the last few rows of test_df

> subset(test_df, condition) #Extracts values in test_df that meet the condition

> aggregate( measurements ~ myfactor, data = mydata, FUN = myfun)
#Allows us to calculate statistics for all values within different groups of a dataframe
#First term is a model formula which relates to measurements within a factor
#FUN argument allows us to specify what function we want
-------------------------------------------------------------------------------------
> summary(x) #Computes several summary statistics on the data in x

> length(x) #Number of elements in x

> min(x) #Minimum value of x

> max(x) #Maaximum value of x

> pmin(x, y) #Pairwise minima of corresponding elements of x and y

> pmax(x, y) #Pairwise maxima of x and y

> range(x) #Difference between maximum and minimum of data in x

> IQR(x) #Interquartile range: difference between 1st and 3rd quartiles of data in x

> sd(x) #Computes the standard deviation of the data in x

> var(x) #Computes the variance of the data in x

> diff(x) #Successive differences of the values in x

> sort(x) #Arranges the elements of x in ascending order
-------------------------------------------------------------------------------------
> is.na() #Used to detect missing values

> dataset1 <- read.table("file1.txt", header = TRUE, sep = " ", na.string = " ", skip = 3)
#Reads a file from the working directory
```

```
#header = TRUE means that it contains a header
#sep = " " means that seperate the values in the table by a " "
#na.string = " " assigns a value of " " to missing values
#skip = 3 skips the first 3 lines
```

Lecture 3

```
> aa %in% bb #Tests whether elements of aa can be found in bb and creates a logical vector

> sum(logical_vector) #Returns the amount of TRUE values

> all.equal(x,y) #Tests if x and y are approximately equal
#(helps when there is rounding error)
-------------------------------------------------------------------------------------
> strptime(vector_here) #Converts vector from string to an internal numeric representation

> format(vector_here) #Converts it back to a string for printing

>ISOdate(vector_here) #Creates date vector when date is known

> ISOdatetime(vector_here) #Used to create a date vector when date and time is known

> library(chron) #Contains more date and time functions, import that to do the following:

> daysSinceTestDate <- chron(dates = testDate, format = c("y-m-d"))
#Creates date vector out of testDate and assigns it to daysSinceTestDate

> as.numeric(daysSinceTestDate) #Prints the number of days since January 1, 1970 to date

> tsTest<- ts( dataframe$column, start = c(1990,1), end = (2020,12), frequency = 12)
#Creates a time series object that has start vector at first month of 1990 and end vector
#at the last month of 2020
```

Lecture 4

```
> barplot(test1) #Graphs a barchart of test1

> barplot(test1, cex.names = .75, cex.axis = .75, main = "TITLE",
xlab = "horizontal label", ylab = "vertical label", xlim = c(0,100), ylim = c(5,10))
#This function contains almost everything we need to know about barplots:
#cex.names = .75 reduces size of region names to .75 of original
#cex.axis = .75 reduces size of labels on vertical axis by .75 of original
#main = "TITLE" sets the main title to "TITLE"
#xlab adds a label for horizontal axis
#ylab adds a label for vertical axis
```

```
#xlim=c(0,100) sets the limits of x values to be between 0 and 100
#ylim = c(5,10) sets the limits of y values to be between 5 and 10
--------------------------------------------------------------------------------------
> dotchart(test2) #Graphs a dotchart of test2
#dotchrt can use any of above arguments in barplot

> dotchart(test2, pch = 16) #pch argument gives a certain shape to dots based on its value
--------------------------------------------------------------------------------------
> barplot(test3, besides = TRUE, legend = TRUE)
#Graphs a bar chart of a complex vector test3
#besides = TRUE causes values in each column to be plotted side by side
#legend = TRUE causes legend to be added
--------------------------------------------------------------------------------------
> hist(test4) #Graphs a histogram of test4

> hist(test4, breaks = "Scott") #Chooses binwidths using Scott way

> hist(test4, breaks = "fd") #Chooses binwidths using Freedman-Diaconis way
```

Please do not solely rely on this, you still need to understand all the concepts we've taken up to week 5. This is merely a nice list of functions (no concept explanations whatsoever) that we might need to use in our upcoming midterm.

Best Wishes,

Zee