Data Science #4 - Max Greene

# Question #1

1(a): A function can be passed arguments as inputs and perform computations on them.

1(b): Sys.Date returns a “Date” object corrosponding to the “system’s idea” of the date, according to the R documentation.

1(c): 1. Everything that exists is an object. 2. Everything that happens is a function call.

1(d): The source code for any function can be seen by entering the function name into the console without ant parentheses or arguments.

1(e): Default arguments can be useful to save time and reduce function complexity while allowing for function versatility. Specifically, they’er useful when you think the user will enter the same value into the function most of the time.

1(f): The *args* function returns the arguments that the functions accepts. For example, args(mean) returns function (x,...)

1(g): Passing functions as arguments increases the versatility of the function. For example, evaluate() in swirl became a function capable of many computations from modulus operations to means to rounding.

1(h): my\_vector[length(my\_vector)] -OR- tail(my\_vector,1) 1(i): paste() combines all string arguments into a single string, seperated by the sep = " " argument.

1(j): the “dot-dot-dot” argument in R allows for an infinite amount of arguments, which can be unpacked based on their assignment.

# Question #2

my\_mean\_func <- function(x)  
 {  
 return(mean(x))  
 }

# Question #3

my\_remain\_func <- function(num, divisor)  
{  
 return(num %% divisor)  
}

# Question #4

4(a): *Scoring functions* are a form of data manipulation that reduces multidimensional data into a single numerical value.

4(b): According to 4.2, *Scores* are functions that extract the features of each entity and map them to a single numerical value.

4(c): An effective scoring system should be easily computable, easily understandable, intuitive (monotonic variable interpretation), produce accurate results on outliers and utilize a normal distribution.

4(d): A ranking is a certain type of score meant to list objects in order of the score. For example, sports team rankings, University rankings, student rankings on GPA, and Google PageRank are all ranking scores.

4(e):

4(f): Z-scores are functions applied to a set of data in order to normalize it. In a data set with non-zero mean and standard deviation, a Z-score normalizes the mean to 0 and the standard deviation to 1, maintaining the relative differences between object values.

# Question #5

5(a):

my\_OBP <- function(hits,walks,atBats)  
{  
 return((hits+walks)/atBats)  
}

5(b):

H <- sample(162:16200,270,replace=TRUE)  
W <- sample(162:1782,270,replace=TRUE)  
AB <- sample(64800:105300,270,replace=TRUE)  
  
OBP <- my\_OBP(H,W,AB)

The sample() function takes a specified number of items of a given vector, creating a somewhat random sampling of data from an evenly distributed vector.

5(c):

mean(OBP)

# Question #6

My function my\_VO2\_max is used to determine the maximum oxygen uptake of an individual. Here, I use it to estimate an average of the population. The average female VO2 max tends to range from 27-30 mL/kg/min and the average male ranges from 35-40 mL/kg/min. It is assumed that larger individuals will need to intake more oxygen to support more cells, so by dividing my mass of the individual, we normalize for mass and create an efficiency metric. VO2 max is used by endurance athletes to determine the efficiency of their oxygen uptake.

my\_VO2\_max <- function(mLO2,duration,mass)  
{  
 return((mLO2/duration)/mass)  
}  
  
mLO2 <- sample(c(5000:100000),500,replace=TRUE)  
duration <- sample(c(5:60),500,replace=TRUE)  
mass <- sample(c(50:120),500,replace=TRUE)  
  
avg\_VO2\_max <- mean(my\_VO2\_max(mLO2,duration,mass))

This is a good scoring function because it is accurate and a real world equation. If the data collected is accurate, this function will return the amount of oxygen used in a certain amount of time, normalized by the weight of the individual.

# Question #7

library(nycflights13)  
  
arr\_delay\_2h <- filter(flights, arr\_delay > 120)  
  
to\_Houston <- filter(flights, dest == "IAH" | dest == "HOU")  
  
United\_American\_Delta <- filter(flights, carrier == "UA" | carrier == "AA" | carrier == "DL" )  
  
Summer <- filter(flights, month == 7 | month == 8 | month == 9)  
  
arrived\_late <- filter(flights, arr\_delay > 120 & dep\_delay < 1)  
  
delayed\_but\_quick <- filter(flights, dep\_delay > 60 & arr\_delay < dep\_delay-30)  
  
early\_morning <- filter(flights , dep\_time >= 0000 & dep\_time <= 0600)

# Question #8

dplyr::between is a shortcut for checking if a value is between two numerical values. For example, if you are checking for flights from midnight to 6:00am, you could have between(dep\_time,0000,0600) instead of dep\_time >= 0000 & dep\_time <= 0600.

# Question #9

filter(flights, is.na(dep\_time))

There are 8,255 objects missing dep\_time. these objects are also missing dep\_delay, arr\_time, arr\_delay, air\_time, and the rest of the data collected from the flight actually taking place. All data scheduling the flights are here. This may mean the fligths were scheduled bu cancelled, possibly to inclement weather.

# Question #10

arrange(flights, desc(is.na(dep\_time)))

# Question #11

arrange(flights, desc(dep\_delay))

# Question #12

arrange(flights, air\_time)