Midterm V3

YI CHEN (yc3356)

Section 003

Instructions (Read this completely first)

You should complete the exam by editting this file directly. Please knit the file often, so that if you make a mistake you catch it before the end of the exam. You will have exactly 1.5 hours (90 minutes) from your start time to complete the exam. **At the end you must turn in your knitted .pdf file and raw .Rmd file on Courseworks.**

When the time is up, you must shut your computer immediately. We will take off points from anyone whose computer is still open after time is up.

You may use your class notes for the exam, but not the internet. You absolutely may not communicate with anyone else during the exam. Doing so will result in an F in this class and likely result in termination from the MA program. Note that your time will be tight so you will not be able to look up every bit of code from your class notes.

Question 0 (5 points)

a. (0.5 points) Place your section number as the date of the document. If you don't know your section number, you can determine it below based on when your lab meets.

- Section 002 Lab meets TR 7:40pm-8:55pm
- Section 003 Lab meets TR 11:40am-12:55pm
- Section 004 Lab meets MW 8:40am-9:55am
- Section 005 Lab meets TR 8:40am-9:55am

b. (0.5 points) Write your name and UNI as the author of the document.

c. (1 point) After you have submitted the exam, fill out the survey on Courseworks. You have until tomorrow evening to do this and in the survey you will have the opportunity to tell me if you see classmates in your vacinity communicating with others during the exam.

d. (3 points) Please present your answers in a readable format. This includes things like indenting your code and generally presenting easy-to-read code. Presentation of the overall Markdown document will be considered as well.

Question 1: Simulations (32 points)

Recall from class that we simulated dart throws using the `drawBoard()` and `scorePositions()` functions included in this document above. For example, the following code simulates 100 throws where the x and y positions of the throws are modeled by $\mathcal{N}(0, 120^2)$ where the standard deviation is 120mm. We then draw the dart board with the `drawBoard()` function, score the throws with the `scorePositions()` function, and plot the locations of the simulated throws and the score of each throw. Note the dimensions of your dart board are a bit different than those used in class.

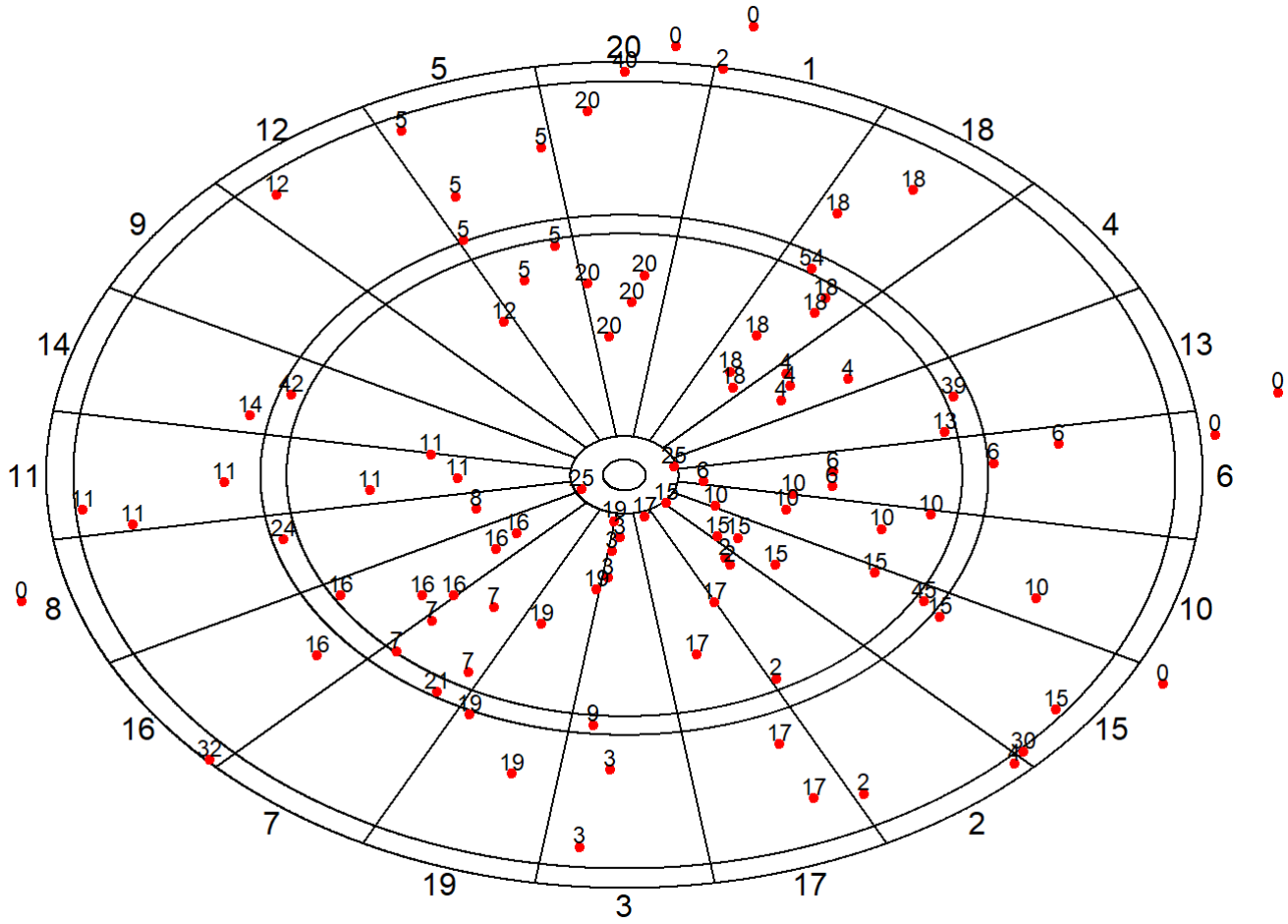
```
set.seed(1)

throws  <- 100
std.dev <- 120

x <- rnorm(throws, sd = std.dev)
y <- rnorm(throws, sd = std.dev)

drawBoard(board)
scores <- scorePositions(x, y, board)

points(x, y, pch = 20, col = "red")
text(x, y + 8, scores, cex = .75)
```



a. (4 points) Write code that simulates 500 throws where x and y are modeled as Uniform(-R,R), where $R = 255$. Make sure to draw the dart board, plot the location of the throws, and the scores of the throws as in the above. Save the simulated values as `x1a` and `y1a`, as they will be used in question 1.d below. Also compute two values `prop.normal` and `prop.unif` which return the proporation of throws from the model in the description and in this question which miss the board (i.e. the score is zero). Print `prop.normal` and `prop.unif`.

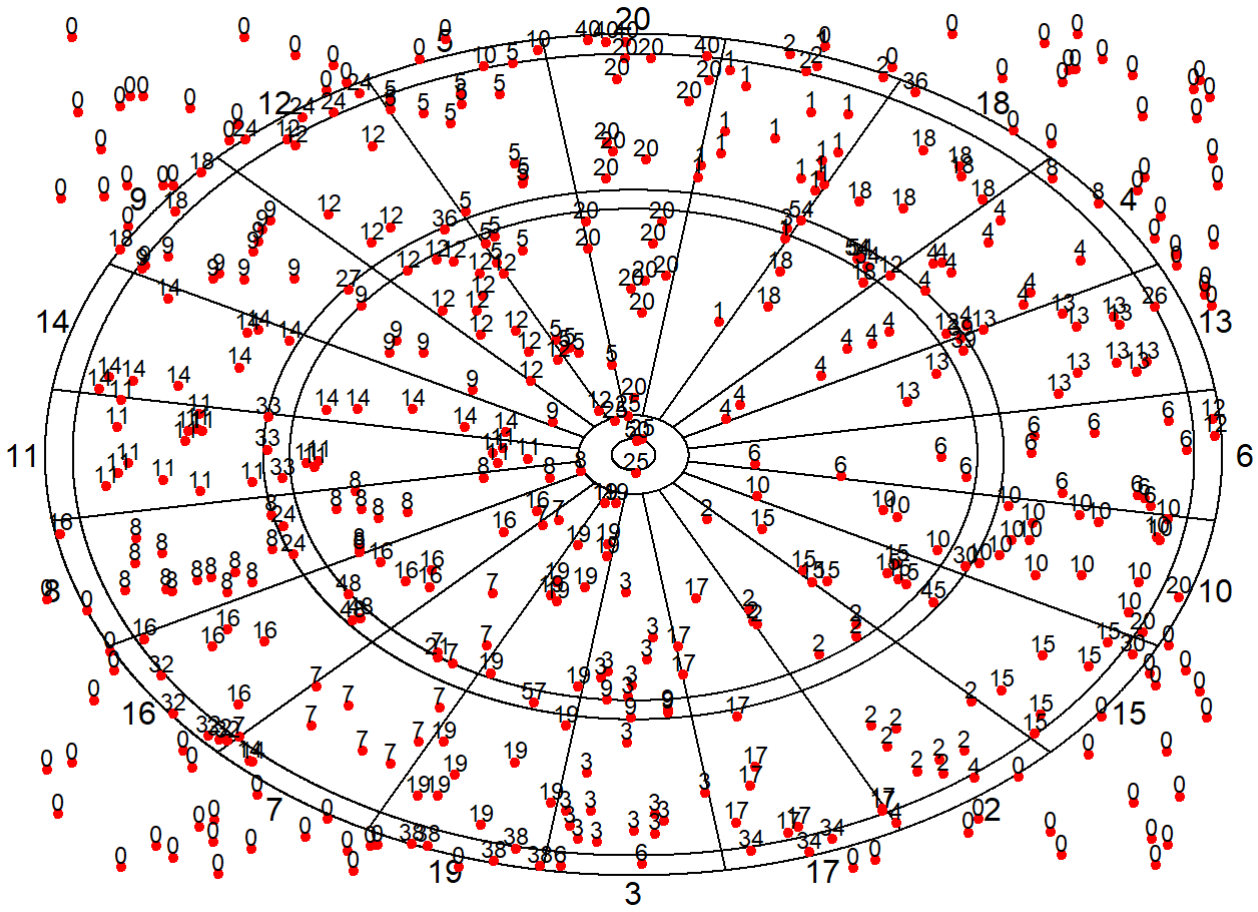
```
set.seed(1)

# Your answer to question 1.a here.  Don't remove the set.seed(1) command.
throws  <- 500
max <- 255
min <- -255

x1a <- runif(n = throws,min = min,max = max)
y1a <- runif(n = throws,min = min,max = max)

drawBoard(board)
scores1a <- scorePositions(x1a, y1a, board)

points(x1a, y1a, pch = 20, col = "red")
text(x1a, y1a + 8, scores1a, cex = .75)
```



```
prop.normal <- mean(scores == 0)
prop.normal
```

```
## [1] 0.06
```

```
prop.unif <- mean(scores1a == 0)
prop.unif
```

```
## [1] 0.198
```

b. (10 points) We have now simulated throws using to both a normal and uniform model, but it turns out that the normal model is too pessimistic compared to the uniform model: the normal model can produce throws far from the center, but uniform model is restricted to [-R,R] in each direction.

Fix this by restricting normal model so each sampled coordinate lies in [-R,R]. To do this, we will sample from a normal distribution, but if the draw lies outside of [-R,R], then toss it out.

Write a function `normal.rejection()` that takes in the following five arguments:

1. `n`, `mean`, `sd` (just as `rnorm()` does)
2. `min.val`, `max.val` (upper and lower limits -R and R)

The function should return a sample of length `n` which is generated by iteratively sampling $\mathcal{N}(\mu, \sigma^2)$ random variables, where μ is specified by the input `mean` and σ by `sd`, but only accepting those draws that fall above `min.val` and below `max.val`. Note that you don't want to sample any random variables unneccessarily, so the code should include a loop that terminates as soon as you have a sample of length `n`.

Note we will use this function to, in turn, generate the x and y simulated throw locations below.

```
# Your answer to question 1.b here.

normal.rejection <- function(n=500,mean=0,sd=120,min.val=-255,max.val=255){
        accepted_x <- vector()
        accepted_y <- vector()
        number <- 0

        while(number<=500){
                x1c <- rnorm(1, sd = sd)
                y1c <- rnorm(1, sd = sd)
                if(x1c < max.val & x1c > min.val & y1c < max.val & y1c > min.val){
                        accepted_y <- c(accepted_y,y1c)
                        accepted_x <- c(accepted_x,x1c)
                        number <- number + 1
                }
        }
        result <- rbind(accepted_x , accepted_y)
        return(result)

}
```

c. (5 points) Write code that simulates 500 throws where x and y are found using your function `normal.rejection()` where the `mean` equals 0 and `sd` matches that used in the example (i.e. `sd` is 120). The input `min.val` and `max.val` should be R and -R from 1.a. Save these values as x1c and y1c to plot in question 1.d below. Finally find the proportion of these throws for which the x location is between 0 and 50. Return this value as `prop.x` and print the result.

If you cannot complete 1.b write the code as if the `normal.rejection()` function exists. Of course, since it doesn't you will have to comment out this code. Otherwise you will have problem knitting your file.

```
set.seed(2)

# Your answer to question 1.c here.  Don't remove the set.seed(2) command.

x1c <- normal.rejection()[1,]
y1c <- normal.rejection()[2,]

prop.x <- mean(x1c > 0 & x1c < 50 )
prop.x
```

```
## [1] 0.1596806
```

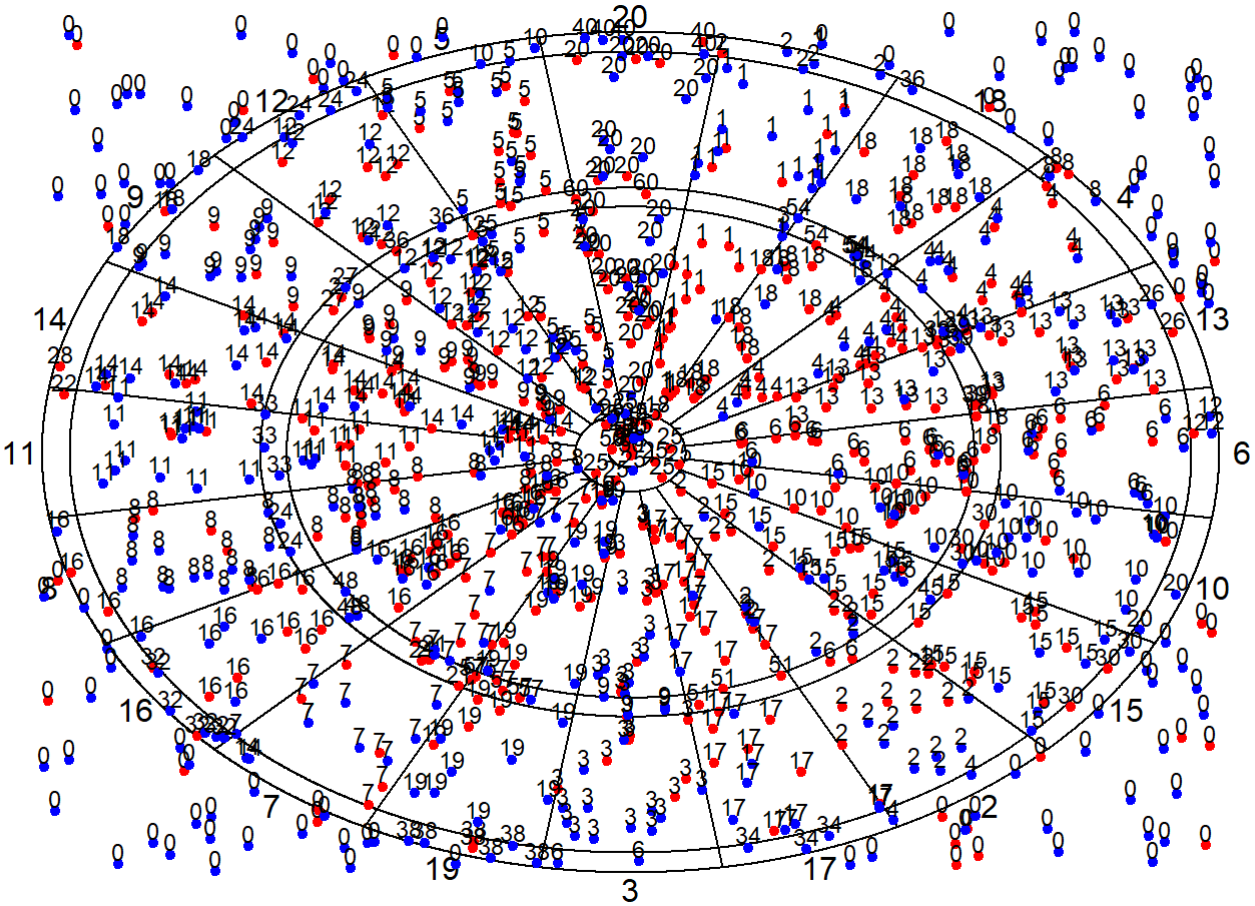
d. (3 points) Draw the dart board, plot the location of the throws, and the scores of the throws from questions 1.a and 1.c. The throws from 1.a should be colored blue and the throws from 1.c should be colored red.

```
# Your answer to question 1.d here.

drawBoard(board)
scores1c <- scorePositions(x1c, y1c, board)

points(x1c, y1c, pch = 20, col = "red")
text(x1c, y1c + 8, scores1c, cex = .75)

points(x1a, y1a, pch = 20, col = "blue")
text(x1a, y1a + 8, scores1a, cex = .75)
```



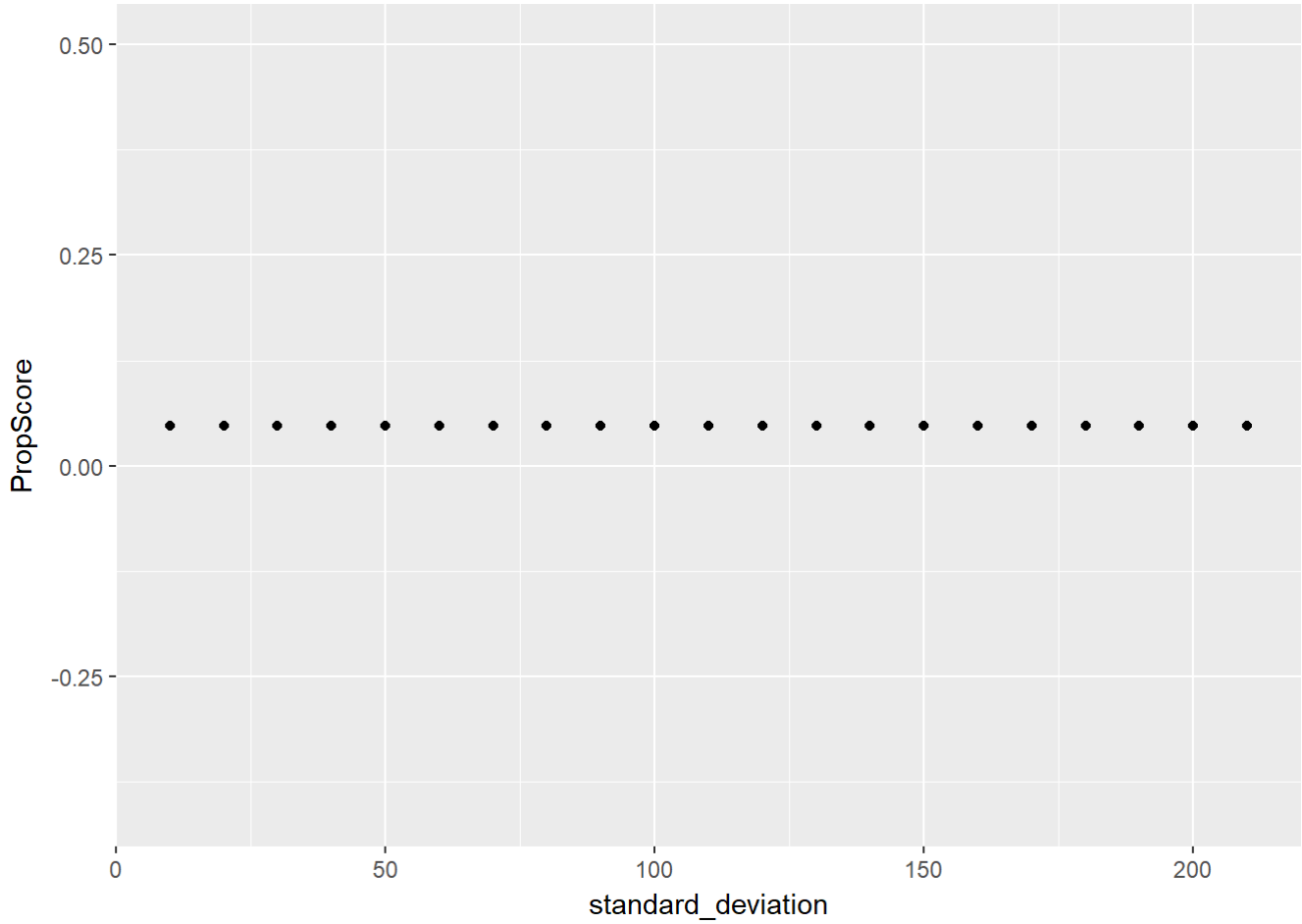
e. (10 points) For each of the standard deviation (`sd`) values $10, 20, \ldots, 210$ (using increments of 10), simulate 1000 throws using the thresholded normal model, i.e. x and y are both generated using `normal.rejection()` with `mean` equal to 0 and `min.val` and `max.val` equal to R and -R from 1.a. For each value of sd, compute the proportion of throws landing outside of the board (in other words, the proportion of throws equal to 0,) and save it in a vector called `PropScore` . Print the first four values of `PropScore` . (If you are unable to write an `normal.rejection()` function in 1.c, then simply simulate from normals as in the question introduction.) Plot these values using `ggplot` with standard deviation on the x-axis and `PropScore` on the y-axis.

```
set.seed(3)

# Your answer to question 1.e here.  Don't remove the set.seed(3) command.
PropScore <- vector()
standard_deviation <- seq(10,210,by = 10)
for( i in standard_deviation){
        x_new <- normal.rejection(n=1000,sd = i)[1,]
        y_new <- normal.rejection(n=1000,sd = i)[2,]
        scores1e <- scorePositions(x1c, y1c, board)
        rate <- mean(scores1e==0)
        PropScore <- c(PropScore,rate)
}
head(PropScore)
```

```
## [1] 0.04790419 0.04790419 0.04790419 0.04790419 0.04790419 0.04790419
```

```
library(ggplot2)
ggplot()+
        geom_point(aes(x=standard_deviation,y=PropScore))
```



Question 2: Character Data (32 points)

The file `rich.html` on the Canvas page is a listing of the 100 richest people in America, according to Forbes magazine (from 2013), which I scraped from `Forbes.com`. We will use the file to practice working with character data. The file `rich_df.txt` is formatted version of the data in the `.html` file.

a. (2 points) Please load `rich.html` onto your computer using `readLines()` and save it as `rich`. Load `rich_df.txt` so that it's a data frame in `R` and save it as `data_rich`.

```
# Your answer to question 2.a here.
rich <- readLines('rich.html')
data_rich <- read.table('rich_df.txt',header = TRUE)
```

b. (10 points) Your task is to extract the net worths of people listed in `rich`. The lines that contain the net worths look like the following:

```
"\t\t<td class=\"worth\">$##,# B</td>"
```

except with the # values are replaced by digits. For example, the first two lines which hold Bill Gates' and Warren Buffet's net worth are

```
"\t\t<td class=\"worth\">$72 B</td>"  "\t\t<td class=\"worth\">$58,5 B</td>"
```

You can find the location of these lines by running the following command:

```
worth.lines <- grep("td class=\"worth\"", rich)
```

Note that the length of `worth.lines` should be 100. Your first step is to create a vector called `net_worthss` that holds the values of net worths recorded in `rich`, in the same format as they appear in the data, meaning the first two values of your vector should be `$72 B` and `$58,5 B` and it should be of length 100. At the end,

print the first five elements of `net_worthss` .

```
# Your answer to question 2.b here.
worth.lines <- grep("\t\t<td class=\"worth\">.*B</td>$",rich,value = TRUE)
length(worth.lines)
```

```
## [1] 100
```

```
net_worthss <- substr(worth.lines,21,nchar(worth.lines)-5)
head(net_worthss,5)
```

```
## [1] "$72 B"   "$58,5 B" "$41 B"   "$36 B"   "$36 B"
```

c. (10 points) The Forbes website writes net worths in the form ``$75,5 B" to mean $75,500,000,000$ dollars or 75.5 billion dollars.

Write code to convert from the Forbes format in your `net_worths` vector to numbers (in billions), and run it to create a numeric vector of net worths, called `net_worths2` . (If you are unable to create a `net_worths` vector in question 2.b, use the column `Networths` in `data_rich` as your starting point for this question.) The first two values of your vector should be the numbers 72 and 58.5 and the vector should be of length 100. At the end, print the frist five values of the `net_worths2` vector.

```
# Your answer to question 2.c here.
net_worths2 <- gsub(",",".",net_worthss)
net_worths2 <- substr(net_worths2,2,nchar(net_worths2)-2)
net_worths2 <- as.numeric(net_worths2) * 1000000000
head(net_worths2,5)
```

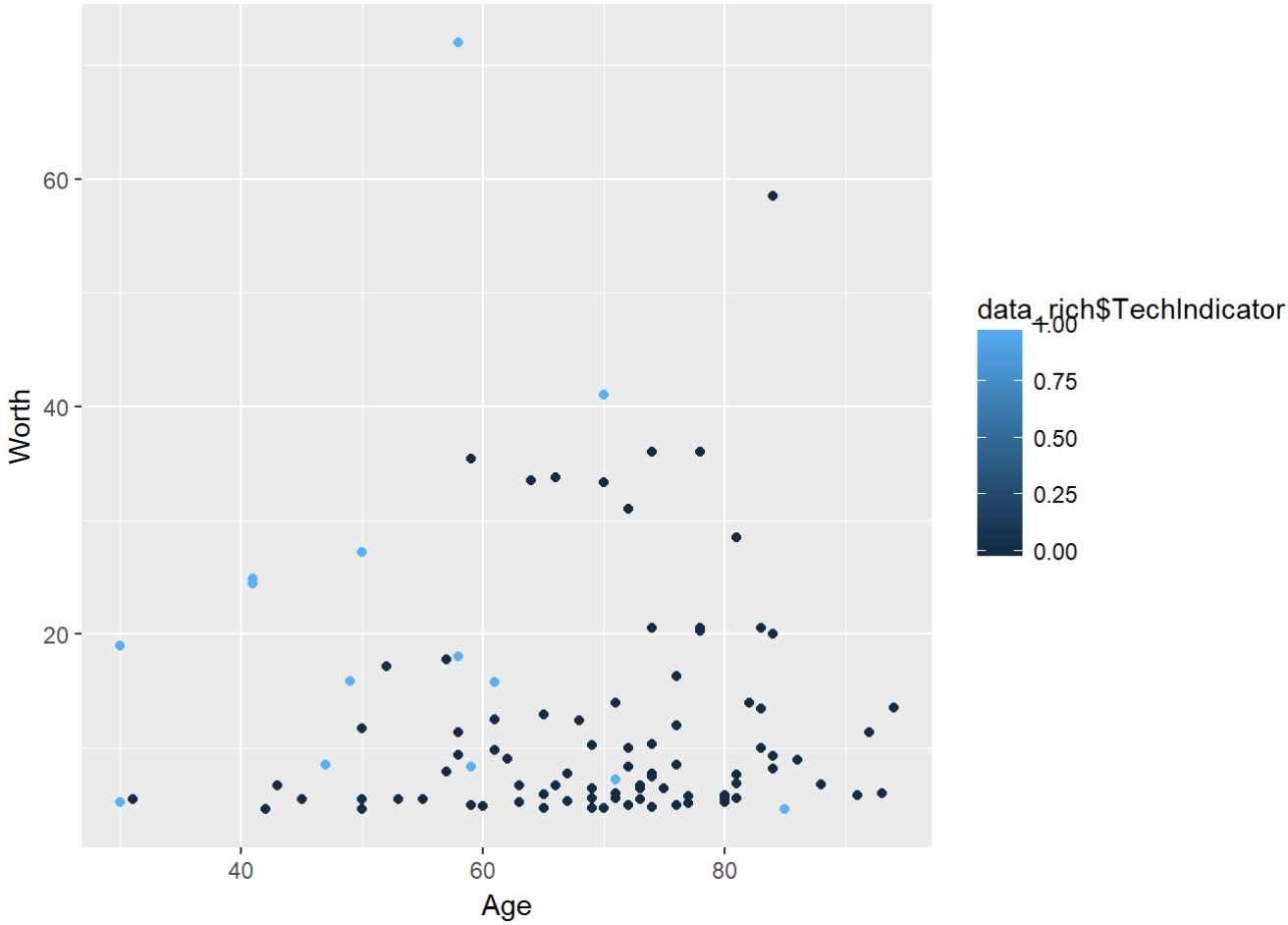
```
## [1] 7.20e+10 5.85e+10 4.10e+10 3.60e+10 3.60e+10
```

d. (6 points) Using `ggplot2` and `data_rich` create a scatterplot with (There is an NA in the Age data, but `ggplot()` will take care of this.)

```
# Your answer to question 2.d here.

ggplot(data=data_rich)+
        geom_point(aes(x=Age,y=Worth,col=data_rich$TechIndicator))
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



e. (4 points) Consider the following block of code: Write a single line of code that provides the same `reported.val` .

```
# Your answer to question 2.e here.
reported.val <- sum(data_rich[data_rich$TechIndicator==1,]$Worth)
reported.val
```

```
## [1] 292
```

Question 3: Cross-Validation (31 points)

Recall from class that we studied the idea that bigger cities tend to produce more economically per capita using the gross metropolitan product (gmp) data, `gmp.txt` . We studied a statistical model for this relationship:

$$Y = \beta_0 X^{\beta_1} + \epsilon,$$

where Y is the per-capita gmp of a city, X is the is the population of the city, β_0, β_1 are parameters, and ϵ is noise. Suppose we are considering three other potential models for this data given below.

- Model A (the same as above):

$$Y = \beta_0 X^{\beta_1} + \epsilon,$$

- Model B (exponential relationship):

$$Y = \beta_0 \exp(\beta_1 X) + \epsilon,$$

- Model C (linear relationship):

$$Y = \beta_0 + \beta_1 X + \epsilon,$$

- Model D (sinusoidal relationship):

$$Y = \beta_0 \sin(\beta_1 X) + \epsilon.$$

The specific form of these models doesn't matter as you won't use them directly.

We plan to choose a model to use by estimating the test mean square error of each model by using 3-fold cross-valiation. Roughly the procedure is as follows, we divide the data into 3 groups, or folds. For each of models A - D in turn, we train the model (optimizing over β_0 and β_1) three times, where each time it is trained on two of the folds, leaving out one of the folds as validation data. Then for the three trained models, we estimate the test error using the validation data, since it was not used to train the model.

Our focus will be on Model A throughout the question, the other models are jsut there to pose concrete examples of other models to consider.

a. (3 points) Please load `gmp.txt` onto your computer save it as `gmp`. Create a new column in the dataframe called `pop` that is created by dividing the `gmp` column by the `pcgmp` column. Print the first three rows of `gmp`.

```
# Your answer to question 3.a here.
```

b. (7 points) In class, we estimated values of the parameters β_0 and β_1 in Model A by minimizing the training MSE of the data, i.e. by minimizing

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \beta_0 X_i^{\beta_1})^2$$

over all β_0, β_1 where n is the number of data points. Write a function `mse.calculator()` that takes as input a dataframe `x` (assumed to have columns titled `gmp` and `pop`) and two estimates `b0` and `b1`. The function should return the training mean square error of the data using the estimated values `b0` and `b1` of β_0 and β_1, respectively.

Test it on the `gmp` data with the values $\beta_0 = 6600$ and $\beta_1 = 0.12$ and show the output.

```
# Your answer to question 3.b here.
```

c. (5 points) Notice the column `fold` in the `gmp` data frame. It consists of the values 1, 2, and 3 indicating the randomly selected fold of that data point. In other words, `fold` is a column with the values 1-3 each occurring 122 times (`gmp` has 366 rows and 366 divided by 3 is 122) in random order. Create a vector `fold.vec` with a single line of code that could have produced this column in the data frame. Print the first ten elements of `fold.vec`.

```
set.seed(1)

# Your answer to question 3.c here.  Do not remove the set.seed(1) command.
```

d. (7 points) Now using cross validation, we'd like to produce an estimate of the test error for Model A. Suppose we trained the model on folds 2 and 3, leaving out fold 1 as validation data, and received the following estimates of the parameters: $\hat{\beta}_0 = 6600, \hat{\beta}_1 = 0.122$ Using fold 1 like a test dataset (since it was not used to train this model), estimate the test error of Model A using your `mse.calculator()` function from 3.a. Note that the folds are determined by the `fold` column in the data.

(If you are unable to write the `mse.calculator()` function, write the code that solves this question as if the `mse.calculator()` function exists. Since it doesn't you will need to comment this code out so it doesn't try to run.)

```
# Your answer to question 3.d here.
```

e. (7 points) Now suppose we have the following three estimates of β_0 and β_1 from our cross-validation procedure:

- Validation Data is fold 1: $\hat{\beta}_0 = 6600, \hat{\beta}_1 = 0.122$
- Validation Data is fold 2: $\hat{\beta}_0 = 6700, \hat{\beta}_1 = 0.130$
- Validation Data is fold 3: $\hat{\beta}_0 = 6580, \hat{\beta}_1 = 0.119$

Calculate the cross-validation estimate of the test error for Model A. Again, you will need to use your `mse.calculator()` function from 3.a.

(If you are unable to write the `mse.calculator()` function, write the code that solves this question as if the `mse.calculator()` function exists. Since it doesn't you will need to comment this code out so it doesn't try to run.)

```
# Your answer to question 3.e here.
gmp <- read.table('gmp.txt',header = TRUE)
gmp$pop <- gmp$gmp/gmp$pcgmp
head(gmp,3)
```

```
##          city        gmp pcgmp fold      pop
## 1 Abilene, TX 3.8870e+09 24490    1 158717.8
## 2   Akron, OH 2.2998e+10 32889    2 699261.2
## 3  Albany, GA 3.9550e+09 24269    2 162965.1
```

f. (2 points) Assume that you receive the following estimates of the test error for the four models using cross validation. Which model do you select?

- Model A: CV Error =
- Model B: CV Error =
- Model C: CV Error =
- Model D: CV Error =

Type your answer here: