

Lab: Week 4

36-350 – Statistical Computing

Week 4 – Fall 2020

Name: Kyle Wagner

Andrew ID: Kowagner

You must submit **your own** lab as a PDF file on Gradescope.

Apply: Base R

You are given the following 8×8 matrix:

```
set.seed(1001)
mat = matrix(rnorm(64),nrow=8)
mat[5,8] = mat[6,7] = mat[4,2] = NA
```

Question 1

(4 points)

Notes 4A (6,10,12)

Compute the mean for each row and for each column using both `apply()` and either `rowMeans()` or `colMeans()`. (So there should be four function calls overall.) Deal with the NAs by passing (an) additional argument(s) to these functions, when possible.

```
apply(mat,1,mean,na.rm = TRUE)
```

```
## [1]  0.29520657  0.51323956 -0.46939506 -0.46438383 -0.19427723  0.02530351
## [7]  0.06268717  0.16192061
```

```
apply(mat,2,mean,na.rm = TRUE)
```

```
## [1] -0.1141279 -0.1176691  0.3097320 -0.3976337  0.2625528 -0.1828815
0.0382058
## [8]  0.2301255
```

```
rowMeans(mat,TRUE)
```

```
## [1] 0.29520657 0.51323956 -0.46939506 -0.46438383 -0.19427723 0.02530351
## [7] 0.06268717 0.16192061
```

```
colMeans(mat,TRUE)
```

```
## [1] -0.1141279 -0.1176691 0.3097320 -0.3976337 0.2625528 -0.1828815
0.0382058
## [8] 0.2301255
```

Question 2

(4 points)

Function writing review

How does the `Income` variable in R's `state.x77` matrix correlate with other variables? Write a function called `cor_var()` that takes two inputs: `v1`, a numeric vector; and `v2`, another numeric vector whose default value is `state.x77[, "Income"]`. Its output should be the correlation of `v1` and `v2`, computed via the `cor()` function. Check that `cor_var(v1=state.x77[, "Life Exp"])` gives you 0.3402553, and `cor_var(v1=state.x77[, "Income"])` gives you 1.

```
cor_var <- function(v1,v2 = state.x77[, "Income"]) {
  return(cor(v1,v2))
}
```

```
cor_var(v1=state.x77[, "Life Exp"])
```

```
## [1] 0.3402553
```

```
cor_var(v1=state.x77[, "Income"])
```

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

Question 3

(4 points)

Notes 4A (6-7,9-10)

Using `apply()` and the function `cor_var()` that you defined in the last question, calculate the correlation between each one of the 8 variables in the `state.x77` matrix and the `Population` variable. Display these correlations.

```
apply(as.matrix(colnames(state.x77)),1,function(x) {return(cor_var(v1 = state.x77[,x],v2 = state.x77[, "Population"])
```

```
## [1] 1.00000000 0.20822756 0.10762237 -0.06805195 0.34364275 -0.09848975
## [7] -0.33215245 0.02254384
```

Question 4

(4 points)

Notes 4A (6,10)

Using `apply()` and the base R `stats` package function `cor()`, display the Spearman correlation between each one of the eight variables in the `state.x77` matrix and the `Frost` variable. (Note that `Spearman` is not the default value for the `method` argument to the `cor()` function.)

```
apply(as.matrix(colnames(state.x77)),1,function(x) {return(cor(state.x77[,x],state.x77[, "Frost"],method
```

```
## [1] -0.4588526 0.1968638 -0.6831936 0.2983910 -0.5438432 0.3985351 1.0000000
## [8] 0.1122878
```

Variations on Apply: Base R

Question 5

(4 points)

Notes 4B (6)

Create a data frame called `state.df` from the matrix `state.x77` and the factors `state.region` and `state.division`. Be sure to name the two new columns appropriately. Using `state.df` and `tapply()`, compute the average population in each of the four defined regions of the U.S. Display the name of the region has the largest average population (and only that name). Then compute the average population in each of the nine defined divisions of the U.S., and display the name of the division has the largest average population (and only that name). Hint: the names may be displayed using a combination of `names()` and `which.max()`.

```
class(state.x77)
```

```
## [1] "matrix" "array"
```

```
state.df = as.data.frame(state.x77)
state.df$region = state.region
state.df$division = state.division
```

```
tapply(state.df$Population,INDEX = state.df$region,mean)
```

```
##      Northeast      South North Central      West
##      5495.111      4208.125      4803.000      2915.308
```

```
div_avg <- tapply(state.df$Population,INDEX = state.df$division,mean)
names(which.max(div_avg))
```

```
## [1] "Middle Atlantic"
```

Question 6

(4 points)

Notes 4A (5) and Notes 4B (3-4)

Split the rows of the data frame `state.df` by state divisions, and call the resulting list `state.df.by.div`. Then use `lapply()` to display just the first two rows of each data frame in the list `state.df.by.div`.

```
state.df.by.div = split(state.df,f = state.df$division)
```

```
my.fun = function(x) {return(x[1:2,])}
```

```
lapply(state.df.by.div,my.fun)
```

```
## $'New England'
##           Population Income Illiteracy Life Exp Murder HS Grad Frost Area
## Connecticut      3100    5348         1.1   72.48    3.1   56.0   139  4862
## Maine              1058    3694         0.7   70.39    2.7   54.7   161 30920
##           region      division
## Connecticut Northeast New England
## Maine         Northeast New England
##
## $'Middle Atlantic'
##           Population Income Illiteracy Life Exp Murder HS Grad Frost Area
## New Jersey      7333    5237         1.1   70.93    5.2   52.5   115  7521
## New York        18076   4903         1.4   70.55   10.9   52.7    82 47831
##           region      division
## New Jersey Northeast Middle Atlantic
## New York   Northeast Middle Atlantic
##
## $'South Atlantic'
##           Population Income Illiteracy Life Exp Murder HS Grad Frost Area
## Delaware         579    4809         0.9   70.06    6.2   54.6   103  1982
## Florida          8277   4815         1.3   70.66   10.7   52.6    11 54090
##           region      division
## Delaware         South South Atlantic
## Florida          South South Atlantic
##
## $'East South Central'
##           Population Income Illiteracy Life Exp Murder HS Grad Frost Area
## Alabama          3615   3624         2.1   69.05   15.1   41.3    20 50708
## Kentucky         3387   3712         1.6   70.10   10.6   38.5    95 39650
##           region      division
## Alabama          South East South Central
## Kentucky         South East South Central
##
## $'West South Central'
##           Population Income Illiteracy Life Exp Murder HS Grad Frost Area
## Arkansas         2110   3378         1.9   70.66   10.1   39.9    65 51945
## Louisiana        3806   3545         2.8   68.76   13.2   42.2    12 44930
##           region      division
## Arkansas         South West South Central
## Louisiana        South West South Central
##
## $'East North Central'
```

```
##      Population Income Illiteracy Life Exp Murder HS Grad Frost Area
## Illinois      11197   5107         0.9   70.14   10.3   52.6   127 55748
## Indiana       5313   4458         0.7   70.88    7.1   52.9   122 36097
##              region      division
## Illinois North Central East North Central
## Indiana  North Central East North Central
##
## $'West North Central'
##      Population Income Illiteracy Life Exp Murder HS Grad Frost Area
## Iowa      2861   4628         0.5   72.56    2.3   59.0   140 55941
## Kansas    2280   4669         0.6   72.58    4.5   59.9   114 81787
##              region      division
## Iowa  North Central West North Central
## Kansas North Central West North Central
##
## $Mountain
##      Population Income Illiteracy Life Exp Murder HS Grad Frost Area
## Arizona      2212   4530         1.8   70.55    7.8   58.1    15 113417
## Colorado     2541   4884         0.7   72.06    6.8   63.9   166 103766
##              region division
## Arizona  West Mountain
## Colorado  West Mountain
##
## $Pacific
##      Population Income Illiteracy Life Exp Murder HS Grad Frost Area
## Alaska      365   6315         1.5   69.31   11.3   66.7   152 566432
## California  21198   5114         1.1   71.71   10.3   62.6    20 156361
##              region division
## Alaska    West Pacific
## California West Pacific
```

Below, we read in a data table showing the fastest women's 100-meter sprint times.

```
sprint.df = read.table("http://www.stat.cmu.edu/~pfreeman/women_100m_with_header.dat",
                       header=TRUE,stringsAsFactors=FALSE)
class(sprint.df)
```

```
## [1] "data.frame"
```

```
head(sprint.df)
```

```
## Rank Time Wind First.Name      Last.Name Country Birthdate Race
## 1   1 10.49  0.0  Florence Griffith-Joyner USA 21.12.59 1q1
## 2   2 10.61  1.2  Florence Griffith-Joyner USA 21.12.59 1
## 3   3 10.62  1.0  Florence Griffith-Joyner USA 21.12.59 1q3
## 4   4 10.64  1.2  Carmelita Jeter USA 24.11.79 1
## 5   5 10.65  1.1   Marion Jones USA 12.10.75 1
## 6   6 10.67 -0.1  Carmelita Jeter USA 24.11.79 1
##      Location      Date
## 1 Indianapolis 16.07.1988
```

```
## 2 Indianapolis 17.07.1988
## 3      Seoul 24.09.1988
## 4      Shanghai 20.09.2009
## 5 Johannesburg 12.09.1998
## 6 Thessaloniki 13.09.2009
```

Question 7

(4 points)

Review of string processing

Extract the last four digits of each entry of the `Date` column. (Hint: you will have to use `as.character()` to convert `sprint.df$Date` from a factor variable to strings.) Create a new data frame called `new.sprint.df` that combines `sprint.df` and a new column called `Year` that contains your extracted four-digit years. Display the first five rows and all nine columns of `new.sprint.df`. Display the class of the newly created `Year` column.

```
extract <- substr(as.character(sprint.df$Date),7,11)

new.sprint.df <- sprint.df
new.sprint.df$Year <- extract
new.sprint.df[1:5,]
```

```
##   Rank  Time Wind First.Name      Last.Name Country Birthdate Race
## 1    1 10.49  0.0  Florence Griffith-Joyner    USA  21.12.59  1q1
## 2    2 10.61  1.2  Florence Griffith-Joyner    USA  21.12.59   1
## 3    3 10.62  1.0  Florence Griffith-Joyner    USA  21.12.59  1q3
## 4    4 10.64  1.2   Carmelita      Jeter    USA  24.11.79   1
## 5    5 10.65  1.1     Marion      Jones    USA  12.10.75   1
##      Location      Date Year
## 1 Indianapolis 16.07.1988 1988
## 2 Indianapolis 17.07.1988 1988
## 3      Seoul 24.09.1988 1988
## 4      Shanghai 20.09.2009 2009
## 5 Johannesburg 12.09.1998 1998
```

```
class(new.sprint.df$Year)
```

```
## [1] "character"
```

Question 8

(4 points)

Notes 4B (6)

Using `tapply()` and the newly created `Year` column, compute the median 100-meter sprint time in each year of the data frame. Call the resulting vector `med.time.by.year`. Create a table of median times. Which median time appears the most, and how many times does it appear? When is the last year that that particular median time appeared in the data?

```
med.time.by.year <- tapply(new.sprint.df$Time, INDEX = new.sprint.df$Year, median)
sort(table(med.time.by.year), decreasing = TRUE)
```

```
## med.time.by.year
## 11.03 11.04 11 11.01 11.02 11.015 11.05 11.06 11.07 10.98 10.99
##      7      7      6      5      4      2      2      2      2      1      1
## 10.995 11.005 11.035 11.045 11.055 11.08
##      1      1      1      1      1      1
```

```
filter(new.sprint.df, new.sprint.df$Time == 11.03)
```

```
## Time Series:
## Start = 1
## End = 2144
## Frequency = 1
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
## 1      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 2      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 3      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 4      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 5      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 6      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 7      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 8      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 9      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 10     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 11     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 12     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 13     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 14     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 15     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 16     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 17     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 18     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 19     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 20     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 21     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 22     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 23     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 24     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 25     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 26     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 27     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 28     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 29     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 30     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 31     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 32     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 33     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 34     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 35     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
## 36     NA      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

## 1063	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1064	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1065	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1066	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1067	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1068	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1069	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1070	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1071	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1072	99043	1199.8	3098	9591	9422	2720	10920	2503	14304	51704	3536
## 1073	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1074	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1075	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1076	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1077	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1078	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1079	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1080	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1081	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1082	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1083	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1084	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1085	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1086	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1087	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1088	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1089	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1090	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1091	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1092	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1093	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1094	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1095	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1096	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1097	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1098	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1099	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1100	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1101	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1102	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1103	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1104	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1105	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1106	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1107	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1108	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1109	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1110	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1111	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1112	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1113	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1114	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1115	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
## 1116	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
## 2143    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## 2144    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
```

The time 11.03 occurs the most as 7 times. The last year that particular median time appeared was in 201

Below, we read in a data table related to the political economy of strikes.

```
strikes.df = read.csv("http://www.stat.cmu.edu/~pfreeman/strikes.csv")
class(strikes.df)
```

```
## [1] "data.frame"
```

```
head(strikes.df)
```

```
##      country year strike.volume unemployment inflation left.parliament
## 1 Australia 1951           296           1.3        19.8           43.0
## 2 Australia 1952           397           2.2        17.2           43.0
## 3 Australia 1953           360           2.5         4.3           43.0
## 4 Australia 1954             3           1.7         0.7           47.0
## 5 Australia 1955           326           1.4         2.0           38.5
## 6 Australia 1956           352           1.8         6.3           38.5
##      centralization density
## 1      0.3748588      NA
## 2      0.3751829      NA
## 3      0.3745076      NA
## 4      0.3710170      NA
## 5      0.3752675      NA
## 6      0.3716072      NA
```

```
dim(strikes.df) # Note that since 18 × 35 = 630 > 625, some years missing from some countries
```

```
## [1] 625    8
```

Question 9

(4 points)

Notes 4A (5) and Notes 4B (5)

Split `strikes.df` by country, using the `split()` function. Call the resulting list `strikes.by.country`. Using `strikes.by.country` and `sapply()`, compute the average centralization metric (a quantity related to unionization) for each country over the range of years in the file. Display the names of the countries that had the highest and lowest average centralization metric (and only the names of those countries).

```
strikes.by.country <- split(strikes.df, f = strikes.df$country)
my.fun = function(x) {return(mean(x$centralization))}
res <- sapply(strikes.by.country, my.fun)
high <- names(which.max(res))
low <- names(which.min(res))
cat("Highest:", high, "\nLowest:", low)
```

```
## Highest: Austria
## Lowest: Canada
```

Question 10

(4 points)

Notes 4B (5)

Using `strikes.by.country` and `sapply()`, compute a summary of the long-term centralization metric for each country. Study the output—do its dimensions make sense to you?

```
my.fun = function(x) {return(summary(x$centralization))}
res <- sapply(strikes.by.country, my.fun)
length(strikes.by.country)
```

```
## [1] 18
```

```
res
```

```
##      Australia  Austria  Belgium      Canada  Denmark  Finland
## Min.    0.3701921 0.9951362 0.7451018 4.985230e-06 0.4951243 0.7453985
## 1st Qu. 0.3723613 0.9963630 0.7480245 8.232258e-04 0.4971313 0.7486803
## Median 0.3745076 0.9977592 0.7489699 2.206919e-03 0.5003940 0.7501793
## Mean   0.3746440 0.9976705 0.7494852 2.244134e-03 0.4999586 0.7503741
## 3rd Qu. 0.3763172 0.9988332 0.7514769 3.468929e-03 0.5022077 0.7521806
## Max.    0.3798597 0.9997884 0.7544044 4.849537e-03 0.5048790 0.7549842
##      France  Germany  Ireland      Italy  Japan Netherlands
## Min.    0.0002446096 0.2453393 0.4951136 0.2454353 0.1205130 0.7454194
## 1st Qu. 0.0013202927 0.2477310 0.4974278 0.2490072 0.1233528 0.7474436
## Median 0.0028737475 0.2493486 0.4994846 0.2507560 0.1247869 0.7491107
## Mean   0.0027299088 0.2499682 0.4997119 0.2506995 0.1246753 0.7496027
## 3rd Qu. 0.0042529214 0.2524444 0.5022122 0.2527474 0.1261252 0.7520595
## Max.    0.0049236913 0.2548710 0.5048117 0.2547880 0.1297671 0.7540260
##      New.Zealand  Norway  Sweden Switzerland      UK      USA
## Min.    0.3706028 0.8700540 0.8701569 0.4956250 0.3701746 0.000109027
## 1st Qu. 0.3730609 0.8730289 0.8723843 0.4976971 0.3738972 0.001355673
## Median 0.3761876 0.8750384 0.8756796 0.4993706 0.3756106 0.002406464
## Mean   0.3759404 0.8753418 0.8752538 0.4999900 0.3759468 0.002390639
## 3rd Qu. 0.3786986 0.8780262 0.8778525 0.5024074 0.3785299 0.003252352
## Max.    0.3798821 0.8799584 0.8794025 0.5048787 0.3797725 0.004975356
```

The output dimensions make sense to me, because the number of columns is equal to the number of elements.

Question 11

(4 points)

Notes 4B (5)

Using `strikes.by.country` and just *one* call to `sapply()`, compute the average unemployment rate, average inflation rate, and average strike volume for each country. The output should be a matrix of dimension 3 x 18. Also, within that call, give the output matrix appropriate row names.


```
my.fun = function(x) {
  unemployment = mean(x$unemployment)
  avg_inflation = mean(x$inflation)
  avg_strike = mean(x$strike.volume)
  res_matrix = c("Avg Unemployment" =unemployment,"Average Inflation" = avg_inflation,"Avg Strike Volume" = avg_strike)
  return(res_matrix)
}

res <- sapply(strikes.by.country,my.fun)
dim(res)
```

```
## [1] 3 18
```

```
res
```

```
##           Australia  Austria  Belgium  Canada  Denmark
## Avg Unemployment   3.505714  2.540000   3.646667   6.042857   5.711429
## Average Inflation   6.594286  5.102857   4.150000   4.797143   6.582857
## Avg Strike Volume  378.600000 25.600000 244.000000 749.542857 194.828571
##           Finland  France  Germany  Ireland  Italy
## Avg Unemployment   2.571429  3.182857  3.117143   7.771429   6.725714
## Average Inflation   7.317143  6.948571  3.294286   8.151429   8.005714
## Avg Strike Volume  448.542857 185.400000 43.828571 547.428571 997.685714
##           Japan Netherlands New.Zealand  Norway  Sweden
## Avg Unemployment   1.602857   3.691429   1.002857   1.428571   2.137143
## Average Inflation   5.820000   4.814286   7.691429   6.320000   6.434286
## Avg Strike Volume  165.828571  26.114286 259.257143 75.114286 73.485714
##           Switzerland  UK  USA
## Avg Unemployment   0.3285714  3.451429  5.542857
## Average Inflation   3.4171429  7.105714  4.428571
## Avg Strike Volume   3.6571429 322.714286 448.228571
```

Question 12

(4 points)

Notes 4B (5)

Using `strikes.df`, `split()`, and `sapply()`, compute the average unemployment rate for each country, before and during 1970, and after 1970. Display the output; it should be a numeric vector of length 36. One way to perform the splitting is to define a new column called `pre1970` that indicates that a year column is less than or equal to 1970. Then use both `country` and `pre1970` to do the splitting. If you are not sure how to use both factor variables at once, look at the documentation for `split()`, specifically its argument `f`.

```
#strikes.df <- mutate(strikes.df,
#                      pre1970 = cut(year,
#                                     c(0,1970,Inf),
#                                     c("yes", "no")))

#split_both <- split(strikes.df, f = list(strikes.df$country,strikes.df$pre1970))
#my.fun = function(x) {
#  res = mean(x$unemployment)
#  return(res)
```

```
#}
#res <- sapply(split_both,my.fun)
#length(res)
#res
```

Question 13

(4 points)

Review of matrices

Using the result from above, display the difference in the average unemployment rate before and after 1970 for each country. (To be clear: subtract the pre-1970 results from the post-1970 results.) Which country had the biggest increase in average unemployment from before to after? The biggest decrease? (Hint: use the output from Q12 to populate a matrix, with pre-1970 results in one column and post-1970 results in another.)

```
#yes <- grep("[a-zA-Z]+\\.yes",names(res))
#y <- res[yes]
#no <- grep("[a-zA-Z]+\\.no",names(res))
#n <- res[no]
#dif <- matrix(c(n,y),ncol = 2)

#rownames(dif) <- unique(strikes.df$country)
#difference = dif[,1] - dif[,2]
#difference

#names(which.max(difference))
#names(which.min(difference))
```

Below, we read in Trump's nomination acceptance speech, and process it so as to create individual sentences (minus the final punctuation mark).

```
trump.lines = readLines("http://www.stat.cmu.edu/~pfreeman/trump.txt")
trump.text = paste(trump.lines, collapse=" ")
trump.sentences = strsplit(trump.text,split="\. |\? |\! |\.$")
```

Question 14

(4 points)

Notes 4B

Use an appropriate base R `apply()`-style function to determine the number of characters in each sentence in Trump's speech. Display the result via `table()`. Utilize `which()` to display the (two) most common number of characters in a Trump sentence.

```
res <- lapply(trump.sentences,nchar)
results <- table(res)

names(which.max(results))
```

```
## [1] "23"
```

```
names(sort(results,decreasing = TRUE))[2]
```

```
## [1] "46"
```

```
suppressWarnings(library(tidyverse))
```

```
## -- Attaching packages ----- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2      v purrr  0.3.4
## v tibble  3.0.1      v dplyr  1.0.2
## v tidyr   1.1.2      v stringr 1.4.0
## v readr   1.3.1      v forcats 0.5.0
```

```
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()
```

Question 15

(4 points)

Notes 4C (4-6)

Convert the following base R code so as to utilize pipes. (Feel free to run the base R code first to see the expected output, both here and in other code chunks below. Also: don't forget about dot notation! It will help you at various points...)

```
paste(toupper(letters), collapse="+")
```

```
## [1] "A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+U+V+W+X+Y+Z"
```

```
letters %>%
  toupper %>%
  paste(collapse = "+")
```

```
## [1] "A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+U+V+W+X+Y+Z"
```

FILL ME IN

Question 16

(4 points)

Notes 4C (4-6,8-9)

Convert the following base R code so as to utilize pipes. (`trimws` means “trim white space”; `gsub` was covered at the very end of the notes in Week 3.)

```
trimws(gsub("une", "un", "      Ceci n'est pas une pipe      "))
```

```
## [1] "Ceci n'est pas un pipe"
```

```
"une" %>%  
  gsub("un", "      Ceci n'est pas une pipe      ") %>%  
  trimws()
```

```
## [1] "Ceci n'est pas un pipe"
```

FILL ME IN

Question 17

(4 points)

Notes 4C (4-6,8-9)

Convert the following base R code so as to utilize pipes.

```
state.name[which.max(state.x77[, "Illiteracy"])]
```

```
## [1] "Louisiana"
```

```
state.x77[, "Illiteracy"] %>%  
  which.max %>%  
  state.name[.]
```

```
## [1] "Louisiana"
```

Question 18

(4 points)

Notes 4C (4-6,8-9)

Convert the following base R code so as to utilize pipes. (Hint: when dealing with `words = words[words != ""]`, you'll want to utilize dot notation...and to realize that you can use a dot more than once in an expression. A dot may be used as if it was any other variable.)

```

str.url = "http://www.stat.cmu.edu/~pfreeman/clinton.txt"
lines = readLines(str.url)
text = paste(lines, collapse=" ")
words = unlist(strsplit(text, split="[:space:]|[:punct:]"))
words = words[words != ""]
wordtab = table(words)
wordtab = sort(wordtab, decreasing=TRUE)
head(wordtab, 10)

```

```

## words
## the to and a of we I in you s
## 200 177 163 102 98 93 85 77 76 75

```

```

"http://www.stat.cmu.edu/~pfreeman/clinton.txt" %>%
  readLines %>%
  paste(collapse=" ") %>%
  strsplit(split="[:space:]|[:punct:]") %>%
  unlist() %>%
  .[, != ""] %>%
  table(.) %>%
  sort(decreasing=TRUE) %>%
  head(10)

```

```

## .
## the to and a of we I in you s
## 200 177 163 102 98 93 85 77 76 75

```

Question 19

(4 points)

Pipes + Notes 4D (6)

How does the `Frost` variable in R's `state.x77` matrix correlate with other variables? Cast `state.x77` to a data frame, and, using pipes, generate the correlation matrix for `Frost` and `Life.Exp`. (Note that the act of casting changed the name of the life expectancy column from `Life Exp` to `Life.Exp`.) The off-diagonal elements of the matrix should be 0.262068.

```

datum = state.x77[,c("Frost", "Life Exp")]
state.x77[,c("Frost", "Life Exp")] %>%
  as.data.frame() %>%
  cor()

```

```

##           Frost Life Exp
## Frost      1.000000 0.262068
## Life Exp    0.262068 1.000000

```

Question 20

(4 points)

Pipes + Notes 4D (9)

Take the `state.df` data frame defined below and mutate it so as to create a new column: `GradLit`. This column should have, for each row in the data frame, the percentage of high school graduates divided by the percentage of literate (note: *literate*, not *illiterate*) individuals, times 100. Then pipe the output so as to compute the median value of `GradLit`. (There is a bit of weirdness here: due to environmental issues, your call to `median()` will not work unless it is placed within curly braces. You are only surrounding `median()` with curly braces...not the entire pipe stream!) Your final value should be 53.59844.

```
mutate(state.df, GradLit = 100 * 'HS Grad' / (100 - Illiteracy)) %>%
  {median(.$GradLit)}
```

```
## [1] 53.59844
```

Question 21

(4 points)

Pipes + Notes 4D (5,6,8)

Take the `state.df` data frame and (1) select all states in the South region, and (2) display the result ordered by the decreasing product of income and life expectancy. In the end, display just the state name and the computed product. There is a quirk here: selecting rows can lead to the loss of row names. (This means that here, you will have a final result but not know which states they correspond to.) To preserve the identity of the states, pipe `state.df` to the function `rownames_to_column("give column name here, like State")`, then do the rest of your piping.

```
state.df %>%
  rownames_to_column("State Name") %>%
  filter(., region == "South") %>%
  mutate(., inc_exp = Income * 'Life Exp') %>%
  arrange(., desc(inc_exp)) %>%
  select(., 'State Name', inc_exp)
```

```
##      State Name  inc_exp
## 1      Maryland 372095.8
## 2      Florida 340227.9
## 3      Delaware 336918.5
## 4      Virginia 329446.1
## 5        Texas 296929.2
## 6      Oklahoma 284465.9
## 7        Georgia 280397.1
## 8 North Carolina 268188.8
## 9      Tennessee 267890.3
## 10     Kentucky 260211.2
## 11 West Virginia 251309.2
## 12        Alabama 250237.2
## 13 South Carolina 247034.6
## 14      Louisiana 243754.2
## 15      Arkansas 238689.5
## 16 Mississippi 210942.8
```

Below we read in a data frame `pros.df` containing measurements on men with prostate cancer. For details on the individual columns, see this web page.

```
pros.df = read.table("http://www.stat.cmu.edu/~pfreeman/pros.txt")
head(pros.df)
```

```
##      lcavol  lweight age      lbph svi      lcp gleason pgg45      lpsa
## 1 -0.5798185 2.769459 50 -1.386294 0 -1.386294      6      0 -0.4307829
## 2 -0.9942523 3.319626 58 -1.386294 0 -1.386294      6      0 -0.1625189
## 3 -0.5108256 2.691243 74 -1.386294 0 -1.386294      7     20 -0.1625189
## 4 -1.2039728 3.282789 58 -1.386294 0 -1.386294      6      0 -0.1625189
## 5  0.7514161 3.432373 62 -1.386294 0 -1.386294      6      0  0.3715636
## 6 -1.0498221 3.228826 50 -1.386294 0 -1.386294      6      0  0.7654678
```

Question 22

(4 points)

Pipes + Notes 4D (5-6)

Among the men whose `lcp` value is equal to its minimum value, report the lowest and highest `lpsa` score. (Hint: look up the `range()` function.)

```
pros.df %>%
  filter(.$lcp == min(.$lcp)) %>%
  range(.$lpsa)
```

```
## [1] -1.386294 80.000000
```

Question 23

(4 points)

Pipes + Notes 4D (5-6,8)

Order the rows by decreasing `age`, then decreasing `lpsa` score, and display the rows from men who are older than 70, but only the `age`, `lpsa`, `lcavol`, and `lweight` columns.

```
pros.df %>%
  arrange(.,desc(age),desc(lpsa)) %>%
  filter(age > 70 ) %>%
  select(.,age,lpsa,lcavol,lweight)
```

```
##      age      lpsa      lcavol  lweight
## 47   79  2.5687881  2.7278528 3.995445
## 78   78  3.4355988  2.5376572 4.354784
## 83   77  3.5652984  2.6130067 3.888754
## 72   77  3.0373539  1.1600209 3.341093
```

```
## 90 76 3.9936030 1.5623463 3.695110
## 3 74 -0.1625189 -0.5108256 2.691243
## 61 73 2.8419982 0.4574248 4.524502
## 37 73 2.1575593 1.4231083 3.657131
## 77 72 3.3928291 2.0108950 4.433789
## 70 72 2.9729753 1.1939225 4.780383
## 68 72 2.9626924 2.1983351 4.050915
## 63 72 2.8535925 2.7757089 3.524889
## 33 71 2.0082140 1.2753628 3.037354
```

Question 24

(4 points)

Pipes + Notes 4D (6,8,11)

Display the first six rows of the `lpsa` column only, in decreasing order, with the column renamed as `Log.Prostate.Specific.Antigen`.

```
pros.df %>%
  rename(.,Log.Prostate.Specific.Antigen = lpsa) %>%
  arrange(.,desc(Log.Prostate.Specific.Antigen)) %>%
  select(.,Log.Prostate.Specific.Antigen) %>%
  head(.,6)
```

```
##      Log.Prostate.Specific.Antigen
## 97                      5.582932
## 96                      5.477509
## 95                      5.143124
## 94                      4.684443
## 93                      4.385147
## 92                      4.129551
```

Question 25

(4 points)

Pipes + Notes 4D (6)

We haven't officially covered plotting yet, but... utilize piping and selection to make a basic R plot showing `lpsa` along the x-axis and `lcavol` along the y-axis. Include the argument `pch=19` in the call to `plot()`. Hint: when a two-column data frame is passed to `plot()`, it will by default map the first column to the x-axis and the second column to the y-axis.

```
pros.df %>%
  select(.,lpsa,lcavol) %>%
  plot(pch = 19)
```


