STATS 419 Analysis of Multivariate Analysis

Week 3 Assignment

Michaela Bayerlova
(michaela.bayerlova@wsu.edu)
[]

Instructor: Monte J. Shaffer

22 September 2020

```
library(devtools); # required for function source_url to work
github.path = "https://raw.githubusercontent.com/MichaelaB1/WSU_STATS419_FALL2020/";
source_url(paste0(github.path, "master/functions/libraries.R"));

source_url(paste0(github.path, "master/functions/functions-imdb.R"));
```

1 Matrix

Create the "rotate matrix" functions described in lecture. Apply to the example "myMatrix".

```
source_url(paste0(github.path, "master/functions/functions-matrix.R"));

myMatrix = matrix( c (
                              1, 0, 2,
                              0, 3, 0,
                              4, 0, 5
                                 ), nrow=3, byrow=T);

transposeMatrix = function(mat)
{
  t(mat);
          }
```

```
# clockwise
rotateMatrix90 = function(mat)
{
  t(mat[nrow(mat):1,,drop=FALSE]);
}

rotateMatrix180 = function(mat)
{
  rotateMatrix90(rotateMatrix90(mat));
```

```
}

rotateMatrix270 = function(mat)
{
  rotateMatrix90(rotateMatrix90(rotateMatrix90(mat)));
}


# counter clockwise
rotateMatrix90_cc = function(mat)
{
  apply(t(mat), 2, rev);
}

rotateMatrix180_cc = function(mat)
{
  rotateMatrix90_cc(rotateMatrix90_cc(mat));
}

rotateMatrix270_cc = function(mat)
{
  rotateMatrix90_cc(rotateMatrix90_cc(rotateMatrix90_cc(mat)));
}
```

```
#source_url(paste0("https://raw.githubusercontent.com/MichaelaB1/WSU_STATS419_FALL2020/master/WEEK-03/fu
# Rotate clockwise
rotateMatrix90(myMatrix);
```

```
##      [,1] [,2] [,3]
## [1,]    4    0    1
## [2,]    0    3    0
## [3,]    5    0    2
```

```
rotateMatrix180(myMatrix);
```

```
##      [,1] [,2] [,3]
## [1,]    5    0    4
## [2,]    0    3    0
## [3,]    2    0    1
```

```
rotateMatrix270(myMatrix);
```

```
##      [,1] [,2] [,3]
## [1,]    2    0    5
## [2,]    0    3    0
## [3,]    1    0    4
```

```
# Rotate counter clockwise
rotateMatrix90_cc(myMatrix);
```

```
##      [,1] [,2] [,3]
## [1,]    2    0    5
## [2,]    0    3    0
## [3,]    1    0    4
```

```
rotateMatrix180_cc(myMatrix);
```

```
##      [,1] [,2] [,3]
## [1,]    5    0    4
## [2,]    0    3    0
## [3,]    2    0    1
```

```
rotateMatrix270_cc(myMatrix);
```

```
##      [,1] [,2] [,3]
## [1,]    4    0    1
## [2,]    0    3    0
## [3,]    5    0    2
```

2 IRIS Scatterplot

Recreate the graphic for the IRIS Data Set using R. Same titles, same scales, same colors. See: https://en.wikipedia.org/wiki/Iris_flower_data_set#/media/File:Iris_dataset_scatterplot.svg

```
# Iris Data Set
IRIS_Data = iris
class(iris)
```

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

```
# Scatterplot
pairs(IRIS_Data[1:4], main = "Iris Data (red=setosa,green=versicolor,blue=virginica)",
      pch = 21, bg = c("red", "green", "blue")[unclass(IRIS_Data$Species)])
```

3 IRIS Question

Right 2-3 sentences concisely defining the IRIS Data Set. Maybe search KAGGLE for a nice template. Be certain the final writeup are your own sentences (make certain you modify what you find, make it your own, but also cite where you got your ideas from). NOTE: Watch the video, Figure 8 has a +5 EASTER EGG.

The Iris flower data set is a multivariate data set. The data set contains four measurements (sepals length and width, petals length and width) for 150 records of flowers. Each is represented in the three species of iris: Iris setosa, Iris versicolor and Iris virginica. The Iris setosa is from a wide range across the Arctic sea. The Iris versicolor is found in North America, like Eastern United States and Eastern Canada. The Iris virginica is native to eastern North America.

4 Personality

Import "personality-raw.txt" into R. Remove the V00 column. Create two new columns from the current column "date_test": year and week. Stack Overflow may help: https://stackoverflow.com/questions/22439540/how-to-get-week-numbers-from-dates ... Sort the new data frame by YEAR, WEEK so the newest tests are first ... The newest tests (e.g., 2020 or 2019) are at the top of the data frame. Then remove duplicates using the unique function based on the column "md5_email". Save the data frame in the same "pipe-delimited format" (| is a pipe) with the headers. You will keep the new data frame as "personality-clean.txt" for future work (you will not upload it at this time). In the homework, for this tasks, report how many records your raw dataset had and how many records your clean dataset has.

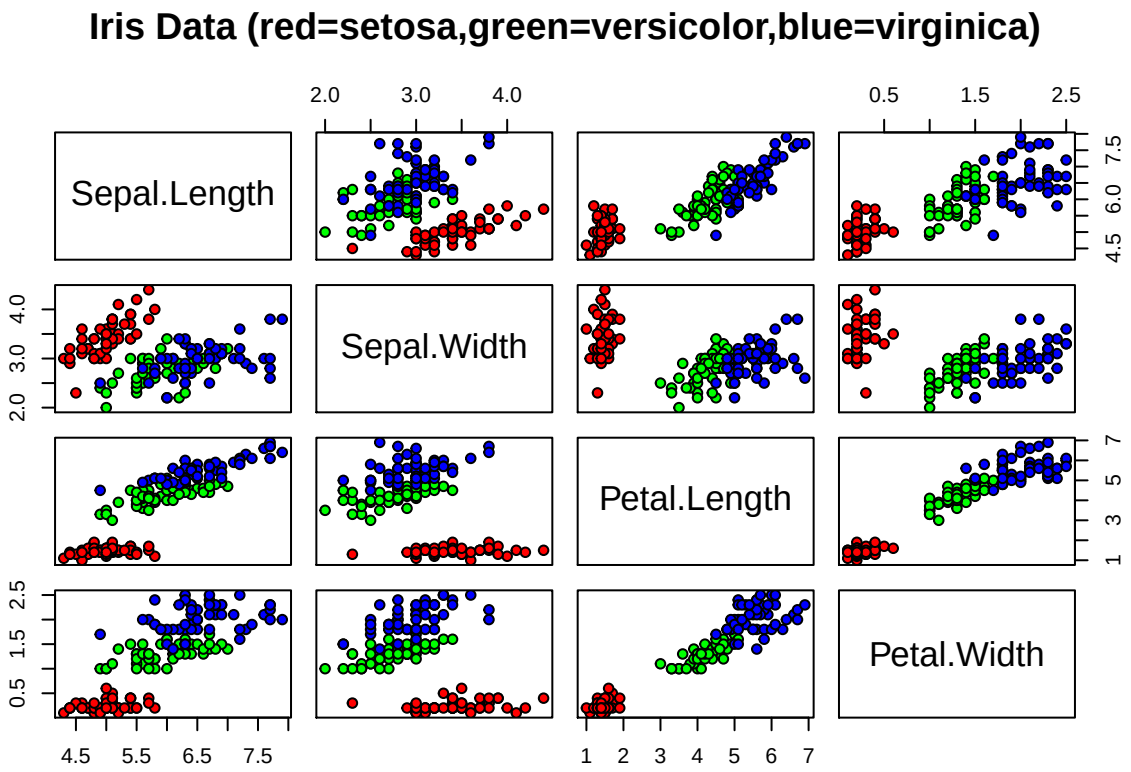


Figure 1: Iris Data scatterplot

```
personality.folder = "C:/Users/michaela.bayerlova/Documents/STATS 419/reading assignments/";
raw.file = paste0(personality.folder, "personality-raw.txt");

#my_data = read.delim("C:\\Users\\michaela.bayerlova\\Documents\\STATS 419\\reading assignments\\persona

my_data = utils::read.csv(raw.file, header=TRUE, sep="|");

head(my_data);
```

```
##                          md5_email       date_test V00 V01 V02 V03 V04 V05 V06
## 1 b62c73cdaf59e0a13de495b84030734e 5/17/2007 10:15   1 4.2 1.8 1.8 2.6 3.4 5.0
## 2 b62c73cdaf59e0a13de495b84030734e 5/17/2007 10:15   1 4.2 1.8 1.8 2.6 3.4 5.0
## 3 d41d8cd98f00b204e9800998ecf8427e 5/17/2007 10:15   1 4.2 1.8 1.8 2.6 3.4 5.0
## 4 b62c73cdaf59e0a13de495b84030734e 5/17/2007 17:06   1 5.0 5.0 1.8 5.0 4.2 2.6
## 5 b62c73cdaf59e0a13de495b84030734e 5/17/2007 17:06   1 5.0 5.0 1.8 5.0 4.2 2.6
## 6 d41d8cd98f00b204e9800998ecf8427e 5/17/2007 17:06   1 5.0 5.0 1.8 5.0 4.2 2.6
##   V07 V08 V09 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21 V22 V23 V24 V25
## 1 2.6 2.6 1.8 1.8 1.8 1.8 5.0 4.2 1.8 1.8 1.8 1.0 2.6 3.4 1.8 1.8 2.6 1.8 3.4
## 2 2.6 2.6 1.8 1.8 1.8 1.8 5.0 4.2 1.8 1.8 1.8 1.0 2.6 3.4 1.8 1.8 2.6 1.8 3.4
## 3 2.6 2.6 1.8 1.8 1.8 1.8 5.0 4.2 1.8 1.8 1.8 1.0 2.6 3.4 1.8 1.8 2.6 1.8 3.4
## 4 2.6 3.4 2.6 4.2 2.6 4.2 3.4 4.2 1.8 3.4 4.2 2.6 4.2 2.6 1.8 2.6 4.2 4.2 4.2
## 5 2.6 3.4 2.6 4.2 2.6 4.2 3.4 4.2 1.8 3.4 4.2 2.6 4.2 2.6 1.8 2.6 4.2 4.2 4.2
## 6 2.6 3.4 2.6 4.2 2.6 4.2 3.4 4.2 1.8 3.4 4.2 2.6 4.2 2.6 1.8 2.6 4.2 4.2 4.2
##   V26 V27 V28 V29 V30 V31 V32 V33 V34 V35 V36 V37 V38 V39 V40 V41 V42 V43 V44
## 1 2.6 1.8 1.0 4.2 2.6 1.8 2.6 4.2 1.8 4.2 3.4 2.6 4.2 2.6 2.6 3.4 2.6 3.4 2.6
## 2 2.6 1.8 1.0 4.2 2.6 1.8 2.6 4.2 1.8 4.2 3.4 2.6 4.2 2.6 2.6 3.4 2.6 3.4 2.6
## 3 2.6 1.8 1.0 4.2 2.6 1.8 2.6 4.2 1.8 4.2 3.4 2.6 4.2 2.6 2.6 3.4 2.6 3.4 2.6
## 4 2.6 1.8 3.4 3.4 3.4 1.8 1.8 3.4 3.4 3.4 2.6 1.8 1.8 3.4 3.4 3.4 4.2 1.8 3.4
## 5 2.6 1.8 3.4 3.4 3.4 1.8 1.8 3.4 3.4 3.4 2.6 1.8 1.8 3.4 3.4 3.4 4.2 1.8 3.4
## 6 2.6 1.8 3.4 3.4 3.4 1.8 1.8 3.4 3.4 3.4 2.6 1.8 1.8 3.4 3.4 3.4 4.2 1.8 3.4
##   V45 V46 V47 V48 V49 V50 V51 V52 V53 V54 V55 V56 V57 V58 V59 V60
## 1 3.4 3.4 3.4 2.6 2.6 3.4 5.0 1.8 2.6 4.2 4.2 3.4 2.6 3.4 3.4 4.2
## 2 3.4 3.4 3.4 2.6 2.6 3.4 5.0 1.8 2.6 4.2 4.2 3.4 2.6 3.4 3.4 4.2
## 3 3.4 3.4 3.4 2.6 2.6 3.4 5.0 1.8 2.6 4.2 4.2 3.4 2.6 3.4 3.4 4.2
## 4 3.4 4.2 3.4 5.0 3.4 3.4 1.8 1.8 3.4 3.4 3.4 4.2 2.6 2.6 2.6 2.6
## 5 3.4 4.2 3.4 5.0 3.4 3.4 1.8 1.8 3.4 3.4 3.4 4.2 2.6 2.6 2.6 2.6
## 6 3.4 4.2 3.4 5.0 3.4 3.4 1.8 1.8 3.4 3.4 3.4 4.2 2.6 2.6 2.6 2.6
```

```
nrow_raw_data <- nrow(my_data);
dim(my_data);
```

```
## [1] 838  63
```

```
# remove column V00
my_data$V00 <- NULL;

# Create two new columns from the current column "date_test":  year and week
d = strptime(my_data$date_test, format="%m/%d/%Y %H:%M");
d.year = as.numeric(strftime(d, format='%Y'));
d.week = as.numeric(strftime(d, format='%W'));
my_data$year = d.year;
my_data$week = d.week;
date_index = which(names(my_data)=="date_test");
```

```
# Remove date_test
my_data$date_test <- NULL;

# Sort the new data frame by YEAR, WEEK so the newest tests are first
# my_data <- arrange(my_data, c("year","week"),);
df.dim = dim(my_data);
df.pos = df.dim[2];
which( names(my_data)=="year" );
```

```
## [1] 62
```

```
which( names(my_data)=="week" );
```

```
## [1] 63
```

```
my_data = my_data[, c(1,62,63, 2:61)];

if(date_index == 1)
{
  reorder = c( (df.pos-1):df.pos, 2:(df.pos-2) );
}else{
     reorder = c( 1:(date_index-1), (df.pos-1):df.pos,  2:(df.pos-2) );
     }

my_data = my_data[, reorder];
my_data_sorted = my_data[ order(-my_data[,date_index],-my_data[,(1+date_index)]), ];
head(my_data_sorted[,1:5]);
```

```
##                           md5_email V59 V60 year week
## 19 42235880933cfbca2103b1b5cb22b5bb   5   5 2007   22
## 28 cb916b1a2f0428f21335bf12f59641e2   5   5 2007   23
## 61 1664a456689f5602bb6954eb897e423e   5   5 2008   13
## 65 ade6e800f168cc2e638538cef9e6e9a6   5   5 2008   20
## 76 1664a456689f5602bb6954eb897e423e   5   5 2008   27
## 86 72707bb71794071355aa539019e040ea   5   5 2008   27
```

```
# Remove duplicates using the unique function based on the column "md5_email", only leave most recent
#my_data <- my_data[!duplicated(my_data£md5_email),];
u = unique( my_data_sorted["md5_email"] );
my_data_clean = my_data_sorted[!duplicated(my_data_sorted["md5_email"]), ];

# In the homework, for this tasks, report how many records your raw dataset had and
# how many records your clean dataset has
nrow_clean_data <- nrow(my_data_clean)
dim(my_data_clean);
```

```
## [1] 678  63
```

```
#install.packages("data.table")
#library(data.table);
#fwrite(my_data_clean,"personality-clean.txt", sep="|",col.names = TRUE, row.names = FALSE);
```

```
clean.file = paste0(personality.folder, "personality-clean.txt");


utils::write.table(my_data_clean, file=clean.file, quote=FALSE, col.names=TRUE, row.names=FALSE, sep="|"


#utils::read.csv(mycache, header=TRUE, sep="|");
```

5 Variance and Z-scores

Write functions for doSummary and sampleVariance and doMode ... test these functions in your homework on the "monte.shaffer@gmail.com" record from the clean dataset. Report your findings. For this "monte.shaffer@gmail.com" record, also create z-scores. Plot(x,y) where x is the raw scores for "monte.shaffer@gmail.com" and y is the z-scores from those raw scores. Include the plot in your assignment, and write 2 sentences describing what pattern you are seeing and why this pattern is present.

```
library(humanVerseWSU);
```

```
##
## Attaching package: 'humanVerseWSU'
```

```
## The following objects are masked _by_ '.GlobalEnv':
##
##     rotateMatrix, rotateMatrix180, rotateMatrix270, rotateMatrix90,
##     rotateMatrixAngle, transposeMatrix
```

```
x.norm = rnorm(100,0,1);
s.norm = doStatsSummary ( x.norm );
str(s.norm);  # mode is pretty meaningless on this data
```

```
## List of 32
##  $ length           : int 100
##  $ length.na        : int 0
##  $ length.good      : int 100
##  $ mean             : num 0.162
##  $ mean.trim.05     : num 0.163
##  $ mean.trim.20     : num 0.121
##  $ median           : num 0.109
##  $ MAD              : num 0.966
##  $ IQR              : num 1.37
##  $ quartiles        : Named num [1:3] -0.529 0.109 0.84
##   ..- attr(*, "names")= chr [1:3] "25%" "50%" "75%"
##  $ deciles          : Named num [1:9] -1.163 -0.738 -0.41 -0.142 0.109 ...
##   ..- attr(*, "names")= chr [1:9] "10%" "20%" "30%" "40%" ...
##  $ centiles         : Named num [1:99] -2.35 -1.76 -1.66 -1.62 -1.5 ...
##   ..- attr(*, "names")= chr [1:99] "1%" "2%" "3%" "4%" ...
##  $ median.weighted  : num 0.966
##  $ MAD.weighted     : num 0.109
##  $ max              : num 2.42
##  $ min              : num -2.39
##  $ range            : num 4.8
##  $ xlim             : num [1:2] -2.39 2.42
##  $ max.idx          : num 83
```

```
##  $ min.idx          : num 15
##  $ freq.max         : num [1:100] -2.39 -2.35 -1.74 -1.66 -1.62 ...
##  $ mode             : num [1:100] -2.39 -2.35 -1.74 -1.66 -1.62 ...
##  $ which.min.freq   : num [1:100] -2.39 -2.35 -1.74 -1.66 -1.62 ...
##  $ ylim             : int [1:2] 1 1
##  $ sd               : num 1.06
##  $ var              : num 1.13
##  $ var.naive        :List of 3
##   ..$ x.bar: num 0.162
##   ..$ s.var: num 1.13
##   ..$ s.sd : num 1.06
##  $ var.2step        :List of 3
##   ..$ x.bar: num 0.162
##   ..$ s.var: num 1.13
##   ..$ s.sd : num 1.06
##  $ shapiro          :List of 4
##   ..$ statistic: Named num 0.988
##   .. ..- attr(*, "names")= chr "W"
##   ..$ p.value  : num 0.524
##   ..$ method   : chr "Shapiro-Wilk normality test"
##   ..$ data.name: chr "xx"
##   ..- attr(*, "class")= chr "htest"
##  $ shapiro.is.normal:List of 3
##   ..$ 0.10: logi TRUE
##   ..$ 0.05: logi TRUE
##   ..$ 0.01: logi TRUE
##  $ outliers.z       :'data.frame':   0 obs. of  2 variables:
##   ..$ value    : Factor w/ 0 levels:
##   ..$ direction: Factor w/ 0 levels:
##  $ outliers.IQR     :'data.frame':   0 obs. of  3 variables:
##   ..$ value    : Factor w/ 0 levels:
##   ..$ fence    : Factor w/ 0 levels:
##   ..$ direction: Factor w/ 0 levels:
```

```
x.unif = runif(100,0,1);
s.unif = doStatsSummary ( x.unif );
str(s.unif);  # mode is pretty meaningless on this data
```

```
## List of 32
##  $ length           : int 100
##  $ length.na        : int 0
##  $ length.good      : int 100
##  $ mean             : num 0.475
##  $ mean.trim.05     : num 0.472
##  $ mean.trim.20     : num 0.46
##  $ median           : num 0.422
##  $ MAD              : num 0.33
##  $ IQR              : num 0.438
##  $ quartiles        : Named num [1:3] 0.272 0.422 0.71
##   ..- attr(*, "names")= chr [1:3] "25%" "50%" "75%"
##  $ deciles          : Named num [1:9] 0.116 0.207 0.293 0.34 0.422 ...
##   ..- attr(*, "names")= chr [1:9] "10%" "20%" "30%" "40%" ...
##  $ centiles         : Named num [1:99] 0.0221 0.0243 0.0502 0.0542 0.0741 ...
##   ..- attr(*, "names")= chr [1:99] "1%" "2%" "3%" "4%" ...
```

```
##  $ median.weighted  : num 0.33
##  $ MAD.weighted     : num 0.422
##  $ max              : num 0.986
##  $ min              : num 0.0177
##  $ range            : num 0.969
##  $ xlim             : num [1:2] 0.0177 0.9865
##  $ max.idx          : num 26
##  $ min.idx          : num 88
##  $ freq.max         : num [1:100] 0.0177 0.0221 0.0243 0.051 0.0544 ...
##  $ mode             : num [1:100] 0.0177 0.0221 0.0243 0.051 0.0544 ...
##  $ which.min.freq   : num [1:100] 0.0177 0.0221 0.0243 0.051 0.0544 ...
##  $ ylim             : int [1:2] 1 1
##  $ sd               : num 0.279
##  $ var              : num 0.078
##  $ var.naive        :List of 3
##   ..$ x.bar: num 0.475
##   ..$ s.var: num 0.078
##   ..$ s.sd : num 0.279
##  $ var.2step        :List of 3
##   ..$ x.bar: num 0.475
##   ..$ s.var: num 0.078
##   ..$ s.sd : num 0.279
##  $ shapiro          :List of 4
##   ..$ statistic: Named num 0.95
##   .. ..- attr(*, "names")= chr "W"
##   ..$ p.value  : num 0.000787
##   ..$ method   : chr "Shapiro-Wilk normality test"
##   ..$ data.name: chr "xx"
##   ..- attr(*, "class")= chr "htest"
##  $ shapiro.is.normal:List of 3
##   ..$ 0.10: logi FALSE
##   ..$ 0.05: logi FALSE
##   ..$ 0.01: logi FALSE
##  $ outliers.z       :'data.frame':   0 obs. of  2 variables:
##   ..$ value    : Factor w/ 0 levels:
##   ..$ direction: Factor w/ 0 levels:
##  $ outliers.IQR     :'data.frame':   0 obs. of  3 variables:
##   ..$ value    : Factor w/ 0 levels:
##   ..$ fence    : Factor w/ 0 levels:
##   ..$ direction: Factor w/ 0 levels:
```

```
# https://raw.githubusercontent.com/MonteShaffer/humanVerseWSU/master/humanVerseWSU/R/functions-stats.R
# https://raw.githubusercontent.com/MonteShaffer/humanVerseWSU/master/humanVerseWSU/R/functions-vector.

#####################################################################

doStatsSummary = function(x)
    {
    result = list();
        result$length = length(x);
    xx = stats::na.omit(x);
        result$length.na = length(x) - length(xx);
        result$length.good = length(xx);
    result$mean = mean(xx);
```

```
    result$mean.trim.05 = mean(xx, trim=0.05);
    result$mean.trim.20 = mean(xx, trim=0.20);

    result$median = stats::median(xx);
    result$MAD = stats::mad(xx);
    result$IQR = stats::IQR(xx);
    result$quartiles = stats::quantile(xx, prob=c(.25,.5,.75));
    result$deciles = stats::quantile(xx, prob=seq(0.1,0.9,by=0.1) );
    result$centiles = stats::quantile(xx, prob=seq(0.01,0.99,by=0.01) );

    result$median.weighted = matrixStats::weightedMad(xx);
    result$MAD.weighted = matrixStats::weightedMedian(xx);

    result$max = max(xx);
    result$min = min(xx);
    result$range = result$max - result$min;
    result$xlim = range(xx);

    result$max.idx = whichMax(x);
    result$min.idx = whichMin(x);

    result$mode = result$freq.max = doMode(x);  # elements with highest frequency
    result$which.min.freq = doModeOpposite(x);

    result$ylim = c( freqMin(xx), freqMax(xx) );

    # you could later get indexes of each mode(freq.max)/freq.min using findAllIndexesWithValueInVector

    result$sd = stats::sd(xx);
    result$var = stats::var(xx);

    result$var.naive = doSampleVariance(x,"naive");
    result$var.2step = doSampleVariance(x,"2step");


    ## normality
    result$shapiro = stats::shapiro.test(xx);
    result$shapiro.is.normal = list("0.10" = isTRUE(result$shapiro$p.value > 0.10), "0.05" = isTRUE(res

    result$outliers.z = findOutliersUsingZscores(x);
    result$outliers.IQR = findOutliersUsingIQR(x);

    #resultfz = calculateZscores(x);

    result;
  }

#########################################################################

doSampleVariance = function(x, method="two-pass")
    {
    x = stats::na.omit(x);
    if(method=="naive")
```

```
        {
        n = 0;
        sum = 0;
        sum2 = 0;

        for(i in 1:length(x))  ## stats::na.omit(x)
            {
            n = n + 1;
            sum = sum + x[i];
            sum2 = sum2 + x[i]*x[i];
            }

        if(n < 2) { return(NULL);} #
            x.bar = sum/n;
            s.var = (sum2 - (sum*sum)/n)/(n-1);

        } else  {
                # two-pass algorithm # testing
                n = sum = sum2 = 0;
                ## first pass
                for(i in 1:length(x))  ## stats::na.omit(x)
                    {
                    n = n + 1;
                    sum = sum + x[i];
                    }
        if(n < 2) { return(NULL);} #
                x.bar = sum/n;
                ## second pass
                for(i in 1:length(x))  ## stats::na.omit(x)
                    {
                    deviation = x[i] - x.bar;
                    sum2 = sum2 + deviation * deviation;
                    }
                s.var = sum2/(n-1);
                }

        s.sd = sqrt(s.var);
    list("x.bar"=x.bar,"s.var"=s.var,"s.sd"=s.sd);
    }

#####################################################################

doMode = function(x) # alias ?
    {
    whichMaxFreq(x);
  }


whichMaxFreq = function(x)  # doMode
    {
    x.table = as.data.frame( table(x) );
        freq.max = max( x.table$Freq );
    x.list = x.table[x.table$Freq==freq.max,];
```

```
    xs = as.numeric( as.vector (x.list$x) );
    xs;
    }

# R does not have a "mode" function built in that will capture ties.
#   This function will.
# In the process, I wrote other functions that are also not robust in R.
# For example ?which.max versus my function ?whichMax

which.max( c(87, presidents[1:30], 87) );
```

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

```
whichMax( c(87, presidents[1:30], 87) );
```

```
## [1]  1  3 32
```

```
## a function can also be referenced using class::method notation

base::which.max( c(87, presidents[1:30], 87) );
```

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

```
#humanVerseWUS::whichMax( c(87, presidents[1:30], 87) );

# typos ...
humanVerseWSU::whichMax( c(87, presidents[1:30], 87) );
```

```
## [1]  1  3 32
```

```
## this will prevent confusion if functions have the same name (in different packages)

#####################################################################

# not a requirement for your homework, but here is a function that will do it.
calculateZscores = function(x, x.bar=NULL, s.hat=NULL)
    {
  if(is.numeric(x.bar) && is.numeric(s.hat)) { return ((x - x.bar) / s.hat);}
  # maybe throw a warning if one is null, but not the other
  if( (is.null(x.bar) + is.null(s.hat)) == 1)
      {
      warning("Only one value was entered for x.bar / s.hat ... Computing these values instead.")
      }


    dsv = doSampleVariance(x);

    x.bar = dsv$x.bar;
    s.hat = dsv$s.sd;

    if(is.null(s.hat)) { return (NULL); }  # we take care of division by zero in our custom sampleVaria

    (x - x.bar) / s.hat;
    }
```

5.1 Variance

5.1.1 Naive

```
# https://raw.githubusercontent.com/MonteShaffer/humanVerseWSU/master/humanVerseWSU/R/functions-standar

v.norm = doSampleVariance(x.norm, "naive");


# if x is really small
vsmall.df = as.data.frame( t(unlist(v.norm)) );

x.small = x.norm;
for(i in 1:20)
  {
  # every loop make it 1000 times smaller
  # notice I am looping over "i" but not using it.
  x.small = standardizeToFactor(x.small, 1/1000);
  v.small = doSampleVariance(x.small, "naive");
    v.row = t(unlist(v.small));
  vsmall.df = rbind(vsmall.df, v.row);
  }

vsmall.df;
```

```
##           x.bar         s.var        s.sd
## 1  1.616831e-01  1.133672e+00 1.06474e+00
## 2  1.616831e-04  1.133672e-06 1.06474e-03
## 3  1.616831e-07  1.133672e-12 1.06474e-06
## 4  1.616831e-10  1.133672e-18 1.06474e-09
## 5  1.616831e-13  1.133672e-24 1.06474e-12
## 6  1.616831e-16  1.133672e-30 1.06474e-15
## 7  1.616831e-19  1.133672e-36 1.06474e-18
## 8  1.616831e-22  1.133672e-42 1.06474e-21
## 9  1.616831e-25  1.133672e-48 1.06474e-24
## 10 1.616831e-28  1.133672e-54 1.06474e-27
## 11 1.616831e-31  1.133672e-60 1.06474e-30
## 12 1.616831e-34  1.133672e-66 1.06474e-33
## 13 1.616831e-37  1.133672e-72 1.06474e-36
## 14 1.616831e-40  1.133672e-78 1.06474e-39
## 15 1.616831e-43  1.133672e-84 1.06474e-42
## 16 1.616831e-46  1.133672e-90 1.06474e-45
## 17 1.616831e-49  1.133672e-96 1.06474e-48
## 18 1.616831e-52 1.133672e-102 1.06474e-51
## 19 1.616831e-55 1.133672e-108 1.06474e-54
## 20 1.616831e-58 1.133672e-114 1.06474e-57
## 21 1.616831e-61 1.133672e-120 1.06474e-60
```

```
# if x is really big
vlarge.df = as.data.frame( t(unlist(v.norm)) );

x.large = x.norm;
for(i in 1:20)
  {
```

```
  # every loop make it 1000 times larger
  # notice I am looping over "i" but not using it.
  x.large = standardizeToFactor(x.large, 1000);
  v.large = doSampleVariance(x.large, "naive");
    v.row = t(unlist(v.large));
  vlarge.df = rbind(vlarge.df, v.row);
  }

vlarge.df;
```

```
##           x.bar         s.var        s.sd
## 1  1.616831e-01  1.133672e+00 1.06474e+00
## 2  1.616831e+02  1.133672e+06 1.06474e+03
## 3  1.616831e+05  1.133672e+12 1.06474e+06
## 4  1.616831e+08  1.133672e+18 1.06474e+09
## 5  1.616831e+11  1.133672e+24 1.06474e+12
## 6  1.616831e+14  1.133672e+30 1.06474e+15
## 7  1.616831e+17  1.133672e+36 1.06474e+18
## 8  1.616831e+20  1.133672e+42 1.06474e+21
## 9  1.616831e+23  1.133672e+48 1.06474e+24
## 10 1.616831e+26  1.133672e+54 1.06474e+27
## 11 1.616831e+29  1.133672e+60 1.06474e+30
## 12 1.616831e+32  1.133672e+66 1.06474e+33
## 13 1.616831e+35  1.133672e+72 1.06474e+36
## 14 1.616831e+38  1.133672e+78 1.06474e+39
## 15 1.616831e+41  1.133672e+84 1.06474e+42
## 16 1.616831e+44  1.133672e+90 1.06474e+45
## 17 1.616831e+47  1.133672e+96 1.06474e+48
## 18 1.616831e+50 1.133672e+102 1.06474e+51
## 19 1.616831e+53 1.133672e+108 1.06474e+54
## 20 1.616831e+56 1.133672e+114 1.06474e+57
## 21 1.616831e+59 1.133672e+120 1.06474e+60
```

```
## from these two experiments it looks okay!

## CS purists say it will fail eventually
## maybe I have to use a smaller n to demo failure?

## examine the function , the failure point is:    sum2 = sum2 + x[i]*x[i];
## [+5 Easter to first x-vec of 100 numbers that causes "naive" to fail!]
## by fail, I mean the "two-pass" approach and built ?sd or ?var function
## shows something entirely different ...
```

5.1.2 Traditional Two Pass

```
v2.norm = doSampleVariance(x.norm, "two-pass");
v2b.norm = doSampleVariance(x.norm);  # default value is "two-pass" in the function
v2c.norm = doSampleVariance(x.norm, "garblideljd=-gook"); # if logic defaults to "two-pass"

unlist(v2.norm);
```

```
##     x.bar     s.var      s.sd
## 0.1616831 1.1336720 1.0647403
```

```
unlist(v2b.norm);
```

```
##     x.bar     s.var      s.sd
## 0.1616831 1.1336720 1.0647403
```

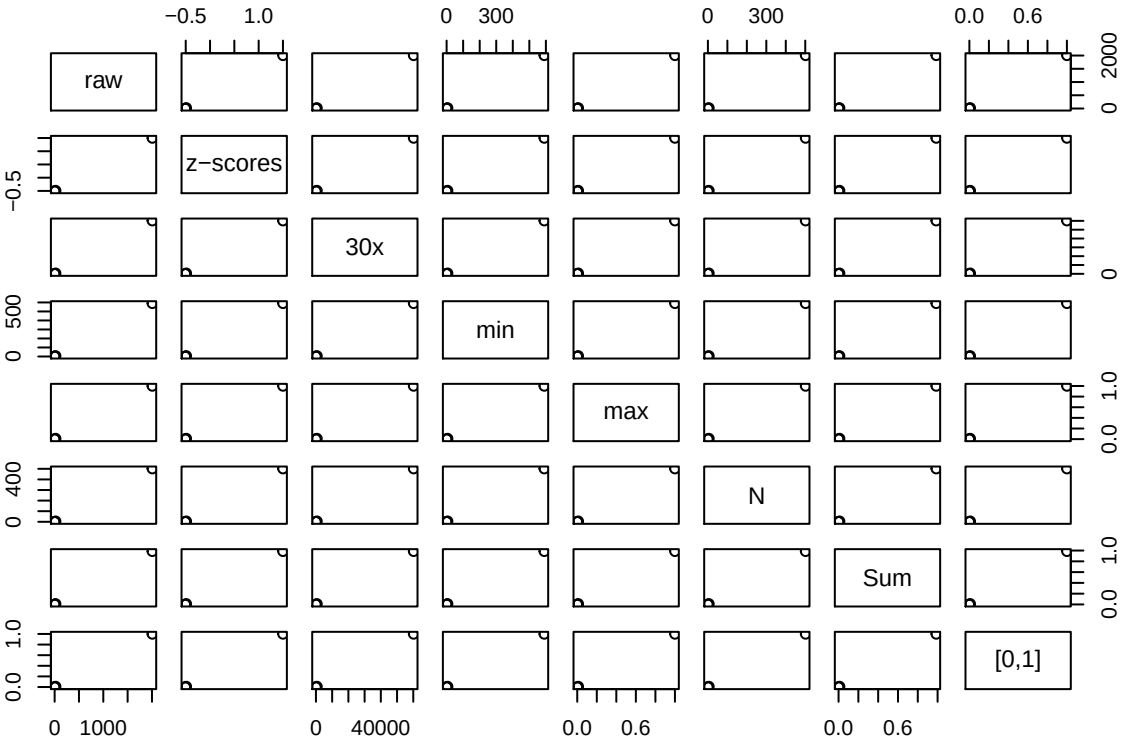
```
unlist(v2c.norm);
```

```
##     x.bar     s.var      s.sd
## 0.1616831 1.1336720 1.0647403
```

5.2 Z-scores

```
# the built in function ?scale you should find useful.
# a z-score is taken on a vector of data requires x.bar and s.hat
# generally, we assume x.bar and s.hat comes from the vector of data.
# It doesn't have to.

library(digest);
md5_monte = digest("monte.shaffer@gmail.com", algo="md5");  # no workee???
md5_monte = "b62c73cdaf59e0a13de495b84030734e";

# jQuery [b62c73cdaf59e0a13de495b84030734e]     https://www.jqueryscript.net/demo/MD5-Hash-String/
# Javascript [b62c73cdaf59e0a13de495b84030734e] http://md5.mshaffer.com/
# PHP     [b62c73cdaf59e0a13de495b84030734e]    https://onlinegdb.com/rJUGCTkrw
# Python enthusiasts:  I recommend WingIDE  ... https://wingware.com/
# [+5 investigate the issue... write a Python function that passes in a string
#   and returns a md5 string, write an onlinegdb.com example for C,
#   and write an onlinegdb.com example for C++ ... summarize your findings.]

row = my_data_clean[my_data_clean$md5_email == md5_monte, ];
vec.start = getIndexOfDataFrameColumns(row,"V01");  # 5
vec.end = getIndexOfDataFrameColumns(row,"V60"); # 64

vec = as.numeric( row[vec.start:vec.end] );  # vector functions require "vector form"
# recall the concept of a vector basis? (e.g., basis of vector space)
# linear combinations of this basis?


vdf = as.data.frame( t(vec) );  # dim(vdf) tells me to transpose it.
  myRows = c("raw");

z.vec = calculateZscores(vec);
  vdf = rbind(vdf,z.vec);     myRows=c(myRows,"z-scores");

z.vec30 = standardizeToFactor(vec, 30);
  vdf = rbind(vdf,z.vec30);  myRows=c(myRows,"30x");

z.vecmin = standardizeToMin(vec);  # like z-scores, should rewrite to allow it be passed in, by default
  vdf = rbind(vdf,z.vecmin);  myRows=c(myRows,"min");

z.vecmax = standardizeToMax(vec);  # like z-scores, should rewrite to allow it be passed in, by default
  vdf = rbind(vdf,z.vecmax);  myRows=c(myRows,"max");
```

```
z.vecN = standardizeToN(vec);  # like z-scores, should rewrite to allow it be passed in, by default it
  vdf = rbind(vdf,z.vecN);  myRows=c(myRows,"N");

z.vecSum = standardizeToSum (vec);  # like z-scores, should rewrite to allow it be passed in, by defaul
  vdf = rbind(vdf,z.vecSum);  myRows=c(myRows,"Sum");

z.vecBound = standardizeFromOneRangeToAnother(vec, c(0,1) );  # like z-scores, should rewrite to allow
  vdf = rbind(vdf,z.vecBound);  myRows=c(myRows,"[0,1]");

rownames(vdf) = myRows;

tvdf = as.data.frame( t(vdf) ); # why transpose it?

graphics::plot( tvdf );
```




```
# linear transformations are "linear" ... should not be surprising
# how does perfect linearity relate to "correlation"?

# Multiplying by a negative number (not shown) is also a vector-basis manipulation.
# As is rotating by an angle.
# What is an example of a nonlinear combination?
# Hint look at your will/denzel problem.
# plot(will$movies.50[,c(1,6,7:10)]); ... one relationship is strong, nonlinear
```

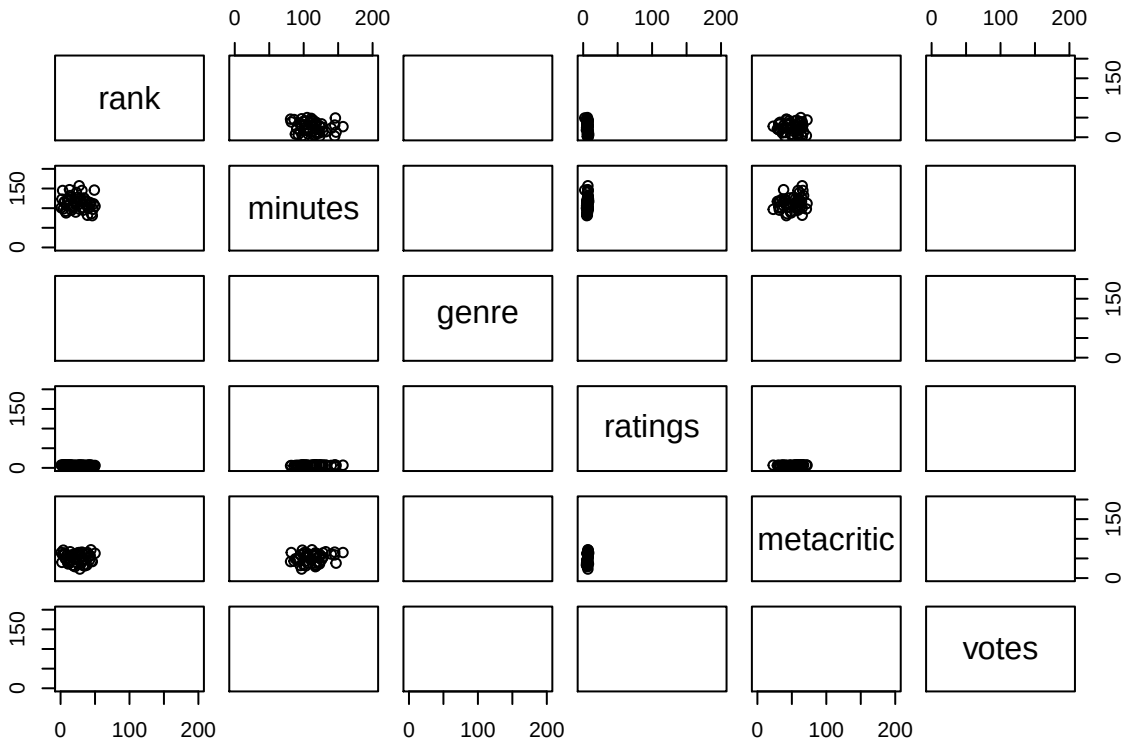


Figure 2: Will Smith scatterplot: IMDB(2020)

6 Will vs. Denzel

Compare Will Smith and Denzel Washington. [See 03_n greater 1-v2.txt for the necessary functions and will-vs-denzel.txt for some sample code and in DROPBOX: __student_access__\unit_01_exploratory_data_analysis\week_02\
You will have to create a new variable $millions.2000 that converts each movie's $millions based on the $year of the movie, so all dollars are in the same time frame. You will need inflation data from about 1980-2020 to make this work.

```
source_url(paste0("https://raw.githubusercontent.com/MichaelaB1/WSU_STATS419_FALL2020/master/functions/f
```

6.1 Will Smith

```
nmid = "nm0000226";
    will = grabFilmsForPerson(nmid);
    plot(will$movies.50[,c(1,6,7:10)], ylim=c(0,200), xlim=c(0,200));
```

```
    boxplot(will$movies.50$millions);
```

```
        widx =  which.max(will$movies.50$millions);
    will$movies.50[widx,];
```

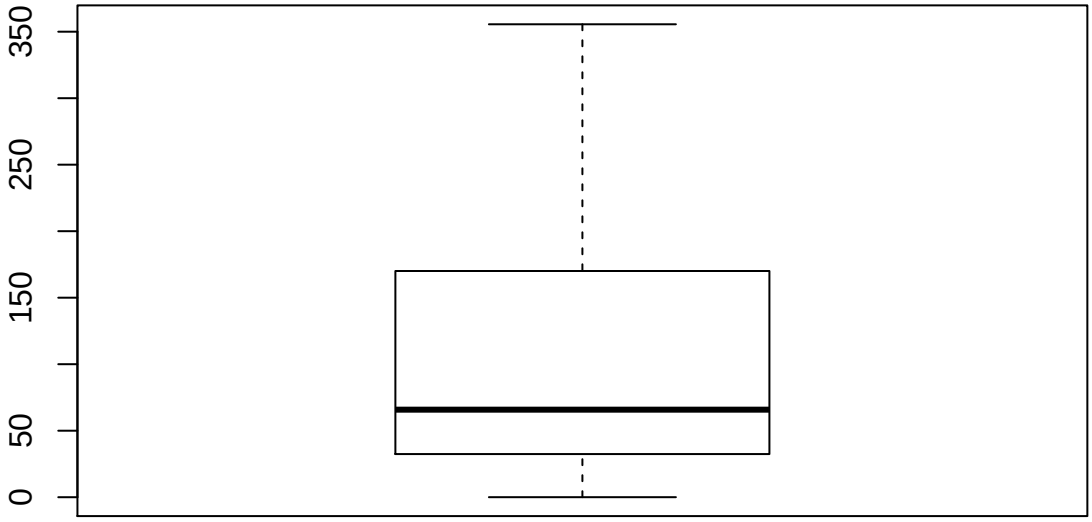


Figure 3: Will Smith boxplot raw millions: IMDB(2020)

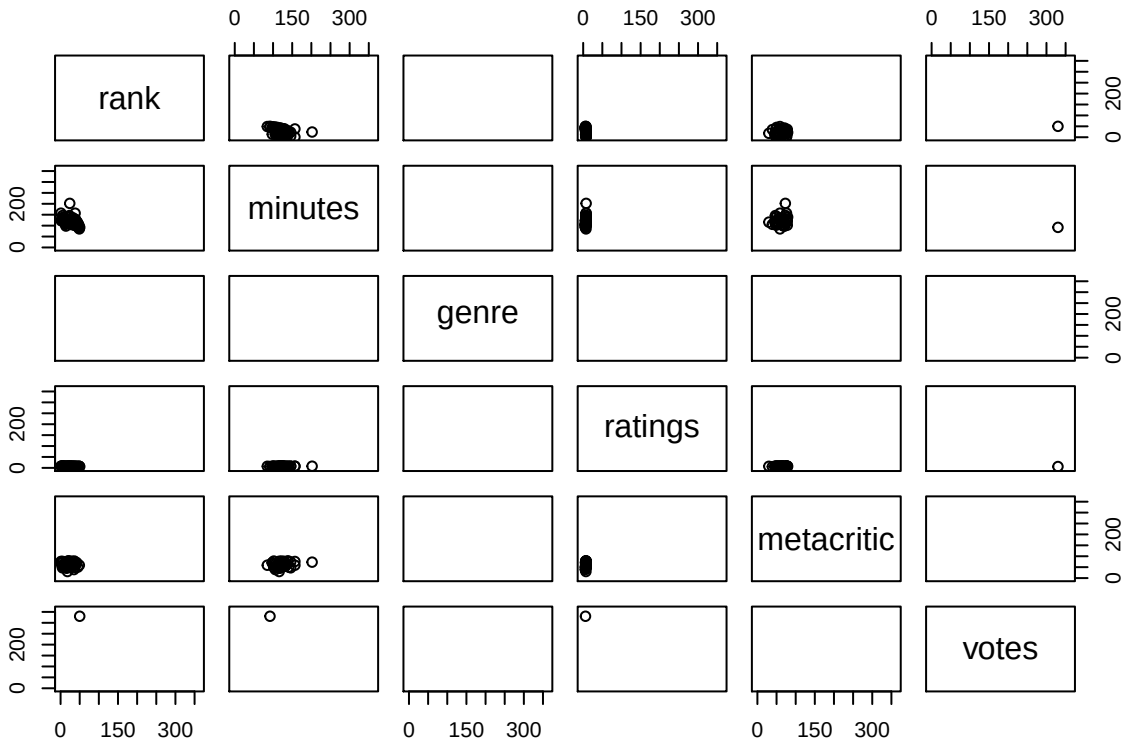


Figure 4: Denzel Washington scatterplot: IMDB(2020)

```
##    rank   title      ttid year rated minutes                      genre ratings
## 15   15 Aladdin tt6139732 2019     6     128 Adventure, Family, Fantasy       7
##    metacritic  votes millions
## 15         53 216958   355.56
```

```
        summary(will$movies.50$year);
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1993    2001    2006    2007    2014    2020
```

6.2 Denzel Washington

```
nmid = "nm0000243";
    denzel = grabFilmsForPerson(nmid);
    plot(denzel$movies.50[,c(1,6,7:10)], ylim=c(0,360), xlim=c(0,360));
```

```
    boxplot(denzel$movies.50$millions);
```

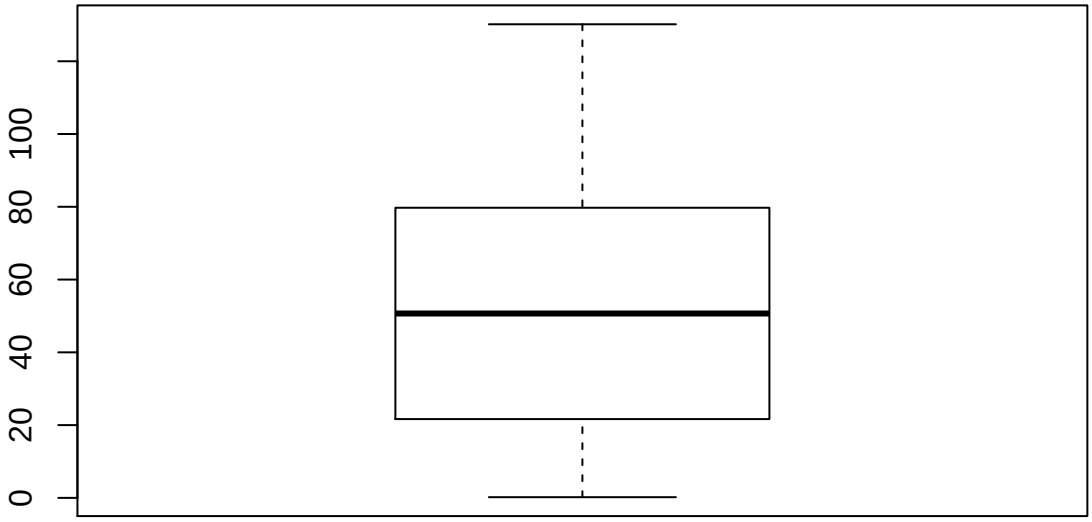


Figure 5: Denzel Washington boxplot raw millions: IMDB(2020)

```
        didx =  which.max(denzel$movies.50$millions);
    denzel$movies.50[didx,];
```

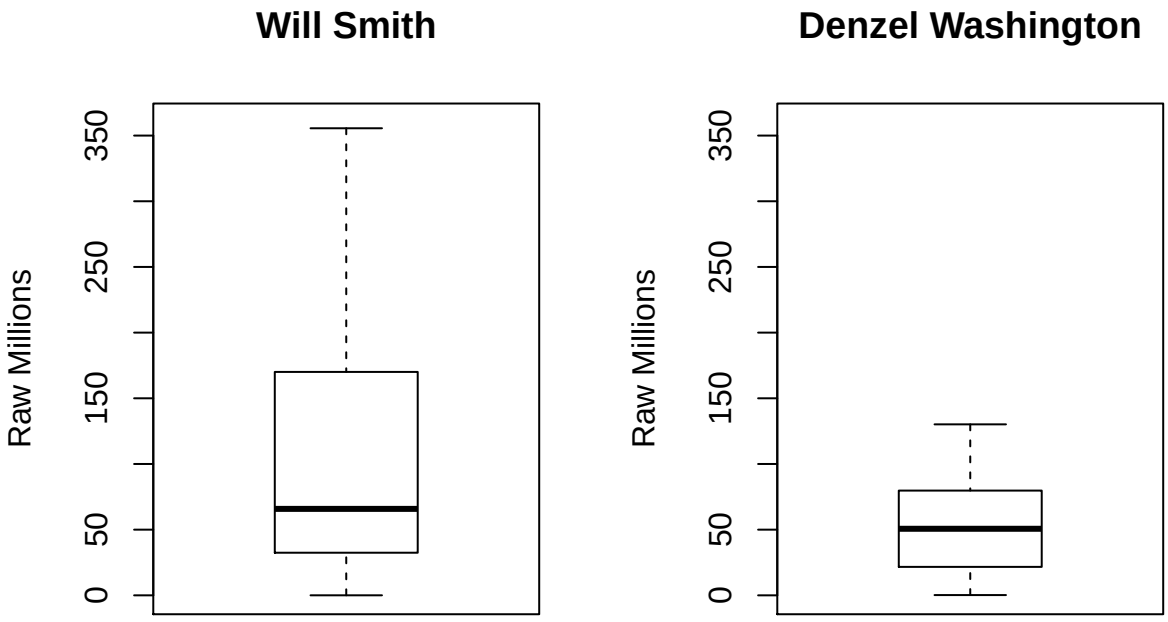
```
##   rank             title      ttid year rated minutes                    genre
## 1    1 American Gangster tt0765429 2007    16     157 Biography, Crime, Drama
##   ratings metacritic  votes millions
## 1     7.8         76 384303   130.16
```

```
        summary(denzel$movies.50$year);
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1981    1993    1999    2000    2008    2018
```

6.3 Boxplot of Top-50 movies using Raw Dollars

```
par(mfrow=c(1,2));
    boxplot(will$movies.50$millions, main=will$name, ylim=c(0,360), ylab="Raw Millions" );
    boxplot(denzel$movies.50$millions, main=denzel$name, ylim=c(0,360), ylab="Raw Millions" );
```




```
    par(mfrow=c(1,1));
```

6.4 Film Count Of Will Smith

```
new.will = will$movies.50;
new.will$nmid = will$nmid;
new.will$name = will$name;
new.will$countfilms = will$countfilms$totalcount;
new.will = new.will[, c(12,13,14, 1:11)];
```

6.5 Film Count Of Denzel Washington

```
new.denzel = denzel$movies.50;
new.denzel$nmid = denzel$nmid;
new.denzel$name = denzel$name;
new.denzel$countfilms = denzel$countfilms$totalcount;
new.denzel = new.denzel[, c(12,13,14, 1:11)];
```

6.6 Combined Dataframe of Will Smith and Denzel Washington

```
df.will.denzel = rbind(new.will, new.denzel);
```

7 Side by side comparison

Build side-by-side box plots on several of the variables (including #6) to compare the two movie stars. After each box plot, write 2+ sentence describing what you are seeing, and what conclusions you can logically make. You will need to review what the box plot is showing with the box portion, the divider in the box, and the whiskers.

7.1 Adjusted Dollars (2000)

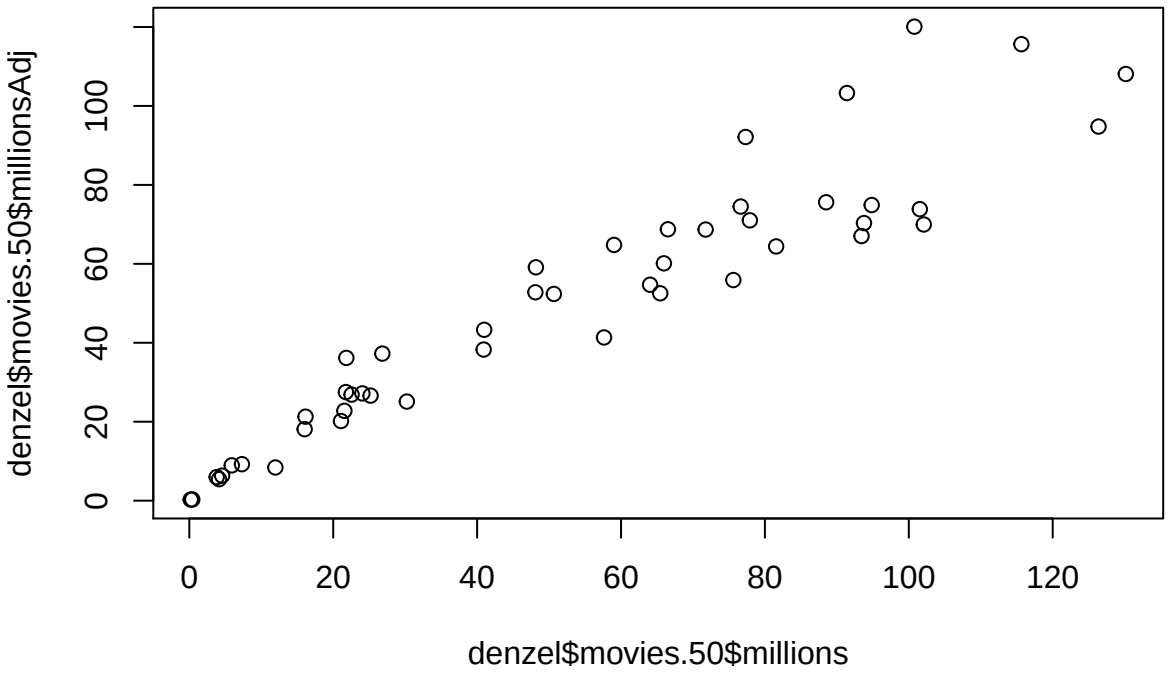
```
# https://raw.githubusercontent.com/MonteShaffer/humanVerseWSU/master/humanVerseWSU/R/functions-inflati

humanVerseWSU::loadInflationData();
str(denzel$movies.50);
```

```
## 'data.frame':    50 obs. of  11 variables:
##  $ rank      : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ title     : chr  "American Gangster" "Training Day" "Inside Man" "The Equalizer" ...
##  $ ttid      : chr  "tt0765429" "tt0139654" "tt0454848" "tt0455944" ...
##  $ year      : num  2007 2001 2006 2014 2004 ...
##  $ rated     : chr  "16" "16" "12" "16" ...
##  $ minutes   : num  157 122 129 132 146 138 126 118 125 115 ...
##  $ genre     : chr  "Biography, Crime, Drama" "Crime, Drama, Thriller" "Crime, Drama, Mystery" "Acti
##  $ ratings   : num  7.8 7.7 7.6 7.2 7.7 7.3 7 6.9 7.7 6.7 ...
##  $ metacritic: num  76 69 76 57 47 76 59 53 66 52 ...
##  $ votes     : num  384303 382426 332305 326504 324434 ...
##  $ millions  : num  130.2 76.6 88.5 101.5 77.9 ...
```

```
denzel$movies.50 = standardizeDollarsInDataFrame(denzel$movies.50, 2000, "millions", "year", "millionsA
str(denzel$movies.50);
```

```
## 'data.frame':    50 obs. of  12 variables:
##  $ rank       : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ title      : chr  "American Gangster" "Training Day" "Inside Man" "The Equalizer" ...
##  $ ttid       : chr  "tt0765429" "tt0139654" "tt0454848" "tt0455944" ...
##  $ year       : num  2007 2001 2006 2014 2004 ...
##  $ rated      : chr  "16" "16" "12" "16" ...
##  $ minutes    : num  157 122 129 132 146 138 126 118 125 115 ...
##  $ genre      : chr  "Biography, Crime, Drama" "Crime, Drama, Thriller" "Crime, Drama, Mystery" "Acti
##  $ ratings    : num  7.8 7.7 7.6 7.2 7.7 7.3 7 6.9 7.7 6.7 ...
##  $ metacritic : num  76 69 76 57 47 76 59 53 66 52 ...
##  $ votes      : num  384303 382426 332305 326504 324434 ...
##  $ millions   : num  130.2 76.6 88.5 101.5 77.9 ...
##  $ millionsAdj: num  108.1 74.5 75.6 73.9 71 ...
```

```
plot(denzel$movies.50$millions,denzel$movies.50$millionsAdj);
```




```
## you should repeat for will (Will Smith) ...
```

7.2 Total Votes (Divide by 1,000,000)

7.3 Average Ratings

7.4 Year? Minutes?

7.5 Metacritic (NA values)