IMGT680 Homework 1

Brandon Hufstetler, Garrett Alarcon, Jinan Andrews, Anson Cheng, Nick Forrest, and Nestor Herandez

1 July 2019

Chapter 1

GETTING STARTED WITH R

Comments, indents, and semicolons

```
# Anything prefaced by a pound sign (#) is a comment.
# Comments are not executed by R. Instead, they explain what the code is doing.
# Indented code (that is not a comment) will run in R as if it was on one line
# Code separated by semicolons will run as if the code was on separate lines,
# with the semicolon marking the line break
```

Open a dataset and display the data

Commentary: The commands below read and output the "Cars" data set as well as specific portions of the data set for the user to see. The setwd() command is used to make ~/IMGT680 the working directory.

```
filepath <- ""
setwd("~/IMGT680")
cars <- read.csv(file = "cars.txt", stringsAsFactors = FALSE)
cars # To display the whole dataset, type the dataset name
```

```
head(cars) # Display the first few records of a dataset
```

	mpg	cylinders	cubicinches	hp	weightlbs	time.to.60	year	brand
	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>	<chr>
1	14.0	8	350	165	4209	12	1972	US.
2	31.9	4	89	71	1925	14	1980	Europe.
3	17.0	8	302	140	3449	11	1971	US.
4	15.0	8	400	150	3761	10	1971	US.
5	30.5	4	98	63	2051	17	1978	US.
6	23.0	8	350	125	3900	17	1980	US.

6 rows

```
  names(cars) # Display variable names of a data frame, one kind of data in R
```

```
## [1] "mpg"         "cylinders"   "cubicinches" "hp"          "weightlbs"
## [6] "time.to.60"  "year"        "brand"
```

```
  cars$weight # Look at only the weight variable within data frame cars
```

```
##   [1] 4209 1925 3449 3761 2051 3900 4363 4312 3530 2050 2245 2188 4141 3664
##  [15] 3381 4360 2020 3433 2278 2430 2019 2600 3012 4054 1968 1795 1773 4657
##  [29] 3574 2380 2130 3278 2506 2648 1985 3415 1835 2720 3955 3265 3897 4638
##  [43] 3645 3520 3086 2635 3755 2395 1940 3060 4464 3190 3609 2158 4380 4278
##  [57] 2930 2075 1937 3821 2945 2379 2910 2110 4237 3525 1950 1965 1825 3880
##  [71] 3102 2640 2288 2545 2219 3015 3085 2515 2265 2350 4325 4952 3425 2694
##  [85] 2220 1613 2774 3465 2125 2720 1975 2300 4100 3329 2255 3907 4499 3139
##  [99] 3830 3781 4997 4906 2126 2200 2265 2635 2335 2065 2671 3504 2279 2933
## [113] 4335 1795 2130 3785 2740 3039 2045 4341 2202 1963 4668 2745 1834 2565
## [127] 4654 2230 2582 1800 2300 2678 4951 3672 4440 4190 1955 3169 2945 2979
## [141] 2700 2171 2665 1760 3605 2957 3360 2735 3693 3205 2542 2560 4425 3158
## [155] 2464 3302 2124 2135 2634 3410 3003 4080 2984 2904 2254 3365 3439 2585
## [169] 2807 2125 2290 2830 1867 2575 2226 2189 4385 3672 3570 2045 2990 2145
## [183] 4215 3436 2395 3070 3459 3940 4129 4735 2620 2670 4422 4082 4077 2144
## [197] 2490 1985 1937 1945 2246 2639 3420 3630 2401 3850 2110 3021 4098 2900
## [211] 2815 3445 1755 2074 3233 2615 1975 4215 2595 4295 3193 2795 4457 3820
## [225] 4042 2003 4615 1649 2587 2790 3563 2625 2592 4498 3150 2130 4096 2965
## [239] 1875 2220 4055 3121 4220 2672 4354 3399 3735 2085 2155 2525 2120 2660
## [253] 2950 3988 2725 2372 3840 1800 2835 3288 3353
```

Matrices

Commentary: The commands below creates a matrix based on the specified number of columns, rows, and values.

```
  # Create a matrix with three rows, two columns, and every value equal to 0.0
  mat <- matrix(0.0, nrow = 3, ncol = 2); mat
```

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

```
  colnames(mat) <- c("Var 1", "Var 2") # Give a matrix variable names
  colnames(mat) # Display variable names of a matrix
```

```
## [1] "Var 1" "Var 2"
```

Subset data and declare new variables

Commentary: The commands below creates subsets within data that has been read into the system and creates new variables.

```
  cars.rsub <- cars[1:50,] # Subset the data by rows
  cars.csub <- cars[,1:3] # Subset by columns
  cars.rcsub <- cars[c(1,3,5), c(2,4)] # Subset by specific rows and columns
  cars.vsub <- cars[which(cars$mpg> 30),] # Subset by a logical condition
# To declare new variables, type the variable name, a left-arrow, then the value of the
 variable
  firstletter <- 'a'
  weight <- cars$weight
```

Display more than one figure at a time

Commentary: The commands below plots figures based on the number of specified rows and figures within each row.

```
  par(mfrow=c(1,1)) # plots one figure; the default setting
  par(mfrow=c(2,3)) # plots six figures: three in the top row, three in the bottom row
  # Plots will fill the plot space row by row
```

Download and install an R Package

Commentary: The command below installs/retrieves the ggplot2 package to enable the user to plot data in a graph.

```
  #install.packages("ggplot2")
  library(ggplot2)
```

Chapter 2

READ IN THE CARS AND CARS2 DATASETS

```
  setwd("~/IMGT680") # sets the working directory
  cars <- read.csv("cars.txt", stringsAsFactors = FALSE) # reads text file Cars.txt
  cars2 <- read.csv("cars2.txt", stringsAsFactors = FALSE) # reads text file Cars.txt
```

MISSING DATA

```
  # Look at four variables from cars
  cars.4var <- cars[, c(1, 3, 4, 8)] # renames data from columns 1,3,4,8 to cars.4var
  head(cars.4var) # displays first few records of each column
```

	mpg	cubicinches	hp	brand
	<dbl>	<int>	<int>	<chr>
1	14.0	350	165	US.

	mpg <dbl>	cubicinches <int>	hp <int>	brand <chr>
2	31.9	89	71	Europe.
3	17.0	302	140	US.
4	15.0	400	150	US.
5	30.5	98	63	US.
6	23.0	350	125	US.

6 rows

```
  # Make certain entries missing
  cars.4var[2,2] <- cars.4var[4,4] <- NA # sets data from row 2, column 2 and row 4, col
umn 4 to "NA".
  head(cars.4var)
```

	mpg <dbl>	cubicinches <int>	hp <int>	brand <chr>
1	14.0	350	165	US.
2	31.9	*NA*	71	Europe.
3	17.0	302	140	US.
4	15.0	400	150	*NA*
5	30.5	98	63	US.
6	23.0	350	125	US.

6 rows

```
  # Replace missing values with constants
  cars.4var[2,2] <- 0 # sets data from row 2, column 2 to zero
  cars.4var[4,4] <- "Missing" # sets data from row 4, column 4 to "Missing"
  head(cars.4var)
```

	mpg <dbl>	cubicinches <dbl>	hp <int>	brand <chr>
1	14.0	350	165	US.
2	31.9	0	71	Europe.
3	17.0	302	140	US.
4	15.0	400	150	Missing
5	30.5	98	63	US.
6	23.0	350	125	US.

6 rows

```
  # Replace values with mean and mode
  cars.4var[2,2] <- mean(na.omit(cars.4var$cubicinches)) # sets data from row 2, column
 2 to the mean for CubicInches (201.5346)
  our_table <- table(cars.4var$brand) # sets data from "brand" column to new variable, o
ur_table
  our_mode <- names(our_table)[our_table == max(our_table)] # sets the mode of our_table
to new variable, our_mode
  cars.4var[4,4] <- our_mode # sets row 4, column 4 to the the mode (US)
  head(cars.4var)
```

	mpg <dbl>	cubicinches <dbl>	hp <int>	brand <chr>
1	14.0	350.0000	165	US.
2	31.9	200.7625	71	Europe.
3	17.0	302.0000	140	US.
4	15.0	400.0000	150	US.
5	30.5	98.0000	63	US.
6	23.0	350.0000	125	US.

6 rows

Commentary: The commands below fill in the missing values with random numbers found within that variable's other entries.

```
  # Generate random observations
  obs_brand <- sample(na.omit(cars.4var$brand), 1) # generates 1 random number for brand
columnand sets to new variable, obs_brand
  obs_cubicinches <- sample(na.omit(cars.4var$cubicinches), 1) # generates 1 random numb
er for cubicinches column and sets to new variable obs_cubicinches
  cars.4var[2,2] <- obs_cubicinches # set row 2, column 2 to obs_cubicinches
  cars.4var[4,4] <- obs_brand # set row 4, column 4 to obs_brand
  head(cars.4var)
```

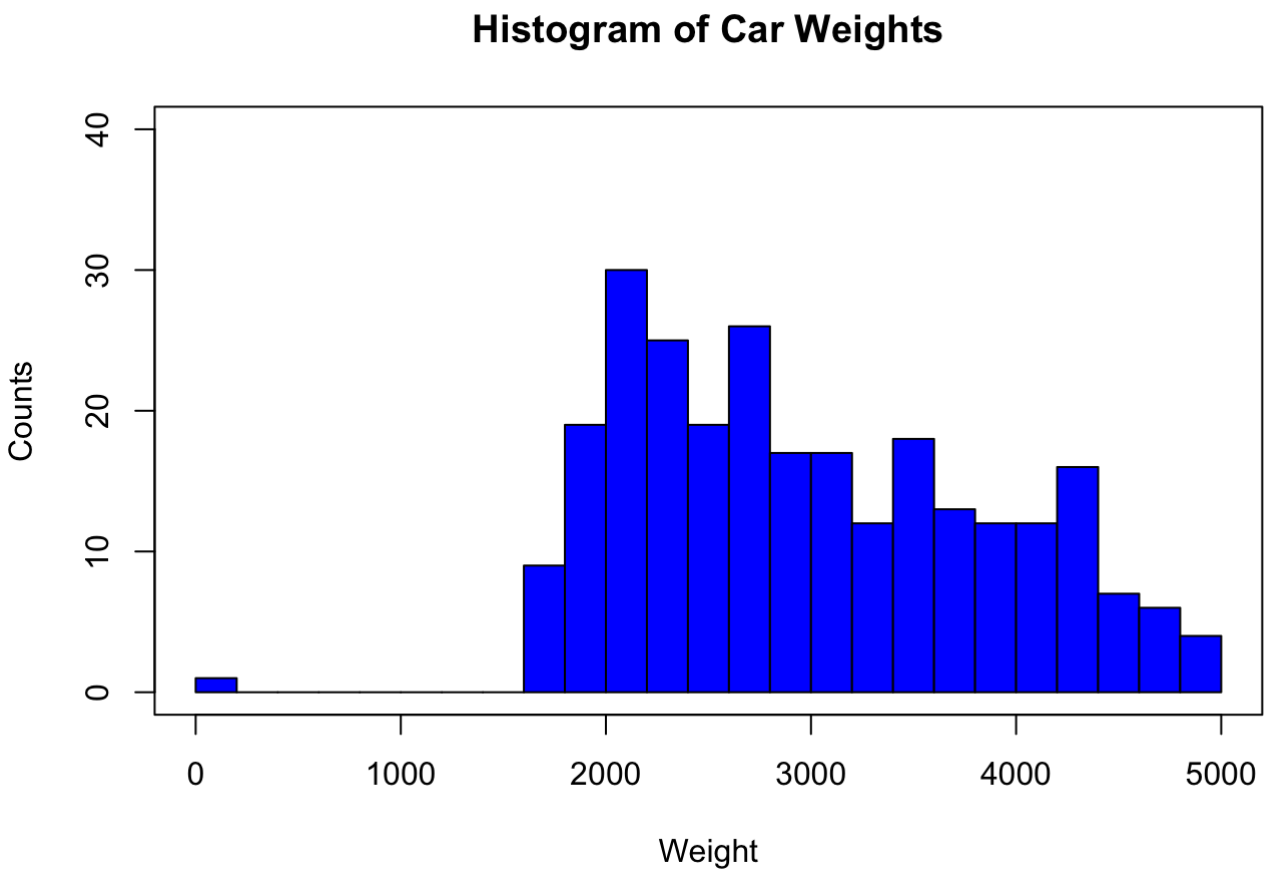
	mpg <dbl>	cubicinches <dbl>	hp <int>	brand <chr>
1	14.0	350	165	US.
2	31.9	120	71	Europe.
3	17.0	302	140	US.
4	15.0	400	150	Japan.
5	30.5	98	63	US.
6	23.0	350	125	US.

6 rows

CREATE A HISTOGRAM

Distribution and outliers are easy to spot in a histogram

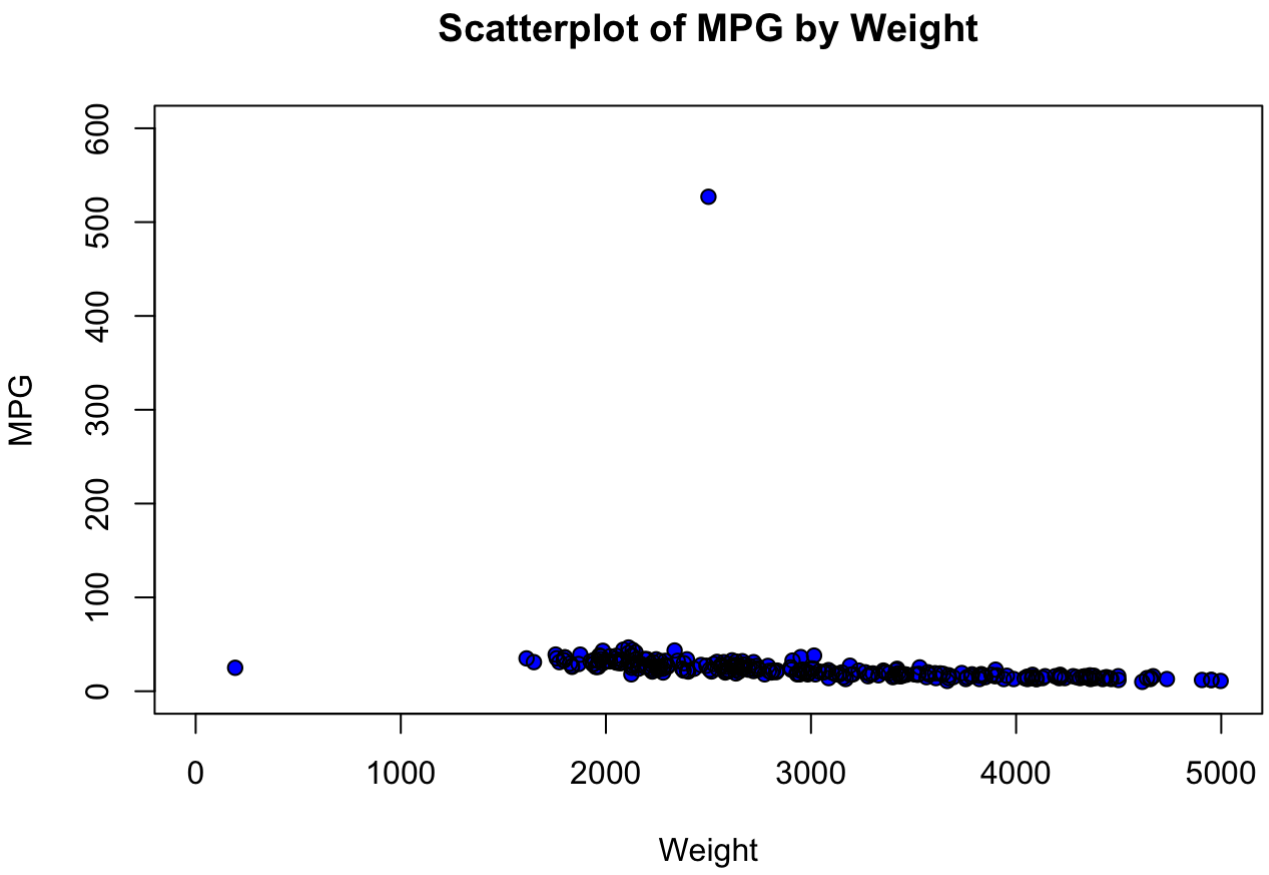
```
# Set up the plot area
par(mfrow = c(1,1)) # creates a multi-paneled plot with 1 row by 1 column

# Create the histogram bars
hist(cars2$weight, # extracts data from weight column in cars2.txt data
     breaks = 30, # sets bin width to 30
     xlim = c(0, 5000), # sets x-axis from 0 to 50000
     col = "blue", # outputs blue colored columns
     border = "black", # outputs a black border around each column
     ylim = c(0, 40), # sets y-axis from 0 to 40
     xlab = "Weight", # labels x-axis
     ylab = "Counts", # labels y-axis
     main = "Histogram of Car Weights") # creates graph title


# Make a box around the plot
box(which = "plot", lty = "solid", col = "black") # creates a black box border around
the plot
```



CREATE A SCATTERPLOT

Outliers are easily spotted

```
  plot(cars2$weight, cars2$mpg, # extracts data from weight and mpg columns in cars2.txt
data
       xlim = c(0, 5000), ylim = c(0, 600), # sets x-axis from 0 to 5000
       xlab = "Weight", ylab = "MPG", # labels x-axis
       main = "Scatterplot of MPG by Weight", # creates graph title
       type = "p", pch = 16, col = "blue") # creates circular points filled in with the
 color blue
  #Add open black circles
  points(cars2$weight, cars2$mpg,
         type = "p", col = "black") # creates circular points filled in with the color b
lack
```



DESCRIPTIVE STATISTICS

```
  mean(cars$weight) # Mean
```

```
## [1] 3005.49
```

```
  median(cars$weight) # Median
```

```
## [1] 2835
```

```
  length(cars$weight) # Number of observations
```

```
## [1] 261
```

```
  sd(cars$weight) # Standard deviation
```

```
## [1] 852.6456
```

```
  summary(cars$weight) # Min, Q1, Median, Mean, Q3, Max
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1613    2246    2835    3005    3664    4997
```

TRANSFORMATIONS

Outputs suppressed to save space and because they are plotted below.

Min–max normalization works by seeing how much greater the field value is than the minimum value min(X), and scaling this difference by the range.

```
  # Min-max normalization
  summary(cars$weight)
  mi <- min(cars$weight) # minimum number
  ma <- max(cars$weight) # maximum number
  minmax.weight <- (cars$weight - mi)/(ma - mi) # min-max normalization
  minmax.weight
```

Z-score standardization, which is very widespread in the world of statistical analysis, works by taking the difference between the field value and the field mean value, and scaling this difference by the SD of the field values.

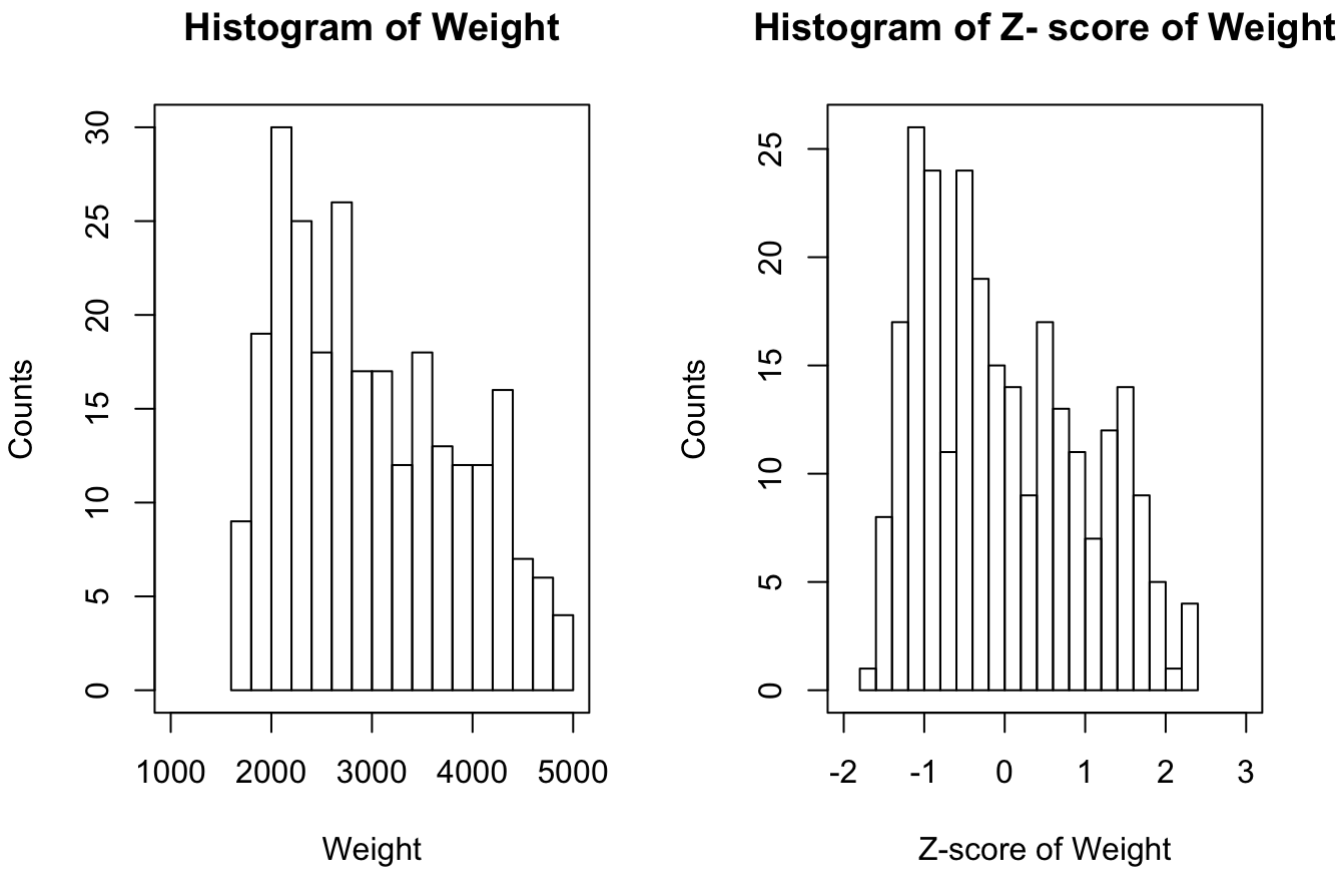
```
  # Z-score standarization
  m <- mean(cars$weight) # mean
  s <- sd(cars$weight) # standard deviation
  z.weight <- (cars$weight - m)/s # standard normal deviation
  z.weight
  length(cars$weight) # length of vector
```

Decimal scaling ensures that every normalized value lies between –1 and 1.

```
  # Decimal scaling
  max(abs(cars$weight)) # 4 digits
  d.weight <- cars$weight/(10^4) # scaled by 10^4
  d.weight
```

SIDE-BY-SIDE HISTOGRAMS

```
par(mfrow = c(1,2)) # creates a multi-paneled plot with 1 row by 2 columns
# Create two histograms
# Creates a histogram of weight
hist(cars$weight, breaks = 20, xlim = c(1000, 5000),
     main = "Histogram of Weight", xlab = "Weight", ylab = "Counts")
box(which = "plot", lty = "solid", col = "black")
# Creates a histogram of weight using z-score
hist(z.weight, breaks = 20, xlim = c(-2, 3),
     main = "Histogram of Z- score of Weight",
     xlab = "Z-score of Weight", ylab = "Counts")
box(which = "plot", lty = "solid", col = "black")
```



SKEWNESS

Skewness describes the distance between the median and the mean. Positive skewness means the mean is greater than the median and negative skewness means the mean is less than the median. Zero skewness means the mean is equal to the mode (when the data is unimodal).

```
  (3*(mean(cars$weight) - median(cars$weight)))/sd(cars$weight) # skewness formula for c
ars data
```

```
## [1] 0.5998638
```

```
  (3*(mean(z.weight) - median(z.weight)))/sd(z.weight) # skewness formula for z-score of
cars data
```

```
## [1] 0.5998638
```

TRANSFORMATIONS FOR NORMALITY

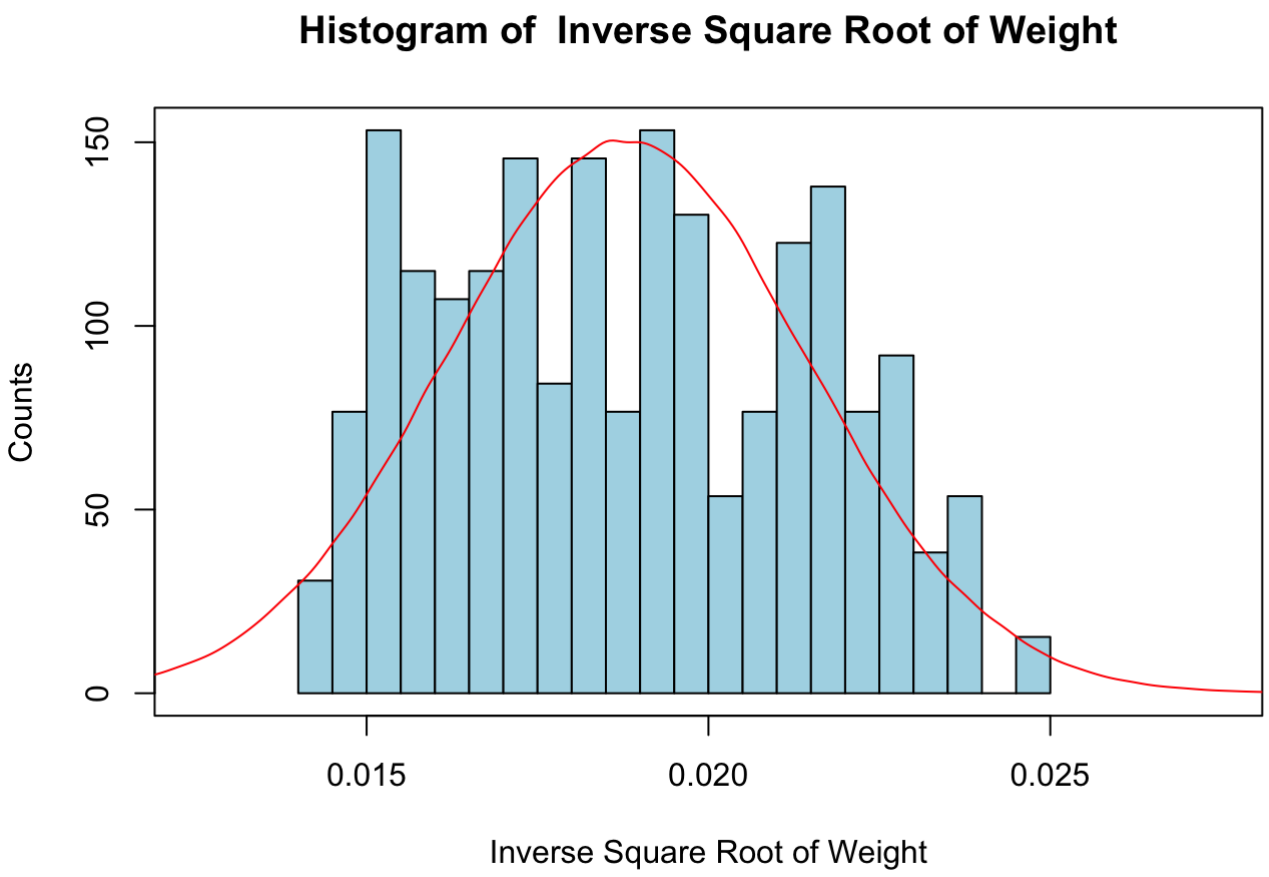
These transformations attempt to reduce skewness and make the data “more normally distributed”.

```
  sqrt.weight <- sqrt(cars$weight) # Square root
  sqrt.weight_skew <- (3*(mean(sqrt.weight) - median(sqrt.weight))) / sd(sqrt.weight) #
 skewness formula for square root of weight data
  ln.weight <- log(cars$weight) # Natural log
  ln.weight_skew <- (3*(mean(ln.weight) - median(ln.weight))) / sd(ln.weight) # skewness
formula for natural log of weight data
  invsqrt.weight <- 1 / sqrt(cars$weight) # Inverse square root
  invsqrt.weight_skew <- (3*(mean(invsqrt.weight) - median(invsqrt.weight))) /sd(invsqr
t.weight) # skewness formula for natural log of weight data
```

HISTOGRAM WITH NORMAL DISTRIBUTION OVERLAY

The inverse square transformation has eliminated skewness, but it is still not normal. This is clearly seen in the histogram with a normal overlay.

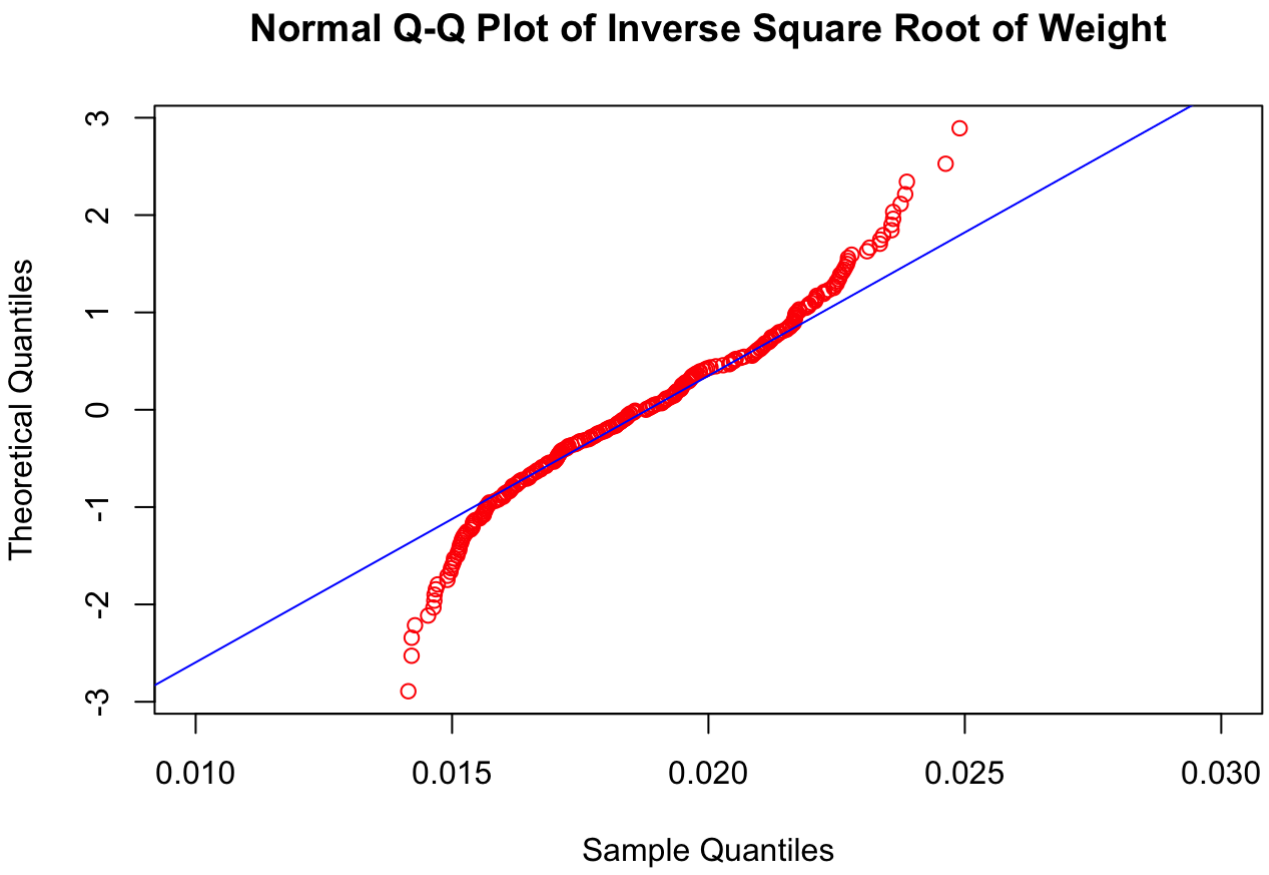
```
  par(mfrow=c(1,1)) # creates a multi-paneled plot with 1 row by 1 column
  x <- rnorm(1000000, mean = mean (invsqrt.weight), sd = sd(invsqrt.weight)) # generates
normal random numbers using mean and standard deviation
  hist(invsqrt.weight,
       breaks = 30,
       xlim = c(0.0125, 0.0275),
       col = "lightblue", prob = TRUE, border = "black",
       xlab = "Inverse Square Root of Weight",
       ylab = "Counts", main = "Histogram of  Inverse Square Root of Weight")
  box(which = "plot", lty = "solid", col="black")

  # Overlay with  Normal density
  lines(density(x), col = "red")
```



NORMAL Q-Q PLOT

The Q-Q plot shows that our data shows systematic deviations from non-normality.

```
  qqnorm(invsqrt.weight, datax = TRUE, col = "red", ylim = c(0.01, 0.03), # generates no
rmal Quantile-Quantile (QQ) plot of the inverse sqrt values
         main = "Normal Q-Q Plot of Inverse Square Root of Weight")
  qqline(invsqrt.weight, col = "blue", datax = TRUE) # overlays theoretical QQ line
```



DE-TRANSFORM DATA

```
# Transform x using y = 1 / sqrt(x)
x <- cars$weight[1]; y <- 1 / sqrt(x)

# Detransform x using x = 1 / (y)^2
detransformedx <- 1 / y^2
x; y; detransformedx
```

```
## [1] 4209
```

```
## [1] 0.01541383
```

```
## [1] 4209
```

CREATE INDICATOR VARIABLES

```
  north_flag <- east_flag <- south_flag <- c(rep(NA, 10)) # combines replications of the
the value "NA" 10 times
  region <- c(rep(c("north", "south", "east", "west"),2), "north", "south") # combines r
eplications of N/S/E/W 2 times, North, and South

  # Change the region variable to indicators
  # Utilizing looping and if/else statements to set N/E/S to zeros and ones
  for (i in 1:length(region)) {
    if(region[i] == "north") north_flag[i] = 1
    else north_flag[i] = 0
    if(region[i] == "east") east_flag[i] = 1
    else east_flag[i] = 0
    if(region[i] == "south") south_flag[i] = 1
    else south_flag[i] = 0
  }
  north_flag; east_flag; south_flag
```

```
##  [1] 1 0 0 0 1 0 0 0 1 0
```

```
##  [1] 0 0 1 0 0 0 1 0 0 0
```

```
##  [1] 0 1 0 0 0 1 0 0 0 1
```

```
### INDEX FIELDS
  # Data frames have an index field;
  # the left-most column
  cars
```

mpg <dbl>	cylinders <int>	cubicinches <int>	hp <int>	weightlbs <int>	time.to.60 <int>	year <int>	brand <chr>
14.0	8	350	165	4209	12	1972	US.
31.9	4	89	71	1925	14	1980	Europe.
17.0	8	302	140	3449	11	1971	US.
15.0	8	400	150	3761	10	1971	US.
30.5	4	98	63	2051	17	1978	US.
23.0	8	350	125	3900	17	1980	US.
13.0	8	351	158	4363	13	1974	US.
14.0	8	440	215	4312	9	1971	US.
25.4	5	183	77	3530	20	1980	Europe.
37.7	4	89	62	2050	17	1982	Japan.

```
cars[order(cars$mpg),]
```

	mpg	cylinders	cubicinches	hp	weightlbs	time.to.60	year	brand
	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>	<chr>
227	10.0	8	360	215	4615	14	1971	US.
14	11.0	8	350	180	3664	11	1974	US.
101	11.0	8	400	150	4997	14	1974	US.
82	12.0	8	429	198	4952	12	1974	US.
97	12.0	8	350	180	4499	13	1974	US.
102	12.0	8	400	167	4906	13	1974	US.
133	12.0	8	455	225	4951	11	1974	US.
7	13.0	8	351	158	4363	13	1974	US.
47	13.0	8	318	150	3755	14	1977	US.
51	13.0	8	400	150	4464	12	1974	US.

1-10 of 261 rows Previous **1** 2 3 4 5 6 … 27 Next

```
# For vectors or matrices,
# add a column to act as an index field
x <- c(1,1,3:1,1:4,3); y <- c(9,9:1)
z <- c(2,1:9)
matrix <- t(rbind(x,y,z)); matrix # rbind = row bind
```

```
##       x y z
##  [1,] 1 9 2
##  [2,] 1 9 1
##  [3,] 3 8 2
##  [4,] 2 7 3
##  [5,] 1 6 4
##  [6,] 1 5 5
##  [7,] 2 4 6
##  [8,] 3 3 7
##  [9,] 4 2 8
## [10,] 3 1 9
```

```
indexed_m <- cbind(c(1:length(x)), matrix); indexed_m # cbind = column bind
```

```
##          x y z
##  [1,]  1 1 9 2
##  [2,]  2 1 9 1
##  [3,]  3 3 8 2
##  [4,]  4 2 7 3
##  [5,]  5 1 6 4
##  [6,]  6 1 5 5
##  [7,]  7 2 4 6
##  [8,]  8 3 3 7
##  [9,]  9 4 2 8
## [10,] 10 3 1 9
```

```
  indexed_m[order(z),]
```

```
##          x y z
##  [1,]  2 1 9 1
##  [2,]  1 1 9 2
##  [3,]  3 3 8 2
##  [4,]  4 2 7 3
##  [5,]  5 1 6 4
##  [6,]  6 1 5 5
##  [7,]  7 2 4 6
##  [8,]  8 3 3 7
##  [9,]  9 4 2 8
## [10,] 10 3 1 9
```

DUPLICATE RECORDS

```
  # For number of duplicate records, use anyDuplicated
  anyDuplicated(cars)
```

```
## [1] 0
```

```
  # To examine each record, use Duplicated
  duplicated(cars)
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [155] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [166] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [177] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [188] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [199] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [210] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [221] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [232] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [243] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [254] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
  # 'True': record is a duplicate,
  # 'False': record is not a duplicate

  # Let's duplicate the first record
  new.cars <- rbind(cars, cars[1,])

  # Check for duplicates
  anyDuplicated(new.cars)
```

```
## [1] 262
```

```
  # The 262nd record is a duplicate
  duplicated(new.cars)
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [155] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [166] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [177] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [188] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [199] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [210] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [221] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [232] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [243] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [254] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
```