Data Wrangling Assessment Task 2: Creating and pre-processing synthetic data

Insert student name and number here

If you have any questions regarding the assignment instructions and the R Markdown template, please post it on the discussion board **Questions on completing Assessment 2.**

Setup

Insert and load the packages you need to produce the report here:

```
knitr::opts_chunk$set(echo = TRUE)

library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 4.1.3
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(magrittr)
```

```
## Warning: package 'magrittr' was built under R version 4.1.3
```

```
library(dplyr) # For Wrangling Data
```

```
## Warning: package 'dplyr' was built under R version 4.1.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr) # Fore Reading and Writing Data.
```

```
## Warning: package 'tidyr' was built under R version 4.1.3
```

```
##
## Attaching package: 'tidyr'
```

```
## The following object is masked from 'package:magrittr':
##
##     extract
```

```
library(outliers)
```

```
## Warning: package 'outliers' was built under R version 4.1.3
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.1.3
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## Warning: package 'ggplot2' was built under R version 4.1.3
```

```
## Warning: package 'tibble' was built under R version 4.1.3
```

```
## Warning: package 'readr' was built under R version 4.1.3
```

```
## Warning: package 'purrr' was built under R version 4.1.3
```

```
## Warning: package 'stringr' was built under R version 4.1.3
```

```
## Warning: package 'forcats' was built under R version 4.1.3
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x lubridate::as.difftime() masks base::as.difftime()
## x lubridate::date()        masks base::date()
## x tidyr::extract()         masks magrittr::extract()
## x dplyr::filter()          masks stats::filter()
## x lubridate::intersect()   masks base::intersect()
## x dplyr::lag()             masks stats::lag()
## x purrr::set_names()       masks magrittr::set_names()
## x lubridate::setdiff()     masks base::setdiff()
## x lubridate::union()       masks base::union()
```

```
library(deducorrect)
```

```
## Warning: package 'deducorrect' was built under R version 4.1.3
```

```
## Loading required package: editrules
```

```
## Warning: package 'editrules' was built under R version 4.1.3
```

```
## Loading required package: igraph
```

```
## Warning: package 'igraph' was built under R version 4.1.3
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:purrr':
##
##     compose, simplify
```

```
## The following object is masked from 'package:tibble':
##
##     as_data_frame
```

```
## The following object is masked from 'package:tidyr':
##
##     crossing
```

```
## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union
```

```
## The following objects are masked from 'package:lubridate':
##
##     %--%, union
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
##
## Attaching package: 'editrules'
```

```
## The following objects are masked from 'package:igraph':
##
##     blocks, normalize
```

```
## The following object is masked from 'package:purrr':
##
##     reduce

## The following objects are masked from 'package:tidyr':
##
##     contains, separate

## The following object is masked from 'package:dplyr':
##
##     contains
```

```
library(deductive)
```

```
## Warning: package 'deductive' was built under R version 4.1.3
```

```
library(validate)
```

```
## Warning: package 'validate' was built under R version 4.1.3

##
## Attaching package: 'validate'

## The following objects are masked from 'package:igraph':
##
##     compare, hierarchy

## The following object is masked from 'package:ggplot2':
##
##     expr

## The following object is masked from 'package:dplyr':
##
##     expr
```

```
library(Hmisc)
```

```
## Warning: package 'Hmisc' was built under R version 4.1.3

## Loading required package: lattice

## Loading required package: survival

## Loading required package: Formula

##
## Attaching package: 'Hmisc'
```

```
## The following objects are masked from 'package:validate':
##
##     label, label<-

## The following objects are masked from 'package:dplyr':
##
##     src, summarize

## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
library(MVN)
```

```
## Warning: package 'MVN' was built under R version 4.1.3
```

```
library(readr)
library(openxlsx)
```

```
## Warning: package 'openxlsx' was built under R version 4.1.3
```

```
library(tinytex)
```

```
## Warning: package 'tinytex' was built under R version 4.1.3
```

```
#library(xlsx)
```

Data Wrangling Assessment Task 2: Creating and pre-processing synthetic data

Creating Synthetic Customer Data

As a Data Analyst for Uber, we have been tasked with generating 2 synthetic Data Sets and merging them together to create a larger data set, and inspect and clean them of any NA and Outlier values and deal with them accordingly.

We start off with our customer data set which we will refer to cust_df, which will contain some customer related data with 100 observations and the following variables:

- **CustomerID**
- **RiderRating**
- **UberPass**
- **TripID**
- **UberServices**
- **TimeRequested**
- **SuburbDropOff**
- **TravelTime**
- **TripDistance**
- **SurgeMultiplier**
- **BaseFare**
- **MinuteRate**
- **RatePerKM**

Customer ID

We start off with **CustomerID**, which is a vector created using the **sample()** function, creating a vector of 100 rows, created using random numbers between a range of 10000 and 11000. This will be our row for **CustomerID** and will be the first column of our **cust_df** data frame as per the code below. As **CustomerID** values are unique in nature, we set **replace = FALSE** to prevent any duplicates.

- ....

```
## Creating Synthetic Data Set

#### Creating Customer ID Variable
#### Creating CUSTOMER DATA FRAME
#For our dataset, Customer ID between 10000 and 11000

# Set the possible outcomes
x <- 10000:11000

# Set the sample size
n <- 100

# Set the value of the seed
SEED <- 234

set.seed(SEED)

CustomerID <- sample(x, n, replace = FALSE )
CustomerID
```

```
##   [1] 10736 10030 10289 10685 10998 10273 10492 10951 10384 10195 10579 10054
##  [13] 10309 10318 10078 10237 10042 10369 10873 10812 10487 10064 10381 10527
##  [25] 10354 10600 10276 10707 10994 10645 10151 10281 10111 10146 10041 10982
##  [37] 10841 10414 10718 10211 10077 10098 10887 10162 10754 10390 10220 10085
##  [49] 10655 10326 10870 10711 10057 10240 10536 10556 10774 10694 10576 10325
##  [61] 10987 10348 10441 10233 10606 10270 10856 10551 10717 10184 10489 10000
##  [73] 10321 10332 10187 10589 10833 10596 10310 10166 10070 10557 10280 10507
##  [85] 10192 10217 10705 10644 10727 10755 10391 10019 10603 10797 10845 10097
##  [97] 10809 10802 10430 10048
```

```
cust_df <- as.data.frame(CustomerID)
```

Customer Rating

Uber works on a system where riders and drivers can provide ratings to each other. For our customer ratings column, we use the **runif()** function to generate a **RiderRating** column of 100 observations between values of 4.2 and 5 as per the code below. **runif()** is used here as we're not making assumptions about the distribution of the rating.

```
## Customer Rating
set.seed(123)

cust_df$RiderRating <- runif(100, 4.2, 5)
```

Uber One

Uber One is a membership program created by Uber which allows members access to special discounts across its service offerings including Uber Eats (Uber, 2022). This column will identify whether the customer in our dataset is a member.

This variable will be a boolean with responses including **TRUE** and **FALSE**. We assume that 25% of the customers in our dataset are members. In creating the variable we set probability as 25% TRUE for members, and 75% False for non members as per the code below. As with **Customer ID** we will be attaching each column to our **cust_df** as they are created.

```
#Uber One
#Uber provide a membership service
# Services
a <- c(TRUE, FALSE)

n <- 100

SEED <- 215

set.seed(SEED)

# Generate the synthetic data
cust_df$UberPass <- sample(a, n, replace = TRUE, prob = c(0.25, 0.75))
#cust_df$UberPass
```

Trip ID

TripID, is a variable representing unique identifier for requested trips. similar to the CustomerID variable, we use the **sample()** function, to create a variable using random numbers generated using a range of numbers between 10000 and 60000 with 100 observations. **TripIDs** are unique, and again we set **replace = FALSE** to prevent any duplicates.

```
################    Trip ID
# Set the possible outcomes
y <- 10000:60000

# Set the sample size
n <- 100

# Set the value of the seed
SEED <- 105

set.seed(SEED)

cust_df$TripID <- sample(y, n, replace = FALSE )
#cust_df$TripID
```

Service Levels

Uber provides different service levels that provide customers options depending on their requests and budget (Uber, 2022).
For the purposes of our data set, we will focus on the following from least to most expensive:

UberX - which is a basic level of service.
UberComfort - which provides a comfortable mid-sized car with top-rated drivers.
UberXL - which provides a larger vehichle for comfort and can accomodate more passengers or groups.
UberPremier - which is a luxury option where a luxury vehicle is used for passengers riding to special events.

we use the **sample()** function to randomly generate our **UberService** column, and we also use **prob** to determine the chance of the service being used, with the cheaper options, being having greater probability.

As per the below code, we set probability(stackexchange, 2011):

- 40% - UberX
- 25% - UberComfort
- 20% - UberXL
- 15% - UberPremier

```
# Services

#https://stats.stackexchange.com/questions/14158/how-to-generate-random-categorical-data
Services <- c("UberX", "UberComfort", "UberXL", "UberPremier")

n <- 100

SEED <- 205

set.seed(SEED)

# Generate the synthetic data
cust_df$UberServices <- sample(Services, n, replace = TRUE, prob = c(0.40, 0.25, 0.2, 0.15 ))
# Display the synthetic data
#We set probability of services "Pool", "UberX", "UberComfort", "UberXL", "UberPremier" to 0.25, .40, 0
#cust_df$UberServices
```

Time Requested

This variable shows when the UBER was ordered. For our dataset, our trip data is focused on a week between 28/03/2022 and 03/04/2022. We set the range for our date and time using the following code:

TimeRange <- seq(as.POSIXct('2022-03-28 00:00:00 AEDT'), as.POSIXct('2022-04-03 23:59:59 AEDT'), by = "sec")

The **as.POSIXct** function allows us to generate our beginning and end dates, **by = sec** allows time to be divided in second increments.

Once we have our range of dates, we then use the sample() function to pull 100 randomly chosen dates and times to create the column for our **cust_df** dataframe.

```
# Generating a random range of time and date within a week (Stackoverflow.com, 2017)
# <https://stackoverflow.com/questions/45633452/generate-random-times-in-sample-of-posixct>
#POSIXct gives date and time

TimeRange <- seq(as.POSIXct('2022-03-28 00:00:00 AEDT'), as.POSIXct('2022-04-03 23:59:59 AEDT'), by = "s


set.seed(2022)
```

```r
n <- 100

cust_df$TimeRequested <- sample(TimeRange, n, replace = TRUE)
```

Destination

For Destination, our sample data will assume that our customers will be ordering an Uber from the centre of Parramatta CBD and travelling to a number of suburbs of varying distances. We then create a vector with our chosen suburbs of the following suburbs:

- Darlinghurst
- Sydney
- Kings Cross
- Glebe
- Blacktown
- Ryde
- Winston Hills
- Strathfield
- Burwood
- Canley Heights
- Baulkham Hills
- Pennant Hills
- Eastwood
- Enfield
- Ashfield
- Leichardt
- Fairfield
- Sefton
- Lidcombe
- Greystanes
- Carlingford
- Chester Hill
- Auburn
- North Rocks
- Merrylands
- Guildford

We again use the sample function to create our SuburbDropOff variable oof 100 observations which will serve to tell us where the customers in our data set are going.

```r
#Destination

Destination <- c("Darlinghurst", "Sydney", "Kings Cross", "Glebe", "Blacktown", "Ryde", "Winston Hills"

n <- 100
SEED <- 768
set.seed(SEED)

cust_df$SuburbDropOff <- sample(Destination, n, replace = TRUE)
```

Travel Time

Travel Time is a variable that will give us the duration of the customer's trip. We used Google Maps(Google, 2022), to get estimates of travel times and distance from parramatta to each suburb. Suburbs that are in relatively similar travel window by time are then grouped together into 4 districts.

From the code below:
cust_df$TravelTime[cust_df$SuburbDropOff == "Merrylands"| cust_df$SuburbDropOff == "Guildford"|cust_df$SuburbDrop
== "Parramatta"| cust_df$SuburbDropOff == "North Parramatta"] <- runif(d1, 5, 10)

We use the runif() function to generate random travel times. As per the above code, we see that a lower limit and upper limit of 5 minutes and 10 minutes is chosen respectively. As the values are generated for each district, they are then added to the column we call TravelTime.

```
### Travel Time

set.seed(5000)

# District 1 -
d1 <- sum(cust_df$SuburbDropOff == "Merrylands"| cust_df$SuburbDropOff == "Guildford"| cust_df$SuburbDr

cust_df$TravelTime[cust_df$SuburbDropOff == "Merrylands"| cust_df$SuburbDropOff == "Guildford"| cust_df


#District 2 -
d2 <- sum(cust_df$SuburbDropOff == "Fairfield"| cust_df$SuburbDropOff == "Sefton"|cust_df$SuburbDropOff

cust_df$TravelTime[cust_df$SuburbDropOff == "Fairfield"| cust_df$SuburbDropOff == "Sefton"|cust_df$Subu



#District 3
d3 <- sum(cust_df$SuburbDropOff == "Blacktown"| cust_df$SuburbDropOff == "Ryde"|cust_df$SuburbDropOff =


cust_df$TravelTime[cust_df$SuburbDropOff == "Blacktown"| cust_df$SuburbDropOff == "Ryde"|cust_df$Suburb



#District 4

d4 <- sum(cust_df$SuburbDropOff == "Darlinghurst"| cust_df$SuburbDropOff == "Sydney"| cust_df$SuburbDro


cust_df$TravelTime[cust_df$SuburbDropOff == "Darlinghurst"| cust_df$SuburbDropOff == "Sydney"|cust_df$Su
```

Travel Distance

Similar to the creation of our TravelTime variable, google maps (google, 2022) is used to get estimates for travel distances for each of our destinations. Using the same groups created for our **TravelTime** column, we use the **runif()** function to generate random travel distances for each group depending on how far that group is from our assumed pick up spot of Parramatta CBD.

The randomness can account for differing routes and traffic levels.

```r
# Travel Distance

set.seed(6000)

# District 1 -
k1 <- sum(cust_df$SuburbDropOff == "Merrylands"| cust_df$SuburbDropOff == "Guildford"| cust_df$SuburbDr

cust_df$TripDistance[cust_df$SuburbDropOff == "Merrylands"| cust_df$SuburbDropOff == "Guildford"| cust_


#District 2 -
k2 <- sum(cust_df$SuburbDropOff == "Fairfield"| cust_df$SuburbDropOff == "Sefton"|cust_df$SuburbDropOff

cust_df$TripDistance[cust_df$SuburbDropOff == "Fairfield"| cust_df$SuburbDropOff == "Sefton"|cust_df$Su



#District 3
k3 <- sum(cust_df$SuburbDropOff == "Blacktown"| cust_df$SuburbDropOff == "Ryde"|cust_df$SuburbDropOff =


cust_df$TripDistance[cust_df$SuburbDropOff == "Blacktown"| cust_df$SuburbDropOff == "Ryde"|cust_df$Subu



#District 4

k4 <- sum(cust_df$SuburbDropOff == "Darlinghurst"| cust_df$SuburbDropOff == "Sydney"| cust_df$SuburbDro


cust_df$TripDistance[cust_df$SuburbDropOff == "Darlinghurst"| cust_df$SuburbDropOff == "Sydney"|cust_df
```

Surge Pricing

Uber utilises surge pricing in times where there is increased demand for their services during peak periods (Uber, 2022). A look at their website indicates that the busiest times are generally on Friday nights, Saturday mornings, Saturday nights and Sunday Mornings.

In order to simulate the effect of surge pricing, we use the **mutate()** function below to create a number of arguments to with respect to the random values generated in our **TimeRequested** Column.

Between 5pm and 7pm we utilize a multiplier of 1.5 on Friday and Saturday, a multiplier of 2 between 7pm and midnight on Saturday and Friday. A multiplier of 1.5 between midnight and 5am on Saturday and Sunday mornings.

```r
#Surge Variable:
#Uber utilises increases the price of using their platform during periods where the demand for rides is

cust_df %<>%
  mutate(SurgeMultiplier= case_when(is.na(TimeRequested) ~ "1.0",
                                    TimeRequested <= "2022-04-01 17:00:00" ~ "1.0",
                                    TimeRequested <= "2022-04-01 19:00:00" ~ "1.5",
                                    TimeRequested <= "2022-04-01 23:59:59" ~ "2.0",
                                    TimeRequested <= "2022-04-02 05:00:00" ~ "1.5",
```

```
                                    TimeRequested <= "2022-04-02 17:00:00" ~ "1.0",
                                    TimeRequested <= "2022-04-02 19:00:00" ~ "1.5",
                                    TimeRequested <= "2022-04-02 23:59:59" ~ "2.0",
                                    TimeRequested <= "2022-04-03 05:00:00" ~ "1.5",
                                    TRUE ~ "1.0"))
```

Fares

Uber utilisese a tiered pricing structure for their services (finder.com, 2021). Uber's fares consist of a **base fare**, a **rate charged per km** and a **rate charged for trip time**. All vary depending on the service level used.

As the base fare and rates vary depending on service level, we use the information generated from our UberService column to create the following columns for our customer data:

- BaseFare
- MinuteRate
- RateperKM

The mutate function is used to create the appropriate base fare and rates in the above columns. The below code shows how we create the columns and how the base fare and rates vary depending on service level.

```
######## FARE - Correlated ######
#https://www.finder.com.au/uber-sydney-fares-services
#Base Rate calculated based on service requested

cust_df %<>%
  mutate(BaseFare = case_when(is.na(UberServices) ~ "2.50",
                                    UberServices == "UberX" ~ "2.50",
                                    UberServices == "UberComfort" ~ "3.38",
                                    UberServices == "UberXL" ~ "4",
                                    UberServices == "UberPremier" ~ "5.5",
                                    TRUE ~ "2.75"))

#Rate per Minute - varies by service requested
cust_df %<>%
  mutate(MinuteRate = case_when(is.na(UberServices) ~ "0.40",
                                    UberServices == "UberX" ~ "0.40",
                                    UberServices == "UberComfort" ~ "0.54",
                                    UberServices == "UberXL" ~ "0.61",
                                    UberServices == "UberPremier" ~ "0.88",
                                    TRUE ~ "0.40"))

#Rate per KM - varies by service requested
cust_df %<>%
  mutate(RatePerKM = case_when(is.na(UberServices) ~ "1.45",
                                    UberServices == "UberX" ~ "1.45",
                                    UberServices == "UberComfort" ~ "1.96",
                                    UberServices == "UberXL" ~ "2.31",
                                    UberServices == "UberPremier" ~ "3.19",
                                    TRUE ~ "1.45"))

#cust_df$TotalFare <- cust_df$BaseFare + (cust_df$TravelTime * cust_df$MinuteRate)
```

Creating Synthetic Driver Data

We will now shift our focus to creating synthetic driver data with information relevant to drivers. Our driver dataframe, Drive_df, will consist of:

- **DriverID**
- **DriverRating**
- **YearsWorked**
- **CarOwnership**
- **CarType**
- **MaxPassengers**
- **TripID**

Driver ID

We will first create a unique identifier for drivers, very much like **customerIDs**. Using the **sample()** function we create 100 observations of random numbers ranged between 40000 and 50000. As DriverIDs are unique, we set **replace = FALSE**

```
#################### DRIVER DATA FRAME

#Create DriverID variable
#For our dataset, DriverID between 40000 and 50000

# Set the possible outcomes
c <- 40000:50000

# Set the sample size
n <- 100

# Set the value of the seed
SEED <- 209

set.seed(SEED)

DriverID <- sample(c, n, replace = FALSE )
DriverID
```

```
##   [1] 40545 46210 47833 41221 46969 48901 44746 46472 49060 48534 45169 42094
##  [13] 47813 44491 45845 49652 41595 41779 46692 48351 45635 40250 43139 41391
##  [25] 42862 40827 41930 49865 48832 49042 42571 46681 44859 41796 46876 44329
##  [37] 44928 43178 48465 41728 42069 48930 46165 44745 47095 40379 44092 41529
##  [49] 44113 43613 43530 48883 47366 46186 44200 48681 45657 41674 44563 41971
##  [61] 45372 46071 46242 44411 43699 47848 46482 49116 44607 43884 43665 42229
##  [73] 46075 45757 46622 40406 48604 43867 40419 47173 40831 43605 49719 41707
##  [85] 47909 42459 47743 46329 49547 41253 48315 43368 47724 42181 43681 43601
##  [97] 47938 47080 48100 47880
```

```
Drive_df <- as.data.frame(DriverID)
```

Driver Rating

Similar to our **RiderRating**, we use the **runif()** function to create our **DriverRating** column.
Uber Drivers are expected to maintain an average rating of 4.6 from their most recent 100 trips to continue using the app (The Guardian, 2019.

In creating our **DriverRating** column, we use a range between 4.5 to 5 in creating data that is realistic.

```
## Driver Rating
set.seed(7778)

Drive_df$DriverRating <- runif(100, 4.5, 5)
```

Years Worked

The number of years the driver has been signed on and working as an uber driver. We set a range between 0 to 10 years and use the **sample()** function to create our **YearsWorked** Column. As the values do not need to be unique, we set **replace = False**.

```
#Drive_DF - Number of years worked

# Set the possible outcomes
d <- 0:10

# Set the sample size
n <- 100

# Set the value of the seed
SEED <- 150

set.seed(SEED)

Drive_df$YearsWorked <- sample(d, n, replace = TRUE)
```

Car Ownership

This column indicates whether the driver is renting or owns their car used for Uber. **sample()** function is used to generate our **CarOwnership** column with values as **Rent** and **Own**.

```
#Car Ownership
SEED <-333
set.seed(SEED)
ownership <- c("Rent", "Own")
Drive_df$CarOwnership <- sample(ownership, n, replace = TRUE)
Drive_df$CarOwnership
```

```
##   [1] "Own"  "Rent" "Rent" "Own"  "Rent" "Own"  "Own"  "Own"  "Rent" "Own"
##  [11] "Rent" "Rent" "Own"  "Rent" "Rent" "Rent" "Rent" "Own"  "Rent" "Own"
##  [21] "Rent" "Own"  "Own"  "Own"  "Rent" "Rent" "Own"  "Rent" "Own"  "Own"
##  [31] "Own"  "Rent" "Rent" "Own"  "Own"  "Rent" "Own"  "Own"  "Own"  "Own"
##  [41] "Rent" "Rent" "Rent" "Rent" "Rent" "Rent" "Own"  "Rent" "Own"  "Rent"
##  [51] "Rent" "Rent" "Rent" "Rent" "Rent" "Own"  "Own"  "Own"  "Own"  "Own"
```

```
##  [61] "Rent" "Own"  "Rent" "Rent" "Rent" "Rent" "Own"  "Rent" "Own"  "Own"
##  [71] "Own"  "Own"  "Rent" "Rent" "Own"  "Own"  "Own"  "Rent" "Rent" "Rent"
##  [81] "Own"  "Rent" "Rent" "Rent" "Rent" "Own"  "Rent" "Rent" "Own"  "Own"
##  [91] "Own"  "Own"  "Rent" "Own"  "Rent" "Rent" "Own"  "Own"  "Rent" "Own"
```

Car Type

CarType shows us the type of car the driver is using. Among Uber's vehicle requirements (Uber, 2022), is that the vehicle be a 4 door car or passenger van.

As we are focusing on a limited range of services, our data will consist of vehicles that Uber would find appropriate. We will focus on the followoing types of vehichles:

- **4-Door Sedan/Hatch**
- **SUV**
- **Luxury Sedan**

Our column for car type is based on the service that the driver can provide. Based off **UberServices** data provided in our **cust_df** data set.

4-Door Sedan/Hatch are suitable for **UberX** and **UberComfort** services, **SUVs** are suitable for **UberXL** as they can sit 5 passengers, we have **Luxury Sedan** suitable for **Uber Premier**.

As with the Base Fare and Rates we generated, we use the **mutate()** function to create our **CarType** data from the **UberServices** variable from our **cust_df** column.

```
#Car Type
# Creating Variable for Services Offered
Services <- c("UberX", "UberComfort", "UberXL", "UberPremier")

n <- 100

SEED <- 205

set.seed(SEED)

# Generate the synthetic data
OfferedService <- sample(Services, n, replace = TRUE, prob = c(0.40, 0.25, 0.2, 0.15 ))
# Display the synthetic data

as.factor(OfferedService)
```

```
##   [1] UberPremier UberX       UberComfort UberXL      UberXL      UberX
##   [7] UberX       UberPremier UberPremier UberPremier UberX       UberX
##  [13] UberX       UberX       UberComfort UberX       UberComfort UberX
##  [19] UberComfort UberX       UberComfort UberPremier UberXL      UberComfort
##  [25] UberX       UberX       UberPremier UberX       UberPremier UberX
##  [31] UberX       UberXL      UberPremier UberXL      UberComfort UberX
##  [37] UberPremier UberPremier UberXL      UberComfort UberX       UberX
##  [43] UberComfort UberXL      UberXL      UberPremier UberComfort UberX
##  [49] UberXL      UberXL      UberX       UberX       UberPremier UberXL
##  [55] UberXL      UberX       UberPremier UberX       UberXL      UberX
##  [61] UberX       UberComfort UberX       UberXL      UberComfort UberX
##  [67] UberPremier UberXL      UberComfort UberX       UberX       UberXL
```

```
##  [73] UberXL      UberX       UberX       UberX       UberX       UberX
##  [79] UberXL      UberComfort UberX       UberXL      UberX       UberXL
##  [85] UberX       UberX       UberX       UberX       UberX       UberComfort
##  [91] UberX       UberComfort UberX       UberComfort UberComfort UberX
##  [97] UberX       UberXL      UberComfort UberComfort
## Levels: UberComfort UberPremier UberX UberXL
```

```
#We set probability of services "Pool", "UberX", "UberComfort", "UberXL", "UberPremier" to 0.25, .40, 0
Drive_df$OfferedService
```

```
## NULL
```

```
#Creating variable for type of car used by driver.
Drive_df %<>%
  mutate(CarType = case_when(is.na(OfferedService) ~ "Hatch",
                                  OfferedService == "UberX" ~ "4-Door Sedan/Hatch",
                                  OfferedService == "UberComfort" ~ "4-Door Sedan/Hatch",
                                  OfferedService == "UberXL" ~ "SUV",
                                  OfferedService == "UberPremier" ~ "Luxury Sedan",
                                  TRUE ~ "Hatch"))
```

Max Passengers

This column will provide information on the maximum number of passengers a vehicle can sit.
* **4-Door Sedan/Hatch** can seat 3 passengers, * **SUVs** can seat 5 passengers * **Luxury Sedan** can set up to 3 passengers.

We use the **mutate** function to create our **MaxPassenger** column from our **CarType**.

```
# Creating variable for Max Passengers
Drive_df %<>%
  mutate(MaxPassengers = case_when(is.na(CarType) ~ "3",
                                  CarType == "4-Door Sedan/Hatch" ~ "3",
                                  CarType == "4-Door Sedan/Hatch" ~ "3",
                                  CarType == "SUV" ~ "5",
                                  CarType == "Luxury Sedan" ~ "3",
                                  TRUE ~ "3"))
```

Trip ID

TripID as stated before, is a unique identifier created when our customer requests a ride. When the app matches the rider and driver, information from our **cust_df** and **Driver_df** is shared. As this column will be the common variable between both data sets, we create a copy of **TripID** from our **cust_df** using **<-**.

```
#Trip ID
# Set the possible outcomes


Drive_df$TripID <- cust_df$TripID
Drive_df$TripID
```

```
##   [1] 36065 12565 59787 33182 56206 11658 47032 35631 25381 35448 19015 50628
##  [13] 41814 49384 45958 47447 31618 27202 40757 58169 11467 31902 27639 40650
##  [25] 29982 57809 51182 26859 32411 53450 52818 23595 22516 49882 26299 27677
##  [37] 43581 31850 28973 57273 11829 48927 38443 28997 41967 30924 54971 39494
##  [49] 58342 37850 29710 31957 39634 39041 49934 18729 38728 16375 21545 21245
##  [61] 33583 52287 51607 58630 25236 45575 52849 17301 33388 13083 32400 10581
##  [73] 32927 57182 18975 48479 44420 42470 35410 10626 14184 21224 53212 22012
##  [85] 55615 18733 12611 36240 57443 42184 16178 53499 14165 12991 26808 44128
##  [97] 55755 37859 21308 17414
```

Inputting Missing Values and Outliers into Data Sets

Missing values and Outliers can occur in datasets for a variety of reasons. They can occur as a result of errors from data entry, measurements, experimental errors, intentional errors, data processing errors and sampling errors.

In creating realistic data sets we will place three NA values in each data set (**cust_df**, and **Driver_df**) and place Outliers in the following numeric variables:

- RiderRating
- TravelTime
- TripDistance
- BaseFare
- MinuteRate
- RatePerKM
- DriverRating

This is done manually by subsetting each of the **cust_df** and **Driver_df** dataframe by **row and column** numbers I've selected at random.

For our **cust_df** data frame, missing values, 2 are placed in **RiderRating** column, and one is placed in **UberServices**.

For **Driver_df**, two missing values are placed in **DriverRating** column, and one in **CarType** column.

```
#############################################################################################################
#                                                NAs and Outliers
#############################################################################################################
summary(cust_df)
```

```
##    CustomerID     RiderRating     UberPass          TripID
##  Min.   :10000   Min.   :4.200   Mode :logical   Min.   :10581
##  1st Qu.:10216   1st Qu.:4.396   FALSE:74        1st Qu.:23325
##  Median :10402   Median :4.573   TRUE :26        Median :35429
##  Mean   :10451   Mean   :4.599                   Mean   :35420
##  3rd Qu.:10706   3rd Qu.:4.804                   3rd Qu.:48591
##  Max.   :10998   Max.   :4.995                   Max.   :59787
##  UberServices       TimeRequested                  SuburbDropOff
##  Length:100         Min.   :2022-03-28 07:44:45   Length:100
##  Class :character   1st Qu.:2022-03-29 14:05:38   Class :character
##  Mode  :character   Median :2022-03-31 01:00:05   Mode  :character
##                     Mean   :2022-03-31 06:00:31
##                     3rd Qu.:2022-04-01 16:44:43
##                     Max.   :2022-04-03 22:44:47
```

```
##    TravelTime      TripDistance    SurgeMultiplier      BaseFare
##  Min.   : 5.037   Min.   : 2.078   Length:100         Length:100
##  1st Qu.:12.585   1st Qu.: 7.388   Class :character   Class :character
##  Median :17.906   Median :13.216   Mode  :character   Mode  :character
##  Mean   :18.103   Mean   :13.495
##  3rd Qu.:22.236   3rd Qu.:18.185
##  Max.   :34.219   Max.   :33.959
##   MinuteRate         RatePerKM
##  Length:100         Length:100
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
summary(Drive_df)
```

```
##     DriverID      DriverRating    YearsWorked    CarOwnership
##  Min.   :40250   Min.   :4.510   Min.   : 0.00   Length:100
##  1st Qu.:42789   1st Qu.:4.667   1st Qu.: 2.75   Class :character
##  Median :45049   Median :4.789   Median : 5.00   Mode  :character
##  Mean   :45121   Mean   :4.776   Mean   : 5.26
##  3rd Qu.:47729   3rd Qu.:4.893   3rd Qu.: 8.00
##  Max.   :49865   Max.   :4.999   Max.   :10.00
##    CarType          MaxPassengers          TripID
##  Length:100         Length:100         Min.   :10581
##  Class :character   Class :character   1st Qu.:23325
##  Mode  :character   Mode  :character   Median :35429
##                                        Mean   :35420
##                                        3rd Qu.:48591
##                                        Max.   :59787
```

```
#Cust_fd  -  Placing NA values into cust_df
cust_df[2, 2] = NA #Places NA value in 2nd row  in RiderRating Column
cust_df[93, 2] = NA #Places NA value in 2nd row  in RiderRating Column
cust_df[53, 5] = NA #Places NA value in 2nd row  in UberServices


#Drive_df  -  Placing NA values into Drive_df
Drive_df[10, 2] = NA #Places NA value in 2nd row  in DriverRating Column
Drive_df[49, 5] = NA #Places NA value in 2nd row  in CarType Column
Drive_df[56, 6] = NA #Places NA value in 2nd row  in DriverRating Column



#Cust_df  -  Outliers
#Rider Rating
cust_df[23, 2] = 2 # Outlier number entered into row 23 of RiderRating Column

#Travel Time
cust_df[52, 8] = 50 # Outlier number entered into row 52 of TravelTime Column

#Trip Distance
```

```
cust_df[70, 9] = 45 # Outlier number entered into row 70 of TripDistance Column

#Base Fare
cust_df[83, 11 ] = 0.20 # Outlier number entered into row 83 of BaseFare Column

#Minute Rate
cust_df[83 , 12] = 0.35 # Outlier number entered into row 83 of MinuteRate Column

#RatePerKM
cust_df[83, 13] = 1  # Outlier number entered into row 83 of RatePerKM Column



#Drive_df  - Outliers
#DriverRating
Drive_df[99, 2] = 3 # Outlier number entered into row 99 of DriverRating Column

#Years Worked
Drive_df[50, 3] = 18 # Outlier number entered into row 50 of YearsWorked Column

#MaxPassengers
Drive_df[43, 6] = 7 # Outlier number entered into row 50 of MaxPassengers Column
```

Merging Data Set

Now that we have our separate dataframes it is time to merge them. We achieve this using the **merge.data.frame** function.

The result of the merge, creates a data frame with 100 observations and 19 variables when we use the **str()** function.

We then use **as.numeric()** function to convert the following columns to numeric data:

- TripID
- CustomerID
- RiderRating
- TravelTime
- TripDistance
- SurgeMultiplier
- BaseFare
- MinuteRate
- RatePerKM
- DriverID
- YearsWorked
- MaxPassengers

as.factor() function is used on **UberServices**, **UberSuburbDropOff**, **CarType** and **CarOwnership**

```
#############################################################################################################
######################### MERGING DATASETS ##################################################################

UBER <- merge.data.frame(cust_df, Drive_df)
```

```r
str(UBER)
```

```
## 'data.frame':    100 obs. of  19 variables:
##  $ TripID         : int  10581 10626 11467 11658 11829 12565 12611 12991 13083 14165 ...
##  $ CustomerID     : int  10000 10166 10487 10273 10077 10030 10705 10797 10184 10603 ...
##  $ RiderRating    : num  4.7 4.29 4.91 4.24 4.31 ...
##  $ UberPass       : logi  TRUE FALSE FALSE TRUE TRUE FALSE ...
##  $ UberServices   : chr  "UberXL" "UberComfort" "UberComfort" "UberX" ...
##  $ TimeRequested  : POSIXct, format: "2022-04-03 12:34:29" "2022-03-30 11:04:15" ...
##  $ SuburbDropOff  : chr  "Blacktown" "Auburn" "Eastwood" "Chester Hill" ...
##  $ TravelTime     : num  20.6 15.6 19.6 14.1 17.4 ...
##  $ TripDistance   : num  16.63 9.11 14.31 9.37 19.12 ...
##  $ SurgeMultiplier: chr  "1.0" "1.0" "1.0" "1.0" ...
##  $ BaseFare       : chr  "4" "3.38" "3.38" "2.50" ...
##  $ MinuteRate     : chr  "0.61" "0.54" "0.54" "0.40" ...
##  $ RatePerKM      : chr  "2.31" "1.96" "1.96" "1.45" ...
##  $ DriverID       : int  42229 47173 45635 48901 42069 46210 47743 42181 43884 47724 ...
##  $ DriverRating   : num  4.91 4.95 4.95 4.81 4.99 ...
##  $ YearsWorked    : num  9 2 5 4 2 2 9 9 9 0 ...
##  $ CarOwnership   : chr  "Own" "Rent" "Rent" "Own" ...
##  $ CarType        : chr  "SUV" "4-Door Sedan/Hatch" "4-Door Sedan/Hatch" "4-Door Sedan/Hatch" ...
##  $ MaxPassengers  : chr  "5" "3" "3" "3" ...
```

```r
colnames(UBER)
```

```
##  [1] "TripID"          "CustomerID"      "RiderRating"     "UberPass"
##  [5] "UberServices"    "TimeRequested"   "SuburbDropOff"   "TravelTime"
##  [9] "TripDistance"    "SurgeMultiplier" "BaseFare"        "MinuteRate"
## [13] "RatePerKM"       "DriverID"        "DriverRating"    "YearsWorked"
## [17] "CarOwnership"    "CarType"         "MaxPassengers"
```

```r
#### Converting Data Type

UBER$TripID <- as.numeric(UBER$TripID)
UBER$CustomerID <- as.numeric(UBER$CustomerID)
UBER$RiderRating <- as.numeric(UBER$RiderRating)
UBER$TravelTime <- as.numeric(UBER$TravelTime)
UBER$TripDistance <- as.numeric(UBER$TripDistance)
UBER$SurgeMultiplier <- as.numeric(UBER$SurgeMultiplier)
UBER$BaseFare <- as.numeric(UBER$BaseFare)
UBER$MinuteRate <- as.numeric(UBER$MinuteRate)
UBER$RatePerKM <- as.numeric(UBER$RatePerKM)
UBER$DriverID <- as.numeric(UBER$DriverID)
UBER$DriverRating <- as.numeric(UBER$DriverRating)
UBER$YearsWorked <- as.numeric(UBER$YearsWorked)
UBER$MaxPassengers <- as.numeric(UBER$MaxPassengers)
UBER$UberServices <- factor(UBER$UberServices, levels = c("UberX", "UberComfort", "UberXL", "UberPremie
UBER$SuburbDropOff <- as.factor(UBER$SuburbDropOff)
UBER$CarOwnership <- as.factor(UBER$CarOwnership)
UBER$CarType <- as.factor(UBER$CarType)
UBER$MaxPassengers <- as.numeric(UBER$MaxPassengers)

summary(UBER)
```

```
##      TripID        CustomerID     RiderRating     UberPass
##  Min.   :10581   Min.   :10000   Min.   :2.000   Mode :logical
##  1st Qu.:23325   1st Qu.:10216   1st Qu.:4.389   FALSE:74
##  Median :35429   Median :10402   Median :4.569   TRUE :26
##  Mean   :35420   Mean   :10451   Mean   :4.570
##  3rd Qu.:48591   3rd Qu.:10706   3rd Qu.:4.803
##  Max.   :59787   Max.   :10998   Max.   :4.995
##                                  NA's   :2
##       UberServices TimeRequested                    SuburbDropOff
##  UberX      :45    Min.   :2022-03-28 07:44:45   Eastwood  : 7
##  UberComfort:20    1st Qu.:2022-03-29 14:05:38   Fairfield : 7
##  UberXL     :21    Median :2022-03-31 01:00:05   Burwood   : 6
##  UberPremier:13    Mean   :2022-03-31 06:00:31   Greystanes: 6
##  NA's       : 1    3rd Qu.:2022-04-01 16:44:43   Blacktown : 5
##                    Max.   :2022-04-03 22:44:47   Merrylands: 5
##                                                  (Other)   :64
##    TravelTime      TripDistance    SurgeMultiplier    BaseFare
##  Min.   : 5.037   Min.   : 2.078   Min.   :1.00    Min.   :0.200
##  1st Qu.:12.585   1st Qu.: 7.388   1st Qu.:1.00    1st Qu.:2.500
##  Median :17.906   Median :13.216   Median :1.00    Median :3.380
##  Mean   :18.415   Mean   :13.806   Mean   :1.07    Mean   :3.388
##  3rd Qu.:22.443   3rd Qu.:18.202   3rd Qu.:1.00    3rd Qu.:4.000
##  Max.   :50.000   Max.   :45.000   Max.   :2.00    Max.   :5.500
##
##    MinuteRate       RatePerKM        DriverID      DriverRating
##  Min.   :0.3500   Min.   :1.000   Min.   :40250   Min.   :3.000
##  1st Qu.:0.4000   1st Qu.:1.450   1st Qu.:42789   1st Qu.:4.664
##  Median :0.5400   Median :1.960   Median :45049   Median :4.784
##  Mean   :0.5388   Mean   :1.972   Mean   :45121   Mean   :4.758
##  3rd Qu.:0.6100   3rd Qu.:2.310   3rd Qu.:47729   3rd Qu.:4.894
##  Max.   :0.8800   Max.   :3.190   Max.   :49865   Max.   :4.999
##                                                   NA's   :1
##   YearsWorked    CarOwnership               CarType   MaxPassengers
##  Min.   : 0.00   Own :48      4-Door Sedan/Hatch:65   Min.   :3.000
##  1st Qu.: 2.75   Rent:52      Luxury Sedan      :14   1st Qu.:3.000
##  Median : 5.00                SUV               :20   Median :3.000
##  Mean   : 5.36                NA's              : 1   Mean   :3.465
##  3rd Qu.: 8.00                                        3rd Qu.:3.000
##  Max.   :18.00                                        Max.   :7.000
##                                                       NA's   :1
```

Create/mutate at least one variable from the existing ones

Now that we have our merged data set, we will use the **mutate** function to create the **FARETOTAL** using the data we have created from the **BaseFare**, **TravelTime**, **MinuteRate**, **TripDistance**, **RatePerKM** and **SurgeMultiplier** columns.

As noted previously, Uber's fares consist of a base fare as well as rates for distance travelled and time taken. This is also affected by a surge pricing multiplier. This is reflected in the code below.

```
####FARE

UBER %<>%
  mutate(FARETOTAL = (BaseFare + (TravelTime * MinuteRate) + (TripDistance * RatePerKM) * SurgeMultipli
```

Scanning Data Set for Missing Values

Using **sum(is.na())**, we can identify the number of missing values in out **UBER** data frame. From the code, we can identify 6 missing values.

colnames() prints gives us the column/ variable names where there are missing values.

Using the **which(is.na())** and subsetting the data frame by the variable name, provide us with the rows of missing values.

When we find the missing values, we can then use the **impute()** function to replace the values as follows:
* **RiderRating** we can impute using the mean as it is numeric data. * **UberServices** we can impute using mode as it is categorical data.
* **DriverRating** we can impute using the mean. * **CarType** we impute using mode as it is categorical data. * **MaxPassengers** as this is numeric data with discrete values, we can impute the missing value with the median.

Once we have replaced the missing values, we can use **sum(is.na(UBER))** to check to make sure all the missing values have been dealt with.

```
##### STEP 4 - SCAN DATA SET FOR MISSING VALUES ###

sum(is.na(UBER)) # Shows us the number of NA values in data frame. In this case there are 6
```

```
## [1] 6
```

```
# Identifying variables containing NA values.

col_names <- colnames(UBER)[colSums(is.na(UBER)) > 0]
col_names
```

```
## [1] "RiderRating"   "UberServices"  "DriverRating"  "CarType"
## [5] "MaxPassengers"
```

```
# The code above showed which columns had missing values, now we can use which() function to find the c

which(is.na(UBER$RiderRating))           # NA value found in Row 6, and 10
```

```
## [1]  6 10
```

```
which(is.na(UBER$UberServices))          #NA value found in Row 6, 10 and 61
```

```
## [1] 61
```

```
which(is.na(UBER$DriverRating))          # NA value found in row 51
```

```
## [1] 51
```

```
which(is.na(UBER$CarType))               # NA value found in row 98
```

```
## [1] 98
```

```r
which(is.na(UBER$MaxPassengers))          # NA value found in row 16
```

```
## [1] 16
```

```r
#RiderRating is missing as it is continuous numeric data it can be replaced by the mean
UBER$RiderRating %<>% impute(UBER$RiderRating, fun = mean)


#UberServices - As this variable is categorical, we can apply the mode to replace NA value
UBER$UberServices %<>% impute(UBER$UberServices, fun = mode)

# Driver Rating - we can replace with mean like with the RiderRating
UBER$DriverRating %<>% impute(UBER$DriverRating, fun = mean)


# Car Type
UBER$CarType %<>% impute(UBER$CarType, fun = mode)

#Max Passenger - replace with median
UBER$MaxPassengers %<>% impute(UBER$MaxPassengers, fun = median)


sum(is.na(UBER)) # Nil NA values
```

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

Scan Numeric Variables for Outliers

Univariate Outlier Detection

We use Tukey's box plot method in each of our numeric values to detect outliers. Using the code below gives us the following results:

- **RiderRating** - we detect an outlier with a value of 2
- **TravelTime** - we detect an outlier with a value of 50
- **TripDistance** - we detect an outlier with a value of 45
- **BaseFare** - No out liers detected. Further analysis may be necessary.
- **DriverRating** - Outlier with value of 3 detected.
- **YearsWorked** - outlier with value of 18 detected.
- **MinuteRate** - no outliers detected, further analysis may be required.
- **RatePerKM** - No outliers detected, further analysis may be required.
- **MaxPassengers** - 22 outliers detected, further investigation required.
- **SurgeMultiplier** - 11 outliers detected. However, if we look at the output, we can see these are not real outliers. They are only detected as outliers, because the amount of rides with surge pricing compared to rides without, is imbalanced. Hence, no need to impute or delete values in SurgeMultiplier.

From the code below code, we can calculate the Inter-Quantile Range (IQR) for RiderRating, TravelTime, TripDistance, DriverRating and YearsWorked variables.

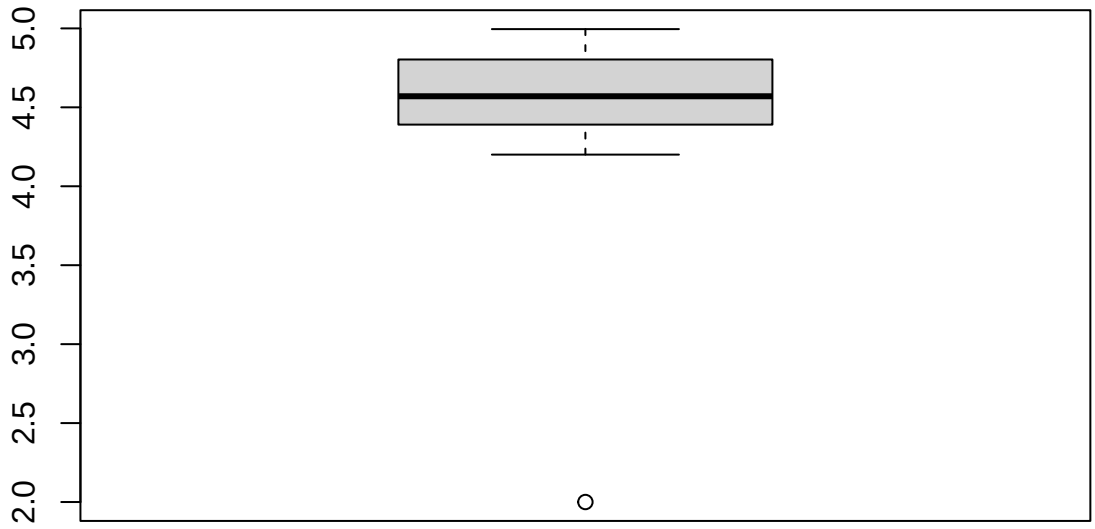
The Cap function below allows us to capture outliers which lie 1.5 times above the upperfence and 1.5 times the lower fence :

cap <- function(x){ quantiles <- quantile(x, c(0.05, 0.25, 0.75, 0.95) , na.rm = TRUE)* x[x < quantiles[2] - 1.5 * IQR(x, na.rm = TRUE)] <- quantiles[1]** x[x > quantiles[3] + 1.5 * IQR(x, na.rm = TRUE)] <- quantiles[4]* x}

The following code is then used to impute the mean or median values:

*RiderRating_cap <- as.data.frame(sapply(UBERRiderRating, FUN = cap))**summary(UBERRiderRating) summary(RiderRating_cap) #Impute with the Mean UBERRiderRating[RiderRatingup_outliers] <- −mean(UBERRiderRating) UBER$RiderRating*

```
###### OUTLIERS
#Univariate Outlier detection using box plots.

########################Rider Rating###################################
bp_RiderRating <- boxplot(UBER$RiderRating)
```




```
RiderRating_outliers <- bp_RiderRating$out
RiderRating_outliers
```

```
## [1] 2
```

```
# We find one outlier with a value of 2

# We Calculate the quartiles using the quantiles() function:
q1RiderRating <- quantile(UBER$RiderRating, probs = 0.25)
q3RiderRating <- quantile(UBER$RiderRating, probs = 0.75)
```

```r
iqrRiderRating <- q3RiderRating - q1RiderRating

# Once we have the IQR, we can calculate the upper and lower fence:
# Use any of the above methods to determine Q3 and Q1, and then:
RiderRatinglower_fence <- q1RiderRating - (1.5 * iqrRiderRating) # Recall that the lower fence is Q1 mi
RiderRatingupper_fence <- q3RiderRating + (1.5 * iqrRiderRating) # Recall that the upper fence is Q3 pl
RiderRatingup_outliers <- which(UBER$RiderRating > RiderRatingupper_fence)
RiderRatinglow_outliers <- which(UBER$RiderRating < RiderRatinglower_fence)
length(RiderRatingup_outliers)
```

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

```r
length(RiderRatinglow_outliers)
```

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

```r
RiderRatinglow_outliers
```

```
## [1] 33
```

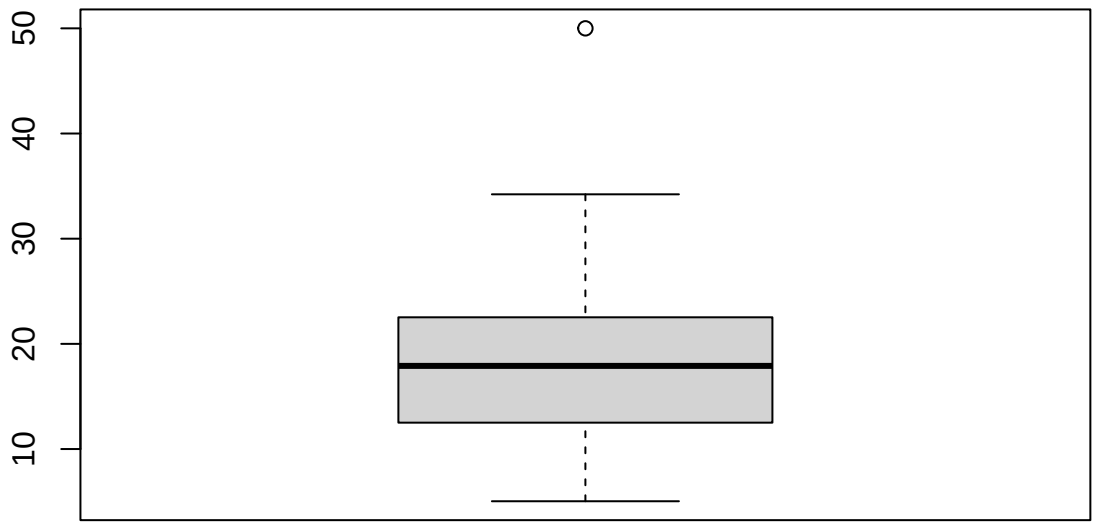
```r
RiderRatingup_outliers
```

```
## integer(0)
```

```r
UBER$RiderRating[RiderRatingup_outliers]
```

```
## numeric(0)
```

```r
##################Travel Time ###################################
bp_TravelTime <- boxplot(UBER$TravelTime)
```




```
TravelTime_outliers <- bp_TravelTime$out
TravelTime_outliers
```

```
## [1] 50
```

```
#we find one outlier with a value of 50

# We Calculate the quartiles using the quantiles() function:
q1TravelTime <- quantile(UBER$TravelTime, probs = 0.25)
q3TravelTime <- quantile(UBER$TravelTime, probs = 0.75)
iqrTravelTime <- q3TravelTime - q1TravelTime

# Once we have the IQR, we can calculate the upper and lower fence:
# Use any of the above methods to determine Q3 and Q1, and then:
TravelTimelower_fence <- q1TravelTime - (1.5 * iqrTravelTime) # Recall that the lower fence is Q1 minus
TravelTimeupper_fence <- q3TravelTime + (1.5 * iqrTravelTime) # Recall that the upper fence is Q3 plus
TravelTimeup_outliers <- which(UBER$TravelTime > TravelTimeupper_fence)
TravelTimelow_outliers <- which(UBER$TravelTime < TravelTimelower_fence)
length(TravelTime_outliers)
```

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

```
length(TravelTimelow_outliers)
```

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

```
TravelTimelow_outliers
```

```
## integer(0)
```

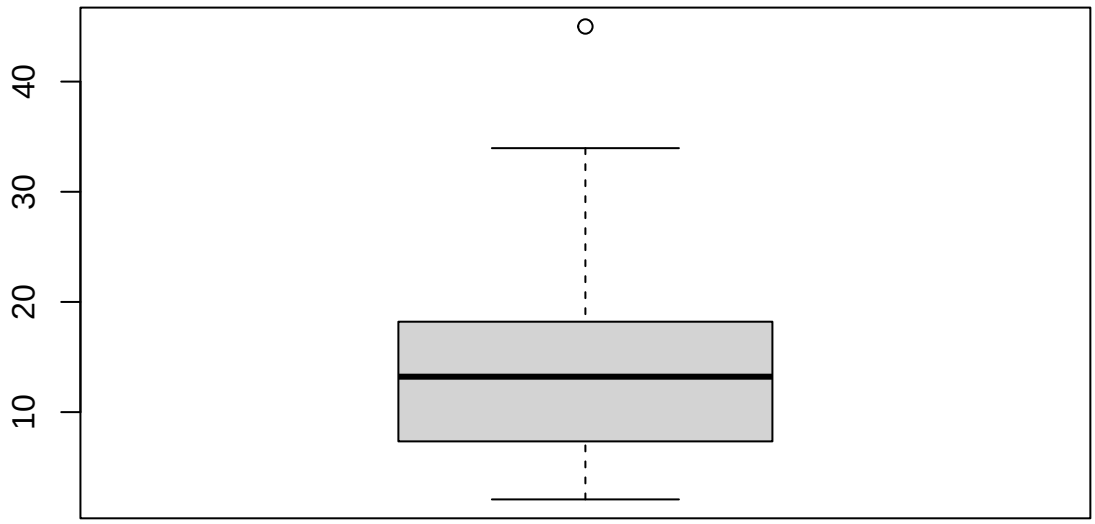
```
TravelTimeup_outliers
```

```
## [1] 43
```

```
UBER$TravelTime[TravelTimeup_outliers]
```

```
## [1] 50
```

```
################ Trip Distance ##################
bp_TripDistance <- boxplot(UBER$TripDistance)
```




```
TripDistance_outliers <- bp_TripDistance$out
TripDistance_outliers
```

```
## [1] 45
```

```r
#We find one outlier with a value of 45

# We Calculate the quartiles using the quantiles() function:
q1TripDistance <- quantile(UBER$TripDistance, probs = 0.25)
q3TripDistance <- quantile(UBER$TripDistance, probs = 0.75)
iqrTripDistance <- q3TripDistance - q1TripDistance

# Once we have the IQR, we can calculate the upper and lower fence:
# Use any of the above methods to determine Q3 and Q1, and then:
TripDistancelower_fence <- q1TripDistance - (1.5 * iqrTripDistance) # Recall that the lower fence is Q1
TripDistanceupper_fence <- q3TripDistance + (1.5 * iqrTripDistance) # Recall that the upper fence is Q3
TripDistanceup_outliers <- which(UBER$TripDistance > TripDistanceupper_fence)
TripDistancelow_outliers <- which(UBER$TripDistance < TripDistancelower_fence)
length(TripDistance_outliers) # There are none
```

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

```r
length(TripDistancelow_outliers) # Theres one
```

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

```r
TripDistancelow_outliers            #gives us location
```

```
## integer(0)
```

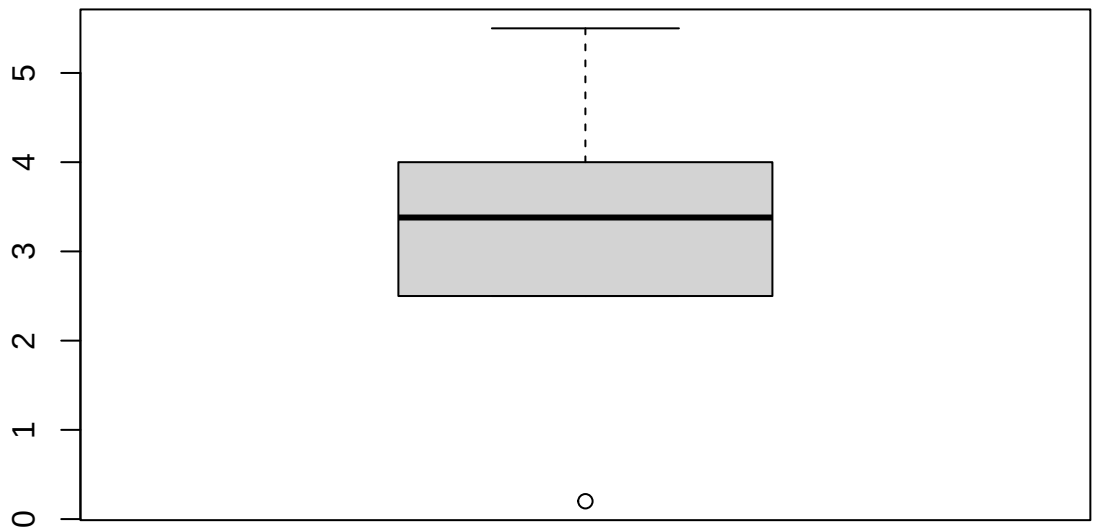
```r
TripDistanceup_outliers              #This gives the locations (observation numbers) of the outliers
```

```
## [1] 9
```

```r
UBER$TripDistance[TripDistanceup_outliers]
```

```
## [1] 45
```

```r
#################  #Base Fare   ###############
bp_BaseFare <- boxplot(UBER$BaseFare)
```




```
BaseFare_outliers <- bp_BaseFare$out
BaseFare_outliers
```

```
## [1] 0.2
```

```
#no out liers detected

# We Calculate the quartiles using the quantiles() function:
q1BaseFare <- quantile(UBER$BaseFare, probs = 0.25)
q3BaseFare <- quantile(UBER$BaseFare, probs = 0.75)
iqrBaseFare <- q3BaseFare - q1BaseFare

# Once we have the IQR, we can calculate the upper and lower fence:
# Use any of the above methods to determine Q3 and Q1, and then:
BaseFarelower_fence <- q1BaseFare - (1.5 * iqrBaseFare) # Recall that the lower fence is Q1 minus the i
BaseFareupper_fence <- q3BaseFare + (1.5 * iqrBaseFare) # Recall that the upper fence is Q3 plus the in
BaseFareup_outliers <- which(UBER$BaseFare > BaseFareupper_fence)
BaseFarelow_outliers <- which(UBER$BaseFare < BaseFarelower_fence)
length(BaseFare_outliers) # There are none
```

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

```
length(BaseFarelow_outliers) # Theres one
```

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

```
BaseFarelow_outliers             #gives us location
```

```
## [1] 86
```

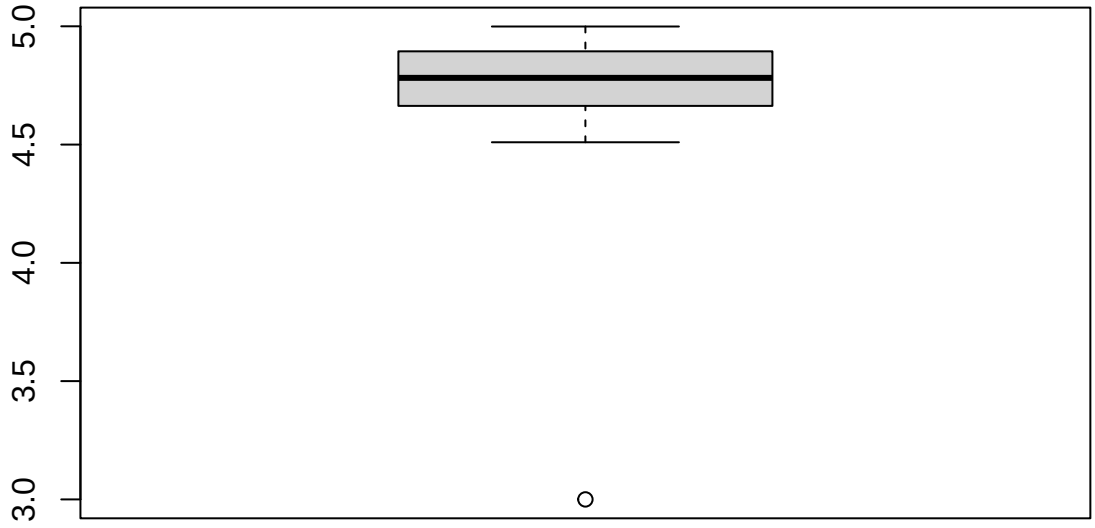
```
BaseFareup_outliers               #This gives the locations (observation numbers) of the outliers
```

```
## integer(0)
```

```
UBER$BaseFare[BaseFareup_outliers]
```

```
## numeric(0)
```

```
############## DriverRating
bp_DriverRating <- boxplot(UBER$DriverRating)
```




```
DriverRating_outliers <- bp_DriverRating$out
DriverRating_outliers
```

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

```r
# we detect one outlier of 3

# We Calculate the quartiles using the quantiles() function:
q1DriverRating <- quantile(UBER$DriverRating, probs = 0.25)
q3DriverRating <- quantile(UBER$DriverRating, probs = 0.75)
iqrDriverRating <- q3DriverRating - q1DriverRating

# Once we have the IQR, we can calculate the upper and lower fence:
# Use any of the above methods to determine Q3 and Q1, and then:
DriverRatinglower_fence <- q1DriverRating - (1.5 * iqrDriverRating) # Recall that the lower fence is Q1
DriverRatingupper_fence <- q3DriverRating + (1.5 * iqrDriverRating) # Recall that the upper fence is Q3
DriverRatingup_outliers <- which(UBER$BaseFare > DriverRatingupper_fence)
DriverRatinglow_outliers <- which(UBER$BaseFare < DriverRatinglower_fence)
length(DriverRating_outliers)
```

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

```r
length(DriverRatinglow_outliers)
```

```
## [1] 86
```

```r
DriverRatinglow_outliers
```

```
##  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
## [20]  20  21  22  23  24  26  27  29  30  31  32  33  34  35  36  37  38  40  43
## [39]  44  46  47  48  49  50  54  55  56  57  59  60  62  63  64  65  66  67  69
## [58]  70  71  72  73  74  75  76  77  78  79  80  82  83  84  86  87  88  89  90
## [77]  91  92  93  94  95  96  97  98  99 100
```

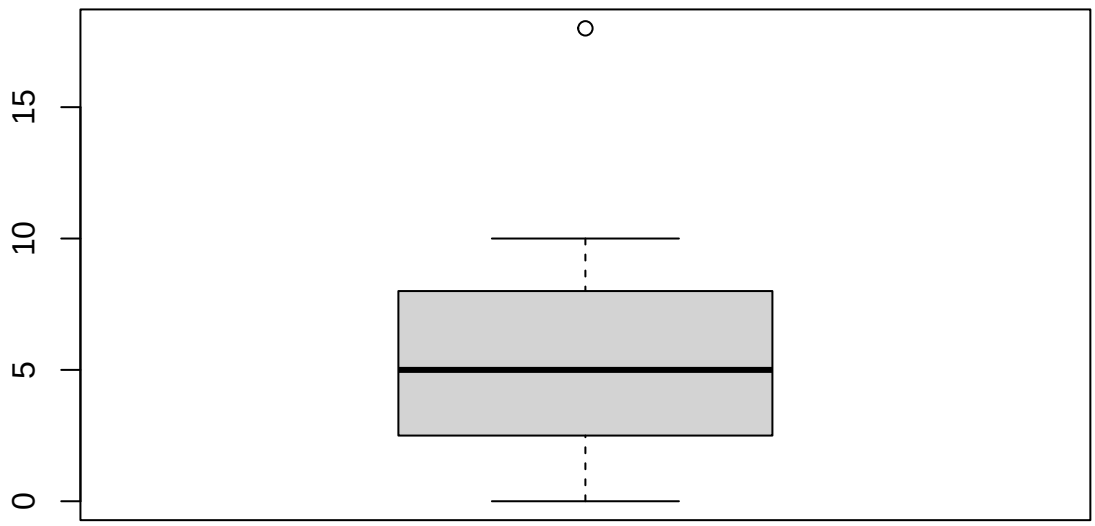
```r
DriverRatingup_outliers
```

```
##  [1] 25 28 39 41 42 45 51 52 53 58 61 68 81 85
```

```r
UBER$DriverRating[DriverRatingup_outliers]
```

```
##  [1] 4.753797  4.623381  4.922740  4.672311  4.513690  4.891965  4.757893*
##  [8] 4.881112  4.738868  4.813336  4.577104  4.810751  4.679728  4.708751
```

```r
###########     Years Worked
bp_YearsWorked <- boxplot(UBER$YearsWorked)
```




```
YearsWorked_outliers <- bp_YearsWorked$out
YearsWorked_outliers
```

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

```
#We find one outlier of 18

q1YearsWorked <- quantile(UBER$YearsWorked, probs = 0.25)
q3YearsWorked <- quantile(UBER$YearsWorked, probs = 0.75)
iqrYearsWorked <- q3YearsWorked - q1YearsWorked

# Once we have the IQR, we can calculate the upper and lower fence:
# Use any of the above methods to determine Q3 and Q1, and then:
YearsWorkedlower_fence <- q1YearsWorked - (1.5 * iqrYearsWorked) # Recall that the lower fence is Q1 mi
YearsWorkedupper_fence <- q3YearsWorked + (1.5 * iqrYearsWorked) # Recall that the upper fence is Q3 pl
YearsWorkedup_outliers <- which(UBER$YearsWorked > YearsWorkedupper_fence)
YearsWorkedlow_outliers <- which(UBER$YearsWorked < YearsWorkedlower_fence)
length(YearsWorked_outliers) # There are none
```

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

```
length(YearsWorkedlow_outliers) # Theres one
```

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

```
YearsWorkedlow_outliers             #gives us location
```

```
## integer(0)
```

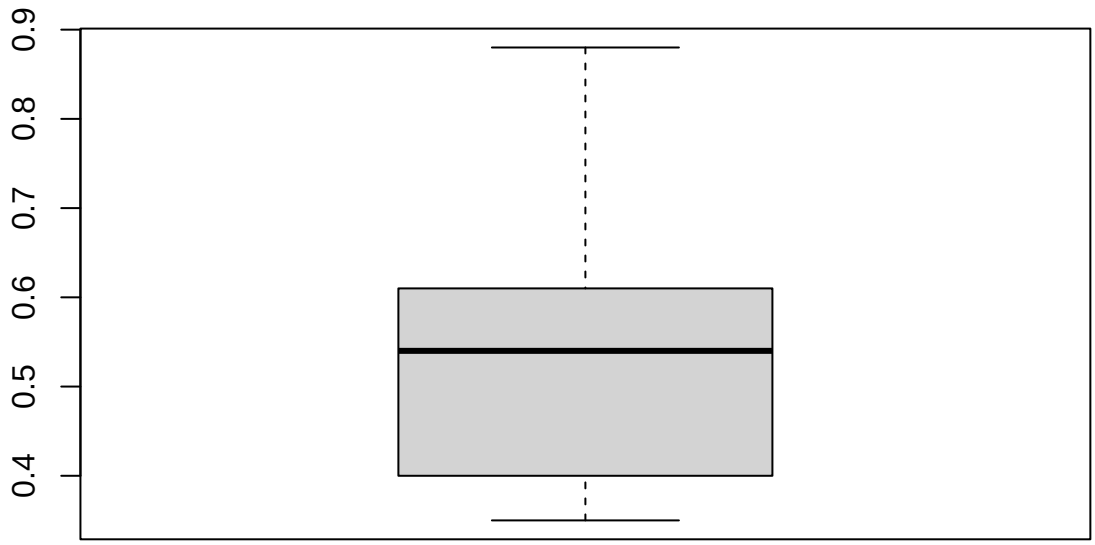
```
YearsWorkedup_outliers              #This gives the locations (observation numbers) of the outliers
```

```
## [1] 55
```

```
UBER$YearsWorked[YearsWorkedup_outliers]
```

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

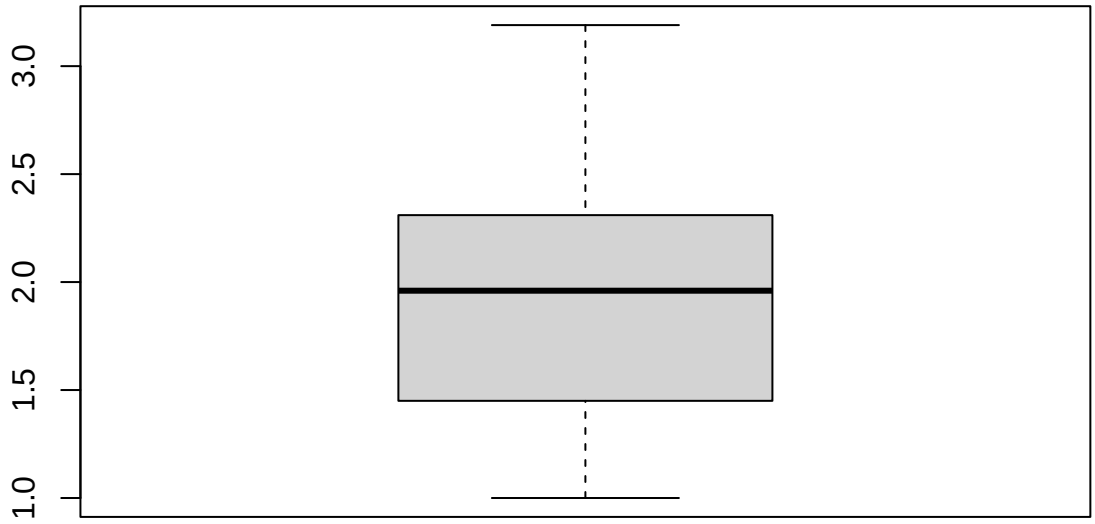
```
#Minute Rate
bp_MinuteRate <- boxplot(UBER$MinuteRate)
```




```
MinuteRate_outliers <- bp_MinuteRate$out
MinuteRate_outliers
```

```
## numeric(0)
```

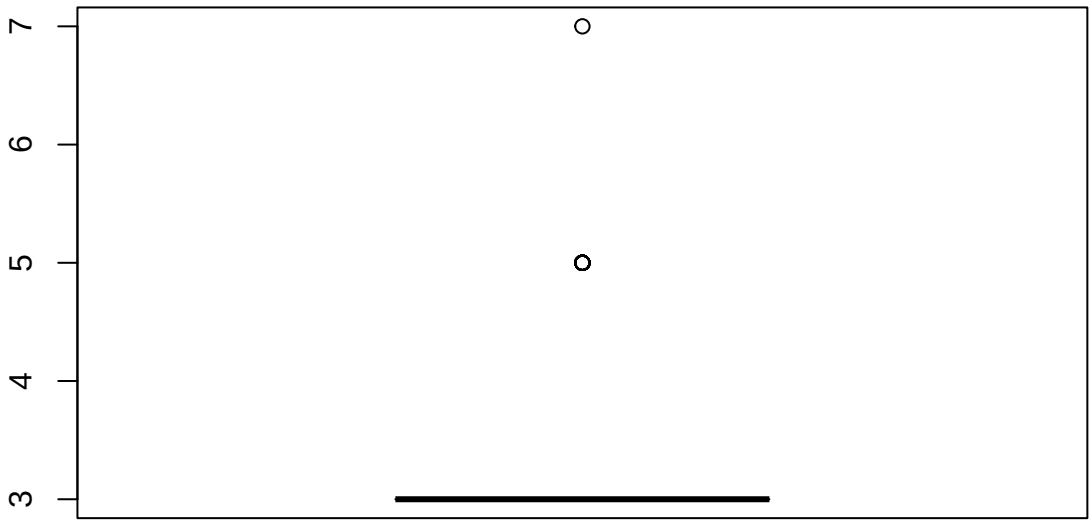
```
#We do not find any outliers present - more investigation may be needed


#RatePerKM
bp_RatePerKM <- boxplot(UBER$RatePerKM)
```




```
RatePerKM_outliers <- bp_RatePerKM$out
RatePerKM_outliers
```

```
## numeric(0)
```

```
#We do not find any outliers present - more investigation may be needed


#MaxPassengers
bp_MaxPassengers <- boxplot(UBER$MaxPassengers)
```

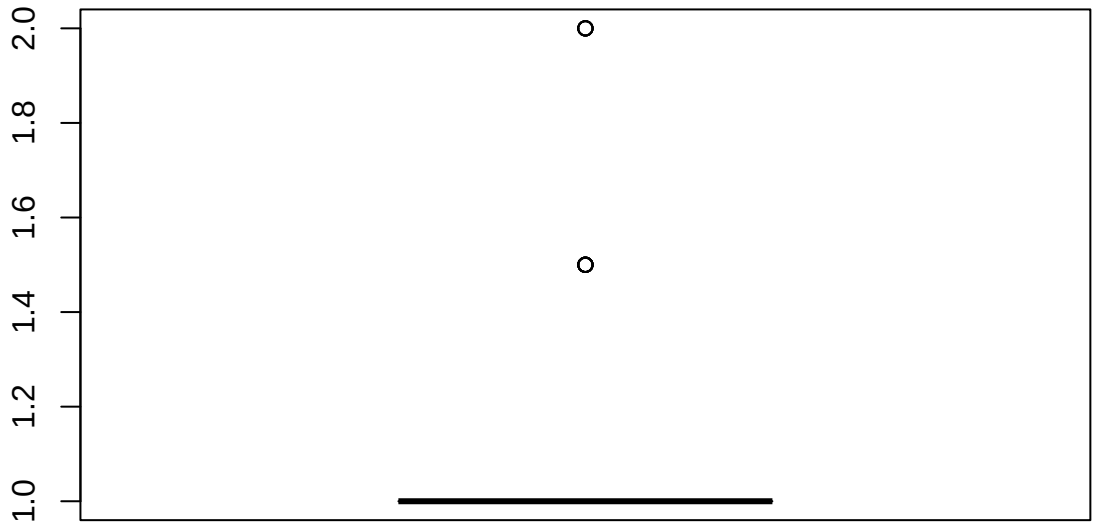



```
MaxPassengers_outliers <- bp_MaxPassengers$out
MaxPassengers_outliers
```

```
##  [1] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 7 5 5 5 5 5 5 5
```

```
# we find 22 values detected as outliers. more investigation needed.
```

```
################   Surge Multiplier   #################
bp_SurgeMultiplier <- boxplot(UBER$SurgeMultiplier)
```




```
SurgeMultiplier_outliers <- bp_SurgeMultiplier$out
SurgeMultiplier_outliers
```

```
##  [1] 2.0 1.5 1.5 2.0 2.0 2.0 1.5 1.5 1.5 1.5
```

```
#We find 11 outliers - more investigation is needed
```

```
#############################################################################################################
############               Impute outliers                ################


# Function
# This is a user-defined function, that will appear in our environment for our use
# It will cap outliers.

cap <- function(x){
  quantiles <- quantile( x, c(0.05, 0.25, 0.75, 0.95 ) , na.rm = TRUE)
  x[ x < quantiles[2] - 1.5 * IQR(x, na.rm = TRUE) ] <- quantiles[1]
  x[ x > quantiles[3] + 1.5 * IQR(x, na.rm = TRUE) ] <- quantiles[4]
```

```
  x
}
```

```
################## Rider Rating
RiderRating_cap <- as.data.frame(sapply(UBER$RiderRating, FUN = cap))

summary(UBER$RiderRating)
```

```
##
##  2 values imputed to 4.57007

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   4.393   4.570   4.570   4.803   4.995
```

```
summary(RiderRating_cap)
```

```
##  sapply(UBER$RiderRating, FUN = cap)
##  Min.   :2.000
##  1st Qu.:4.393
##  Median :4.570
##  Mean   :4.570
##  3rd Qu.:4.803
##  Max.   :4.995
```

```
#Impute with the Mean
UBER$RiderRating[RiderRatingup_outliers] <- mean(UBER$RiderRating)
UBER$RiderRating
```

```
##   [1] 4.703377  4.288908  4.911631  4.236445  4.314240  4.570070* 4.987966
##   [8] 4.725407  4.551865  4.570070* 4.394896  4.304557  4.802646  4.849912
##  [15] 4.609204  4.365225  4.547914  4.580253  4.965467  4.734444  4.499570
##  [22] 4.573423  4.916036  4.830557  4.752564  4.921839  4.851712  4.641148
##  [29] 4.219691  4.456299  4.675314  4.233648  2.000000  4.582237  4.454545
##  [36] 4.495076  4.236665  4.724565  4.311045  4.396870  4.373126  4.754243
##  [43] 4.553760  4.803580  4.431328  4.768146  4.906414  4.835474  4.732092
##  [50] 4.481438  4.565292  4.913935  4.430062  4.914441  4.886262  4.274876
##  [57] 4.530979  4.302025  4.297519  4.572770  4.839140  4.995416  4.462337
##  [64] 4.742057  4.321956  4.340042  4.690217  4.806768  4.350153  4.503853
##  [71] 4.558813  4.282340  4.622484  4.919860  4.376095  4.531637  4.658107
##  [78] 4.836374  4.648758  4.562667  4.635253  4.507176  4.275873  4.970419
##  [85] 4.848051  4.534117  4.317691  4.722482  4.386427  4.282292  4.825835
##  [92] 4.952374  4.200500  4.385301  4.909175  4.766824  4.963603  4.412778
##  [99] 4.419507  4.527182
```

```
################## Travel Time
TravelTime_cap <- as.data.frame(sapply(UBER$TravelTime, FUN = cap))

summary(UBER$TravelTime)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   5.037  12.585  17.906  18.415  22.443  50.000
```

```
summary(TravelTime_cap)
```

```
##  sapply(UBER$TravelTime, FUN = cap)
##  Min.   : 5.037
##  1st Qu.:12.585
##  Median :17.906
##  Mean   :18.415
##  3rd Qu.:22.443
##  Max.   :50.000
```

```
# Impute with the Mean
UBER$TravelTime[TravelTimeup_outliers] <- mean(UBER$TravelTime)


################ Trip Distance
TripDistance_cap <- as.data.frame(sapply(UBER$TripDistance, FUN = cap))

summary(UBER$TripDistance)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.078   7.388  13.216  13.806  18.202  45.000
```

```
summary(TripDistance_cap)
```

```
##  sapply(UBER$TripDistance, FUN = cap)
##  Min.   : 2.078
##  1st Qu.: 7.388
##  Median :13.216
##  Mean   :13.806
##  3rd Qu.:18.202
##  Max.   :45.000
```

```
# Impute with the Mean
UBER$TripDistance[TripDistanceup_outliers] <- mean(UBER$TripDistance)



############### Base Fare
BaseFare_cap <- as.data.frame(sapply(UBER$BaseFare, FUN = cap))

summary(UBER$BaseFare)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.200   2.500   3.380   3.388   4.000   5.500
```

```
summary(BaseFare_cap)
```

```
##  sapply(UBER$BaseFare, FUN = cap)
##  Min.   :0.200
##  1st Qu.:2.500
##  Median :3.380
##  Mean   :3.388
##  3rd Qu.:4.000
##  Max.   :5.500
```

```
# Impute with the Median as mean would not match with discrete levels
UBER$BaseFare[BaseFareup_outliers] <- median(UBER$BaseFare)
```

```
############## Driver Rating
DriverRating_cap <- as.data.frame(sapply(UBER$DriverRating, FUN = cap))

summary(UBER$DriverRating)
```

```
##
##  1 values imputed to 4.757893
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   4.665   4.782   4.758   4.893   4.999
```

```
summary(DriverRating_cap)
```

```
##  sapply(UBER$DriverRating, FUN = cap)
##  Min.   :3.000
##  1st Qu.:4.665
##  Median :4.782
##  Mean   :4.758
##  3rd Qu.:4.893
##  Max.   :4.999
```

```
# Impute with the Mean
UBER$DriverRating[DriverRatingup_outliers] <- mean(UBER$DriverRating)
```

```
############## Years Worked

YearsWorked_cap <- as.data.frame(sapply(UBER$YearsWorked, FUN = cap))

summary(UBER$YearsWorked)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    2.75    5.00    5.36    8.00   18.00
```

```
summary(YearsWorked_cap)
```

```
##  sapply(UBER$YearsWorked, FUN = cap)
##  Min.   : 0.00
##  1st Qu.: 2.75
##  Median : 5.00
##  Mean   : 5.36
##  3rd Qu.: 8.00
##  Max.   :18.00
```

```
# Impute with the Mean
UBER$YearsWorked[YearsWorkedup_outliers] <- median(UBER$YearsWorked)
```

```
# A boxplot is really just a list that gets plotted.
# It contains the statistics, which look suspiciously like our summary() output
# It contains the number of observations in the dataset, n
# It contains the confidence intervals around the mean
# It contains the outliers, "out"
# It contains any grouping variables (useful in multi-variate boxplots)

# Therefore, if we wanted to extract the outliers from the boxplot, we could extract
# them as we would any other element of a list:
```

Multvariate

In detecting Multivariate outliers, we attempt to calculate multivariate outliers, however, due to some of the variables showing 0 IQR, the function produces an error.

Some values, such as the basefare or rates used in calculating the total fare may be dependent on categorical data such as the type of service requested and can be detected using an alternative approach.

Using **groupby** function, if we group **BaseFare** by service type. In **BaseFare_stats** we can see the min value for UberX is $0.20, well below the normal base of $2.50.

UBER$BaseFare[UBER$UberServices == "UberX" & UBER$BaseFare != 2.50] <- 2.50

MinuteRate the same method works for MinuteRate, an outlier of $0.35 is detected for a requested UberX service, we can then replace this with the correct value using the groupby method.

RatePerKM Using the same approach an outlier with a value of 1 is detected as a min as part of the UberX service. We replace the outlier value with the correct value using group by approach.

Max Passengers Outlier of 7 passengers is detected for a 4-Door Hatch/ Sedan. We correct this outlier by replacing it with the correct value of 3.

```
################  Multivariate Outlier Detection ######################
#Use of MVN to detect Multivariate Outliers among numeric variables


## Below subset does not work as at least one column has IQR = 0
#UBER_set1 <- UBER %>% select(MaxPassengers, RatePerKM)
#head(UBER_set1)
#results <- UBER_set1 %>%
#  MVN::mvn(multivariateOutlierMethod = "quan",
#           showOutliers = TRUE)
```

```
## Below subset does not work as at least one column has IQR = 0
#UBER_set2 <- UBER %>% select(MinuteRate, TravelTime)
#head(UBER_set2)
#results <- UBER_set1 %>%
#  MVN::mvn(multivariateOutlierMethod = "quan",
#           showOutliers = TRUE)



## Below subset does not work as at least one column has IQR = 0
#UBER_set3 <- UBER %>% select(BaseFare, TripDistance, RatePerKM)
#head(UBER_set3)
#results <- UBER_set1 %>%
#  MVN::mvn(multivariateOutlierMethod = "quan",
#           showOutliers = TRUE)



## Below subset does not work as at least one column has IQR = 0
#UBER_set4 <- UBER %>% select(BaseFare, MaxPassengers)
#head(UBER_set4)
#results <- UBER_set1 %>%
#  MVN::mvn(multivariateOutlierMethod = "quan",
#           showOutliers = TRUE)



######    alternate method.


#Driver Rating by Uber Services
BaseFare_stats <- UBER %>%
group_by(UberServices) %>%
summarise(mean_BaseFare = mean(BaseFare, na.rm = TRUE),
          median_BaseFare = median(BaseFare, na.rm = TRUE),
          minimum_BaseFare = min(BaseFare, na.rm = TRUE),
          max_BaseFare = max(BaseFare, na.rm = TRUE),
          sd_BaseFare = sd(BaseFare, na.rm = TRUE),
          n = n()) %>%
  ungroup()



#Replace outliers for UberX service in BaseFare with correct Base amount.
UBER$BaseFare[UBER$UberServices == "UberX" & UBER$BaseFare != 2.50] <- 2.50


#MinuteRate by Uber Services
MinuteRate_stats <- UBER %>%
group_by(UberServices) %>%
```

```
summarise(mean_MinuteRate = mean(MinuteRate, na.rm = TRUE),
          median_MinuteRate = median(MinuteRate, na.rm = TRUE),
          minimum_MinuteRate = min(MinuteRate, na.rm = TRUE),
          max_MinuteRate = max(MinuteRate, na.rm = TRUE),
          sd_MinuteRate = sd(MinuteRate, na.rm = TRUE),
          n = n()) %>%
  ungroup()


#Replace outliers for UberX service in MinuteRate with correct Base amount.
UBER$MinuteRate[UBER$UberServices == "UberX" & UBER$MinuteRate != 0.4] <- 0.4


#RatePerKM by Uber Services
RatePerKM_stats <- UBER %>%
group_by(UberServices) %>%
summarise(mean_RatePerKM = mean(RatePerKM, na.rm = TRUE),
          median_RatePerKM = median(RatePerKM, na.rm = TRUE),
          minimum_RatePerKM = min(RatePerKM, na.rm = TRUE),
          max_RatePerKM = max(RatePerKM, na.rm = TRUE),
          sd_RatePerKM = sd(RatePerKM, na.rm = TRUE),
          n = n()) %>%
  ungroup()


#Replace outliers for UberX service in RatePerKM with correct Base amount.
UBER$RatePerKM[UBER$UberServices == "UberX" & UBER$RatePerKM != 1.45] <- 1.45


#Max Passengers by Car Type
MaxPassengers_stats <- UBER %>%
group_by(CarType) %>%
summarise(mean_MaxPassengers = mean(MaxPassengers, na.rm = TRUE),
          median_MaxPassengers = median(MaxPassengers, na.rm = TRUE),
          minimum_MaxPassengers = min(MaxPassengers, na.rm = TRUE),
          max_MaxPassengersM = max(MaxPassengers, na.rm = TRUE),
          sd_MaxPassengers = sd(MaxPassengers, na.rm = TRUE),
          n = n()) %>%
  ungroup()


#Replace outliers for UberX service in RatePerKM with correct Base amount.
UBER$MaxPassengers[UBER$CarType == "UberX" & UBER$MaxPassengers != 3] <- 3
```

Apply a Data Transformation to a Variable

Transformation of Trip Distance

The variable **TripDistance** is selected for further examination and gain a better understanding of its distribution. Looking at the Histogram, we can observe that the data distribution is skewed towards the right.

To reduce skewedness of the distribution, we apply a **log** tansformation. Observing the Histogram for the log10 distribution below, the transformed data looks skewed to the left.

Applying the **Square Root** transformation to the **TripDistance** data yields great results, we can observe a more symmetrical distribution in the Histogram for Squaroot transformation below.

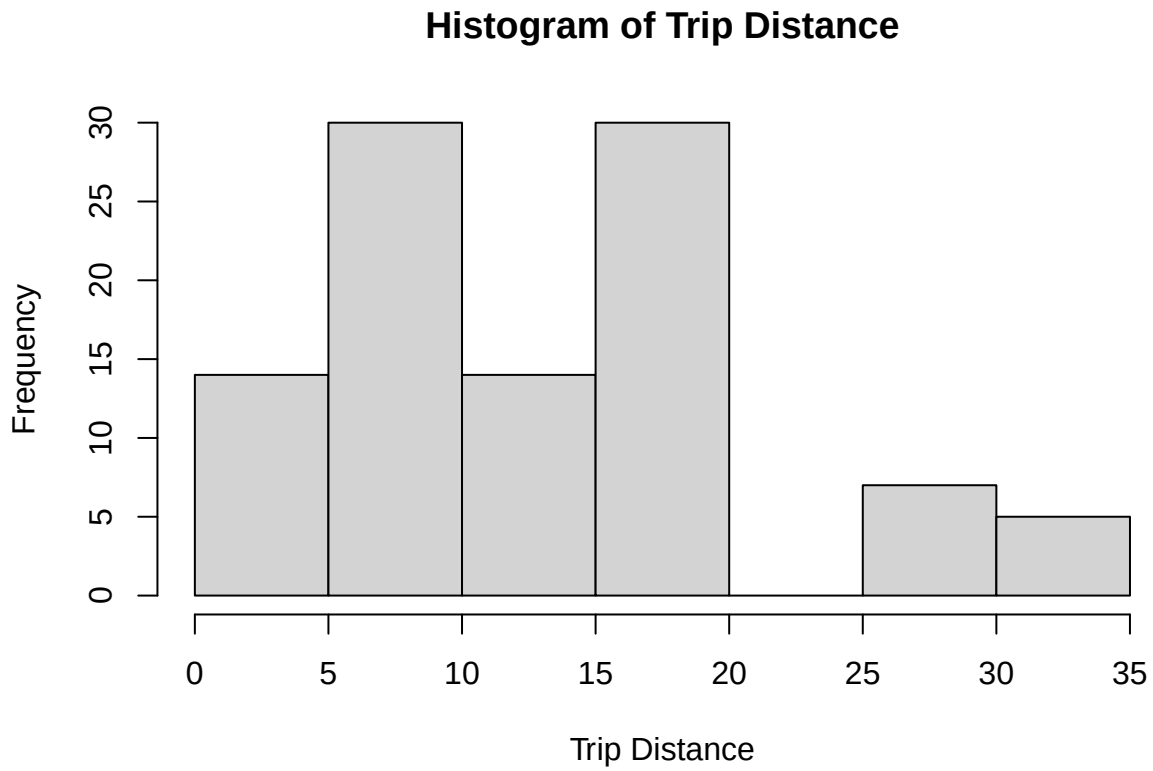
Fare Total

We have also chosen **FARETOTAL** for further examination. Examining the histogram for the column, below, we can observe it is skewed to the right.

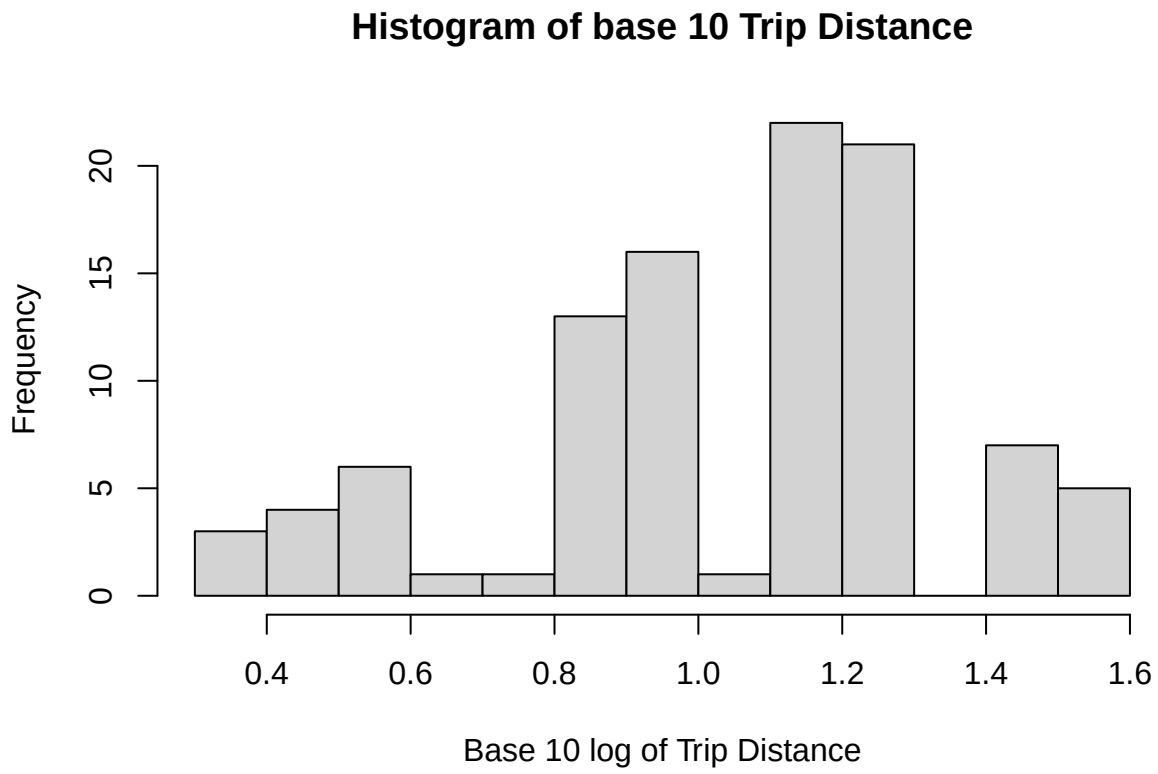
Applying the **SquareRoot** transformation, we can observe a more symmetrical distribution from the histogram below.

With **TripDistance** and **FARETOTAL** transformed into a symmetrical distribution. Transformation of data can aide with analysis by transforming complex non-linear relationships into linear relationships, they can be easier to work with. They also enable statistical analysis techniques like parametric tests and linear regression, which require data to be normally distributed.

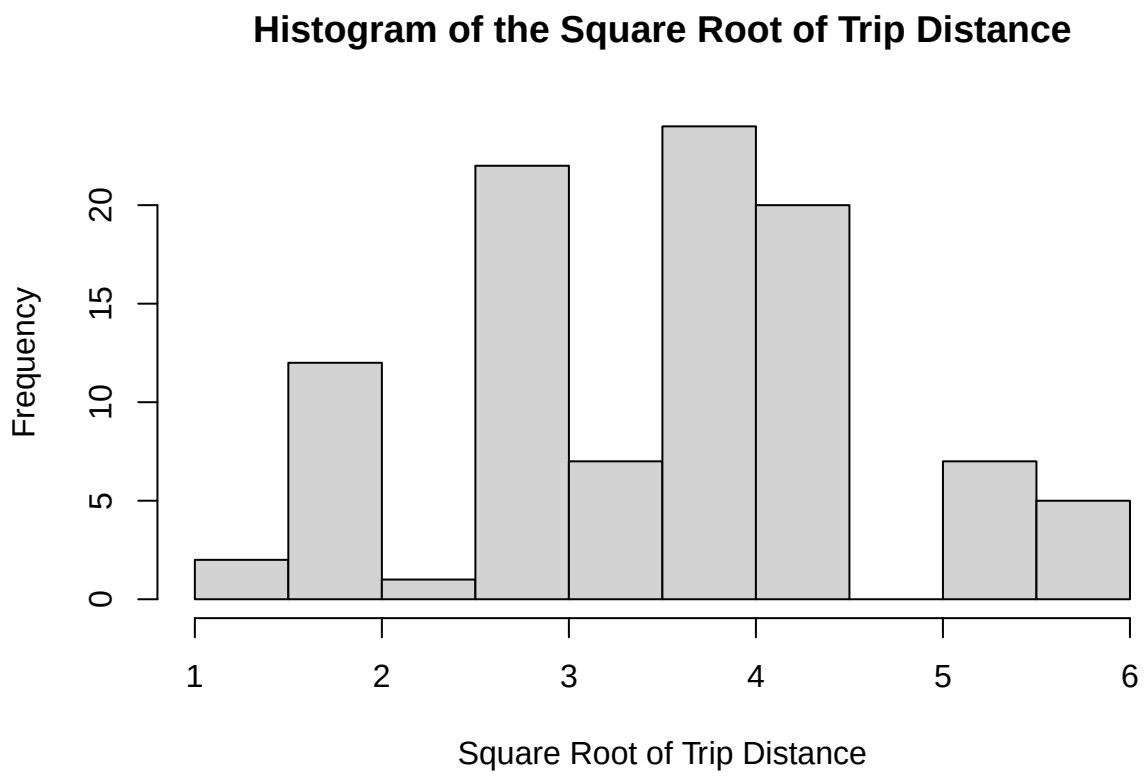
```
### Transformation

## Trip Distance


#Histogram of Trip Distance
hist(UBER$TripDistance, breaks = 10,
      main = "Histogram of Trip Distance",
      xlab = "Trip Distance")
```



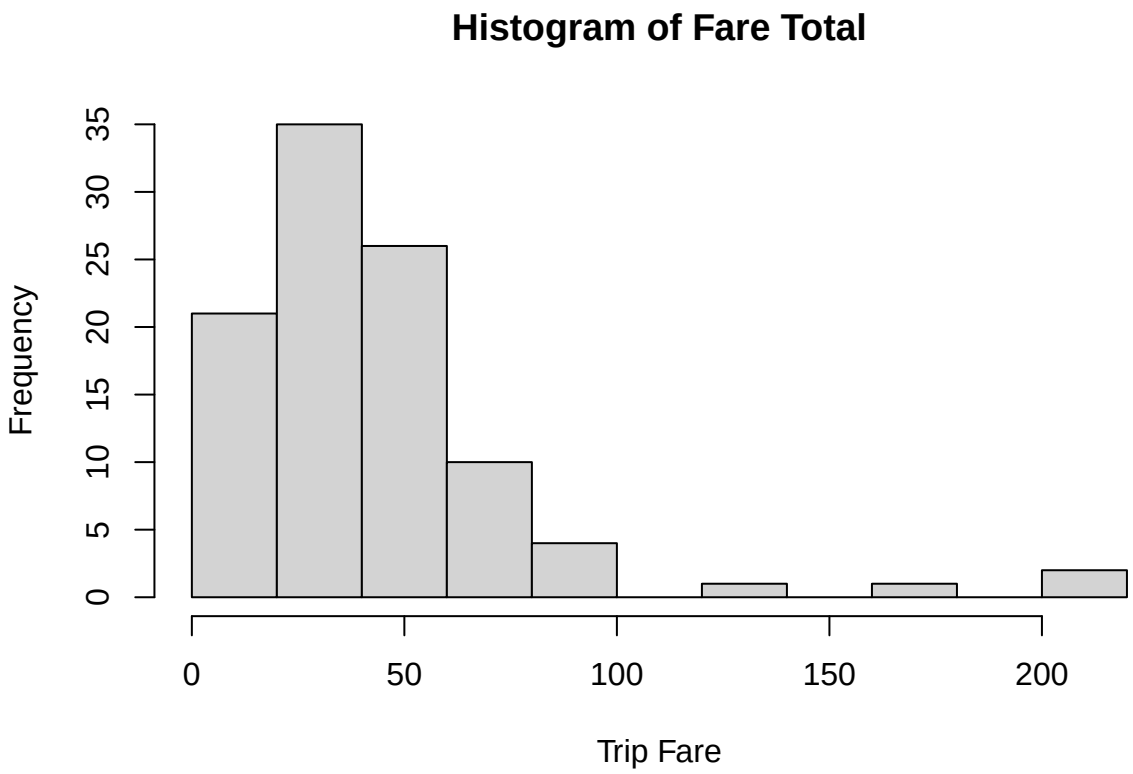
```r
# Log 10 Transformation
log_Distance <- log10(UBER$TripDistance)

# Histogram of Log10 Transformed Trip Distance
hist(log_Distance, breaks = 10,
     main = "Histogram of base 10 Trip Distance",
     xlab = "Base 10 log of Trip Distance")
```



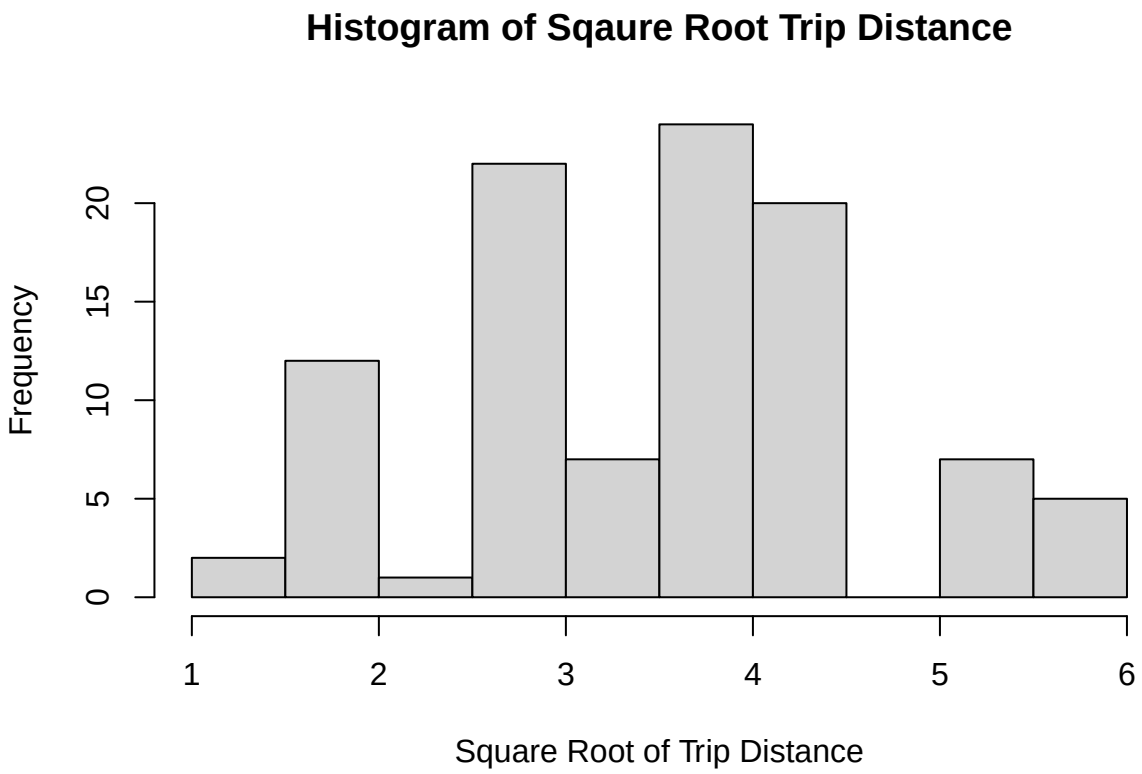

```r
# Sqare Root Transformation
sqrt_Distance <- sqrt(UBER$TripDistance)

# Histogram of SQRT Transformed Trip Distance
hist(sqrt_Distance, breaks = 10,
     main = "Histogram of the Square Root of Trip Distance",
     xlab = "Square Root of Trip Distance")
```




```
## Fare Total
#Histogram of Fare Total
hist(UBER$FARETOTAL,
      main = "Histogram of Fare Total",
      xlab = "Trip Fare")
```




```
# Sqare Root Transformation
sqrt_Fare <- sqrt(UBER$TripDistance)

# Histogram of Square Root Transformed Trip Distance
hist(sqrt_Fare,
     main = "Histogram of Sqaure Root Trip Distance",
     xlab = "Square Root of Trip Distance")
```



Summarise Statistics

Below, we create a number of summary statistics using the **groupby()**, **summarise()** and **pipes** functions in the code below allows us to calculate the Mean, Median, Minimum, Maximum and Standard Deviations.

From the code below we have Summary statistics for:

- Rider Rating by Uber Services
- Driver Rating by Uber Services
- Travel Time by Suburb Drop Off
- Travel Distance by Suburb Drop Off
- Fare Total By Suburb Drop Off

```
#Summarise Statistics

#Rider Rating by Uber Services
RiderRating_stats <- UBER %>%
group_by(UberServices) %>%
summarise(mean_RiderRating = mean(RiderRating, na.rm = TRUE),
          median_RiderRating = median(RiderRating, na.rm = TRUE),
          minimum_RiderRating = min(RiderRating, na.rm = TRUE),
          max_RiderRating = max(RiderRating, na.rm = TRUE),
          sd_RiderRating = sd(RiderRating, na.rm = TRUE),
          n = n()) %>%
  ungroup()
```

```r
#Driver Rating by Uber Services
DriverRating_stats <- UBER %>%
group_by(UberServices) %>%
summarise(mean_DriverRating_stats = mean(DriverRating, na.rm = TRUE),
          median_DriverRating_stats = median(DriverRating, na.rm = TRUE),
          minimum_DriverRating_stats = min(DriverRating, na.rm = TRUE),
          max_DriverRating_stats = max(DriverRating, na.rm = TRUE),
          sd_DriverRating_stats = sd(DriverRating, na.rm = TRUE),
          n = n()) %>%
  ungroup()

# Travel Time by Suburb Drop Off
TravelTime_stats <- UBER %>%
group_by(SuburbDropOff) %>%
summarise(mean_TravelTime = mean(TravelTime, na.rm = TRUE),
          median_TravelTime = median(TravelTime, na.rm = TRUE),
          minimum_TravelTime = min(TravelTime, na.rm = TRUE),
          max_TravelTime = max(TravelTime, na.rm = TRUE),
          sd_TravelTime = sd(TravelTime, na.rm = TRUE),
          n = n()) %>%
  ungroup()

# Travel Distance by Suburb Drop Off
TripDistance_stats <- UBER %>%
group_by(SuburbDropOff) %>%
summarise(mean_TripDistance = mean(TripDistance, na.rm = TRUE),
          median_TripDistance = median(TripDistance, na.rm = TRUE),
          minimum_TripDistance = min(TripDistance, na.rm = TRUE),
          max_TripDistance = max(TripDistance, na.rm = TRUE),
          sd_TripDistance = sd(TripDistance, na.rm = TRUE),
          n = n()) %>%
  ungroup()

# Fare Total By Suburb Drop Off
FARETOTAL_stats <- UBER %>%
group_by(SuburbDropOff) %>%
summarise(mean_FARETOTAL = mean(FARETOTAL, na.rm = TRUE),
          median_FARETOTAL = median(FARETOTAL, na.rm = TRUE),
          minimum_FARETOTAL = min(FARETOTAL, na.rm = TRUE),
          max_FARETOTAL = max(FARETOTAL, na.rm = TRUE),
          sd_FARETOTAL = sd(FARETOTAL, na.rm = TRUE),
          n = n()) %>%
  ungroup()
```

Write Data Frames to xlsx file

from http://www.sthda.com/english/wiki/writing-data-from-r-to-excel-files-xls-xlsx

```r
#(STHDA, 2022)
write.xlsx(cust_df, file = "customerdata.xlsx", sheetName = "Sheet1",
  colNames = TRUE, rowNames = TRUE, append = FALSE)
```

```
write.xlsx(Drive_df, file = "Driverdata.xlsx", sheetName = "Sheet1",
  colNames = TRUE, rowNames = TRUE, append = FALSE)

write.xlsx(UBER, file = "UBERDATA.xlsx", sheetName = "Sheet1",
  colNames = TRUE, rowNames = TRUE, append = FALSE)
```

References

Uber.com 2022, *UBER Uber One Member Benefits and Invite-Only Perks for Rides, Deliveries, Groceries and More.*, UBER, viewed 4 April 2022, https://www.uber.com/us/en/u/uber-one/

Uber.com 2022, *Always The Ride You Want*, UBER, viewed 4 April 2022, https://www.uber.com/au/en/ride/

Caracal, Stackexhange.com, 2011, *How To Generate Random Categorical Data?* Stackexchange.com, Viewed 3 April, 2022, https://stats.stackexchange.com/questions/14158/how-to-generate-random-categorical-data

AEF, Stackoverflow.com, 2017, *Generate Random Times in Sample of POSIXct*, Stackoverflow.com, viewed 3 April 2022, https://stackoverflow.com/questions/45633452/generate-random-times-in-sample-of-posixct

google.com 2022, *Google Maps*, google.com, viewed 4 April 2022, https://www.google.com/maps

Uber.com 2022, *Making The Most of Your Time on the Road*, UBER, viewed 4 April 2022, https://www.uber.com/en-AU/blog/making-the-most-of-your-time-on-the-road/

Yip, S 2021, *Uber Sydney: Fare Estimates, Vehicle Types and Airport Pick-up Points, finder.com.au, viewed 4 April 2022,
https://www.finder.com.au/uber-sydney-fares-services

Paul, K 2019, *Uber to Ban Riders with Low Ratings: Will you Pass the Test.* The Guardian, viewed 5 April 2022, https://www.theguardian.com/technology/2019/may/31/uber-to-ban-riders-with-low-ratings#:~:text=Drivers%20on%20the%20platform%2C%20for,to%20stay%20on%20the%20app.

Uber.com 2022, *Sydney Vehicle Requirements*, UBER, viewed 5 April 2022, https://www.uber.com/au/en/drive/sydney/vehicle-requirements/

STHDA.com 2022, *Writing Data From R to Excel Files (XLS/XLSX)*, STHDA.com, viewed 10/04/2022, http://www.sthda.com/english/wiki/writing-data-from-r-to-excel-files-xls-xlsx

Garrett Grolemund, Hadley Wickham (2011). Dates and Times Made Easy with lubridate. Journal of Statistical Software, 40(3), 1-25. URL https://www.jstatsoft.org/v40/i03/.

Stefan Milton Bache and Hadley Wickham (2022). magrittr: A Forward-Pipe Operator for R. R package version 2.0.2. https://CRAN.R-project.org/package=magrittr

Hadley Wickham, Romain François, Lionel Henry and Kirill Müller (2022). dplyr: A Grammar of Data Manipulation. R package version 1.0.8. https://CRAN.R-project.org/package=dplyr

Hadley Wickham and Maximilian Girlich (2022). tidyr: Tidy Messy Data. R package version 1.2.0. https://CRAN.R-project.org/package=tidyr

Lukasz Komsta (2022). outliers: Tests for Outliers. R package version 0.15. https://CRAN.R-project.org/package=outliers

Wickham et al., (2019). Welcome to the tidyverse. Journal of Open Source Software, 4(43), 1686, https://doi.org/10.21105/joss.01686

Mark van der Loo, Edwin de Jonge and Sander Scholtus (2015). deducorrect: Deductive Correction, Deductive Imputation, and Deterministic Correction. R package version 1.3.7. https://CRAN.R-project.org/package=deducorrect

Mark van der Loo and Edwin de Jonge (2021). deductive: Data Correction and Imputation Using Deductive Methods. R package version 1.0.0. https://CRAN.R-project.org/package=deductive

Mark P. J. van der Loo, Edwin de Jonge (2021). Data Validation Infrastructure for R. Journal of Statistical Software, 97(10), 1-31. doi:10.18637/jss.v097.i10

Frank E Harrell Jr (2021). Hmisc: Harrell Miscellaneous. R package version 4.6-0. https://CRAN.R-project.org/package=Hmisc

Korkmaz S, Goksuluk D, Zararsiz G. MVN: An R Package for Assessing Multivariate Normality. The R Journal. 2014 6(2):151-162.

Hadley Wickham, Jim Hester and Jennifer Bryan (2022). readr: Read Rectangular Text Data. R package version 2.1.2. https://CRAN.R-project.org/package=readr

Philipp Schauberger and Alexander Walker (2021). openxlsx: Read, Write and Edit xlsx Files. R package version 4.2.5. https://CRAN.R-project.org/package=openxlsx

Adrian Dragulescu and Cole Arendt (2020). xlsx: Read, Write, Format Excel 2007 and Excel 97/2000/XP/2003 Files. R package version 0.6.5. https://CRAN.R-project.org/package=xlsx

Yihui Xie (2022). tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents. R package version 0.38.

Yihui Xie (2019) TinyTeX: A lightweight, cross-platform, and easy-to-maintain LaTeX distribution based on TeX Live. TUGboat 40 (1): 30–32. https://tug.org/TUGboat/Contents/contents40-1.html

IMPORTANT NOTE

The report must be uploaded to Assignment 1 section in Canvas as a **PDF** document with R codes and outputs showing. The easiest way to do this is to: 1) Run all R chunks 2) **Preview** your notebook in **HTML** (by clicking Preview Notebook) 3) **Open in Browser (Chrome)** 4) **Right Click on the report in Chrome** 5) **Click Print** and Select the Destination Option to **Save as PDF**. 6) Now upload this PDF report as one single file via the Assignment 1 page in Canvas. **Remember to DELETE the instructional text provided in the template. Failure to do this will INCREASE the SIMILARITY INDEX reported in TURNITIN**

If you have any questions regarding the assignment instructions or the R Markdown template, please post them on the discussion board.