Regression

Name: Ryan Donaldson

Date: 02/13/2023

Summary

Linear regression is a statistical method used to find a relationship between predictor values, x, and target values y. The slope of the line and the intercept quantifies the amount that the target values change with respect to the predictor values. In other words, the line of best fit which will show an overall trend in the dataset and make accurate predictions. This can be interpreted as one strength of linear regression as the line of best fit can make the trend easy to interpret. It can also utilize multiple independent, or predictor, variables. However, linear regression does have its drawbacks in that any outliers can make it more difficult to understand the relationship along with variables that might not necessarily have a linear relationship.

Please click here (https://www.kaggle.com/datasets/yasserh/uber-fares-dataset) to access the dataset used in this project. We will use linear regression to predict the fare price of an Uber ride based on the pickup and dropoff location.

Data Cleaning

Before exploring and visualizing data, let's clean our dataset to make our overall linear regression model easier to work with as well as compiling informative graphs later on.

First, we will read the CSV and load the geosphere package so we can better work with latitude and longitude data that the columns will provide. We will also load the dplyr package to make column and row operations and mutations easier.

```
Uber <- read.csv("uber.csv", na.strings="NA", header=TRUE)
if(!require("geosphere")) {
  install.packages("geosphere")
  library("geosphere")
}
```

```
## Loading required package: geosphere
```

```
if(!require("dplyr")) {
  install.packages("dplyr")
  library("dplyr", warn.conflicts=FALSE)
}
```

```
## Loading required package: dplyr
```

```
##
## 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
```

Let's remove any rows which contain any latitude or longitude outliers.

```
exclude_pickup_lon <- filter(Uber, pickup_longitude < -360
         | pickup_longitude > 360
         | pickup_longitude < -180
         | pickup_longitude > 180)
Uber <- anti_join(Uber, exclude_pickup_lon, by="key")

exclude_dropoff_lon <- filter(Uber, dropoff_longitude < -360
         | dropoff_longitude > 360
         | dropoff_longitude < -180
         | dropoff_longitude > 180)
Uber <- anti_join(Uber, exclude_dropoff_lon, by="key")

exclude_pickup_lat <- filter(Uber, pickup_latitude < -90
         | pickup_latitude > 90)
Uber <- anti_join(Uber, exclude_pickup_lat, by="key")

exclude_dropoff_lat <- filter(Uber, dropoff_latitude < -90
         | dropoff_latitude > 90)
Uber <- anti_join(Uber, exclude_dropoff_lat, by="key")
```

Now, let's mutate the data frame and add a new Distance column. Each row will have a value representing the shortest distance between its corresponding pickup and dropoff points. We will use the distHaversine function from geosphere for the math behind this. The `Distance_In_Km` will be represented in kilometers from the function output.

```
print("Adding Distance column, please wait...")
```

```
## [1] "Adding Distance column, please wait..."
```

```
Uber <- Uber %>% rowwise() %>%
  mutate(Distance_In_Km=distHaversine(c(pickup_longitude, pickup_latitude), c(dropoff_lo
ngitude, dropoff_latitude)))
print("Uber dataframe has been mutated")
```

```
## [1] "Uber dataframe has been mutated"
```

Now, we will divide the data into train and test sets using a 80/20 split.

```
set.seed(1234)
i <- sample(1:nrow(Uber), nrow(Uber)*0.80,
replace=FALSE)
train <- Uber[i,]
test <- Uber[-i,]
```

Data Exploration

Next, we will run 5 R functions for data exploration of the data set using the training data. First, let's run the str() function to get a look into the format of the data.

```
str(train)
```

```
## rowws_df [159,990 × 10] (S3: rowwise_df/tbl_df/tbl/data.frame)
##  $ X                : int [1:159990] 20105724 18896163 40414748 11607434 34954013 290
82183 27715186 29305241 29942390 22199820 ...
##  $ key              : chr [1:159990] "2010-03-04 08:53:30.0000006" "2014-10-12 03:30:
41.0000002" "2010-07-16 07:57:00.00000030" "2013-09-14 09:27:00.000000131" ...
##  $ fare_amount      : num [1:159990] 9.7 17 34.3 7.5 13.5 ...
##  $ pickup_datetime  : chr [1:159990] "2010-03-04 08:53:30 UTC" "2014-10-12 03:30:41 U
TC" "2010-07-16 07:57:00 UTC" "2013-09-14 09:27:00 UTC" ...
##  $ pickup_longitude : num [1:159990] -73.8 -74 -73.9 -74 -74 ...
##  $ pickup_latitude  : num [1:159990] 40.8 40.7 40.8 40.7 40.8 ...
##  $ dropoff_longitude: num [1:159990] -73.8 -73.9 -74 -74 -74 ...
##  $ dropoff_latitude : num [1:159990] 40.8 40.8 40.8 40.7 40.8 ...
##  $ passenger_count  : int [1:159990] 1 1 1 1 1 3 1 1 1 1 ...
##  $ Distance_In_Km   : num [1:159990] 3048 6429 9321 1853 5147 ...
##  - attr(*, "groups")= tibble [159,990 × 1] (S3: tbl_df/tbl/data.frame)
##   ..$ .rows: list<int> [1:159990]
##   .. ..$ : int 1
##   .. ..$ : int 2
##   .. ..$ : int 3
##   .. ..$ : int 4
##   .. ..$ : int 5
##   .. ..$ : int 6
##   .. ..$ : int 7
##   .. ..$ : int 8
##   .. ..$ : int 9
##   .. ..$ : int 10
##   .. ..$ : int 11
##   .. ..$ : int 12
##   .. ..$ : int 13
##   .. ..$ : int 14
##   .. ..$ : int 15
##   .. ..$ : int 16
##   .. ..$ : int 17
##   .. ..$ : int 18
##   .. ..$ : int 19
##   .. ..$ : int 20
##   .. ..$ : int 21
##   .. ..$ : int 22
##   .. ..$ : int 23
##   .. ..$ : int 24
##   .. ..$ : int 25
##   .. ..$ : int 26
##   .. ..$ : int 27
##   .. ..$ : int 28
##   .. ..$ : int 29
##   .. ..$ : int 30
##   .. ..$ : int 31
##   .. ..$ : int 32
##   .. ..$ : int 33
##   .. ..$ : int 34
##   .. ..$ : int 35
##   .. ..$ : int 36
```

```
##   .. ..$ : int 37
##   .. ..$ : int 38
##   .. ..$ : int 39
##   .. ..$ : int 40
##   .. ..$ : int 41
##   .. ..$ : int 42
##   .. ..$ : int 43
##   .. ..$ : int 44
##   .. ..$ : int 45
##   .. ..$ : int 46
##   .. ..$ : int 47
##   .. ..$ : int 48
##   .. ..$ : int 49
##   .. ..$ : int 50
##   .. ..$ : int 51
##   .. ..$ : int 52
##   .. ..$ : int 53
##   .. ..$ : int 54
##   .. ..$ : int 55
##   .. ..$ : int 56
##   .. ..$ : int 57
##   .. ..$ : int 58
##   .. ..$ : int 59
##   .. ..$ : int 60
##   .. ..$ : int 61
##   .. ..$ : int 62
##   .. ..$ : int 63
##   .. ..$ : int 64
##   .. ..$ : int 65
##   .. ..$ : int 66
##   .. ..$ : int 67
##   .. ..$ : int 68
##   .. ..$ : int 69
##   .. ..$ : int 70
##   .. ..$ : int 71
##   .. ..$ : int 72
##   .. ..$ : int 73
##   .. ..$ : int 74
##   .. ..$ : int 75
##   .. ..$ : int 76
##   .. ..$ : int 77
##   .. ..$ : int 78
##   .. ..$ : int 79
##   .. ..$ : int 80
##   .. ..$ : int 81
##   .. ..$ : int 82
##   .. ..$ : int 83
##   .. ..$ : int 84
##   .. ..$ : int 85
##   .. ..$ : int 86
##   .. ..$ : int 87
##   .. ..$ : int 88
```

```
##   .. ..$ : int 89
##   .. ..$ : int 90
##   .. ..$ : int 91
##   .. ..$ : int 92
##   .. ..$ : int 93
##   .. ..$ : int 94
##   .. ..$ : int 95
##   .. ..$ : int 96
##   .. ..$ : int 97
##   .. ..$ : int 98
##   .. ..$ : int 99
##   .. .. [list output truncated]
##   .. ..@ ptype: int(0)
```

Next, let's gather an overall basic summary of each column of our training data.

```
summary(train)
```

```
##        X                  key             fare_amount     pickup_datetime
##  Min.   :       1   Length:159990      Min.   :-52.00   Length:159990
##  1st Qu.:13886062   Class :character   1st Qu.:  6.00   Class :character
##  Median :27763618   Mode  :character   Median :  8.50   Mode  :character
##  Mean   :27749116                      Mean   : 11.37
##  3rd Qu.:41600315                      3rd Qu.: 12.50
##  Max.   :55423567                      Max.   :499.00
##
##  pickup_longitude pickup_latitude  dropoff_longitude dropoff_latitude
##  Min.   :-93.82   Min.   :-74.01   Min.   :-75.42    Min.   :-74.02
##  1st Qu.:-73.99   1st Qu.: 40.73   1st Qu.:-73.99    1st Qu.: 40.73
##  Median :-73.98   Median : 40.75   Median :-73.98    Median : 40.75
##  Mean   :-72.51   Mean   : 39.93   Mean   :-72.52    Mean   : 39.93
##  3rd Qu.:-73.97   3rd Qu.: 40.77   3rd Qu.:-73.96    3rd Qu.: 40.77
##  Max.   : 40.81   Max.   : 48.02   Max.   : 40.83    Max.   : 45.03
##                                    NA's   :1         NA's   :1
##  passenger_count   Distance_In_Km
##  Min.   :  0.000   Min.   :      0
##  1st Qu.:  1.000   1st Qu.:   1216
##  Median :  1.000   Median :   2127
##  Mean   :  1.683   Mean   :  20083
##  3rd Qu.:  2.000   3rd Qu.:   3886
##  Max.   :208.000   Max.   :8792737
##                    NA's   :1
```

Our summary above tells us that exploring missing data is really not necessary considering the `dropoff_longitude` and `dropoff_latitude` columns each only contain 1 NA. So, first let's just look at the first few rows.

```
head(train)
```

X	key	fare_amount	pickup_datetime	picku
<int>	<chr>	<dbl>	<chr>	
20105724	2010-03-04 08:53:30.0000006	9.70	2010-03-04 08:53:30 UTC	
18896163	2014-10-12 03:30:41.0000002	17.00	2014-10-12 03:30:41 UTC	
40414748	2010-07-16 07:57:00.00000030	34.27	2010-07-16 07:57:00 UTC	
11607434	2013-09-14 09:27:00.000000131	7.50	2013-09-14 09:27:00 UTC	
34954013	2014-06-20 06:19:25.0000003	13.50	2014-06-20 06:19:25 UTC	
29082183	2011-03-27 17:09:00.000000162	49.80	2011-03-27 17:09:00 UTC	

6 rows | 1-5 of 10 columns

Now, let's look at the last few rows of our training data.

```
tail(train)
```

X	key	fare_amount	pickup_datetime	picku
<int>	<chr>	<dbl>	<chr>	
55251256	2014-08-20 23:51:55.0000002	32.83	2014-08-20 23:51:55 UTC	
9135184	2012-07-09 08:28:31.0000001	4.10	2012-07-09 08:28:31 UTC	
38983892	2010-06-25 12:27:48.0000004	4.50	2010-06-25 12:27:48 UTC	
17122458	2011-04-26 01:03:03.0000001	8.50	2011-04-26 01:03:03 UTC	
28892833	2012-06-06 15:12:44.0000001	31.30	2012-06-06 15:12:44 UTC	
37672342	2012-05-09 21:34:00.000000118	9.30	2012-05-09 21:34:00 UTC	

6 rows | 1-5 of 10 columns

Since we're dealing with over 200,000 records across 9 columns. Let's also explore the column names within our data set in case we need to reference them later when making predictions.

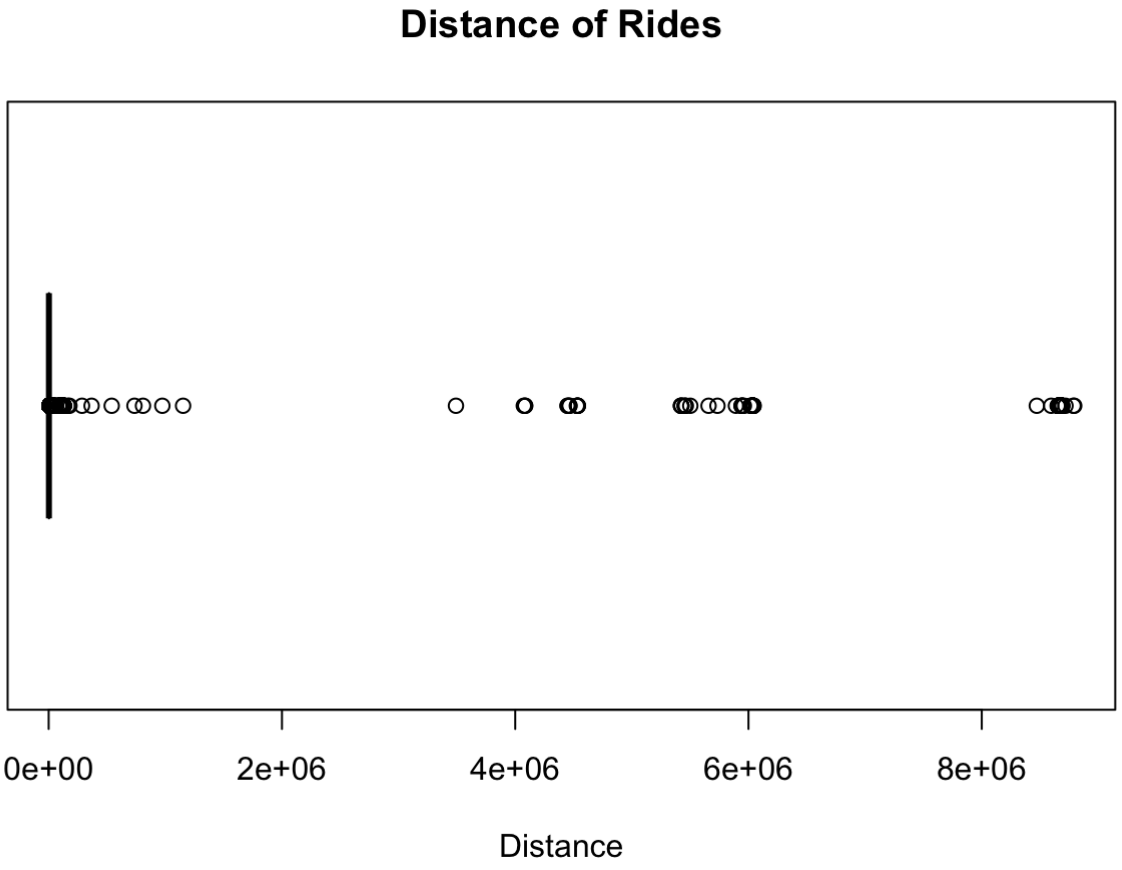
```
names(train)
```

```
##  [1] "X"                "key"              "fare_amount"
##  [4] "pickup_datetime"  "pickup_longitude" "pickup_latitude"
##  [7] "dropoff_longitude" "dropoff_latitude" "passenger_count"
## [10] "Distance_In_Km"
```

Data Visualization

Let's create some informative graphs based on this training data, particuarly distributions of various columns since we're working with such a large dataset. First, let's get a sense of the distribution of ride distances.

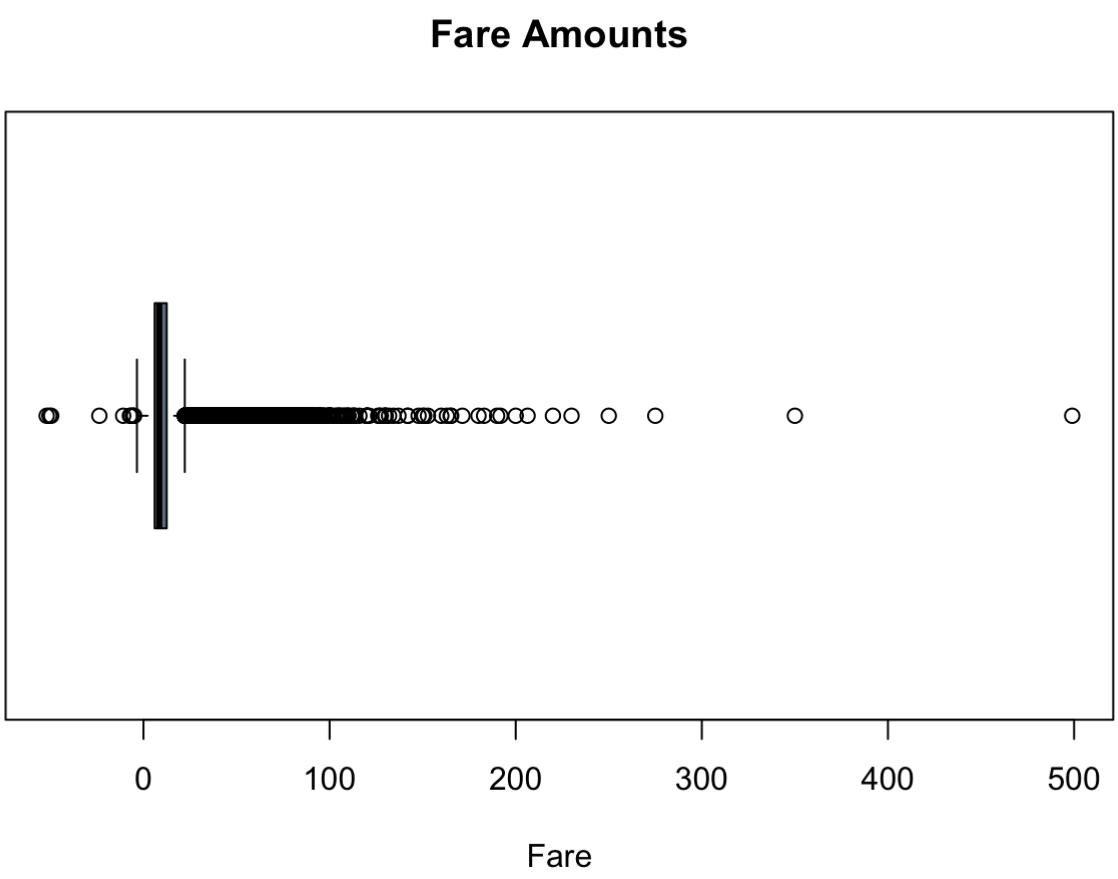
```
boxplot(train$Distance_In_Km, col="slategray", horizontal=TRUE, xlab="Distance", main="D
istance of Rides")
```



Next,

let’s get a sense of the distribution of fare amounts.

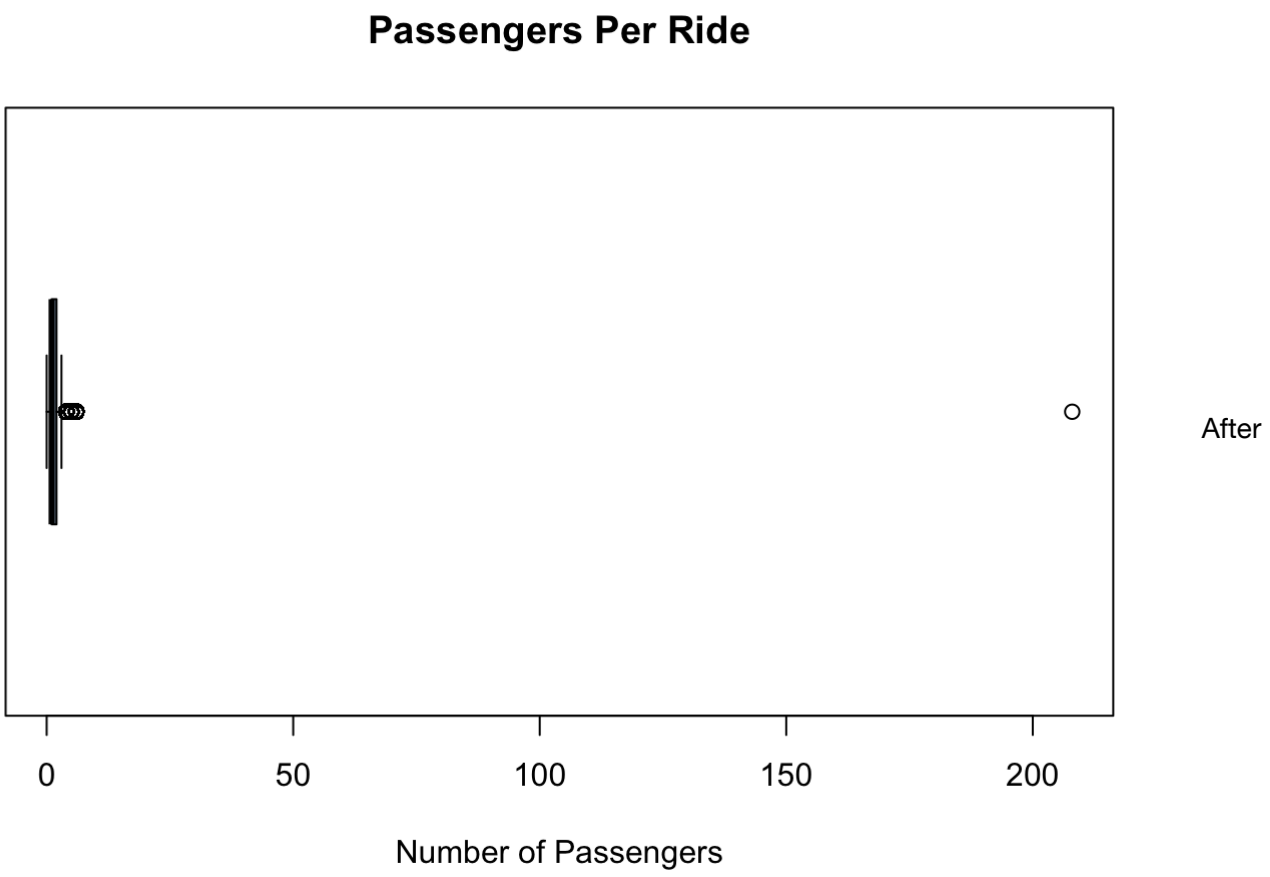
```
boxplot(train$fare_amount, col="slategray", horizontal=TRUE, xlab="Fare", main="Fare Amo
unts")
```



Next,

let's get a sense of the distribution of passenger counts.

```
boxplot(train$passenger_count, col="slategray", horizontal=TRUE, xlab="Number of Passeng
ers", main="Passengers Per Ride")
```



looking at the distribution among these columns, we've identified some outliers that would be best to remove.

We cannot have negative fare amounts, zero passengers, more than 6 passengers and more than 0 km traveled in order to accurately represent a ride. We also should remove large distances as well (let's say more than 50 miles) as well as large fare amounts (let's say more than 60 dollars).

Let's further filter our Uber data set and then do a 80/20 split again.

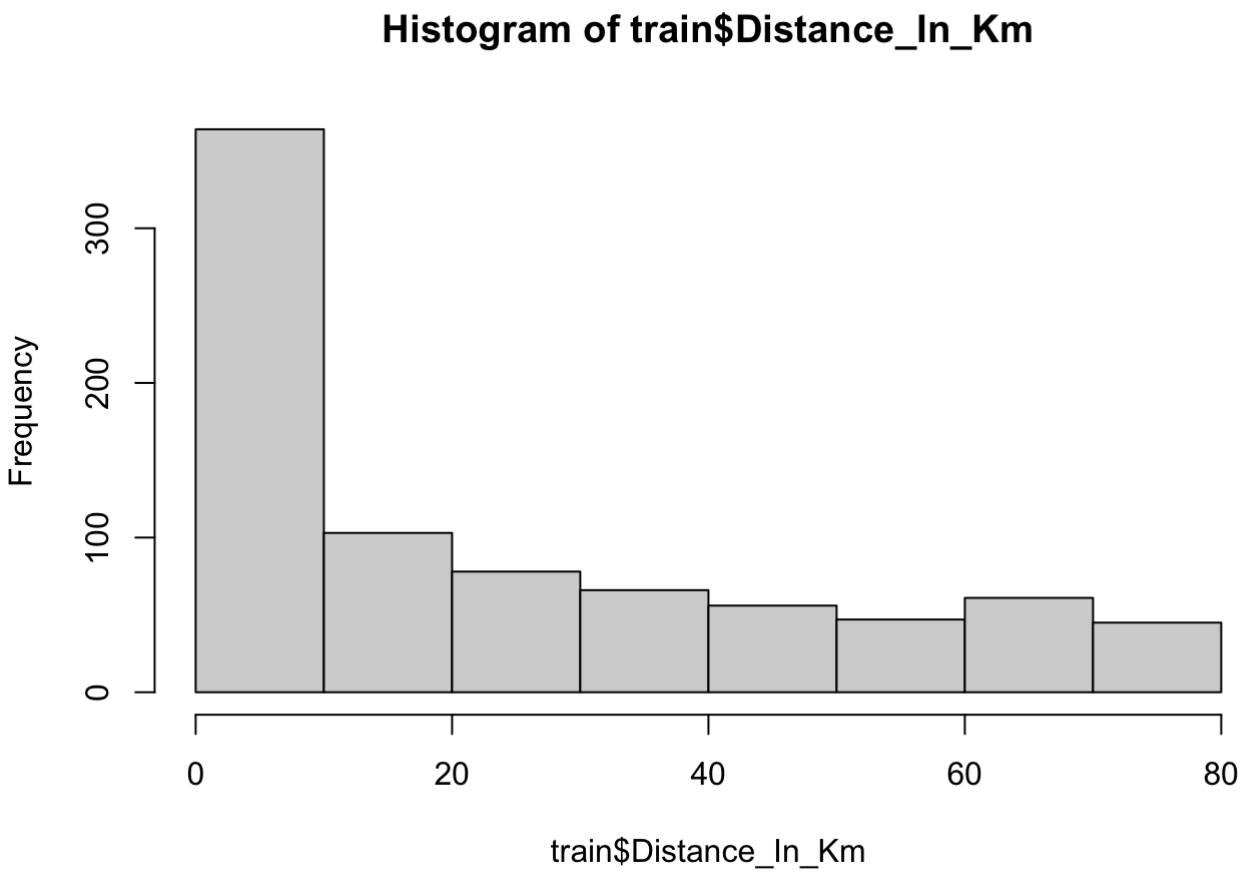
```
exclude_outliers <- filter(Uber, fare_amount < 0
         | fare_amount == 0
         | fare_amount > 60
         | passenger_count == 0
         | passenger_count > 6
         | Distance_In_Km == 0
         | Distance_In_Km > 80)
Uber <- anti_join(Uber, exclude_outliers, by="key")
```

Now, let's divide the new filtered data into train and test sets using a 80/20 split.

```
set.seed(1234)
i <- sample(1:nrow(Uber), nrow(Uber)*0.80,
replace=FALSE)
train <- Uber[i,]
test <- Uber[-i,]
```

Now that we've removed outliers, let's see a histogram of the distance of rides.

```
hist(train$Distance_In_Km)
```



Now

let’s see a histogram corresponding to fare amounts.

```
hist(train$fare_amount)
```

###

Linear Regression Now, we will build a linear regression model and output the summary accordingly.

```
lm1 <- lm(fare_amount~Distance_In_Km, data=train)
summary(lm1)
```

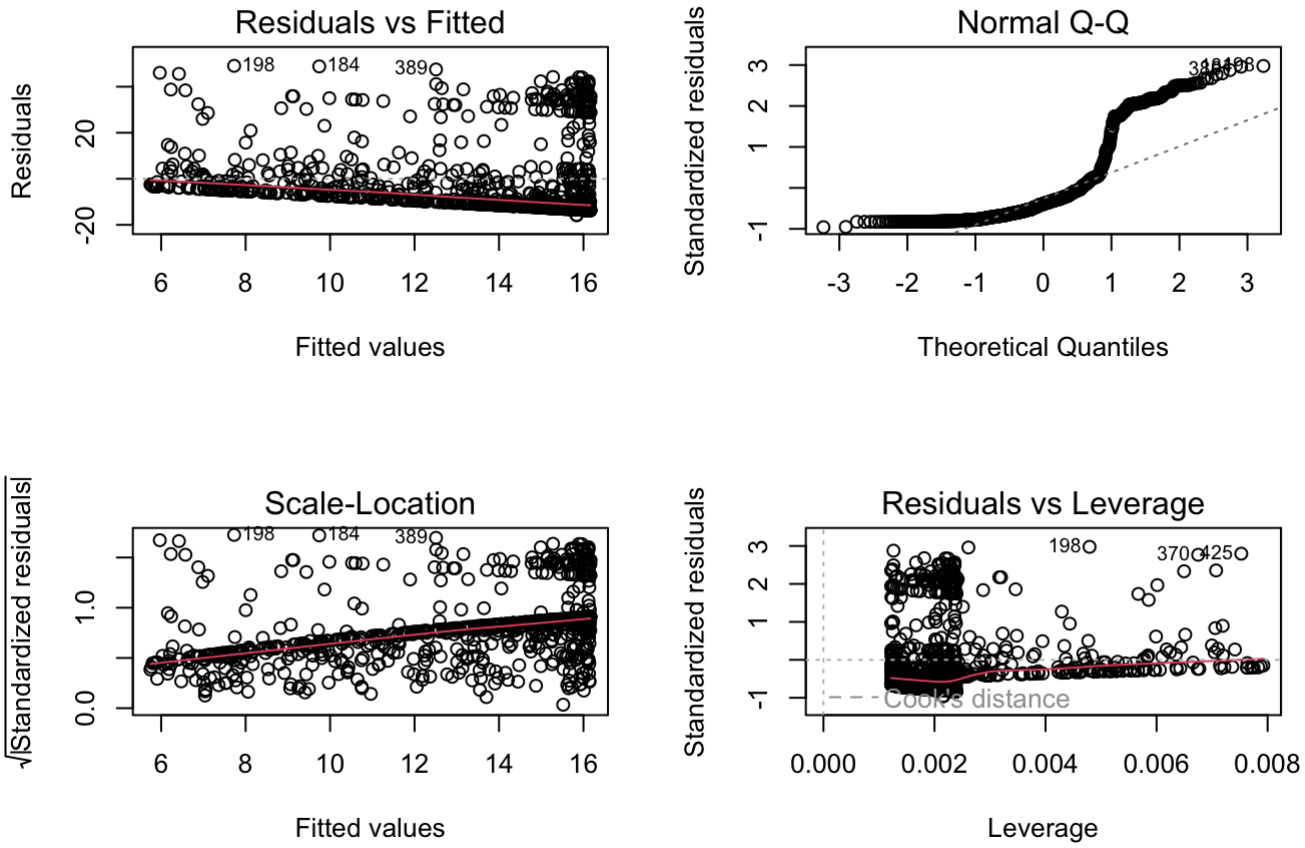
```
##
## Call:
## lm(formula = fare_amount ~ Distance_In_Km, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.832 -11.442  -6.373   2.771  49.072
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    16.17463    0.80996  19.970  < 2e-16 ***
## Distance_In_Km -0.13068    0.02407  -5.428 7.52e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.52 on 818 degrees of freedom
## Multiple R-squared:  0.03477,    Adjusted R-squared:  0.03359
## F-statistic: 29.46 on 1 and 818 DF,  p-value: 7.516e-08
```

Based on the information output from the summary, we see that the minimum residual value is -15.346 while the maximum residual value is 49.072. The residuals show a fairly wide range between the actual observed and the predicted values. The coefficient for Distance_In_Km is -0.13068, which shows a negative linear relationship between Distance_In_Km and the dependent variable, fare_amount, in our case. Based on this summary, the negative linear relationship and wide range of values show that the model may not fit this data well and more factors are needed to build an accurate model.

Residual Plots

Next, we will predict target values, plot the residuals, and provide an explanation of what the plot tells us.

```
par(mfrow=c(2,2))
plot(lm1)
```



Based on the residual plots, we can confirm our observations from the model summary. We know that the line of perfect fit represents where the residuals are equal to zero, and the red lines indicate the confidence of how far off the predicted values are from our actual values. Because there are many points above and below the red lines, this model does not fit the data well, as the variance for the residuals is not constant on these plots. For example, in the Normal Q-Q plot, there are many points above the dashed line, which shows that these residual values are not expected among a normal distribution. Because there are many smaller and larger than expected residuals, some outliers or other factors are still affecting the model's accuracy.

Multiple Linear Regression

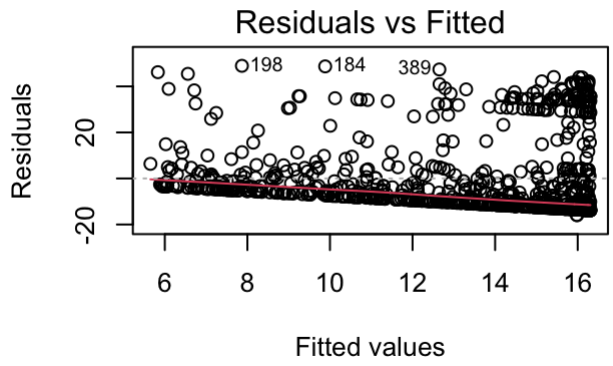
So, let's try using a linear regression model now with multiple predictors and output the summary and residual plots.

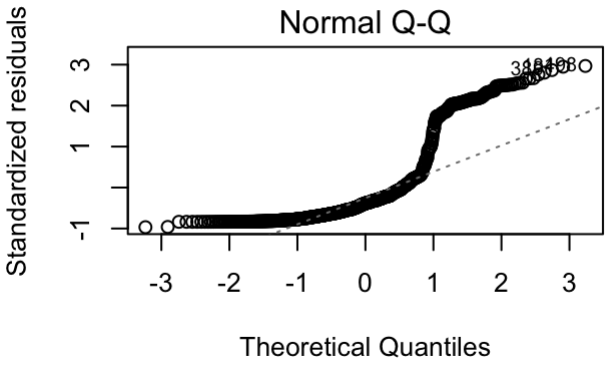
```
lm2 <- lm(fare_amount~Distance_In_Km+passenger_count, data=train)
summary(lm2)
```

```
##
## Call:
## lm(formula = fare_amount ~ Distance_In_Km + passenger_count,
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.975 -11.365  -6.367   2.921  48.932
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     16.59211    1.10708  14.987  < 2e-16 ***
## Distance_In_Km  -0.13074    0.02409  -5.428 7.51e-08 ***
## passenger_count -0.27374    0.49462  -0.553     0.58
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.53 on 817 degrees of freedom
## Multiple R-squared:  0.03513,    Adjusted R-squared:  0.03277
## F-statistic: 14.87 on 2 and 817 DF,  p-value: 4.529e-07
```

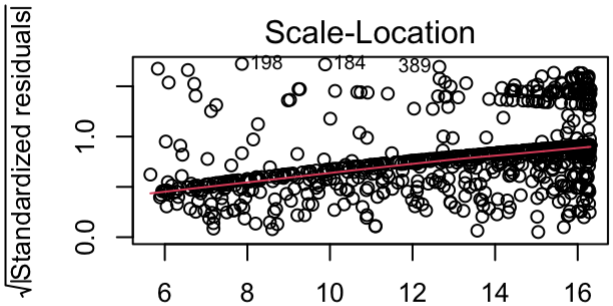
Let's output the residual plots.

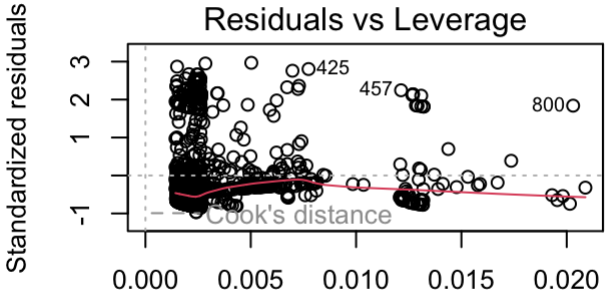
```
par(mfrow=c(2,2))
plot(lm2)
```





###





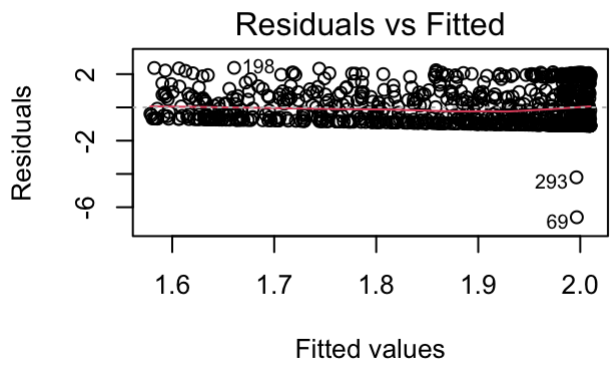
3rd Linear Regression Model Let's build a third linear regression model to try and improve the results.

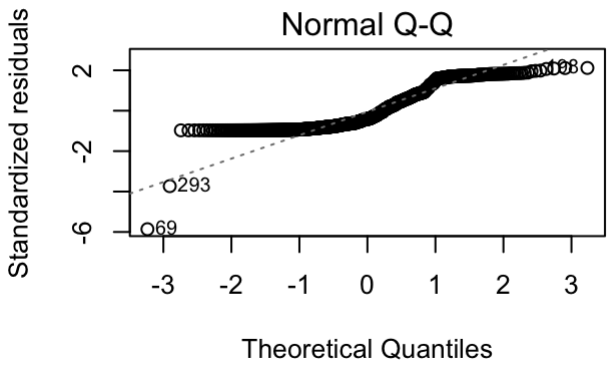
```
lm3 <- lm(log(fare_amount)~Distance_In_Km+passenger_count, data=train)
summary(lm3)
```

```
##
## Call:
## lm(formula = log(fare_amount) ~ Distance_In_Km + passenger_count,
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.6018 -0.9391 -0.4676  0.8216  2.3788
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      2.015794   0.075522  26.691  < 2e-16 ***
## Distance_In_Km  -0.005410   0.001643  -3.293  0.00104 **
## passenger_count -0.005397   0.033742  -0.160  0.87297
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.128 on 817 degrees of freedom
## Multiple R-squared:  0.01312,    Adjusted R-squared:  0.0107
## F-statistic: 5.431 on 2 and 817 DF,  p-value: 0.004538
```

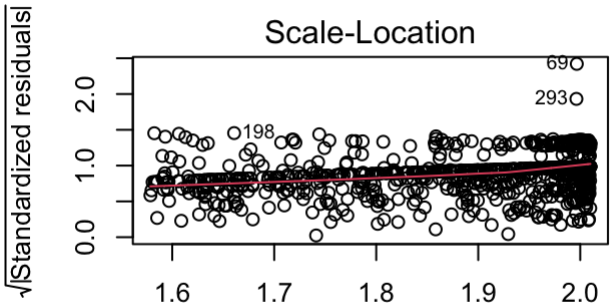
Let's output the residual plots.

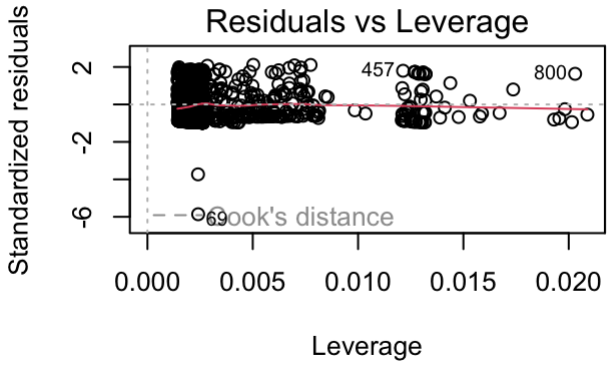
```
par(mfrow=c(2,2))
plot(lm3)
```





###





Result Comparison The third model is the best model after comparing the summary output and residual plots from all three models. The polynomial term in Distance_In_Km and the logarithmic transformation of fare_amount captures the nonlinear relationship between Distance_In_Km and fare_amount better than the other models. It also shows that passenger_count did not really affect fare_amount in any of the models. Overall, the third model appears to be the most appropriate when predicting fare_amount based on Distance_In_Km and passenger_count.

Prediction and Evaluation

Now, we will predict and evaluate on the test data using metrics correlation and mse for the first model.

```
pred1 <- predict(lm1, newdata=test)
correlation1 <- cor(pred1, test$fare_amount)
print(paste("correlation: ", correlation1))
```

```
## [1] "correlation:  0.170735464491682"
```

```
mse1 <- mean((pred1 - test$fare_amount)^2)
print(paste("mse: ", mse1))
```

```
## [1] "mse:  241.394396334485"
```

Next, we will predict and evaluate on the test data using metrics correlation and mse for the second model.

```
pred2 <- predict(lm2, newdata=test)
correlation2 <- cor(pred2, test$fare_amount)
print(paste("correlation: ", correlation2))
```

```
## [1] "correlation:  0.175843684393043"
```

```
mse2 <- mean((pred2 - test$fare_amount)^2)
print(paste("mse: ", mse2))
```

```
## [1] "mse:  240.87312382799"
```

Finally, we will predict and evaluate on the test data using metrics correlation and mse for the third model.

```
pred3 <- predict(lm3, newdata=test)
correlation3 <- cor(pred3, test$fare_amount)
print(paste("correlation: ", correlation3))
```

```
## [1] "correlation:  0.173387981616032"
```

```
mse3 <- mean((pred3 - test$fare_amount)^2)
print(paste("mse: ", mse3))
```

```
## [1] "mse:  348.448711805457"
```

Looking at the three outputs, we can see that all models have the same MSE value but different correlations. We know from class that a correlation coefficient close to 1 or -1 indicates a robust linear relationship between the variables, while a correlation closer to 0 indicates a weak relationship. The second model has a very high negative correlation, which differs from the first or third model. The high negative correlation might mean we are overfitting the data. The third model uses multiple linear and polynomial regression and has a better correlation than the first one. So, these results indicate that the third model gives the best result because it shows a better correlation coefficient while maintaining the same MSE as the other two models.

Sources

- https://dplyr.tidyverse.org/index.html (https://dplyr.tidyverse.org/index.html)
- https://www.rdocumentation.org/packages/geosphere/versions/1.5-18 (https://www.rdocumentation.org/packages/geosphere/versions/1.5-18)
- https://r4ds.had.co.nz/transform.html#transform (https://r4ds.had.co.nz/transform.html#transform)
- https://stackoverflow.com/questions/32363998/function-to-calculate-geospatial-distance-between-two-points-lat-long-using-r (https://stackoverflow.com/questions/32363998/function-to-calculate-geospatial-distance-between-two-points-lat-long-using-r)
- https://stackoverflow.com/questions/40554592/geosphere-disthaversine-dplyr-error-wrong-length-for-vector-should-be-2 (https://stackoverflow.com/questions/40554592/geosphere-disthaversine-dplyr-error-

wrong-length-for-vector-should-be-2)

- https://sebastiansauer.github.io/percentage_plot_ggplot2_V2/ (https://sebastiansauer.github.io/percentage_plot_ggplot2_V2/)
- https://www.youtube.com/watch?v=MHbzCs05Luo (https://www.youtube.com/watch?v=MHbzCs05Luo)
- https://github.com/kjmazidi/Machine_Learning_2nd_edition/tree/master/Part_2_Linear_Models (https://github.com/kjmazidi/Machine_Learning_2nd_edition/tree/master/Part_2_Linear_Models)