Classification

Name: Ryan Donaldson

Date: 02/13/2023

Summary

Linear models predict which category a given input belongs to by creating a linear decision to separate different data classes from each other. The strength of linear models is that they can be used with large datasets and are easy to visualize and implement. The weakness of linear models is that they assume a linear relationship between the features and the target variable.

Please click here (https://www.kaggle.com/datasets/yasserh/uber-fares-dataset) to access the dataset used in this project.

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. We will load the e1071 package to build a Naive Bayes model and caret for useful distribution statistics.

```
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
```

```
if(!require("e1071")) {
  install.packages("e1071")
  library("e1071", warn.conflicts=FALSE)
}
```

```
## Loading required package: e1071
```

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

```
## Loading required package: caret
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

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.

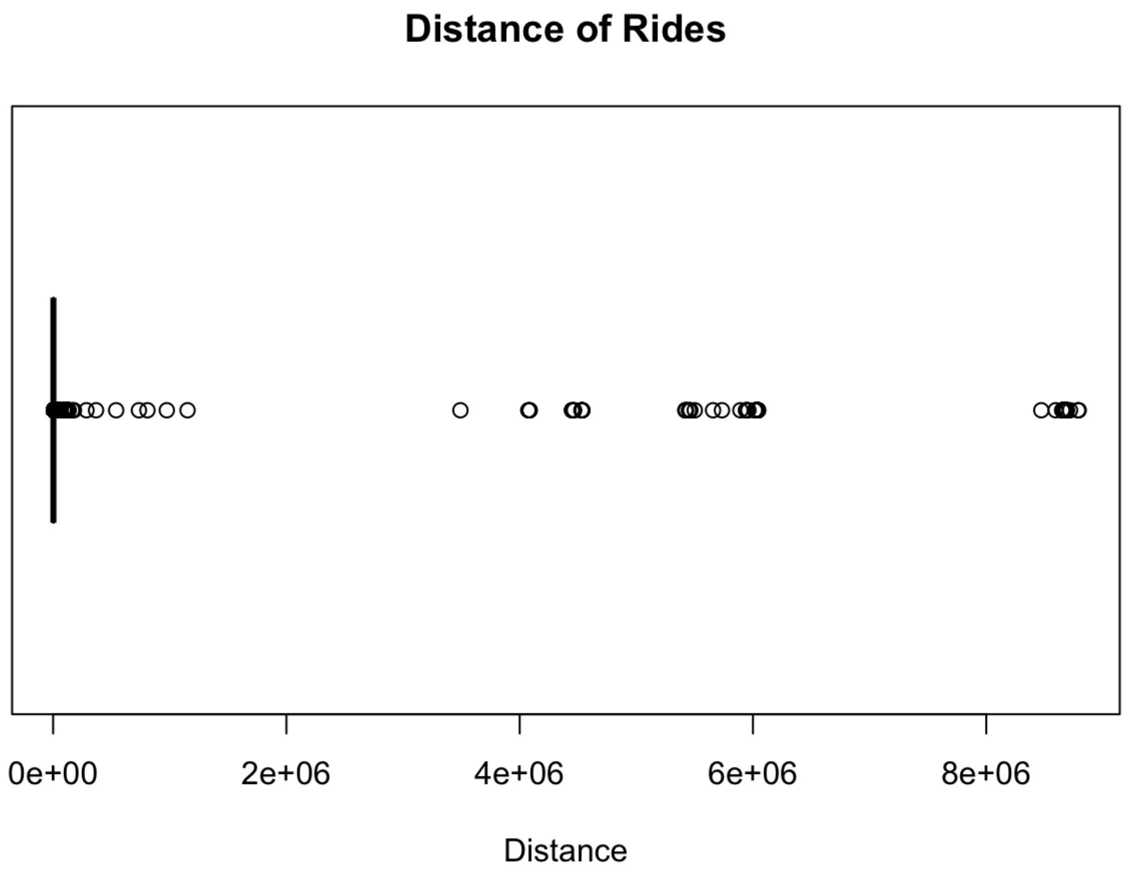
```
names(train)
```

```
##  [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

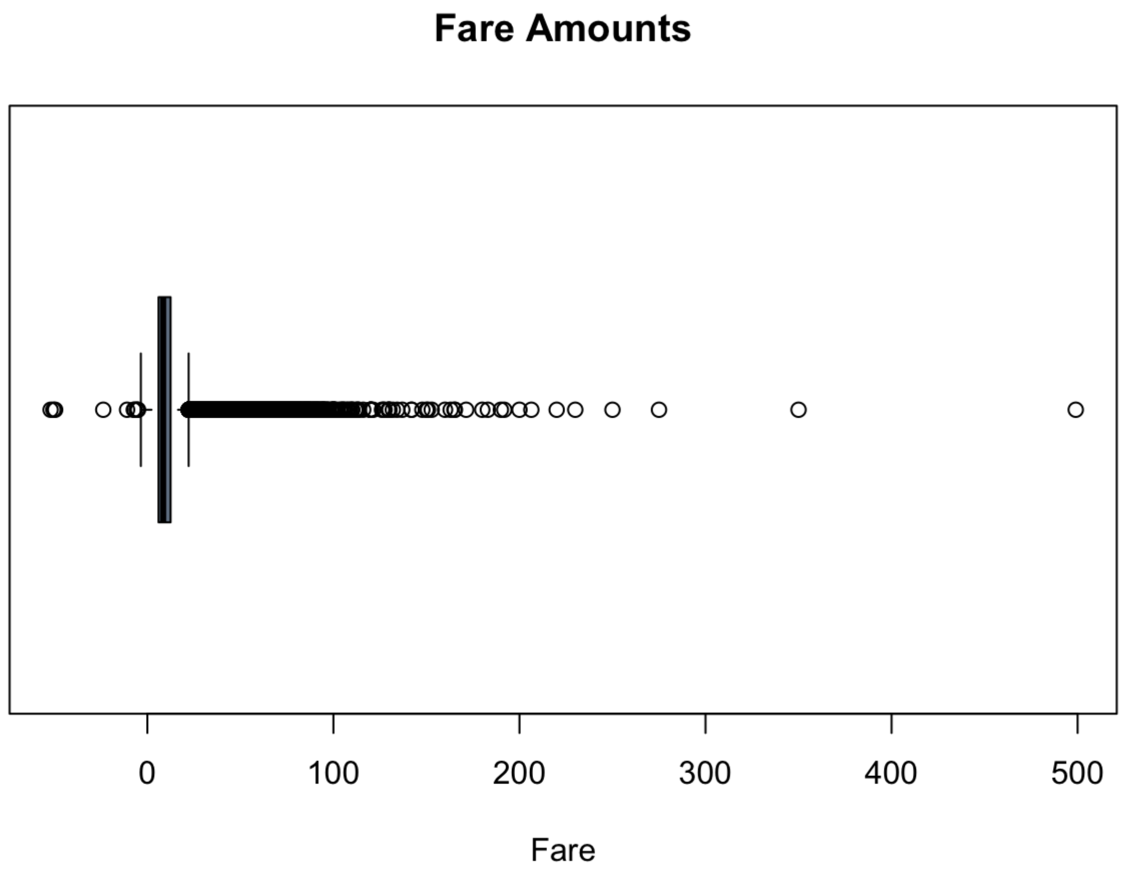
Let’s create some informative graphs based on this training data, particularly distributions of various columns since we’re working with such a large dataset. First, let’s get a sense of the distribution of ride distances.

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



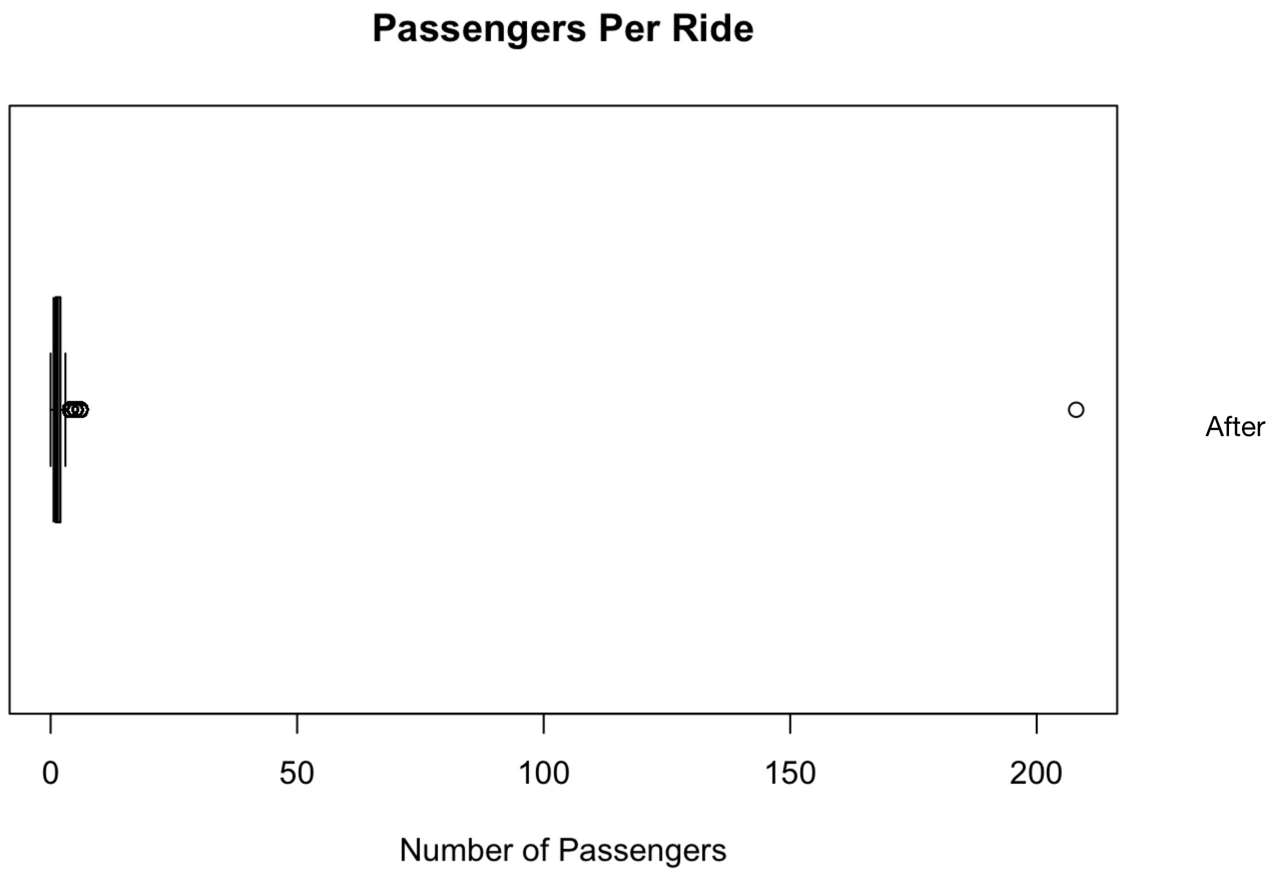
Next, let’s get a sense of the distribution of fare amounts.

```
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.

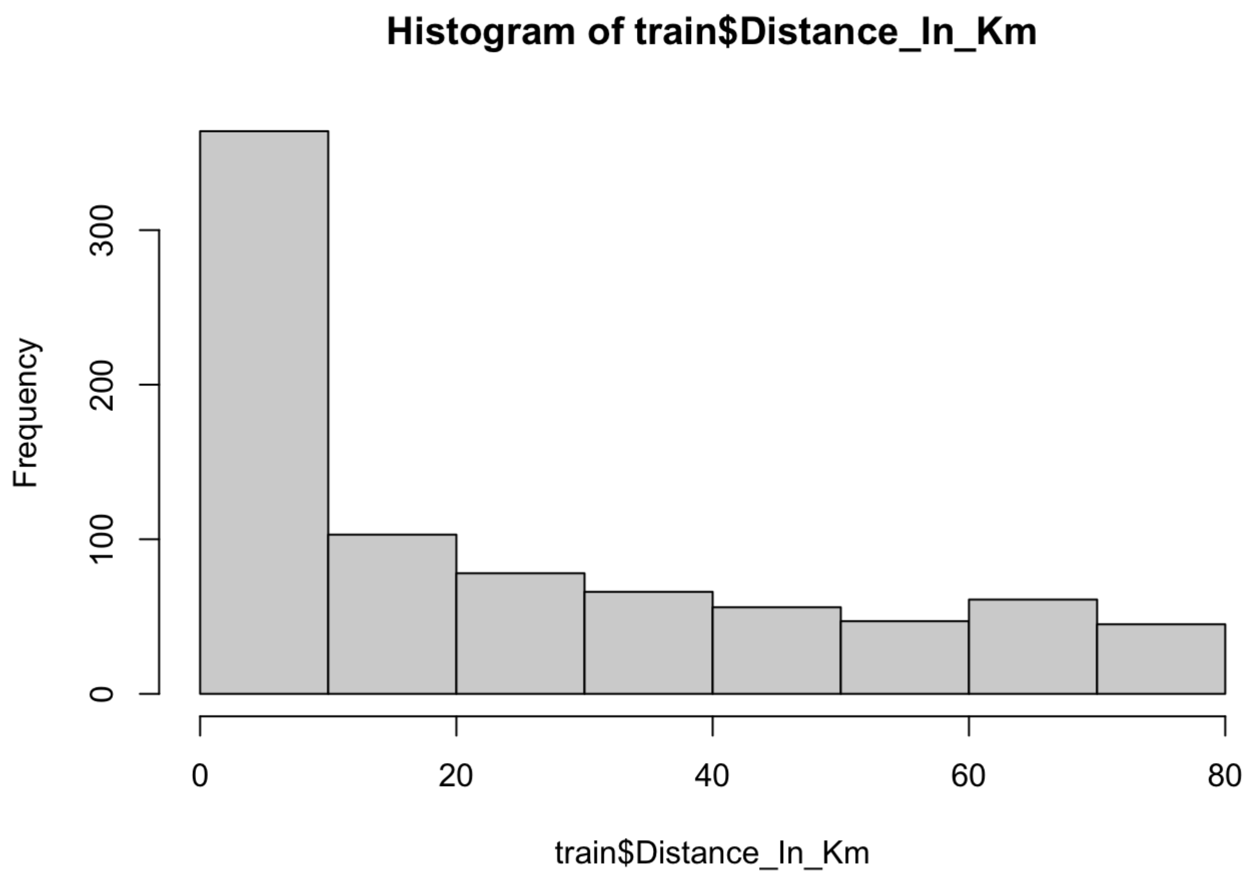
```
exclude_outliers <- filter(Uber, fare_amount < 0
         | fare_amount == 0
         | fare_amount > 60
         | passenger_count == 0
         | passenger_count > 6
         | Distance_In_Km == 0
         | Distance_In_Km > 80)
Uber <- anti_join(Uber, exclude_outliers, by="key")
```

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,]
```

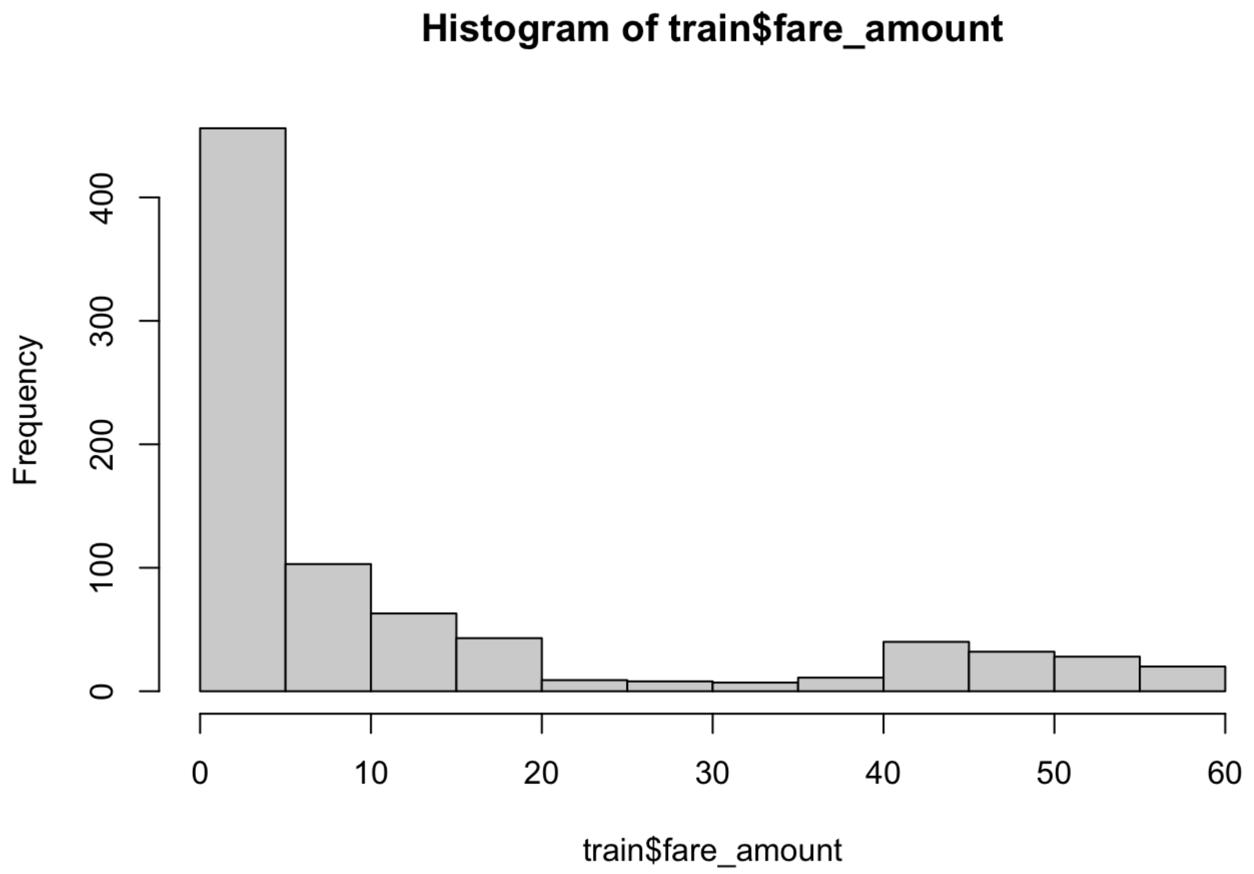
Now that we've removed outliers, let's see a histogram of the distance of rides.

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



Now let's see a histogram corresponding to fare amounts.

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



###

Logistic Regression Next, we will build a logistic regression model and output the summary.

```
glm1 <- glm(as.factor(fare_amount)~Distance_In_Km, data=train, family=binomial)
summary(glm1)
```

```
##
## Call:
## glm(formula = as.factor(fare_amount) ~ Distance_In_Km, family = binomial,
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.4480   0.0024   0.0249   0.0708   0.0899
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)      5.4955     1.2420   4.425 9.66e-06 ***
## Distance_In_Km   0.1751     0.2880   0.608    0.543
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 15.417  on 819  degrees of freedom
## Residual deviance: 13.886  on 818  degrees of freedom
## AIC: 17.886
##
## Number of Fisher Scoring iterations: 13
```

First, the median residual indicates that most residuals are close to zero. Second, the AIC measures the overall quality of the model where lower values indicate that the model is a good fit. We can see that the AIC has a low value of 17.886. With these two factors in mind, we can hypothesize that logistic regression might be a good fit for the data.

Naive Bayes

Next, we will build a Naive Bayes model and output what the model learned.

```
nb1 <- naiveBayes(fare_amount~Distance_In_Km, data=train)
nb1
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##        0.01        0.11         2.5         2.9           3        3.25
## 0.001219512 0.001219512 0.365853659 0.025609756 0.042682927 0.001219512
##         3.3         3.4         3.5         3.7         3.8           4
## 0.017073171 0.001219512 0.018292683 0.013414634 0.001219512 0.004878049
##         4.1         4.5         4.9           5         5.1         5.3
## 0.010975610 0.036585366 0.007317073 0.007317073 0.001219512 0.007317073
##         5.5         5.7           6         6.1         6.2         6.3
## 0.003658537 0.008536585 0.002439024 0.002439024 0.001219512 0.001219512
##         6.5         6.9           7         7.3         7.5         7.7
## 0.020731707 0.006097561 0.004878049 0.003658537 0.003658537 0.006097561
##           8         8.1         8.2         8.5         8.6         8.9
## 0.004878049 0.008536585 0.001219512 0.008536585 0.001219512 0.004878049
##           9         9.3         9.5         9.6         9.7         9.8
## 0.004878049 0.003658537 0.004878049 0.001219512 0.002439024 0.001219512
##          10        10.1        10.2        10.3        10.5        10.9
## 0.004878049 0.003658537 0.001219512 0.001219512 0.007317073 0.001219512
##          11        11.3        11.5        11.7          12        12.1
## 0.003658537 0.007317073 0.001219512 0.002439024 0.003658537 0.007317073
##        12.5        12.9          13        13.3        13.5        13.8
## 0.003658537 0.002439024 0.002439024 0.001219512 0.003658537 0.001219512
##          14        14.1        14.5        14.6        14.9          15
## 0.003658537 0.003658537 0.004878049 0.001219512 0.003658537 0.004878049
##        15.5        15.7        15.9          16        16.5        16.9
## 0.003658537 0.001219512 0.001219512 0.001219512 0.004878049 0.002439024
##          17        17.3        17.5        17.9          18        18.1
## 0.003658537 0.001219512 0.007317073 0.001219512 0.001219512 0.001219512
##        18.9          19        19.3        19.7        19.8          20
## 0.001219512 0.001219512 0.001219512 0.001219512 0.001219512 0.015853659
##        20.5        20.9          23        23.7        24.5          25
## 0.002439024 0.002439024 0.001219512 0.001219512 0.001219512 0.002439024
##          27          28        28.5          29       29.07        29.3
## 0.001219512 0.001219512 0.001219512 0.001219512 0.001219512 0.001219512
##          30          31          32        32.2        32.9          33
## 0.002439024 0.001219512 0.001219512 0.001219512 0.001219512 0.001219512
##       33.33          35        35.5        35.7        37.5          38
## 0.001219512 0.001219512 0.001219512 0.001219512 0.001219512 0.001219512
##          39        39.3       39.33        39.5       39.75          40
## 0.001219512 0.001219512 0.001219512 0.001219512 0.001219512 0.002439024
##        40.5        42.7          45          47          48       49.15
## 0.001219512 0.001219512 0.046341463 0.001219512 0.002439024 0.002439024
##        49.5       49.57       49.75        49.8          50        50.5
## 0.001219512 0.017073171 0.001219512 0.006097561 0.007317073 0.002439024
##          51          52          53       53.64          54        54.5
## 0.002439024 0.021951220 0.001219512 0.001219512 0.001219512 0.002439024
##          55        56.8       57.33          58        58.5       58.57 
## 0.001219512 0.003658537 0.010975610 0.002439024 0.001219512 0.001219512 
##        59.5          60 
## 0.001219512 0.003658537 
## 
## Conditional probabilities:
##        Distance_In_Km
## Y              [,1]      [,2]
##   0.01   2.5479009        NA
##   0.11   2.6177784        NA
##   2.5   22.0808761 22.912853
##   2.9   37.7251378 27.070153
##   3     34.0447226 26.426231
##   3.25   0.4189497        NA
##   3.3   33.0573410 27.472687
##   3.4   13.8409749        NA
##   3.5   28.7409726 21.188613
##   3.7   25.8192337 26.751145
##   3.8   39.0022651        NA
##   4     39.2228944 32.357884
##   4.1   25.0725978 22.363298
##   4.5   20.7843395 22.275927
##   4.9   20.3565608 26.408721
##   5     17.3290563 22.180390
##   5.1    0.7910048        NA
##   5.3   24.8641845 29.274058
##   5.5   17.6310155 25.550488
##   5.7   38.0005633 24.921268
##   6      5.8908103  6.706961
##   6.1    5.3841191  6.429328
##   6.2    0.1113195        NA
##   6.3   38.6222069        NA
##   6.5   35.3511988 27.392645
##   6.9   23.5237228 25.333211
##   7     12.6479180 13.987339
##   7.3   38.4285477 27.306757
##   7.5   31.6824242 33.862995
##   7.7   26.4652365 15.365485
##   8     19.4825927 18.605070
##   8.1   46.5446445 27.403782
##   8.2   10.9496570        NA
##   8.5   40.3814211 20.207567
##   8.6   53.0065837        NA
##   8.9   60.4118282 12.954369
##   9     25.7655882 18.173573
##   9.3   21.6479891 21.030324
##   9.5   36.8210795 39.082656
##   9.6    0.7792364        NA
##   9.7   43.8028491 15.087914
##   9.8    3.1958325        NA
##   10    46.9797054 30.284621
##   10.1  25.0079932 20.927732
##   10.2   0.6064151        NA
##   10.3   1.0103798        NA
##   10.5  33.5703407 31.169566
##   10.9  26.0425422        NA
##   11    52.8850691 26.644111
##   11.3  36.1270052 24.257916
##   11.5  41.5466553        NA
##   11.7  12.0194889  7.388253
##   12    44.1006464 36.819044
##   12.1  19.7886938 13.935128
##   12.5  45.5540423 29.524464
##   12.9   7.6497091 10.423294
##   13    37.4942128 24.596267
##   13.3  13.1846810        NA
##   13.5  17.7674502 13.380645
##   13.8   0.7972678        NA
##   14    27.8763817 25.804370
##   14.1  52.5440523  8.518843
##   14.5  29.1652861 21.022864
##   14.6  54.8227250        NA
##   14.9  19.6471716 20.595603
##   15     2.8195232  1.100715
##   15.5  12.1418029 15.703828
##   15.7   9.9401851        NA
##   15.9  20.1596833        NA
##   16    64.5070556        NA
##   16.5  30.9218245 23.033063
##   16.9  27.4040126 33.224435
##   17    39.2888415 27.736496
##   17.3   0.2226390        NA
##   17.5  27.9379725 25.262726
##   17.9  29.3443625        NA
##   18     4.9584936        NA
##   18.1   0.3373876        NA
##   18.9  43.3468102        NA
##   19     3.7079537        NA
##   19.3  56.2555520        NA
##   19.7  49.4254984        NA
##   19.8   0.4768149        NA
##   20    12.3279851 22.412380
##   20.5  28.0313821 37.240413
##   20.9  54.5514320 31.240035
##   23    34.7792858        NA
##   23.7  62.5384993        NA
##   24.5   4.2576416        NA
##   25    14.1361486 18.790441
##   27    41.4726316        NA
##   28     0.4770395        NA
##   28.5  42.8602912        NA
##   29    23.5022841        NA
##   29.07 61.6608020        NA
##   29.3  27.3202839        NA
##   30    14.2162677  7.302095
##   31     1.3453087        NA
##   32     0.1689370        NA
##   32.2   4.1331101        NA
##   32.9  48.2727208        NA
##   33    70.3464558        NA
##   33.33  0.3369768        NA
##   35     0.4215060        NA
##   35.5   4.0968217        NA
##   35.7  69.4029453        NA
##   37.5  16.2146774        NA
##   38     3.1966026        NA
##   39    32.7575902        NA
##   39.3  71.1297560        NA
##   39.33 27.1882054        NA
##   39.5  56.1052094        NA
##   39.75 51.5469315        NA
##   40     1.2047137  1.308768
##   40.5  22.2395348        NA
##   42.7   0.3444593        NA
##   45    18.2949010 21.766500
##   47     0.3370903        NA
##   48     3.1114200  3.828580
##   49.15  8.5623809  3.056398
##   49.5   9.6229690        NA
##   49.57  4.7925190  3.857396
##   49.75  0.6433941        NA
##   49.8   6.5965257  6.544769
##   50     1.4089737  2.318874
##   50.5  11.6526168  6.823416
##   51     3.3334793  1.130297
##   52    13.4893381 24.157310
##   53     7.8215156        NA
##   53.64 28.0296990        NA
##   54     5.2407085        NA
##   54.5   2.0926973  1.800871
##   55     0.5816101        NA
##   56.8  23.2305708 35.868620
##   57.33  2.2508692  2.738756
##   58     1.6710081  1.576015
##   58.5   0.4042437        NA
##   58.57 49.2121304        NA
##   59.5   6.8566415        NA
##   60    10.6152528 15.110639
```

This output has given us various a priori probabilities and indicates that the model has found around 60 classes, each corresponding to a different, tiny probability. Because of this, the training data may appear scattered within most classes for the corresponding variables.

Prediction and Evaluation

First, let’s predict and evaluate on the test data of the logistic regression model.

```
p1 <- predict(glm1, newdata=test, type="response")
pred <- ifelse(p1 > 0.5, 1, 0)
acc <- mean(p1==test$fare_amount)
print(paste("accuracy =", acc))
```

```
## [1] "accuracy = 0"
```

Now, let's predict and evaluate on the test data of the Naive Bayes model.

```
p2 <- predict(nb1, newdata=test, type="class")
acc2 <- mean(p2==test$fare_amount)
print(paste("accuracy =", acc2))
```

```
## [1] "accuracy = 0.344660194174757"
```

If we assume the logistic regression model had an accuracy of 1.0 compared to the 0.345 accuracies provided by the Naive Bayes model. Based on these accuracy numbers, the logistic regression model could correctly predict all test samples, especially since the mean of the `pred` variable was 1. The logistic regression model would be a better fit for the data. This might be because there is an approximately linear relationship between the features and target variable rather than independence between features assumed by the Naive Bayes model.

Naive Bayes vs Logistic Regression

The strengths of Naive Bayes come from being useful for classification with a large dataset and independence between features. It works well with scattered data and can typically eliminate features that are not relevant to the prediction. However, its weaknesses fall under the fact that there might be a strong relationship between all the features, especially when there is not necessarily an independence between them. Logistic regression also finds its strengths in classification and a large dataset. It is easy to implement and visualize the impact of features on a target variable. However, there might only sometimes be a linear relationship among features. It only works well if there are a few outliers in the dataset so it may require further cleaning.

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://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)
- http://www.sthda.com/english/articles/36-classification-methods-essentials/151-logistic-regression-essentials-in-r/ (http://www.sthda.com/english/articles/36-classification-methods-essentials/151-logistic-regression-essentials-in-r/)