Uber & Lyfr Price Predictions

Hai Vu

December 14, 2022

TABLE OF CONTENTS

I. INTRODUCTION

In this assignment, I aim to utilize various statistical methodologies to perform exploratory analysis as well as predictive analysis on the ride share data set containing ride-sharing records of 2 companies Uber and Lyft in the city of Boston during the month of November and December in 2018, along with the weather information at the time of transaction. Primarily, I will focus on answering 3 main questions:

- Can we predict the fare prices of Uber & Lyft accurately based on the given information?
- What sort of factors influence the changes in price, and how strong are these influences relative to one another?
- What are the main factors influence the surge in prices, as represented by the "surge_multiplier" feature in the data set?

II. ANALYSIS

1. Data cleaning & feature engineering

First, let's import the ride sharing data set. Below is the overview of the original data set:

```
# Import data set and treat blank cells as NA
df_ride <- read.csv("rideshare_kaggle.csv", na.strings=c("","  ","NA"))
str(df_ride, width = 70, strict.width = "cut")
```

```
## 'data.frame':    693071 obs. of  57 variables:
##  $ id                          : chr  "424553bb-7174-41ea-aeb4-fe06d"..
##  $ timestamp                   : num  1.54e+09 1.54e+09 1.54e+09 1.54..
##  $ hour                        : int  9 2 1 4 3 18 5 19 6 10 ...
##  $ day                         : int  16 27 28 30 29 17 26 2 3 27 ...
##  $ month                       : int  12 11 11 11 11 12 11 12 12 11 ...
##  $ datetime                    : chr  "2018-12-16 09:30:07" "2018-11"..
##  $ timezone                    : chr  "America/New_York" "America/Ne"..
##  $ source                      : chr  "Haymarket Square" "Haymarket "..
##  $ destination                 : chr  "North Station" "North Station"..
##  $ cab_type                    : chr  "Lyft" "Lyft" "Lyft" "Lyft" ...
##  $ product_id                  : chr  "lyft_line" "lyft_premier" "ly"..
##  $ name                        : chr  "Shared" "Lux" "Lyft" "Lux Bla"..
##  $ price                       : num  5 11 7 26 9 16.5 10.5 16.5 3 27..
##  $ distance                    : num  0.44 0.44 0.44 0.44 0.44 0.44 1..
##  $ surge_multiplier            : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ latitude                    : num  42.2 42.2 42.2 42.2 42.2 ...
##  $ longitude                   : num  -71 -71 -71 -71 -71 ...
##  $ temperature                 : num  42.3 43.6 38.3 34.4 37.4 ...
```

```
##  $ apparentTemperature        : num  37.1 37.4 32.9 29.6 30.9 ...
##  $ short_summary              : chr  " Mostly Cloudy " " Rain " " C"..
##  $ long_summary               : chr  " Rain throughout the day. " ""..
##  $ precipIntensity            : num  0 0.13 0 0 0 ...
##  $ precipProbability          : num  0 1 0 0 0 0 0 1 0 1 ...
##  $ humidity                   : num  0.68 0.94 0.75 0.73 0.7 0.84 0...
##  $ windSpeed                  : num  8.66 11.98 7.33 5.28 9.14 ...
##  $ windGust                   : num  9.17 11.98 7.33 5.28 9.14 ...
##  $ windGustTime               : int  1545015600 1543291200 154333440..
##  $ visibility                 : num  10 4.79 10 10 10 ...
##  $ temperatureHigh            : num  43.7 47.3 47.5 45 42.2 ...
##  $ temperatureHighTime        : int  1544968800 1543251600 154332000..
##  $ temperatureLow             : num  34.2 42.1 33.1 28.9 36.7 ...
##  $ temperatureLowTime         : int  1545048000 1543298400 154340280..
##  $ apparentTemperatureHigh    : num  38 43.9 44.1 38.5 35.8 ...
##  $ apparentTemperatureHighTime: int  1544968800 1543251600 154332000..
##  $ apparentTemperatureLow     : num  27.4 36.2 29.1 26.2 30.3 ...
##  $ apparentTemperatureLowTime : int  1545044400 1543291200 154339200..
##  $ icon                       : chr  " partly-cloudy-night " " rain"..
##  $ dewPoint                   : num  32.7 41.8 31.1 26.6 28.6 ...
##  $ pressure                   : num  1022 1004 992 1014 998 ...
##  $ windBearing                : int  57 90 240 310 303 294 91 159 30..
##  $ cloudCover                 : num  0.72 1 0.03 0 0.44 1 1 1 1 1 ...
##  $ uvIndex                    : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ visibility.1               : num  10 4.79 10 10 10 ...
##  $ ozone                      : num  304 291 316 291 348 ...
##  $ sunriseTime                : int  1544962084 1543232969 154331943..
##  $ sunsetTime                 : int  1544994864 1543266992 154335336..
##  $ moonPhase                  : num  0.3 0.64 0.68 0.75 0.72 0.33 0...
##  $ precipIntensityMax         : num  0.1276 0.13 0.1064 0 0.0001 ...
##  $ uvIndexTime                : int  1544979600 1543251600 154333800..
##  $ temperatureMin             : num  39.9 40.5 35.4 34.7 33.1 ...
##  $ temperatureMinTime         : int  1545012000 1543233600 154337760..
##  $ temperatureMax             : num  43.7 47.3 47.5 45 42.2 ...
##  $ temperatureMaxTime         : int  1544968800 1543251600 154332000..
##  $ apparentTemperatureMin     : num  33.7 36.2 31 30.3 29.1 ...
##  $ apparentTemperatureMinTime : int  1545012000 1543291200 154337760..
##  $ apparentTemperatureMax     : num  38.1 43.9 44.1 38.5 35.8 ...
##  $ apparentTemperatureMaxTime : int  1544958000 1543251600 154332000..
```

The imported data set contains 693071 observations and 57 variables. At first glance, we can see that there are 2 main groups of information presented, where one is about the information of the ride (starting location, ending location, price, type of cab, etc.) and the other describes the weather conditions surrounding that specific ride. This makes sense since weather status may be a factor in determining the fare price for Uber and Lyft. Below are

the descriptions of the variables in this data set:

Table 1. Description of the data set variables

Summarized list of variables	Description of variables
Hour & day & month	All are integer variables used to record the specific time when each transaction occurred.
source	Starting location of the ride
destination	Ending location of the ride
cab_type	Strings used to record the transaction platform, including Uber and lyft.
product_id	Unique identification of the ride type
name	Strings used to record the ride type (e.g: Uber XL from Uber and Lyft XL from Lyft)
price	Fare price of the ride
distance	Total distance of the ride
surge_multiplier	Values determining how many times the price is surged
latitude	Latitude of each transaction
longitude	Longitude of each transaction
temperature	Temperature at time of transaction
apparentTemperature	Apparent temperature at time of transaction
short_summary	Short description of weather condition at time of transaction
long_summary	Long description of weather condition at time of transaction
precipIntensity	Indicates the amount of precipitation expected within one hour at the current intensity
precipProbability	Probability that the forecast timeframe and location will receive at least 0.01" of rain
Other weather measurements	humidity, wind speed, wind gust, visibility, UV index, etc.
Timestamps of specific events	highest & lowest temperature, wind gust, sunrise, sunset, etc.

While importing the data, I replace all blank cells (if any) by NA values so that they can be examined together. I also trim any unnecessary spaces for the categorical variables:

```
# Remove blank and white space
df_ride2 <- df_ride %>% mutate_if(is.character, str_trim) # remove white
 ↪  space
```

Next, I check to see if there are any variables containing NA in their records:

```
##   Variables No blank counts Blank counts
## 1     price          637976        55095
```

After checking, it seems that "price" is the only variable to contain blank values. However, the missing observations only account for `8.64` percent of the entire data set and thus, the invalid amount not too significant. I also want to check how these missing values are distributed among other variables, such as the type of cab:

```
# Check distribution of NA values by product type
table(df_ride2$name, is.na(df_ride2$price))
```

```
##
##                FALSE  TRUE
##   Black        55095     0
##   Black SUV    55096     0
##   Lux          51235     0
##   Lux Black    51235     0
##   Lux Black XL 51235     0
##   Lyft         51235     0
##   Lyft XL      51235     0
##   Shared       51233     0
##   Taxi             0 55095
##   UberPool     55091     0
##   UberX        55094     0
##   UberXL       55096     0
##   WAV          55096     0
```

It seems that Taxi is the only type of ride that have no information on the fare prices in this data set. Thus, we can go ahead and remove all of the missing observations. Using the distinct function, I also remove any potential duplicates and rename some columns to be more comprehensible:

```
# Remove blank rows
df_ride3 <- na.omit(df_ride2)

# Remove duplicates
df_ride3 <- distinct(df_ride3)

# Rename columns
df_ride3 <- df_ride3 %>%
  dplyr::rename("product" = "name",
                "weather" = "icon")
```

```
## [1] "There are no repeated values in the data set"
```

Moving on, I want to check the number of unique classes within each categorical variables:

Table 2. Unique counts of categorical variables

Variable Name	Unique Count
id	637976

Table 2. Unique counts of categorical variables *(continued)*

Variable Name	Unique Count
datetime	31350
day	17
source	12
destination	12
product_id	12
product	12
long_summary	11
short_summary	9
weather	7
cab_type	2
month	2
timezone	1

Looking at the number of unique classes, I am able to identify a number of irrelevant columns, such as ID and timezone. Furthermore, I also notice several redundant numerical columns like timestamp, product_id, and visibility.1, which are replaceable by datetime, product, and visibility, respectively. Thus, my next step is to remove these variables:

```
# Remove redundant columns
col_remove <-
 ↪  c("id","day","timestamp","timezone","product_id","visibility.1")
df_ride4 <- df_ride3[,!names(df_ride3) %in% col_remove]
```

After that, I want to convert the variables containing timestamp values into readable time format. I decide to use the decimal hours format, where the hours and minutes are converted into continuous decimal numbers representing a 24-hour time range; for example, 4:30 PM would become 16.5 hours. Below is the function I used to do the transformation and an overview of the data set after the time format transformation was done:

```
# Convert time stamp to decimal hours
as_dechour <- function(y){
  y <- format(as.POSIXct(as_datetime(y)), format = "%H:%M")
  sapply(strsplit(y,":"),
         function(x) {
           x <- as.numeric(x)
           x[1]+x[2]/60
         }
  )
}
```

```r
# Check current look
str(df_ride4, width = 70, strict.width = "cut")
```

```
## 'data.frame':    637976 obs. of  55 variables:
##  $ hour                       : int  9 2 1 4 3 18 5 19 6 10 ...
##  $ month                      : int  11 10 10 10 10 11 10 11 11 10 ...
##  $ datetime                   : POSIXlt, format: "2018-12-16 09:30:"..
##  $ source                     : chr  "Haymarket Square" "Haymarket "..
##  $ destination                : chr  "North Station" "North Station"..
##  $ cab_type                   : chr  "Lyft" "Lyft" "Lyft" "Lyft" ...
##  $ product                    : chr  "Shared" "Lux" "Lyft" "Lux Bla"..
##  $ price                      : num  5 11 7 26 9 16.5 10.5 16.5 3 27..
##  $ distance                   : num  0.44 0.44 0.44 0.44 0.44 0.44 1..
##  $ surge_multiplier           : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ latitude                   : num  42.2 42.2 42.2 42.2 42.2 ...
##  $ longitude                  : num  -71 -71 -71 -71 -71 ...
##  $ temperature                : num  42.3 43.6 38.3 34.4 37.4 ...
##  $ apparentTemperature        : num  37.1 37.4 32.9 29.6 30.9 ...
##  $ short_summary              : chr  "Mostly Cloudy" "Rain" "Clear""..
##  $ long_summary               : chr  "Rain throughout the day." "Ra"..
##  $ precipIntensity            : num  0 0.13 0 0 0 ...
##  $ precipProbability          : num  0 1 0 0 0 0 0 1 0 1 ...
##  $ humidity                   : num  0.68 0.94 0.75 0.73 0.7 0.84 0...
##  $ windSpeed                  : num  8.66 11.98 7.33 5.28 9.14 ...
##  $ windGust                   : num  9.17 11.98 7.33 5.28 9.14 ...
##  $ windGustTime               : num  3 4 16 18 23 5 3 13 17 17 ...
##  $ visibility                 : num  10 4.79 10 10 10 ...
##  $ temperatureHigh            : num  43.7 47.3 47.5 45 42.2 ...
##  $ temperatureHighTime        : num  14 17 12 17 16 20 18 22 16 12 ...
##  $ temperatureLow             : num  34.2 42.1 33.1 28.9 36.7 ...
##  $ temperatureLowTime         : num  12 6 11 12 8 11 6 6 11 10 ...
##  $ apparentTemperatureHigh    : num  38 43.9 44.1 38.5 35.8 ...
##  $ apparentTemperatureHighTime: num  14 17 12 17 16 21 17 22 16 12 ...
##  $ apparentTemperatureLow     : num  27.4 36.2 29.1 26.2 30.3 ...
##  $ apparentTemperatureLowTime : num  11 4 8 11 3 12 6 6 9 10 ...
##  $ weather                    : chr  "partly-cloudy-night" "rain" ""..
##  $ dewPoint                   : num  32.7 41.8 31.1 26.6 28.6 ...
##  $ pressure                   : num  1022 1004 992 1014 998 ...
##  $ windBearing                : int  57 90 240 310 303 294 91 159 30..
##  $ cloudCover                 : num  0.72 1 0.03 0 0.44 1 1 1 1 1 ...
##  $ uvIndex                    : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ ozone                      : num  304 291 316 291 348 ...
##  $ sunriseTime                : num  12.1 11.8 11.8 11.9 11.8 ...
##  $ sunsetTime                 : num  21.2 21.3 21.3 21.2 21.2 ...
```

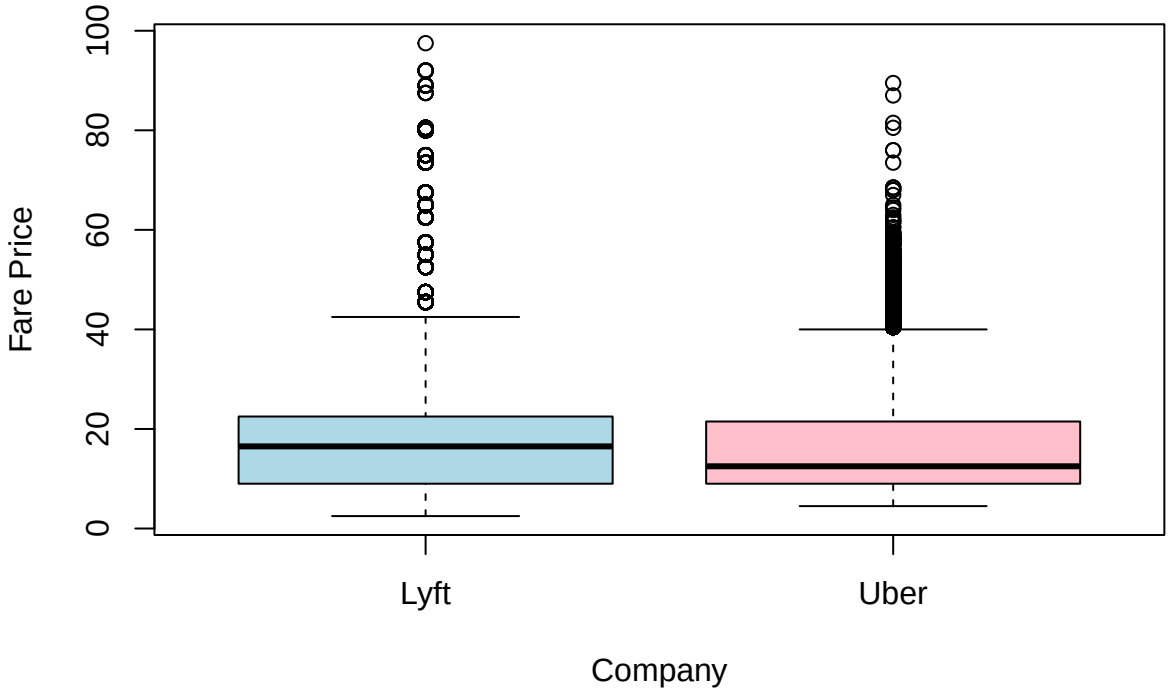
```
##  $ moonPhase                  : num  0.3 0.64 0.68 0.75 0.72 0.33 0...
##  $ precipIntensityMax         : num  0.1276 0.13 0.1064 0 0.0001 ...
##  $ uvIndexTime                : num  17 17 17 16 16 17 17 17 16 17 ...
##  $ temperatureMin             : num  39.9 40.5 35.4 34.7 33.1 ...
##  $ temperatureMinTime         : num  2 12 4 4 11 12 12 5 4 4 ...
##  $ temperatureMax             : num  43.7 47.3 47.5 45 42.2 ...
##  $ temperatureMaxTime         : num  14 17 12 17 16 5 18 22 16 12 ...
##  $ apparentTemperatureMin     : num  33.7 36.2 31 30.3 29.1 ...
##  $ apparentTemperatureMinTime : num  2 4 4 4 8 11 4 11 4 4 ...
##  $ apparentTemperatureMax     : num  38.1 43.9 44.1 38.5 35.8 ...
##  $ apparentTemperatureMaxTime : num  11 17 12 17 16 21 17 22 16 12 ...
##  $ date                       : Date, format: "2018-12-16" ...
##  $ day_of_week                : Factor w/ 7 levels "Monday","Tuesda"..
##  $ day_of_month               : int  16 27 28 30 29 17 26 2 3 27 ...
##  $ hour_of_day                : num  9.5 2 1 4.88 3.82 ...
```

2. Predict fare price using linear regression model

2.1. Feature selection

Since our target variable is the fare price, I want to examine its distribution for both Uber and Lyft:

Figure 1. Boxplots of fare price by company

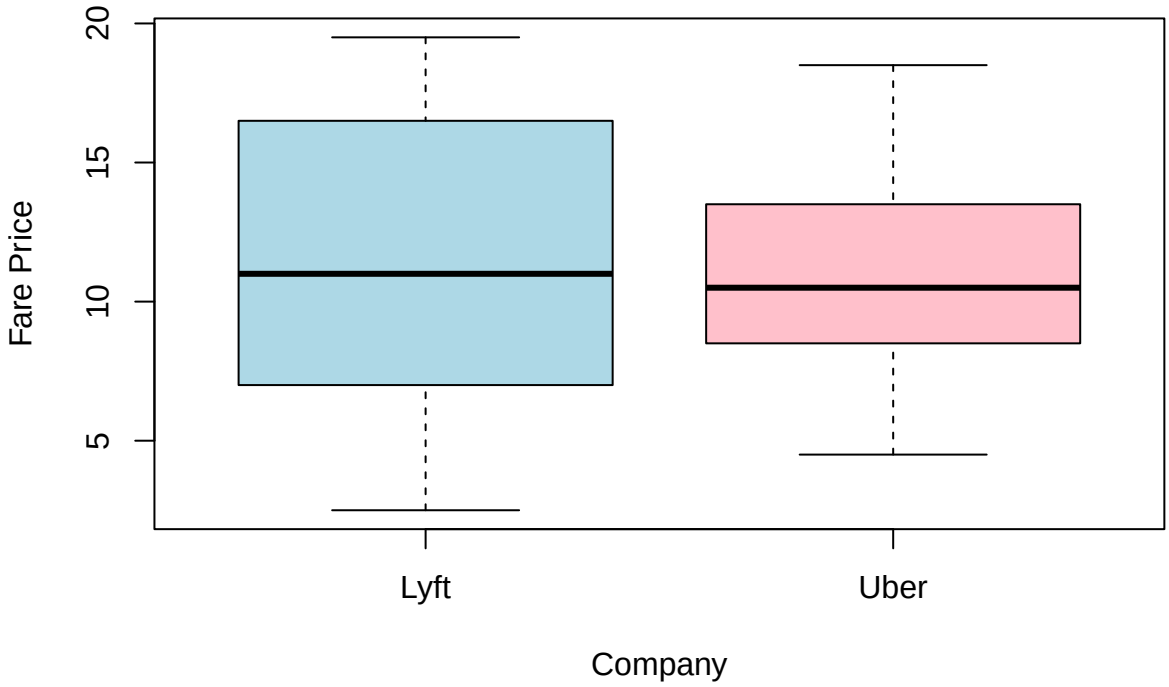


From figure 1, we can see that there are multiple outliers towards the upper (or right) tail of the distribution. Since linear regression results can be skewed by outliers, I will remove these outliers using the Tukey Method (for each group Uber and Lyft separately) with the hope of improving the model accuracy:

```
# Outliers imputation
df_uber <- subset(df_ride4,df_ride4$cab_type == "Uber")
df_lyft <- subset(df_ride4,df_ride4$cab_type == "Lyft")

uber_q25 <- as.numeric(quantile(df_uber$price,.25))
uber_q75 <- as.numeric(quantile(df_uber$price,.75))
uber_iqr <- uber_q75 - uber_q25

lyft_q25 <- as.numeric(quantile(df_lyft$price,.25))
lyft_q75 <- as.numeric(quantile(df_lyft$price,.75))
lyft_iqr <- lyft_q75 - lyft_q25

df_uber$price <- ifelse(df_uber$price > uber_iqr*1.5 | df_uber$price <
 ↪  -uber_iqr*1.5, NA, df_uber$price)
df_lyft$price <- ifelse(df_lyft$price > lyft_iqr*1.5 | df_lyft$price <
 ↪  -lyft_iqr*1.5, NA, df_lyft$price)

df_uber2 <- na.omit(df_uber)
df_lyft2 <- na.omit(df_lyft)
```

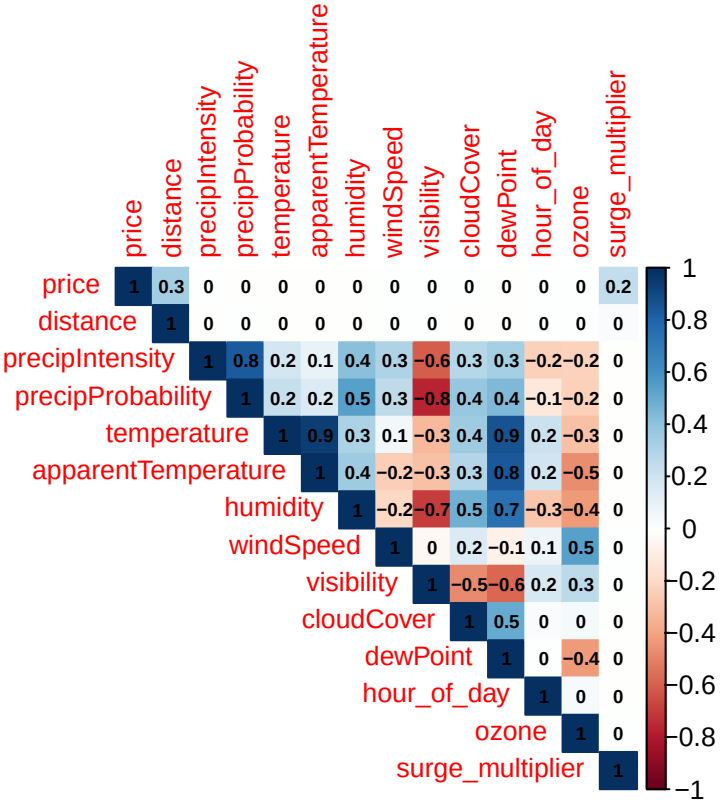
After imputing the outliers, the fare price distribution now looks like the following:

Figure 2. Boxplots of fare price by company (without outliers)



Next, I want to examine the correlation level between some of the numerical variables. The following correlation matrix below presents that information:

Figure 3. Correlation matrix of numerical values



From figure 3, I was able to spot several highly correlated groups of variables:

- Group 1: humidity, dewPoint, visibility, precipIntensity, precipProbability
- Group 2: temperature, apparentTemperature, dewPoint

To avoid multi-collinearity instances, I want to pick only 1 feature from each group that are not highly correlated with one another. Thus, I pick visibility and temperature and I also believe they may have an impact on the pricing of the Uber & Lyft rides. The remaining numerical features are not correlated with one another, and so I decide to use all of them.

As for the categorical variables, I believe the starting location of the trip might influence the fare price more than the destination of the trip because the traffic of Uber/Lyft drivers around the pick-up locations determines can greatly affect pricing (less drivers mean more expensive rides). Meanwhile, the traffic around the destinations are not that important in determining the fare price. Other information that "destination" can offer may be the indication of how far the ride is, however, we already have "distance" as one of our features and thus, I think we can disregard the "destination" feature in this case. I also notice that there are 3 different features describing the weather conditions, therefore, I want to pick only one of them. Based on table 2, we can see that "weather" has the least number of sub-categories, and so, each sub-category may have more significant influence on the fare price. Apart from weather, the product type also greatly determines the fare price as I have already knew from my experience using the ride-share service from both companies before. Lastly, I also think that different days in the week may also tend to have different prices

since perhaps on the weekends rides are less popular because people stay at home more, for instance. In the end, all of my chosen categorical variables are: cab_type, product, source, weather, and day of week.

After picking my features, I convert the categorical variables into the factor data type and prepare to partition the data. The following is the current structure of the data set to be used for model construction:

```
## 'data.frame':    441382 obs. of  15 variables:
##  $ source     : Factor w/ 12 levels "Back Bay","Beacon Hill",..: 7 7..
##  $ cab_type   : Factor w/ 2 levels "Lyft","Uber": 2 2 2 2 2 2 2 2 2 ..
##  $ product    : Factor w/ 10 levels "Black","Lux",..: 9 1 8 10 7 10 ..
##  $ weather    : Factor w/ 7 levels "clear-day","clear-night",..: 3 2..
##  $ price      : num  12 16 7.5 7.5 5.5 8.5 15 8.5 7 9.5 ...
##  $ distance   : num  1.11 1.11 1.11 1.11 1.11 2.48 2.48 2.48 2.48 2...
##  $ temperature: num  40.1 20.4 32.9 41.3 43.5 ...
##  $ windSpeed  : num  3.38 2.94 2.65 8.3 12.13 ...
##  $ visibility : num  9.83 9.83 9.96 4.05 9.8 ...
##  $ pressure   : num  1017 1032 1034 1013 1007 ...
##  $ cloudCover : num  1 0.03 0.64 1 0.53 0.77 0.02 0.03 0.92 0.37 ...
##  $ ozone      : num  282 327 331 325 310 ...
##  $ moonPhase  : num  0.79 0.21 0.21 0.3 0.75 0.33 0.27 0.21 0.33 0.7..
##  $ hour_of_day: num  22.2 10.8 19.2 23.9 19.3 ...
##  $ day_of_week: Factor w/ 7 levels "Monday","Tuesday",..: 5 4 4 7 4 ..
```

I also want to subset the data set into 2 sets, each containing data on only 1 company (either Uber or Lyft). I will then construct separate linear regression model for each of them to predict the fare price of each company independently. Therefore, the "cab_type" variable is no longer needed.

```
# Subset by cab_type
df_uber3 <- subset(df_combine2, df_combine2$cab_type == "Uber")
df_uber3$cab_type <- NULL

df_lyft3 <- subset(df_combine2, df_combine2$cab_type == "Lyft")
df_lyft3$cab_type <- NULL
```

2.2. Data partitioning and standardization

Using the 80/20 ratio, I split the data set into the training and testing sets for both Uber and Lyft records:

```
# Partition data set using 80/20 split
set.seed(888)

# Method 1:
```

```
uber_trainRatio <- createDataPartition(df_uber3$price,p=0.8,list=F,times=1)
uber_train <- df_uber3[uber_trainRatio,]
uber_test <- df_uber3[-uber_trainRatio,]

lyft_trainRatio <- createDataPartition(df_lyft3$price,p=0.8,list=F,times=1)
lyft_train <- df_lyft3[lyft_trainRatio,]
lyft_test <- df_lyft3[-lyft_trainRatio,]
```

The following table shows the size of each data set:

Table 3. Dimension of training and testing set for the linear regression model

	Uber Training Set	Uber Testing Set	Lyft Training Set	Lyft Testing Set
No. of Records	187217	46803	165891	41471
No. of Features	14	14	14	14

Since we are using regression, a distance based method, it is useful to standardize the numerical values so they stay relatively within the same range. Thus, I scale the numerical values using the scale() function provided in R.

```
# Normalize numerical values in each set
num_vars3 <- pick_num[!pick_num %in% c("price")]
uber_train[num_vars3] <- scale(uber_train[num_vars3])
lyft_train[num_vars3] <- scale(lyft_train[num_vars3])
```

2.3. Construct linear regression models to forecast Lyft & Uber prices

I create the first linear regression model to predict the Uber prices using all of my chosen features in the previous part:

```
# Construct linear model to predict Uber prices
lm_uber1 <- lm(data=uber_train,price~.)
summary(lm_uber1)
```

```
##
## Call:
## lm(formula = price ~ ., data = uber_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.9536 -0.8904 -0.1887  0.6315 10.6441
##
## Coefficients:
##                                 Estimate Std. Error  t value Pr(>|t|)
```

```
## (Intercept)                  17.4767133  0.0293608  595.239  < 2e-16 ***
## sourceBeacon Hill             0.0589596  0.0161922    3.641 0.000271 ***
## sourceBoston University      -0.3194589  0.0170024  -18.789  < 2e-16 ***
## sourceFenway                 -0.1380681  0.0167883   -8.224  < 2e-16 ***
## sourceFinancial District     -0.3056898  0.0164676  -18.563  < 2e-16 ***
## sourceHaymarket Square        0.1056423  0.0161068    6.559 5.44e-11 ***
## sourceNorth End               0.3504915  0.0160857   21.789  < 2e-16 ***
## sourceNorth Station          -0.0646430  0.0162984   -3.966 7.30e-05 ***
## sourceNortheastern University -0.2376869  0.0166466  -14.278  < 2e-16 ***
## sourceSouth Station           0.1334107  0.0161683    8.251  < 2e-16 ***
## sourceTheatre District        0.3273024  0.0161297   20.292  < 2e-16 ***
## sourceWest End                0.0460955  0.0162025    2.845 0.004442 **
## productUberPool              -9.0497017  0.0127693 -708.705  < 2e-16 ***
## productUberX                 -8.0841588  0.0127761 -632.755  < 2e-16 ***
## productUberXL                -3.1599839  0.0129719 -243.603  < 2e-16 ***
## productWAV                   -8.0866445  0.0127797 -632.775  < 2e-16 ***
## weatherclear-night            0.0200121  0.0224224    0.893 0.372125
## weathercloudy                 0.0563859  0.0289910    1.945 0.051783 .
## weatherfog                    0.0247424  0.0458633    0.539 0.589556
## weatherpartly-cloudy-day      0.0482602  0.0225633    2.139 0.032447 *
## weatherpartly-cloudy-night    0.0440102  0.0224199    1.963 0.049648 *
## weatherrain                   0.0372778  0.0341947    1.090 0.275643
## distance                      1.7575673  0.0038575  455.628  < 2e-16 ***
## temperature                  -0.0026485  0.0052730   -0.502 0.615468
## windSpeed                     0.0024921  0.0055307    0.451 0.652288
## visibility                   -0.0050676  0.0074137   -0.684 0.494263
## pressure                      0.0008274  0.0096371    0.086 0.931583
## cloudCover                   -0.0192456  0.0085569   -2.249 0.024505 *
## ozone                         0.0053919  0.0078601    0.686 0.492722
## moonPhase                     0.0044093  0.0058216    0.757 0.448815
## hour_of_day                   0.0041484  0.0039445    1.052 0.292940
## day_of_weekTuesday           -0.0122386  0.0135661   -0.902 0.366980
## day_of_weekWednesday         -0.0269711  0.0175996   -1.532 0.125405
## day_of_weekThursday          -0.0278196  0.0150385   -1.850 0.064330 .
## day_of_weekFriday            -0.0209005  0.0171142   -1.221 0.221999
## day_of_weekSaturday          -0.0011004  0.0165440   -0.067 0.946969
## day_of_weekSunday            -0.0100578  0.0146944   -0.684 0.493684
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.439 on 187180 degrees of freedom
## Multiple R-squared:  0.8207, Adjusted R-squared:  0.8207
## F-statistic: 2.38e+04 on 36 and 187180 DF,  p-value: < 2.2e-16
```

```
vif(lm_uber1)
```

```
##                  GVIF Df GVIF^(1/(2*Df))
## source       1.268475 11        1.010868
## product      1.085050  4        1.010255
## weather     46.967669  6        1.378214
## distance     1.344798  1        1.159654
## temperature  2.512830  1        1.585191
## windSpeed    2.764526  1        1.662686
## visibility   4.967317  1        2.228748
## pressure     8.393502  1        2.897154
## cloudCover   6.617425  1        2.572436
## ozone        5.583591  1        2.362962
## moonPhase    3.062996  1        1.750142
## hour_of_day  1.406184  1        1.185826
## day_of_week 19.161592  6        1.278996
```

The r-squared of the Uber training model is 0.8207, which is not bad at all in terms of correlation level. From examining the Variance Inflation Factor (VIF), it seems that "weather" has the highest VIF values and thus, I want to remove this feature to avoid multi-collinearity occurrences.

Next, I build another regression model using stepwise regression because I want to keep only the significant predictors and disregard the others:

```
# Stepwise regression for Uber
step_uber1 <- step(lm(data=uber_train,price~.-weather), direction = "both",
 ↪  trace = FALSE)
summary(step_uber1)
```

```
##
## Call:
## lm(formula = price ~ source + product + distance + cloudCover,
##     data = uber_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.9396 -0.8940 -0.1917  0.6294 10.6459
##
## Coefficients:
##                                Estimate Std. Error  t value Pr(>|t|)
## (Intercept)                   17.507015   0.015186 1152.827  < 2e-16 ***
## sourceBeacon Hill              0.059024   0.016190    3.646 0.000267 ***
## sourceBoston University       -0.319572   0.017001  -18.797  < 2e-16 ***
## sourceFenway                  -0.138150   0.016786   -8.230  < 2e-16 ***
```

```
## sourceFinancial District      -0.305696   0.016466  -18.566  < 2e-16 ***
## sourceHaymarket Square         0.105458   0.016106    6.548 5.85e-11 ***
## sourceNorth End                0.350478   0.016084   21.791  < 2e-16 ***
## sourceNorth Station           -0.064946   0.016296   -3.985 6.74e-05 ***
## sourceNortheastern University -0.237826   0.016645  -14.288  < 2e-16 ***
## sourceSouth Station            0.133401   0.016167    8.252  < 2e-16 ***
## sourceTheatre District         0.327479   0.016127   20.306  < 2e-16 ***
## sourceWest End                 0.046435   0.016201    2.866 0.004155 **
## productUberPool               -9.049642   0.012769 -708.735  < 2e-16 ***
## productUberX                  -8.084170   0.012776 -632.773  < 2e-16 ***
## productUberXL                 -3.159964   0.012971 -243.615  < 2e-16 ***
## productWAV                    -8.086620   0.012779 -632.796  < 2e-16 ***
## distance                       1.757575   0.003857  455.684  < 2e-16 ***
## cloudCover                    -0.004759   0.003327   -1.431 0.152521
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.439 on 187199 degrees of freedom
## Multiple R-squared:  0.8207, Adjusted R-squared:  0.8207
## F-statistic: 5.041e+04 on 17 and 187199 DF,  p-value: < 2.2e-16
```

```
vif(step_uber1)
```

```
##                GVIF Df GVIF^(1/(2*Df))
## source     1.265709 11        1.010768
## product    1.084719  4        1.010217
## distance   1.344493  1        1.159523
## cloudCover 1.000125  1        1.000063
```

The Uber stepwise regression model returns a similar R-squared of `0.8207`, however, the VIF values are now all below 5, indicating that there are no multi-collinearity instances. Since the values of the predictors are scaled beforehand, we can compare the regression coefficients of among the predictors to explore how big of an influence each of them has on the fare price. The following table shows the variable importance for determining the Uber fare price:

Table 4. Features regression coefficients from the linear model predicting Uber prices

Predictors	Coefficients	P-values
productUberPool	-9.0496425	0.0000000
productWAV	-8.0866204	0.0000000
productUberX	-8.0841703	0.0000000
productUberXL	-3.1599643	0.0000000
distance	1.7575755	0.0000000

Table 4. Features regression coefficients from the linear model predicting Uber prices *(continued)*

Predictors	Coefficients	P-values
sourceNorth End	0.3504782	0.0000000
sourceTheatre District	0.3274791	0.0000000
sourceBoston University	-0.3195717	0.0000000
sourceFinancial District	-0.3056960	0.0000000
sourceNortheastern University	-0.2378260	0.0000000
sourceFenway	-0.1381497	0.0000000
sourceSouth Station	0.1334010	0.0000000
sourceHaymarket Square	0.1054576	0.0000000
sourceNorth Station	-0.0649463	0.0000674
sourceBeacon Hill	0.0590240	0.0002668
sourceWest End	0.0464346	0.0041553
cloudCover	-0.0047593	0.1525211

From here, it seems that product, unsurprisingly, has the biggest influence on the Uber fare prices, followed by distance, source. On the other hand, considering a confidence level of 0.95 (or alpha = 0.05), cloudCover (index representing the density and thickness of clouds) is the only insignificant factor in this case.

Similarly, I create another linear model to predict the Lyft fare prices:

```
# Construct linear model to predict Lyft prices
lm_lyft1 <- lm(data=lyft_train,price~.)
summary(lm_lyft1)
```

```
##
## Call:
## lm(formula = price ~ ., data = lyft_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.1368 -1.0063 -0.0931  0.8877 12.5636
##
## Coefficients:
##                                Estimate Std. Error  t value Pr(>|t|)
## (Intercept)                   1.610e+01  3.331e-02  483.250  < 2e-16 ***
## sourceBeacon Hill            -2.526e-01  1.885e-02  -13.399  < 2e-16 ***
## sourceBoston University      -5.024e-01  2.051e-02  -24.499  < 2e-16 ***
## sourceFenway                 -2.380e-01  2.011e-02  -11.832  < 2e-16 ***
## sourceFinancial District     -7.872e-01  1.946e-02  -40.445  < 2e-16 ***
## sourceHaymarket Square       -4.929e-01  1.836e-02  -26.847  < 2e-16 ***
## sourceNorth End               4.007e-02  1.852e-02    2.163   0.0305 *
```

```
## sourceNorth Station           -4.249e-01  1.902e-02  -22.337  < 2e-16 ***
## sourceNortheastern University -1.954e-01  1.982e-02   -9.857  < 2e-16 ***
## sourceSouth Station            9.639e-02  1.870e-02    5.154 2.55e-07 ***
## sourceTheatre District         1.355e-01  1.930e-02    7.020 2.22e-12 ***
## sourceWest End                -2.367e-01  1.881e-02  -12.580  < 2e-16 ***
## productLux Black               3.152e+00  1.563e-02  201.730  < 2e-16 ***
## productLyft                   -6.716e+00  1.219e-02 -551.046  < 2e-16 ***
## productLyft XL                -1.672e+00  1.239e-02 -134.968  < 2e-16 ***
## productShared                 -1.025e+01  1.217e-02 -842.087  < 2e-16 ***
## weatherclear-night            -2.046e-02  2.638e-02   -0.776   0.4380
## weathercloudy                 -4.944e-02  3.416e-02   -1.447   0.1478
## weatherfog                    -6.763e-02  5.443e-02   -1.242   0.2141
## weatherpartly-cloudy-day      -1.378e-02  2.649e-02   -0.520   0.6030
## weatherpartly-cloudy-night    -2.641e-02  2.624e-02   -1.007   0.3141
## weatherrain                   -3.687e-02  4.037e-02   -0.913   0.3610
## distance                       2.049e+00  4.609e-03  444.431  < 2e-16 ***
## temperature                    3.447e-03  6.229e-03    0.553   0.5801
## windSpeed                     -1.365e-04  6.514e-03   -0.021   0.9833
## visibility                    -2.086e-03  8.759e-03   -0.238   0.8118
## pressure                       1.223e-03  1.135e-02    0.108   0.9142
## cloudCover                     5.372e-03  1.013e-02    0.530   0.5959
## ozone                         -3.860e-03  9.256e-03   -0.417   0.6766
## moonPhase                     -6.941e-03  6.853e-03   -1.013   0.3111
## hour_of_day                   -3.509e-03  4.661e-03   -0.753   0.4516
## day_of_weekTuesday            -1.862e-02  1.604e-02   -1.161   0.2458
## day_of_weekWednesday           1.914e-03  2.077e-02    0.092   0.9266
## day_of_weekThursday           -1.884e-02  1.779e-02   -1.059   0.2894
## day_of_weekFriday             -1.865e-02  2.024e-02   -0.921   0.3570
## day_of_weekSaturday            2.217e-03  1.949e-02    0.114   0.9094
## day_of_weekSunday             -2.424e-02  1.737e-02   -1.396   0.1628
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.601 on 165854 degrees of freedom
## Multiple R-squared:  0.8868, Adjusted R-squared:  0.8868
## F-statistic: 3.609e+04 on 36 and 165854 DF,  p-value: < 2.2e-16
```

```
vif(lm_lyft1)
```

```
##                  GVIF Df GVIF^(1/(2*Df))
## source       1.281310 11        1.011331
## product      1.107734  4        1.012872
## weather     46.415285  6        1.376856
## distance     1.374940  1        1.172578
```

```
## temperature  2.511164  1        1.584665
## windSpeed    2.745690  1        1.657013
## visibility   4.964546  1        2.228126
## pressure     8.340039  1        2.887912
## cloudCover   6.640435  1        2.576904
## ozone        5.543538  1        2.354472
## moonPhase    3.038833  1        1.743225
## hour_of_day  1.405932  1        1.185720
## day_of_week 19.157870  6        1.278976
```

The Lyft linear regression models return a higher R-squared of `0.8868` compared to the Uber model, indicating a better fit. However, "weather" is once again attain a high VIF value and thus, needs to be removed. Using the stepwise regression method, I attempt to remove weather along with other insignificant factors.

```
# Stepwise regression for Lyft
step_lyft1 <- step(lm(data=lyft_train,price~.-weather), direction = "both",
 ↪  trace = FALSE)
summary(step_lyft1)
```

```
##
## Call:
## lm(formula = price ~ source + product + distance, data = lyft_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.1108 -1.0070 -0.0948  0.8874 12.5838
##
## Coefficients:
##                                Estimate Std. Error  t value Pr(>|t|)
## (Intercept)                   16.054689   0.015637 1026.741  < 2e-16 ***
## sourceBeacon Hill             -0.252898   0.018848  -13.417  < 2e-16 ***
## sourceBoston University       -0.502200   0.020503  -24.494  < 2e-16 ***
## sourceFenway                  -0.237864   0.020111  -11.827  < 2e-16 ***
## sourceFinancial District      -0.787311   0.019461  -40.456  < 2e-16 ***
## sourceHaymarket Square        -0.492616   0.018357  -26.835  < 2e-16 ***
## sourceNorth End                0.039755   0.018522    2.146   0.0318 *
## sourceNorth Station           -0.424872   0.019018  -22.341  < 2e-16 ***
## sourceNortheastern University -0.195483   0.019820   -9.863  < 2e-16 ***
## sourceSouth Station            0.095823   0.018699    5.124 2.99e-07 ***
## sourceTheatre District         0.135080   0.019298    7.000 2.57e-12 ***
## sourceWest End                -0.237060   0.018811  -12.602  < 2e-16 ***
## productLux Black               3.152229   0.015625  201.741  < 2e-16 ***
## productLyft                   -6.715497   0.012186 -551.077  < 2e-16 ***
## productLyft XL                -1.672383   0.012390 -134.975  < 2e-16 ***
```

```
## productShared                -10.250528   0.012172 -842.133  < 2e-16 ***
## distance                       2.048611   0.004609  444.472  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.601 on 165874 degrees of freedom
## Multiple R-squared:  0.8868, Adjusted R-squared:  0.8868
## F-statistic: 8.121e+04 on 16 and 165874 DF,  p-value: < 2.2e-16
```

```
vif(step_lyft1)
```

```
##               GVIF Df GVIF^(1/(2*Df))
## source   1.277964 11        1.011211
## product  1.107313  4        1.012824
## distance 1.374713  1        1.172481
```

Again, the R-squared value of the stepwise model does not change, but the now only the significant variables are kept and their VIF values are all lower than 5. Next, I compare the regression coefficients of all features to see their impacts on the Lyft prices:

Table 5. Features regression coefficients from the linear model predicting Lyft prices

Predictors	Coefficients	P-values
productShared	-10.2505278	0.0000000
productLyft	-6.7154966	0.0000000
productLux Black	3.1522288	0.0000000
distance	2.0486112	0.0000000
productLyft XL	-1.6723827	0.0000000
sourceFinancial District	-0.7873112	0.0000000
sourceBoston University	-0.5021998	0.0000000
sourceHaymarket Square	-0.4926156	0.0000000
sourceNorth Station	-0.4248724	0.0000000
sourceBeacon Hill	-0.2528984	0.0000000
sourceFenway	-0.2378642	0.0000000
sourceWest End	-0.2370604	0.0000000
sourceNortheastern University	-0.1954830	0.0000000
sourceTheatre District	0.1350796	0.0000000
sourceSouth Station	0.0958229	0.0000003
sourceNorth End	0.0397551	0.0318417

Similar to table 4, product, distance and source are once again, determined to be the top 3 significant factors influencing fare prices by the stepwise linear regression model.

2.4. Linear model performance

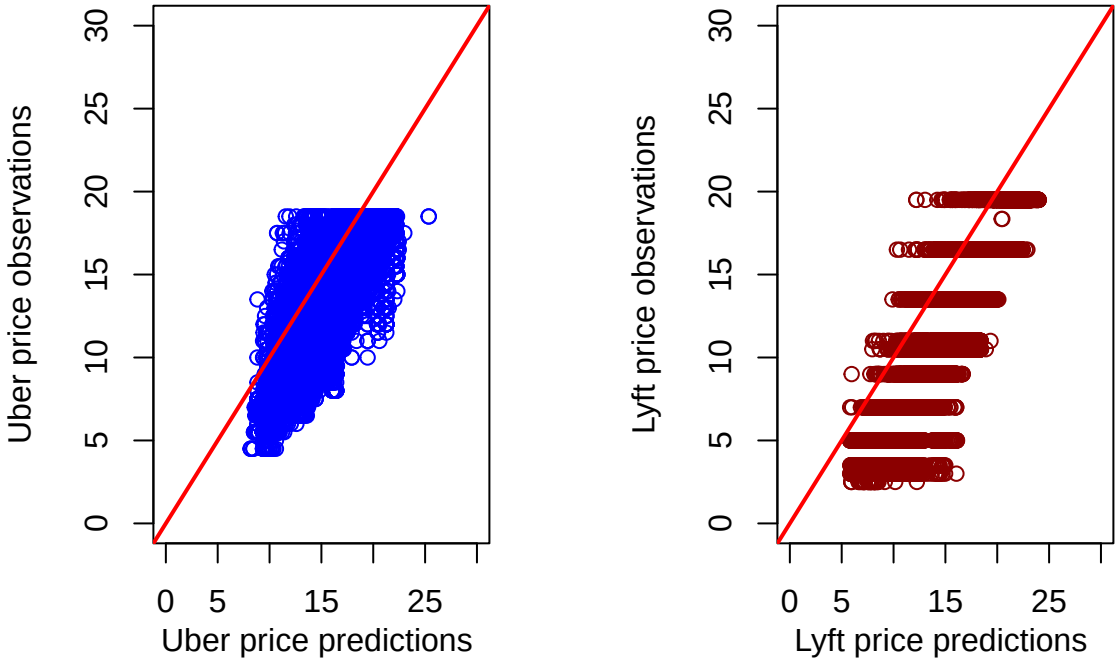
After constructing the regression models, I generate the price predictions using both the training and testing set. The following table presents the performances of those models using different metrics:

Table 6. Linear regression model predictive performance between training and testing

	Uber Training	Uber Testing	Lyft Training	Lyft Testing
RMSE	1.439	3.831	1.601	4.326
MAE	4.358	3.625	1.212	4.056
MAPE	0.099	0.260	0.131	0.284
R-Squared	0.821	0.820	0.887	0.888

From table 6, we can see that Mean Average Percentage Errors (MAPE) are still rather high when using the testing set (around 26% to 28% error) compared to the training set (around 10% to 13% error). The R-squared for the Uber price prediction model got worse during the testing process compared to the training, however, the R-squared of the Lyft model actually improves slightly, indicating that the model does not overfit. The following scatter plot helps visualize the goodness of fit between our model predictions and the actual fare prices of both companies:

Figure 4. Rideshare fare price observations vs linear predictions



From figure 4, we can see that our linear model does not seem to fit very well with this data set and there are still huge margin or errors despite having removed the outliers. For

that reason, I intend to try other predictive methods as well with the hope of predicting the fare price with higher accuracy.

3. Predict fare price using decision tree model

3.1. Decision tree model construction

Decision tree model is very robust to outliers, and thus, I will no longer remove the outliers in fare prices when training this model. Using a similar 80/20 split, I create training and testing data sets for both Uber and Lyft. The following table shows the size of each set:

Table 7. Dimension of training and testing set for the decision tree model

	Uber Training Set	Uber Testing Set	Lyft Training Set	Lyft Testing Set
No. of Records	264457	66111	245929	61479
No. of Features	16	16	16	16

First, I created the decision tree to predict Uber prices, with complexity parameter (CP) value of 0.001:

```
# Regression Trees for Uber
uber_tree1 <- rpart(price ~., data = uber_train2, method = "anova", cp =
 ↪  0.001)
printcp(uber_tree1)
```

```
##
## Regression tree:
## rpart(formula = price ~ ., data = uber_train2, method = "anova",
##     cp = 0.001)
##
## Variables actually used in tree construction:
## [1] distance product
##
## Root node error: 19389981/264457 = 73.32
##
## n= 264457
##
##           CP nsplit rel error   xerror       xstd
## 1  0.6294393      0  1.000000 1.000010 0.00303849
## 2  0.1083906      1  0.370561 0.370564 0.00163562
## 3  0.0669133      2  0.262170 0.262175 0.00137523
## 4  0.0313497      3  0.195257 0.195262 0.00114473
## 5  0.0275477      4  0.163907 0.163912 0.00106288
## 6  0.0193014      5  0.136360 0.136449 0.00092807
## 7  0.0125018      6  0.117058 0.117149 0.00083494
```

```
## 8  0.0102048      7  0.104556 0.104648 0.00081866
## 9  0.0062377      8  0.094352 0.092591 0.00074770
## 10 0.0058544      9  0.088114 0.087689 0.00072048
## 11 0.0052303     10  0.082259 0.082970 0.00071704
## 12 0.0041512     11  0.077029 0.077746 0.00067646
## 13 0.0021855     12  0.072878 0.072976 0.00066494
## 14 0.0017942     13  0.070692 0.070797 0.00066000
## 15 0.0017509     14  0.068898 0.068850 0.00065924
## 16 0.0016516     15  0.067147 0.067251 0.00065816
## 17 0.0014528     16  0.065496 0.065257 0.00064614
## 18 0.0013470     17  0.064043 0.064133 0.00063823
## 19 0.0010000     18  0.062696 0.062792 0.00063738
```

The tree created for Uber used 2 only variables distance and product to make 18 splits and 19 nodes. Similarly, I created another tree to predict the Lyft prices with CP = 0.001:

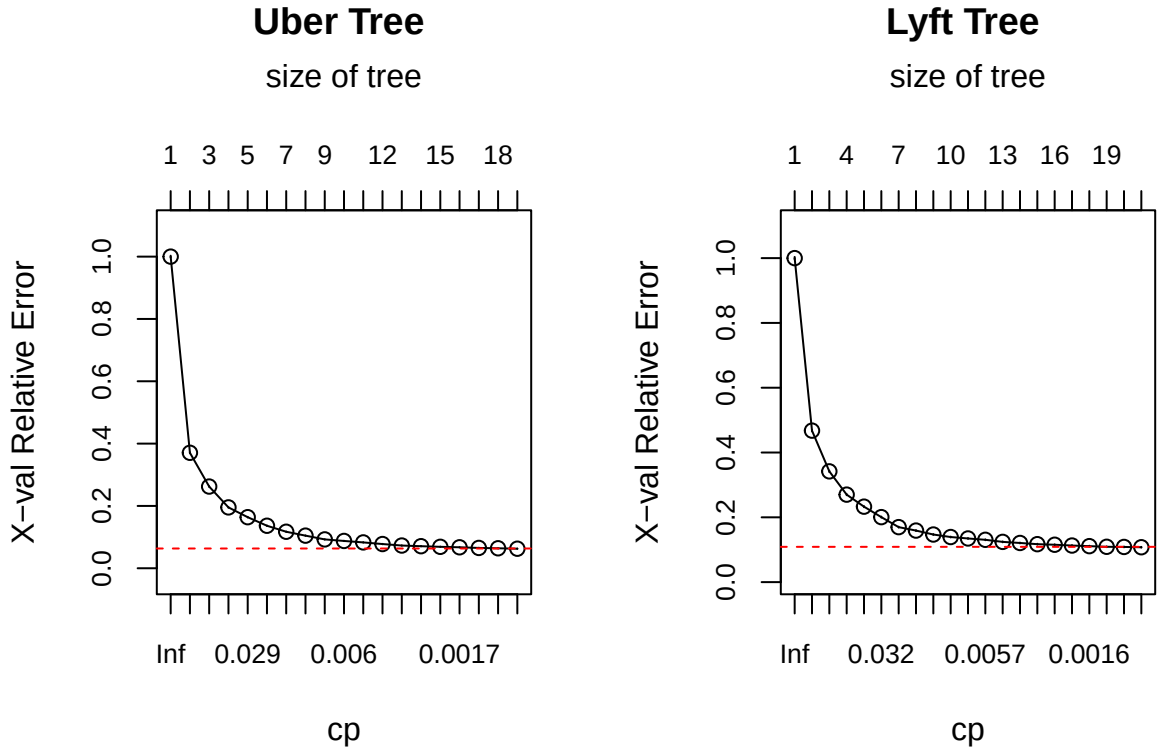
```
# Regression Trees for Lyft
lyft_tree1 <- rpart(price ~., data = lyft_train2, method = "anova", cp =
 ↪  0.001)
printcp(lyft_tree1)
```

```
##
## Regression tree:
## rpart(formula = price ~ ., data = lyft_train2, method = "anova",
##     cp = 0.001)
##
## Variables actually used in tree construction:
## [1] distance product
##
## Root node error: 24745918/245929 = 100.62
##
## n= 245929
##
##           CP nsplit rel error  xerror      xstd
## 1  0.5324369      0   1.00000 1.00002 0.0037589
## 2  0.1257671      1   0.46756 0.46757 0.0022942
## 3  0.0717694      2   0.34180 0.34181 0.0021681
## 4  0.0373906      3   0.27003 0.27004 0.0018588
## 5  0.0323078      4   0.23264 0.23291 0.0017993
## 6  0.0309891      5   0.20033 0.20060 0.0016901
## 7  0.0106246      6   0.16934 0.16971 0.0014580
## 8  0.0085023      7   0.15871 0.15909 0.0014544
## 9  0.0077001      8   0.15021 0.14694 0.0013957
## 10 0.0073729      9   0.14251 0.13918 0.0013218
## 11 0.0069279     10   0.13514 0.13472 0.0013184
```

```
## 12 0.0046536      11    0.12821 0.13061 0.0013036
## 13 0.0030458      12    0.12356 0.12422 0.0013006
## 14 0.0030385      13    0.12051 0.12081 0.0012933
## 15 0.0025574      14    0.11747 0.11713 0.0012836
## 16 0.0024441      15    0.11492 0.11540 0.0012816
## 17 0.0018894      16    0.11247 0.11296 0.0012818
## 18 0.0013650      17    0.11058 0.11107 0.0012740
## 19 0.0012804      18    0.10922 0.10907 0.0012715
## 20 0.0011651      19    0.10794 0.10859 0.0012712
## 21 0.0010000      20    0.10677 0.10765 0.0012706
```

Again, the decision tree model only chooses the distance and product features, but for Lyft, it creates 20 splits and 21 nodes in total. The following figure demonstrates the reduction in cross-validated errors for both models as the value of CP decreases:

Figure 6. Cross−validated errors and complexity parameter



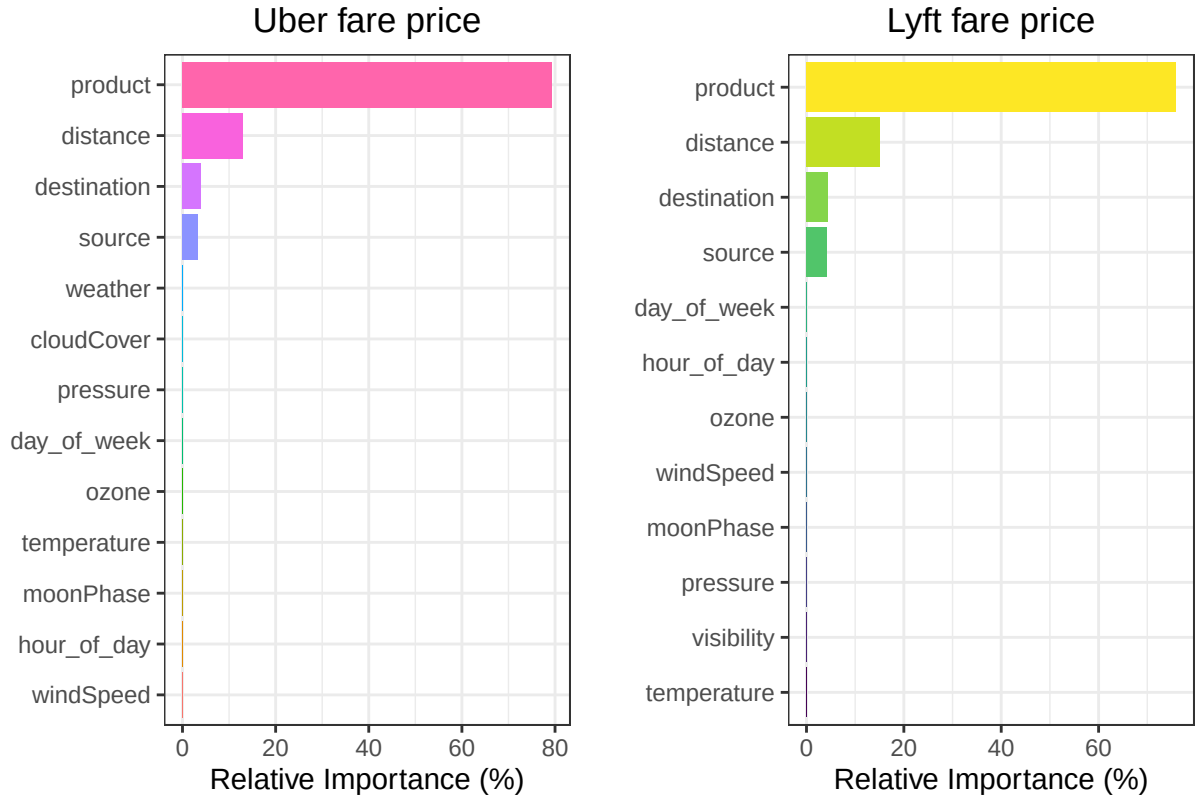

```
# Pruned tree
best_cp_uber <-
 ↪  uber_tree1$cptable[which.min(uber_tree1$cptable[,"xerror"]),"CP"]
best_cp_lyft <-
 ↪  lyft_tree1$cptable[which.min(lyft_tree1$cptable[,"xerror"]),"CP"]
best_cp_uber==best_cp_lyft
```

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

The complexity parameter values that generates the lowest cross-validated error for uber and lyft prices are both `0.001`.

Next, I want to examine the relative importance of the features as determined by the decision tree model. In decision trees, a variable may appear in the tree many times, either as a primary variable in the root node, or as a surrogate variable in the decision node. "An overall measure of variable importance is the sum of the goodness of split measures for each split for which it was the primary variable, plus goodness * (adjusted agreement) for all splits in which it was a surrogate" (Therneau & Atkinson, 2022). In regression trees, the goodness of split measure is often the gap between the observations and the model predictions. This measure is then scaled to sum up to 100%, representing the proportion of influence each variable has on the dependent variable. The figure below helps visualize these importance measures between features:

Figure 7. Variable importance from regression trees



From figure 5, we can see that the regression trees determine product is the most significant factor affecting fare prices, followed by distance, destination and source for both Uber and Lyft. Other features seem to have an inconsiderable impact on the price of the ride-sharing service.

3.2. Decision tree model performance

After having constructed the regression trees, I then use these models to generate pricing predictions using both the training and testing sets of Uber & Lyft. The table below

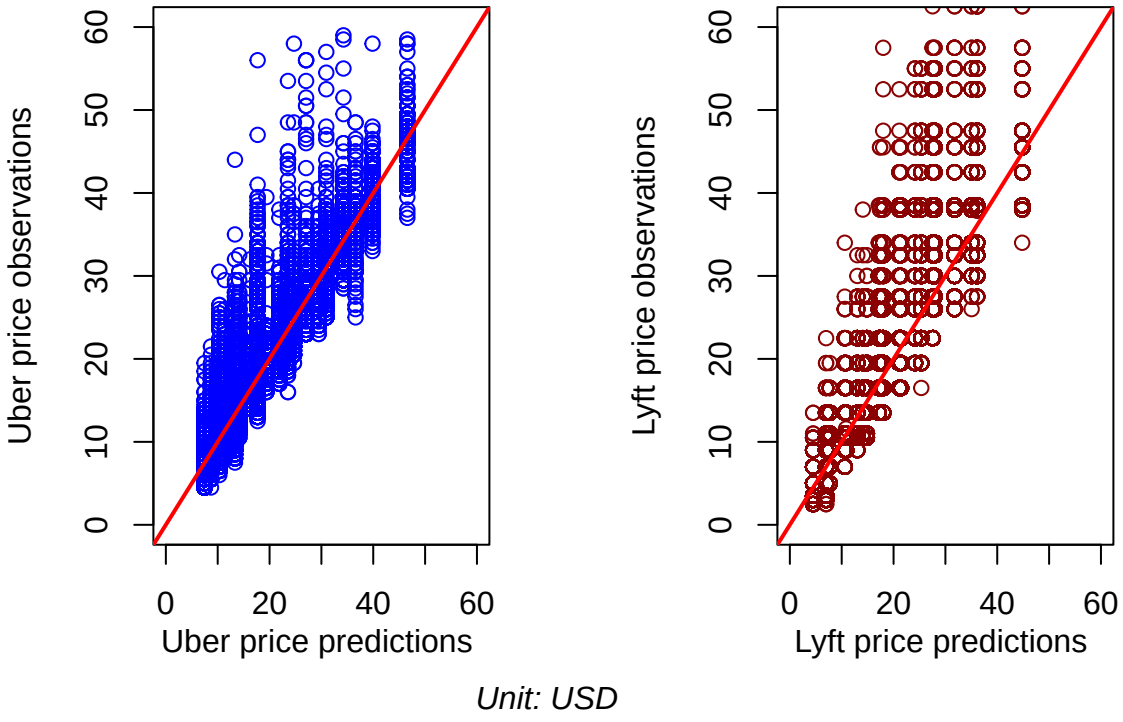
summarizes the predictive accuracy of the model outputs:

Table 8. Regression tree model predictive performance between training and testing

	Uber Training	Uber Testing	Lyft Training	Lyft Testing
RMSE	1.439	3.831	1.601	4.326
MAE	4.358	3.625	1.212	4.056
MAPE	0.099	0.260	0.131	0.284
R-Squared	0.821	0.820	0.887	0.888

From table 8, we can observe that the decision tree models have been able to generate higher r-squared values when predicting both Uber and Lyft prices, along with lower overall RMSE and MAPE. The testing predictions also show some improvements in accuracy compared to the training ones since R-squared slightly increases but MAPE remains similar or lower. The following scatterplots help visualize the predictions fitness with the observations in fare prices:

Figure 8. Rideshare fare price observations vs regression tree predictions



From figure 8, we see that the points are not too densely scattered around the diagonal red lines, similar to what we have seen in figure 4. This represents that the pricing predictions made by the regression trees, despite the progress made versus the linear regression method, are still quite far off from the actual observations.

4. Predict whether price will surge using logistic regression model

In this section we will focus on a new target variable "surge_multipler". From my understanding, the actual price = standard price * surge multiplier. Thus, this variable determines whether the actual price will be surged (multiplier > 1) or not (multiplier = 1). First, we examine the distribution of "surge_multipler" across the 2 companies Lyft & Uber:

```
##
##              1   1.25    1.5   1.75      2    2.5      3
##   Lyft 286433  11085   5065   2420   2239    154     12
##   Uber 330568      0      0      0      0      0      0
```

From here, we can see that the surge multiplier only seems to exist for Lyft, and thus, we will focus only only predicting price surge for Lyft with this data set. In addition, the value of "surge_multipler" = 1 accounts for majority of the data, while the highest surge is 3 times the standard price. Next, we subset the original data set into a smaller set containing only Lyft's data and create a new target categorical feature "lyft_surge" based on the numerical values of "surge_multipler":

```
# Subset original data for only Lyft's data
df_lyft_surge <- df_ride4[df_ride4$cab_type=="Lyft",c(char_vars,pick_num,
                                                      "day_of_week",
                                                      "surge_multiplier")]
df_lyft_surge <- df_lyft_surge[,!names(df_lyft_surge) %in% "cloudCover"]
df_lyft_surge$cab_type <- NULL

# Create new target variable lyft_surge and remove irrelevant ones
df_lyft_surge$price_surge <- if_else(df_lyft_surge$surge_multiplier > 1,1,0)
char_vars_lyft <- names(select_if(df_lyft_surge,is.character))
df_lyft_surge$surge_multiplier <- NULL
df_lyft_surge$price <- NULL

# Convert categorical to factor
for (i in char_vars_lyft){
  df_lyft_surge[,i] <- as.factor(df_lyft_surge[,i])
}

# Scale numerical features
num_vars_lyft <- names(select_if(df_lyft_surge,is.numeric))
num_vars_lyft <- num_vars_lyft[!num_vars_lyft %in% c("price_surge")]
df_lyft_surge2 <- df_lyft_surge
df_lyft_surge2[num_vars_lyft] <- scale(df_lyft_surge[num_vars_lyft])

# Overview of current data set
str(df_lyft_surge2, width = 70, strict.width = "cut")
```

```
## 'data.frame':    307408 obs. of  13 variables:
##  $ source     : Factor w/ 12 levels "Back Bay","Beacon Hill",..: 6 6..
##  $ product    : Factor w/ 6 levels "Lux","Lux Black",..: 6 1 4 3 5 2..
##  $ weather    : Factor w/ 7 levels "clear-day","clear-night",..: 6 7..
##  $ distance   : num  -1.61 -1.61 -1.61 -1.61 -1.61 ...
##  $ temperature: num  0.408 0.592 -0.188 -0.775 -0.32 ...
##  $ windSpeed  : num  0.79 1.846 0.367 -0.285 0.943 ...
##  $ visibility : num  0.587 -1.418 0.587 0.587 0.587 ...
##  $ pressure   : num  0.88 -0.454 -1.321 0.269 -0.87 ...
##  $ ozone      : num  -0.3471 -0.8015 0.0787 -0.8015 1.2235 ...
##  $ moonPhase  : num  -1.142 0.247 0.411 0.697 0.574 ...
##  $ hour_of_day: num  -0.375 -1.453 -1.596 -1.039 -1.192 ...
##  $ day_of_week: Factor w/ 7 levels "Monday","Tuesday",..: 7 2 3 5 4 ..
##  $ price_surge: num  0 0 0 0 0 0 0 0 0 0 ...
```

4.1. Remove class bias with undersampling

The current distribution of the new target feature "price_surge" is as follow:

```
##
##      0      1
## 286433  20975
```

We can see that the classes of the dependent variable is currently unbalanced, with the majority being "1" compared to "0". Thus, we use the undersampling method in order to balance out the classes:

```
# Balance classes of dependent variable
df_surge_freq <-
 ↪  as.data.frame(t(as.data.frame(table(df_lyft_surge2$price_surge))))
rownames(df_surge_freq) <- c("Price-Surge categories","Frequency")
colnames(df_surge_freq) <- c("Not-surge","Surge")
df_surge_freq$`Not-surge` <- as.numeric(df_surge_freq$`Not-surge`)
df_surge_freq$Surge <- as.numeric(df_surge_freq$Surge)

df_surge <- df_lyft_surge2[df_lyft_surge2$price_surge == 1,]
df_nosurge <- df_lyft_surge2[df_lyft_surge2$price_surge == 0,]

no_bias_row <- min(nrow(df_surge),nrow(df_nosurge))
private_yes_nobias <- head(df_nosurge,no_bias_row)
df_col_equal <- rbind(private_yes_nobias,df_surge)
table(df_col_equal$price_surge)
```

```
##
##     0     1
## 20975 20975
```

4.2. Logistic regression model construction

Now that the classes are balanced, we can partition the data set for training and testing and fit a stepwise logistic regression model to predict whether price will surge:

```
# Partition unscaled data
set.seed(888)
surge_trainRatio <-
 ↪  createDataPartition(df_col_equal$price_surge,p=0.8,list=F,times=1)
surge_train <- df_col_equal[surge_trainRatio,]
surge_test <- df_col_equal[-surge_trainRatio,]

# Stepwise logistic regression model
logit_model <- glm(price_surge~., data=surge_train,
 ↪  family=binomial(link="logit")) %>% stepAIC(trace = FALSE)
```

```
##                                    Estimate  Std. Error Pr(>|z|)
## (Intercept)                     9.274445e-01  0.07501516  0.00000
## sourceBeacon Hill              -5.301758e-01  0.05367157  0.00000
## sourceBoston University        -1.883201e-01  0.05436265  0.00053
## sourceFenway                   -1.896376e-01  0.05324627  0.00037
## sourceFinancial District       -8.626058e-01  0.05563884  0.00000
## sourceHaymarket Square         -2.151052e+00  0.07329616  0.00000
## sourceNorth End                -2.174090e+00  0.07333490  0.00000
## sourceNorth Station            -1.430932e+00  0.06097032  0.00000
## sourceNortheastern University -3.886303e-02  0.05430781  0.47423
## sourceSouth Station            -6.631493e-01  0.05545043  0.00000
## sourceTheatre District         -2.884785e-01  0.05319799  0.00000
## sourceWest End                 -1.275511e+00  0.05956656  0.00000
## productLux Black                1.607656e-02  0.03819933  0.67386
## productLux Black XL            -3.382326e-04  0.03812787  0.99292
## productLyft                    -1.804124e-03  0.03817516  0.96231
## productLyft XL                  2.341818e-02  0.03823017  0.54017
## productShared                  -1.764978e+01 66.99726136  0.79221
## weatherclear-night             -4.519911e-02  0.07369042  0.53964
## weathercloudy                  -1.495419e-01  0.06517483  0.02176
## weatherfog                     -2.314925e-02  0.11582984  0.84159
## weatherpartly-cloudy-day       -7.752722e-02  0.06818167  0.25551
## weatherpartly-cloudy-night     -7.894924e-02  0.06651912  0.23528
## weatherrain                    -1.387189e-01  0.06909782  0.04469
```

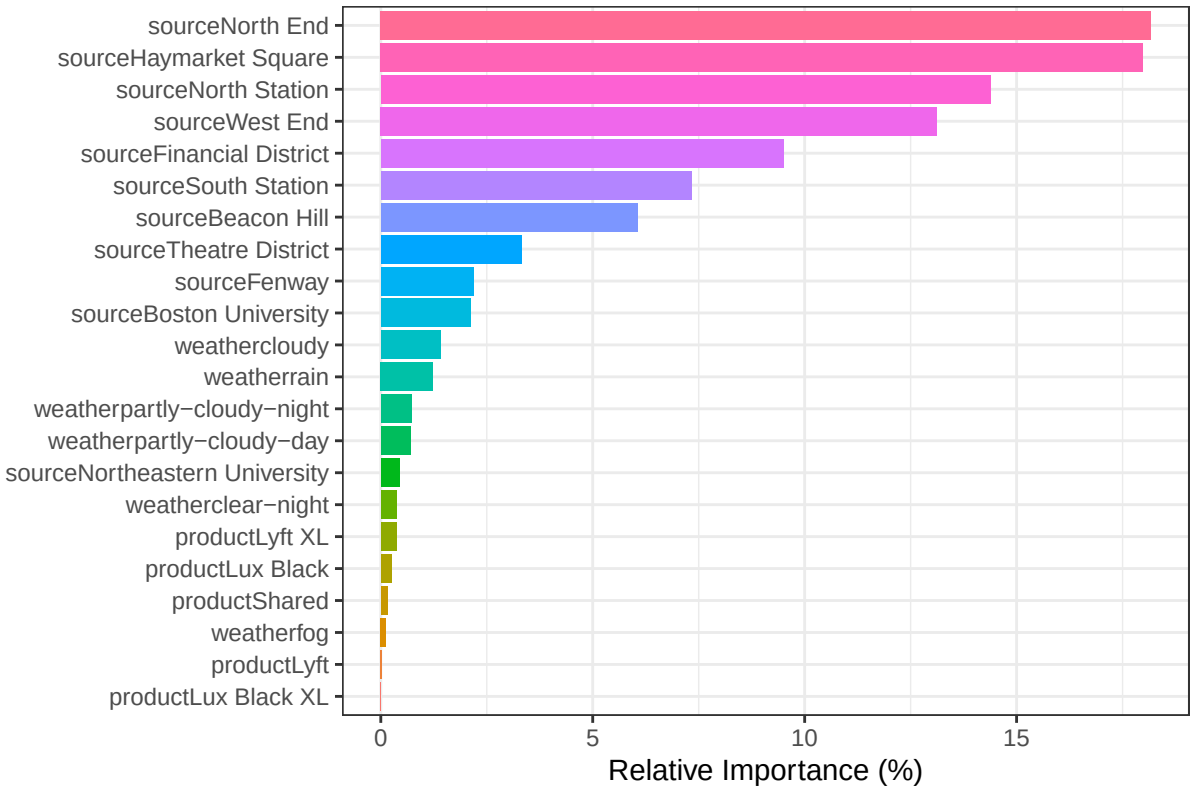
Similar to linear regression, we check the VIF values to see if there are any signs of multicollinearity issues.

```
# Check VIF values
vif(logit_model)
```

```
##             GVIF Df GVIF^(1/(2*Df))
## source  1.004162 11        1.000189
## product 1.000835  5        1.000083
## weather 1.004377  6        1.000364
```

Since all VIFs are less than 5, we can be fairly confident that there are no correlated features in the model and move on with our analysis. Based on the model output, we can observe that most of the independent predictors are deemed significant (based on their P-values) except for "product" feature. To better visualize this information, the variable importance calculated for each feature is plotted below:

Figure 9. Variable importance of logistic regression model



From figure 9, we can determine that the source locations of the rides, along with weather conditions, are the most influential factors affecting whether prices will surge or not. Specifically, rides that start from North End, Haymarket Square, North Station, West End and Financial District are among the top 5 locations that are most likely to experience price surges. This makes sense since all of these locations are often heavily-crowded, which could cause the demand for ride-shares to temporarily outnumbers the supply of drivers at these particular locations during high traffic hours. In addition, cloudy and rainy weathers may also contribute a small part in the surging of prices due to unfavorable road conditions.

4.3. Logistic regression model performance

After fitting the training set, the next step is to generate predictions for the testing set. The training and testing results from the logistic regression model are presented below:

```
# Generate predictions using the training set
logit_predict_train <- predict(logit_model, surge_train, type="response")

# Generate predictions using the testing set
logit_predict_test <- predict(logit_model, surge_test, type="response")
```

- The training results:

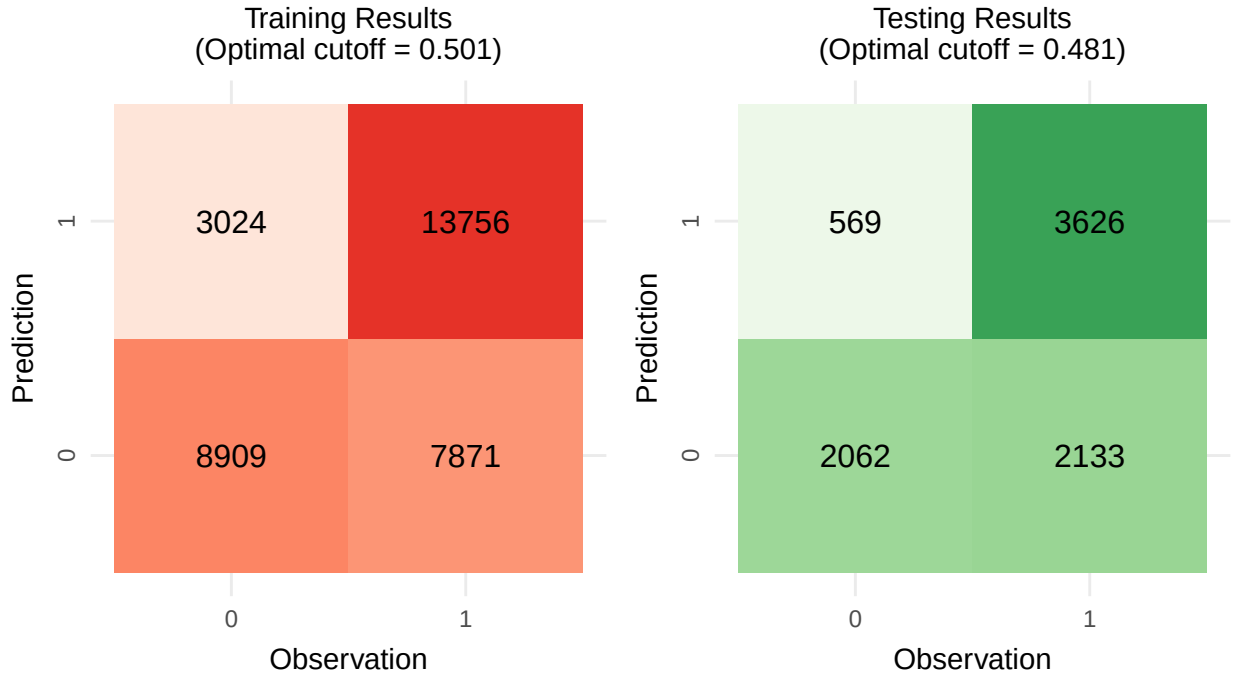
```
##          Sensitivity          Specificity       Pos Pred Value
##            0.6360568            0.7465851            0.8197855
##       Neg Pred Value            Precision               Recall
##            0.5309297            0.8197855            0.6360568
##                   F1           Prevalence       Detection Rate
##            0.7163278            0.6444279            0.4098927
## Detection Prevalence    Balanced Accuracy
##            0.5000000            0.6913209
```

- The testing results:

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.6296232            0.7837324            0.8643623
##       Neg Pred Value            Precision               Recall
##            0.4915375            0.8643623            0.6296232
##                   F1           Prevalence       Detection Rate
##            0.7285513            0.6864124            0.4321812
## Detection Prevalence    Balanced Accuracy
##            0.5000000            0.7066778
```

- Both results are better visualized using the 2 confusion matrices below:

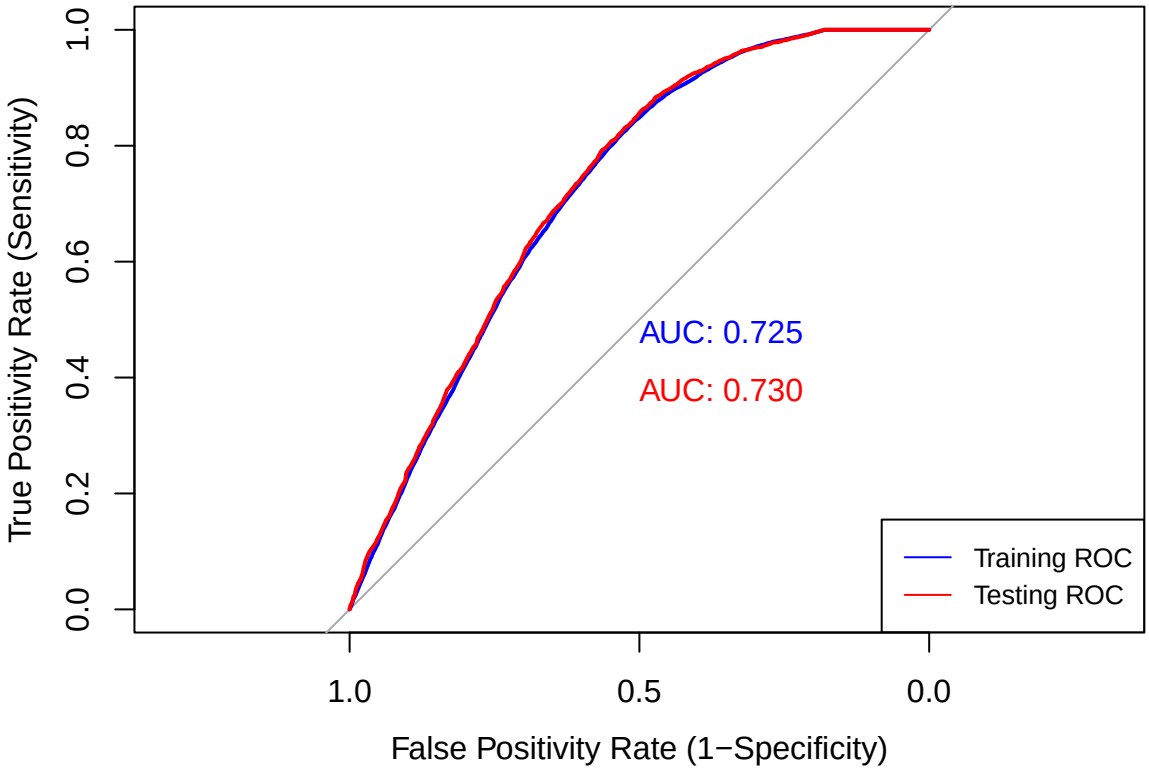
Figure 10. Confusion matrix from the training and testing predictions



The optimal cutoff values that generate the highest accuracy are employed in this model, where the cutoff point is `0.501` for the training set and `0.481` for the testing set. From there, we can see that for the training model, the prediction accuracy is `67.54%` while the accuracy for the testing set is slightly higher, achieving `67.79%`. In addition, the majority of the misclassifications stem from Type 2 errors (or False Negatives cases), meaning that our model tends to predict that price does not surge when in reality, it does. This shows that our logistic regression model is better at predicting when the price will surge compared to predicting when it will not surge.

Finally, let's examine the Area Under the Curve (AUC) values of each model. Since the AUC indicates how strong the model is at distinguishing between classes compared to making random 50-50 predictions, we can compare the true predictive power of these models by looking at their ROC curves:

Figure 11. ROC Curves from the training and testing set



From figure 11, we can see that our testing model performs slightly better than our training model in terms of AUC (`0.73` vs `0.725`), showing that our logistic regression model does not overfit with the training set and can work relatively well with newly introduced data.

III. CONCLUSION

Based on the analysis of this data set, I have learned of a lot about the various factors influencing fare prices of Uber and Lyft. All in all, I found that the product type, the trip distance, and starting location of the ride accounts for the heaviest influence on fare prices for both Uber & Lyft. In addition, the weather conditions are mostly insignificant factors in terms of setting the ride prices. When constructing prediction models to forecast price, I see that the decision tree model performs relatively better than the linear regression model. Although both models manage to generate quite good R-square measures (around 0.8 to 0.9), the predictions generated are still not close enough to the actual observations to be deem meaningful. However, I still believe I have been able to discover the features impacting fare prices the most after conducting my analysis.

As for the classification model created to predict whether price will surge, I have determined that "source" (starting point) and "weather" are the 2 main influences on the surge in price. If users book a ride from North End, Haymarket Square, or South Station, which are quite densely-populated central areas of Boston, they are likely to pay more for the rides

due to the price surge mechanism. Moreover, if the weather is either cloudy or rainy, the price will also be likely to surge as well, and like we mentioned earlier, the product seems to have no impact on whether the price will surge.

Overall, I think these are some interesting insights that we found after analyzing 2 different target features, "price" and "price surge". As a customer, I will focus on these factors to reduce the ride fares whenever possible, such as using shared Uber/Lyft services, travel short distance and choose pick-up location at heavily populated areas to increase the likelihood of having sufficient numbers of Uber/Lyft drivers around.

IV. REFERENCES

Uber and Lyft Dataset Boston, MA | Kaggle. (n.d.). Kaggle: Your Machine Learning and Data Science Community. Retrieved December 14, 2022, from https://www.kaggle.com/datasets/brllrb/uber-and-lyft-dataset-boston-ma

Therneau, T. M., & Atkinson, E. J. (2022, January 24). *An Introduction to Recursive Partitioning using the RPART Routines.* The Comprehensive R Archive Network. Retrieved December 14, 2022, from https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf