IDS575 Project: Final report

Ashish Menkudale (UIN 656130575)

2017-12-01

Contents

List of Figures

List of Tables

Note

- Take an advantage of the cool interactivity of rmarkdown.
  - In the *.html* version of this report, you can navigate through different sections via interactive dynamic index generated on the left side of this page.
  - Also, to see the specific section of the code in the *.html* version, you can click on '*code*' button on the right side at that particular chunk.
  - Or else, you can click the '*code*' button at the very begining of this *.html* version of the report to expand all the code sections.
  - In the *.pdf* version of this report, you can take advantage of the interactive index (Table of content), list of figures, and list of tables as well. Elements in these indices are hyperlinked.
  - Few objects created using '*leaflet*' package are hashed in the *.pdf* version. These objects (graphs, maps) are highly interactive and those cannot be knitted in the *.pdf* version.
- The progress made from mid report can be summarized as further.
  - In feature engineering section, weather related variables are added (section 5.4). These are derived from external dataset.
  - All reference links are mentioned at the appropriate places.
  - xgboosting parameters are optimized. **Tweedie distribution regression** is used. More details are in the respective section.
- To run the .rmd file at your end or to knit it into either *.pdf* or *.html* format, please do further things,
  - change the paths in the code chunks number (**6** - importing initial train and test datasets and **35** - importing external weather dataset).
  - Keep the *fig_pox.tex* file in the working directory.
  - I would recommend switching to '*xelatex*' engine for pdf conversion. (The settings symbol on .rmd file in rstudio > ouput options > advanced > Latex Engine > *xelatex*).
  - From the *MikTeX package manager (Admin)*, please uninstall the outdated *fontspec* package and reinstall updated one.

1 Project report details

This report summarizes progress made with the project so far and important details of this project along statistical models built for the prediction.

Building on to the progress made with exploring rmarkdown features during the assignments, I knitted this webpage and pdf version of this report using the rmakrdown. R markdown is truely amazing tool to present the data. Many times simple bits, code chunks, and knitting different objects produced errors which were never seen before. It took patience and at times I gave up. But eventually, with learning all the ways around a problem, I was able to knit these formats. If you are trying to knit the *.rmd* file at your end, I would recommend switching to '*xelatex*' engine. (the settings symbol on .rmd file in rstudio > ouput options > advanced).

Also with this project, I got proficient in one of the important library called **dplyr**. Its syntax is easy to understand and construct. Throughout the report, it has been extensively used.

Recalling to **the first report** (attached as the appendix at the very end) submitted earlier on blackboard, the goal is to predict *trip_duration*s for taxis in New York City given input features like pickup coordinates, time of the day, vendor id etc. The conventional set of hypothesis is considered for the statistical significance of each variable used for building the model. if β_j for each feature holds the statistical significance. At the end of the model we will test this hypothsis using *feature_importance()*.

Also, in the *first report* we discussed the idea of the using integration of classification and regression technique. Accordingly, I preferred starting with the **eXtreme Gradient Boosting** for the prediction. With this ensemble strategy, linear regression analysis is applied for individual models.

I learned and implemented **feature engineering** *section 5* for the very first time. This is a highlight for me in this project. This helped me to understand how the solutions for the real world data science problem can be derived from *thinking outside the box*.

For majority of the project, different ideas and concepts are integrated after studying various approaches to the problem statement mentioned on the competition webpage. The extensive study helpedd me to develope more perspectives while solving the model. This report includes all the references and links mentioned right where they are used. The major ideas of exploratory data analysis and feature engineering are opted from *here*.

I tried my best to complete this project in the utmost professional manner, labeling each code chunks in the *.rmd* file, labeling each image and each table, defining margins for figures, and defining number of columns while reporting tables. These all things are tanken care through ramrkdown features. Small macros from latex are also imported while resolving knitting issues for example, holding the figures in position after rendering into pdf file or keeping the text wrapped inside the code chunks etc..

Regarding the **ambitious project**, I have attached the pictures of the progress. I started reading the paper and understood almost 60% of the mathematics at this stage. The relevant information is mentioned on **appendix 2** of this report.

2 Details of the data set

Recalling the details of the project, the objective is to predict the *trip_duration* for taxis given further input variables.

- *id* - a unique identifier for each trip
- *vendor_id* - a code indicating the provider associated with the trip record
- *pickup_datetime* - date and time when the meter was engaged
- *dropoff_datetime* - date and time when the meter was disengaged
- *passenger_count* - the number of passengers in the vehicle (driver entered value)
- *pickup_longitude* - the longitude where the meter was engaged
- *pickup_latitude* - the latitude where the meter was engaged
- *dropoff_longitude* - the longitude where the meter was disengaged
- *dropoff_latitude* - the latitude where the meter was disengaged
- *store_and_fwd_flag* - This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip

Let's set working directory and java environment before we begin.

```
# setting working directory
setwd("C:/Users/Dell/OneDrive/UIC/Fall_2017/IDS 575/IDS575_pro1")
# setting java environment for Rjava and Rweka
Sys.setenv(JAVA_HOME = "C:/Program Files/Java/jre8")
```

Let's import necessary libraries before we start.

```
# {r import libraries, echo =TRUE, message = FALSE, warning=FALSE,
# results='hide'} Above setting is used to evalute this code chunk, include this
# chunk into report but not to include the results.

# Importing necessary libraries.
```

```
library("ggplot2")  # visualisation
library("scales")  #date_format (used in scale_x_datetime{ggplot})
library("corrplot")  # visualisation of correlation
library("alluvial")  # visualisation
library("ggmap")  # visualization
library("geosphere")  # geospatial locations
library("leaflet")  # maps
library("leaflet.extras")  # maps
library("maps")  # maps
library("dplyr")  # data processing
library("readr")  # input/output
library("data.table")  # data processing
library("tibble")  # data wrangling
library("tidyr")  # data wrangling
library("dplyr")  # data preprocessing
library("stringr")  # string processing
library("forcats")  # factor processing
library("lubridate")  # date and time
library("xgboost")  # library(geojsonio) #Not available in Kaggle
library("caret")  # modelling
library("Metrics")  # for functions related to accuracy
library("caret")  # set of functions to streamline creating predictive models.
library("lars")  # Lasso
library("sf")  # Data processing
library("sp")  # Data processing with 2D 3D data
library("forecast")  # ARIMA in R
library("quantmod")  # data analysis package
library("pander")  # pretty outputs and table formatting
library("kableExtra")  #formatting
library("knitr")  # format conversion and rendering of this report.
library("stargazer")  # to create latex code
library("bitops")  # Used in decodeLine()
library("grid")  #graphics layout capabilities
library("car")  #for qqplot
library("qqplotr")
library("fitdistrplus")
```

Let's load the data using tibble library and see what we are dealing with here.

```
# read data with tibble.
# https://cran.r-project.org/web/packages/tibble/vignettes/tibble.html tibble()
# is a nice way to create data frames. It encapsulates best practices for data
# frames: It never changes an input's type (i.e., no more stringsAsFactors =
# FALSE!).  This makes it easier to use with list-columns.  It never adjusts the
# names of variables.  It evaluates its arguments lazily and sequentially: It
# never uses row.names().  The whole point of tidy data is to store variables in
# a consistent way.  So it never stores a variable as special attribute.  It only
# recycles vectors of length 1.  This is because recycling vectors of greater
# lengths is a frequent source of bugs.

train <- as.tibble(fread("C:/Users/Dell/OneDrive/UIC/Fall_2017/IDS 575/IDS575_pro1/train.csv"))
test <- as.tibble(fread("C:/Users/Dell/OneDrive/UIC/Fall_2017/IDS 575/IDS575_pro1/test.csv"))
sample_submit <- as.tibble(fread("C:/Users/Dell/OneDrive/UIC/Fall_2017/IDS 575/IDS575_pro1/sample_submis
```

```
# It prints pretty outputs as Read 1458644 rows and 11 (of 11) columns from 0.187
# GB file in 00:00:13 Read 625134 rows and 9 (of 9) columns from 0.066 GB file in
# 00:00:05
```

```
# summary of training dataset in representable format with pander.
pander(summary(train), caption = "Summary of the training data set", digits = 4)
```

Table 1: Summary of the training data set (continued below)

id	vendor_id	pickup_datetime	dropoff_datetime
Length:1458644	Min. :1.000	Length:1458644	Length:1458644
Class :character	1st Qu.:1.000	Class :character	Class :character
Mode :character	Median :2.000	Mode :character	Mode :character
NA	Mean :1.535	NA	NA
NA	3rd Qu.:2.000	NA	NA
NA	Max. :2.000	NA	NA

Table 2: Table continues below

passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude
Min. :0.000	Min. :-121.93	Min. :34.36	Min. :-121.93
1st Qu.:1.000	1st Qu.: -73.99	1st Qu.:40.74	1st Qu.: -73.99
Median :1.000	Median : -73.98	Median :40.75	Median : -73.98
Mean :1.665	Mean : -73.97	Mean :40.75	Mean : -73.97
3rd Qu.:2.000	3rd Qu.: -73.97	3rd Qu.:40.77	3rd Qu.: -73.96
Max. :9.000	Max. : -61.34	Max. :51.88	Max. : -61.34

dropoff_latitude	store_and_fwd_flag	trip_duration
Min. :32.18	Length:1458644	Min. : 1
1st Qu.:40.74	Class :character	1st Qu.: 397
Median :40.75	Mode :character	Median : 662
Mean :40.75	NA	Mean : 959
3rd Qu.:40.77	NA	3rd Qu.: 1075
Max. :43.92	NA	Max. :3526282

```
# caption the table, later to be used in list of tables. digits is for number of
# columns in one row. also we can add attribute, style = grid.
```

```
print(paste0("training data at a glimpse"))
```

```
## [1] "training data at a glimpse"
```

```
glimpse(train)
```

```
## Observations: 1,458,644
## Variables: 11
## $ id                <chr> "id2875421", "id2377394", "id3858529", "id3...
```

```
## $ vendor_id          <int> 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2...
## $ pickup_datetime    <chr> "2016-03-14 17:24:55", "2016-06-12 00:43:35...
## $ dropoff_datetime   <chr> "2016-03-14 17:32:30", "2016-06-12 00:54:38...
## $ passenger_count    <int> 1, 1, 1, 1, 1, 6, 4, 1, 1, 1, 1, 4, 2, 1, 1...
## $ pickup_longitude   <dbl> -73.98215, -73.98042, -73.97903, -74.01004,...
## $ pickup_latitude    <dbl> 40.76794, 40.73856, 40.76394, 40.71997, 40....
## $ dropoff_longitude  <dbl> -73.96463, -73.99948, -74.00533, -74.01227,...
## $ dropoff_latitude   <dbl> 40.76560, 40.73115, 40.71009, 40.70672, 40....
## $ store_and_fwd_flag <chr> "N", "N", "N", "N", "N", "N", "N", "N", "N"...
## $ trip_duration      <int> 455, 663, 2124, 429, 435, 443, 341, 1551, 2...
```

Let's see the testing data.

```
# summary of testing dataset in representable format with pander.
pander(summary(test), caption = "Summary of the testing data set", digits = 4)
```

Table 4: Summary of the testing data set (continued below)

id	vendor_id	pickup_datetime	passenger_count
Length:625134	Min. :1.000	Length:625134	Min. :0.000
Class :character	1st Qu.:1.000	Class :character	1st Qu.:1.000
Mode :character	Median :2.000	Mode :character	Median :1.000
NA	Mean :1.535	NA	Mean :1.662
NA	3rd Qu.:2.000	NA	3rd Qu.:2.000
NA	Max. :2.000	NA	Max. :9.000

Table 5: Table continues below

pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
Min. :-121.93	Min. :37.39	Min. :-121.93	Min. :36.60
1st Qu.: -73.99	1st Qu.:40.74	1st Qu.: -73.99	1st Qu.:40.74
Median : -73.98	Median :40.75	Median : -73.98	Median :40.75
Mean : -73.97	Mean :40.75	Mean : -73.97	Mean :40.75
3rd Qu.: -73.97	3rd Qu.:40.77	3rd Qu.: -73.96	3rd Qu.:40.77
Max. : -69.25	Max. :42.81	Max. : -67.50	Max. :48.86

store_and_fwd_flag
Length:625134
Class :character
Mode :character
NA
NA
NA

```
print(paste0("testing data at a glimpse"))
```

```
## [1] "testing data at a glimpse"
```

```
# summary of testing dataset in representable format with pander.
glimpse(test)
```

```
## Observations: 625,134
## Variables: 9
## $ id                <chr> "id3004672", "id3505355", "id1217141", "id2...
## $ vendor_id         <int> 1, 1, 1, 2, 1, 1, 1, 1, 2, 2, 1, 2, 1, 2, 1...
## $ pickup_datetime   <chr> "2016-06-30 23:59:58", "2016-06-30 23:59:53...
## $ passenger_count   <int> 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 4, 1, 1, 1, 1...
## $ pickup_longitude  <dbl> -73.98813, -73.96420, -73.99744, -73.95607,...
## $ pickup_latitude   <dbl> 40.73203, 40.67999, 40.73758, 40.77190, 40....
## $ dropoff_longitude <dbl> -73.99017, -73.95981, -73.98616, -73.98643,...
## $ dropoff_latitude  <dbl> 40.75668, 40.65540, 40.72952, 40.73047, 40....
## $ store_and_fwd_flag <chr> "N", "N", "N", "N", "N", "N", "N", "N", "N"...
```

According to avaiable information from glimpses and summaries for training and test data we observed above, we can cross verify,

- *vendor_id* is a categorical variable with two levels. 1 and 2.
- *Passenger_count* is also a categorical variable. with two levels Y and N.
- *pickup_datetime, gropoff_datetime* are in datetime format.
- *pickup_longitude*, *pickup_latitude*, *dropoff_latitude* and *dropoff_longitude* are location related variables.
- *passanger_count* ranges from 1 to 9.
- *trip_duration* is our response variable.

3 Data preprocessing

Data preprocessing is an important stage in data analysis. Before fitting any model, it is required that the data is checked for consistency and dirtiness. If required, variables should be converted to suitable datatypes and values should be scaled or transformed before building the model. Missing values and outliers should be taken care of to avoid capturing inherent errors in the model.

3.1 Handling missing data

Before we start any data processing, let's check if there are any missing values in the data. We found that,

```
# Checking for missing values using 'is.na'.
print(paste0("number of missing datapoints in training dataset are ", sum(is.na(train)),
    " and", " number of missing datapoints in testing dataset are ", sum(is.na(test))))
```

```
## [1] "number of missing datapoints in training dataset are 0 and number of missing datapoints in testi
```

Well, this is good thing that there's no missing data in dataset.

3.2 Combining train & test dataset

For ease in operations, and to avoid repeatations of all operations on datapoints, we will first combine training and testing dataset and then perform all the necessary data transformations. Once it is complete, we will split it again for building predictive models. We will store this data into a single dataframe called *combine*. Let's have a look at *combine*,

```
# Using dplyr create a combined dataframe of train and test data points named
# 'combine'.  while doing that, create a new column which takes two values as
# 'train' and 'test'.  If the datapoint was from train dataset, the the value of
# variable dset is 'train'. Else 'test' for that datapoint.  For all the test
# datapoints, put 'NA' as value in columns 'dropoff_time' and 'trip_duration'.
# Using dplyr makes it easy.  %>% operator adds operations in the single command.
# mutate is used for creating new variables.

combine <- bind_rows(train %>% mutate(dset = "train"), test %>% mutate(dset = "test",
    dropoff_datetime = NA, trip_duration = NA))

# chaning the data type of attribute to factor with two levels train and test.
combine <- combine %>% mutate(dset = factor(dset))

glimpse(combine)
```

```
## Observations: 2,083,778
## Variables: 12
## $ id                <chr> "id2875421", "id2377394", "id3858529", "id3...
## $ vendor_id         <int> 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2...
## $ pickup_datetime   <chr> "2016-03-14 17:24:55", "2016-06-12 00:43:35...
## $ dropoff_datetime  <chr> "2016-03-14 17:32:30", "2016-06-12 00:54:38...
## $ passenger_count   <int> 1, 1, 1, 1, 1, 6, 4, 1, 1, 1, 1, 4, 2, 1, 1...
## $ pickup_longitude  <dbl> -73.98215, -73.98042, -73.97903, -74.01004,...
## $ pickup_latitude   <dbl> 40.76794, 40.73856, 40.76394, 40.71997, 40....
## $ dropoff_longitude <dbl> -73.96463, -73.99948, -74.00533, -74.01227,...
## $ dropoff_latitude  <dbl> 40.76560, 40.73115, 40.71009, 40.70672, 40....
## $ store_and_fwd_flag <chr> "N", "N", "N", "N", "N", "N", "N", "N", "N"...
## $ trip_duration     <int> 455, 663, 2124, 429, 435, 443, 341, 1551, 2...
## $ dset              <fctr> train, train, train, train, train, train, ...
```

3.3 Changing datatypes

First we will change the datatype for the variables *pickup_datetime, gropoff_datetime* as they involve date and time using *ymd_hms*. Then we will change the datatype of variables *vendor_id* and *passanger_count* as *factor*.

```
#changing data type of few attributes.
train <- train %>%
  mutate(pickup_datetime = ymd_hms(pickup_datetime),
         #pickup_datetime as datetime in format ymd_hms
         dropoff_datetime = ymd_hms(dropoff_datetime),
         #dropoff_datetime as datetime in format ymd_hms
         vendor_id = factor(vendor_id),
         #vendor_id as factor
         passenger_count = factor(passenger_count))
         #passanger_count id as factor

glimpse(train)
```

```
## Observations: 1,458,644
## Variables: 11
```

```
## $ id                <chr> "id2875421", "id2377394", "id3858529", "id3...
## $ vendor_id         <fctr> 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, ...
## $ pickup_datetime   <dttm> 2016-03-14 17:24:55, 2016-06-12 00:43:35, ...
## $ dropoff_datetime  <dttm> 2016-03-14 17:32:30, 2016-06-12 00:54:38, ...
## $ passenger_count   <fctr> 1, 1, 1, 1, 1, 6, 4, 1, 1, 1, 1, 4, 2, 1, ...
## $ pickup_longitude  <dbl> -73.98215, -73.98042, -73.97903, -74.01004,...
## $ pickup_latitude   <dbl> 40.76794, 40.73856, 40.76394, 40.71997, 40....
## $ dropoff_longitude <dbl> -73.96463, -73.99948, -74.00533, -74.01227,...
## $ dropoff_latitude  <dbl> 40.76560, 40.73115, 40.71009, 40.70672, 40....
## $ store_and_fwd_flag <chr> "N", "N", "N", "N", "N", "N", "N", "N", "N"...
## $ trip_duration     <int> 455, 663, 2124, 429, 435, 443, 341, 1551, 2...
```

```
#ymd_hms => year month day_hour minute second
```

As you can see the datatype has been changed accordingly.

3.4 Consistency check

We should also check whether the *trip_durations* are consistent with the intervals between the *pickup_datetime* and *dropoff_datetime*. Presumably the former were directly computed from the latter, but it is good practise to have it verified. Below, the *check* variable shows "TRUE" if the two intervals are not consistent:

```
# with dplyr, create new variable 'check' if the addition of the difference
# between 'dropoff_datetime' and 'pickup_datetime' (-ve value) and positive
# 'trip_durations' is greater than zero, then date times are not consistent.
# Basically we are checking if the 'trip_duration' is same as the difference
# between 'dropoff_datetime' and 'pickup_datetime'.  If it is incosistent, 'check
# takes value as 'true'.  after creating 'check', subset train data with only 4
# attributes as, 'check', 'pickup_datetime', 'dropoff_datetime', 'trip_duration'
# use group_by on this subset with variable 'check' report counts of check.

train %>% mutate(check = abs(int_length(interval(dropoff_datetime, pickup_datetime)) +
    trip_duration) > 0) %>% dplyr::select(check, pickup_datetime, dropoff_datetime,
    trip_duration) %>% group_by(check) %>% count()
```

```
## # A tibble: 1 x 2
##   check       n
##   <lgl>   <int>
## 1 FALSE 1458644
```

```
# Once I installed fitdistplus library, select statement in dplye had clashes.
# https://stackoverflow.com/questions/24202120/dplyrselect-function-clashes-with-massselect
# From here, the simple solution is to use dplyr::select
```

Zero count of true values concludes the consistency in the *trip_duration* and the difference between *pickup_datetime* and *dropoff_datetime*.

4 Exploratory data analysis

A good understanding of the data can be achieved by visualizing the data. It is good practice to visualize the data before processing. So that, we can have an idea about how our predictive models might behave. We can identify trends and patterns in the data. That will definitely help us with providing more perspectives.

4.1 Target variable - trip_duration

Let's start with our interest. The target variable *trip_duration*. Let's see the distribution of the values. For this histogram, onservations have been log transformed and count has been scaled as its square root. The data transformation provides better understanding of the data even when it has extreme ranges. We can visualize in a single graph in details.

```
# chunk option as

# {r trip_duration distribution, fig.align = 'center', warning = FALSE, fig.cap
# =' Distribution of trip duration (transformed)', out.width='100%'}

train %>% # dplyr functionality. in the train,
ggplot(aes(trip_duration)) + # map trip_duration
geom_histogram(fill = "red", bins = 150) + # using histogram and bin size of 150
scale_x_log10() + # scale x axis using log transformation
scale_y_sqrt()
```

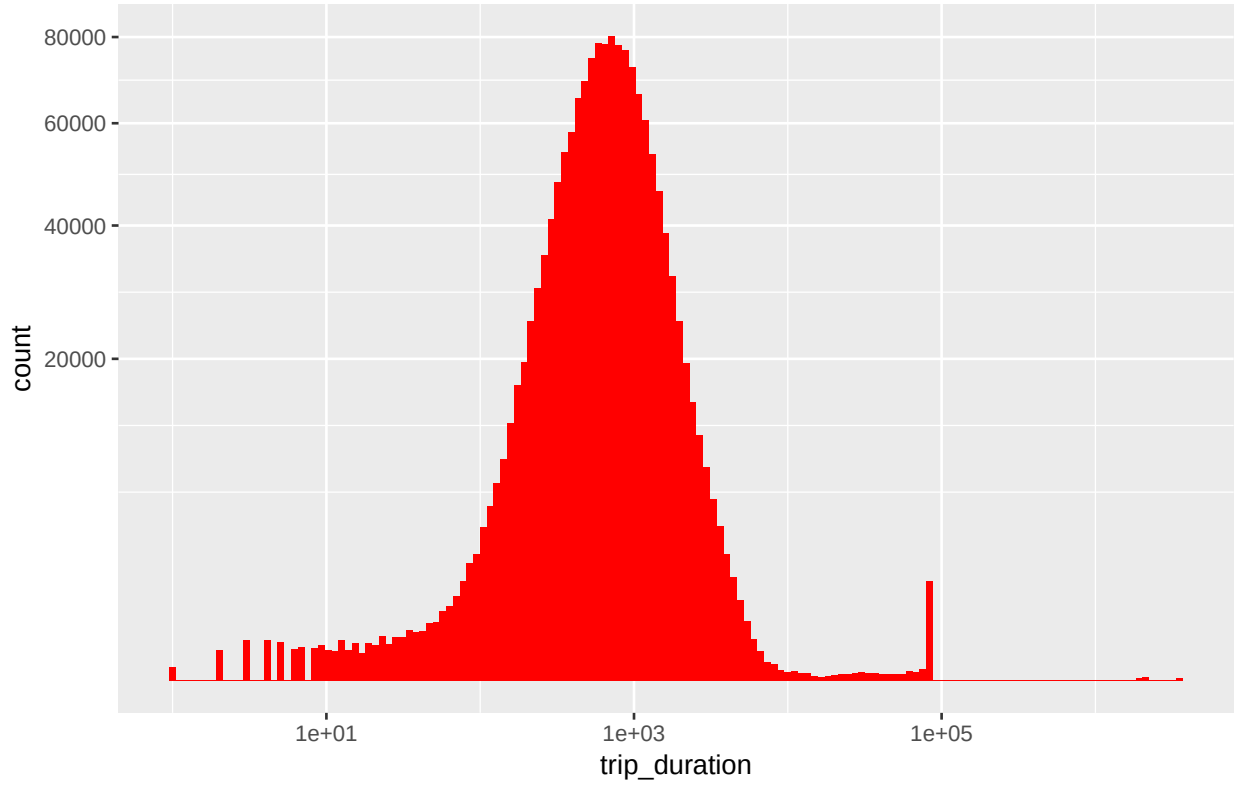


Figure 1: Distribution of trip duration (transformed)

```
# scale y axis using square root
```

- It seems that primarily all rides follow a rather smooth distribution that looks almost log-normal with a peak just short of 1000 seconds, i.e. about 27 minutes.
- There are several suspiciously short rides with less than 10 seconds duration.
- Additionally, there is a strange delta-shaped peak of *trip_duration* just before the 1e+05 seconds mark. Possibly outliers.

Let's see the details about these datapoints.

```
outlies <- train %>% # in train,
arrange(desc(trip_duration)) %>% # arrange datapoints with descending values of trip_duration
dplyr::select(trip_duration, pickup_datetime, dropoff_datetime, everything()) %>%
    # Display these variables first and then rest variable values for selected
# datapoints
head(5)
# first 5 datapoints just to see

# print the dataframe
pander(outlies, caption = "outliers", digits = 4)
```

Table 7: outliers (continued below)

trip_duration	pickup_datetime	dropoff_datetime	id
3526282	2016-02-13 22:46:52	2016-03-25 18:18:14	id0053347
2227612	2016-01-05 06:14:15	2016-01-31 01:01:07	id1325766
2049578	2016-02-13 22:38:00	2016-03-08 15:57:38	id0369307
1939736	2016-01-05 00:19:42	2016-01-27 11:08:38	id1864733
86392	2016-02-15 23:18:06	2016-02-16 23:17:58	id1942836

Table 8: Table continues below

vendor_id	passenger_count	pickup_longitude	pickup_latitude
1	1	-73.78	40.65
1	1	-73.98	40.74
1	2	-73.92	40.74
1	1	-73.79	40.64
2	2	-73.79	40.64

dropoff_longitude	dropoff_latitude	store_and_fwd_flag
-73.98	40.75	N
-73.99	40.73	N
-73.98	40.76	N
-73.96	40.77	N
-73.99	40.76	N

The trip duration is cecrtainly unbelievable as the time is above 24 hours till 12 days. We will deal with these outliers later.

4.2 Predictors

Let's visualize other input variables and their corresponding effect with respect to the target variable.

4.2.1 Vendor_id

So moving on, Let's observe our target variable with respect to the *vendor_id*.

```
train %>% # in train,
ggplot(aes(vendor_id, fill = vendor_id)) + # plot vendor_id with it's count
geom_bar()
```

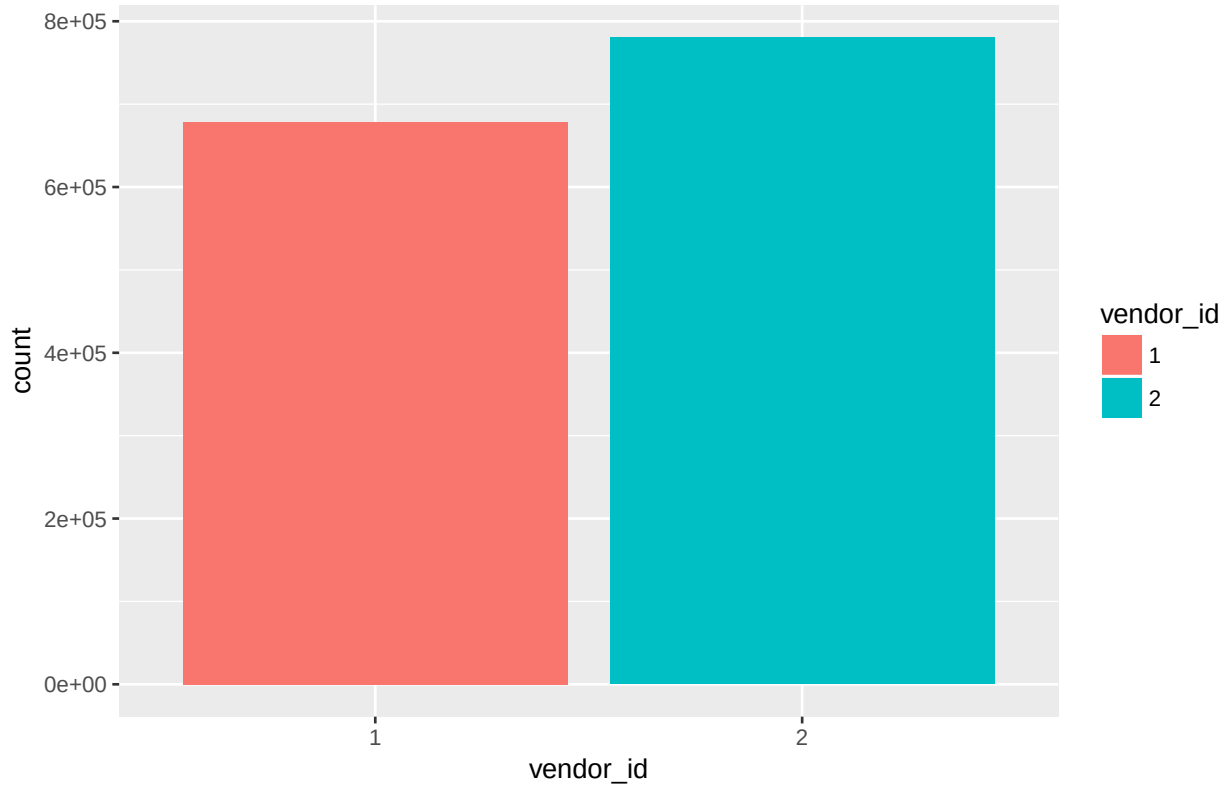


Figure 2: Number of trips by each vendor

```
# bar plot
```

4.2.2 Passanger-count

It seems we have sufficient datapoints for both taxi service providers. Also, Let's address the next variabel *passanger_count*.

```
train %>% # In train,
group_by(passenger_count) %>% # group dataset by passanger_count
count()
```

```
## # A tibble: 10 x 2
##    passenger_count       n
##              <fctr>   <int>
##  1               0      60
##  2               1 1033540
##  3               2  210318
##  4               3   59896
##  5               4   28404
##  6               5   78088
##  7               6   48333
```

```
##  8                7        3
##  9                8        1
## 10                9        1
```

```
# report counts
```

8 and 9 passanger in one trip sounds a bit extreme. It could be incocsistency in the data. However, the number of datapoints for these variables are considerably less comapared to the total number of datapoints. Vast majority is for single passanger.

Also, we can see the relation between *passanger_ount* and *trip_duration* by further boxplot.

```
train %>% # in train,
ggplot(aes(passenger_count, trip_duration, color = passenger_count)) + # plot passanger_count against t
# different count of passangers.
geom_boxplot() + # plot boxplot
facet_wrap(~vendor_id) + # group by data using vendor_id
scale_y_log10() + # scale the data because with original datapoints due to extreme outliers, the
# box plot was not readable.
theme(legend.position = "none") + # axes are self exploraotry we do not need redundant legend. It consu
# space.
labs(y = "Trip duration in seconds", x = "Number of passengers (right: vendor 1 and left: vendor 2)")
```

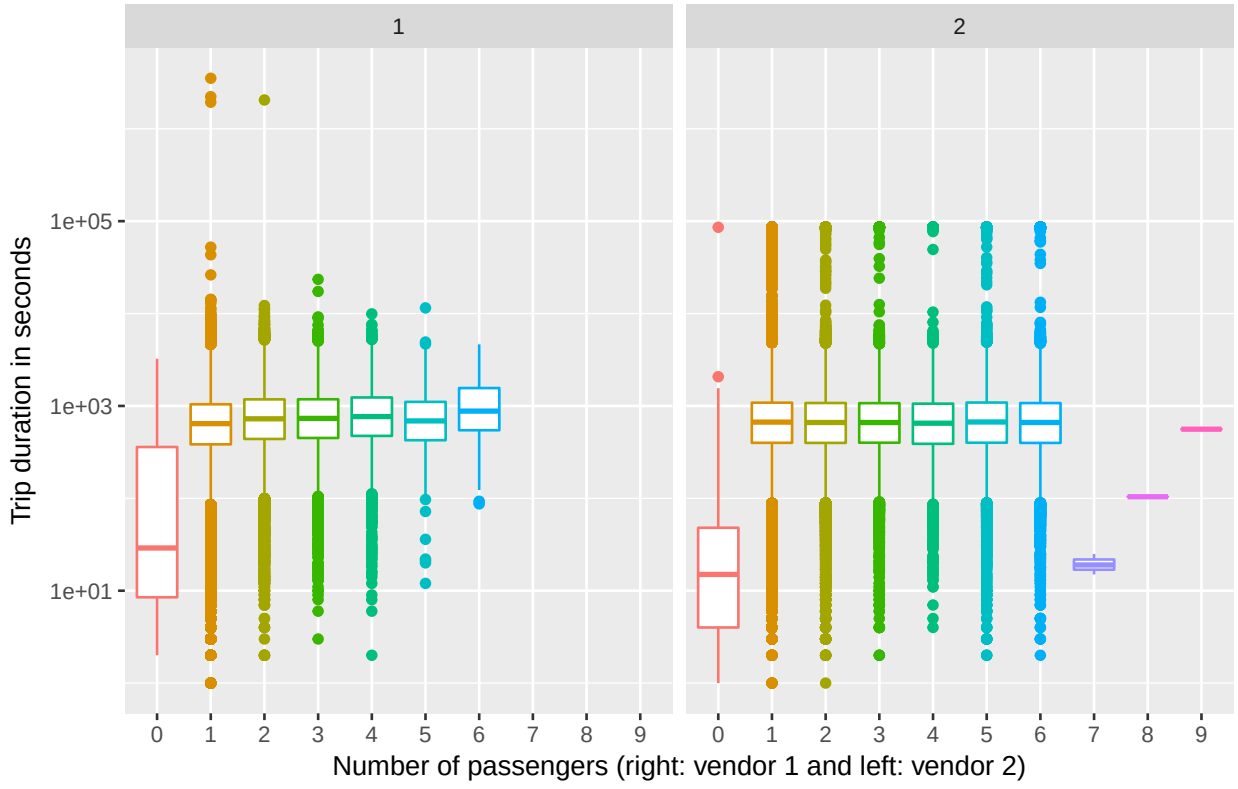


Figure 3: passanger count against trip duration boxplot for both vendors

```
# add labels.
```

There's almost similar trend for both vendors.

4.2.3 count of trips w.r.t. time

Let's see thedistribution of datapoints over time. For multiple plots we will use the function called multiplot which intakes the values as graph objects and their desired layout in matrix format and outputs a single object combining all these graph in presentable format. I have mentioned the source for this function.

```
# Define multiple plot function
# http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/ ggplot
# objects can be passed in to plotlist (as a list of ggplot objects) - cols:
# Number of columns in layout - layout: A matrix specifying the layout. If
# present, 'cols' is ignored.  If the layout is something like matrix(c(1,2,3,3),
# nrow=2, byrow=TRUE), This shows there are 3 graph objects to be plotted then
# plot 1 will go in the upper left, 2 will go in the upper right, and 3 will go
# all the way across the bottom.  nrow defines number of rows in the layout.

multiplot <- function(..., plotlist = NULL, file, cols = 1, layout = NULL) {

    # Make a list from the ... arguments and plotlist
    plots <- c(list(...), plotlist)

    numPlots = length(plots)

    # If layout is NULL, then use 'cols' to determine layout
    if (is.null(layout)) {
        # Make the panel ncol: Number of columns of plots nrow: Number of rows needed,
        # calculated from # of cols
        layout <- matrix(seq(1, cols * ceiling(numPlots/cols)), ncol = cols, nrow = ceiling(numPlots/co
    }

    if (numPlots == 1) {
        print(plots[[1]])

    } else {
        # Set up the page
        grid.newpage()
        pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

        # Make each plot, in the correct location
        for (i in 1:numPlots) {
            # Get the i,j matrix positions of the regions that contain this subplot
            matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))
            # Match position of layouot and graph objects and print results accordingly.
            print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row, layout.pos.col = matchidx$co
        }
    }
}
```

```
p1 <- train %>% # In train,
ggplot(aes(pickup_datetime)) + # using variable pickup_datetime,
geom_histogram(fill = "green", bins = 120) + # create the histogram
labs(x = "Pickup dates")
# define the label
```

```r
# similar with variable dropoff_datetime
p2 <- train %>% # In train,
ggplot(aes(dropoff_datetime)) + # plot dropoff_datetime
geom_histogram(fill = "orange", bins = 120) + # distribution of counts
labs(x = "Dropoff dates")
# labels

# define layout for this visual. Column_1 row_1 [fig_1] row_2 [fig_2]
layout <- matrix(c(1, 2), 2, 1, byrow = FALSE)

# using multiplot, plot these two graphs.
multiplot(p1, p2, layout = layout)
```

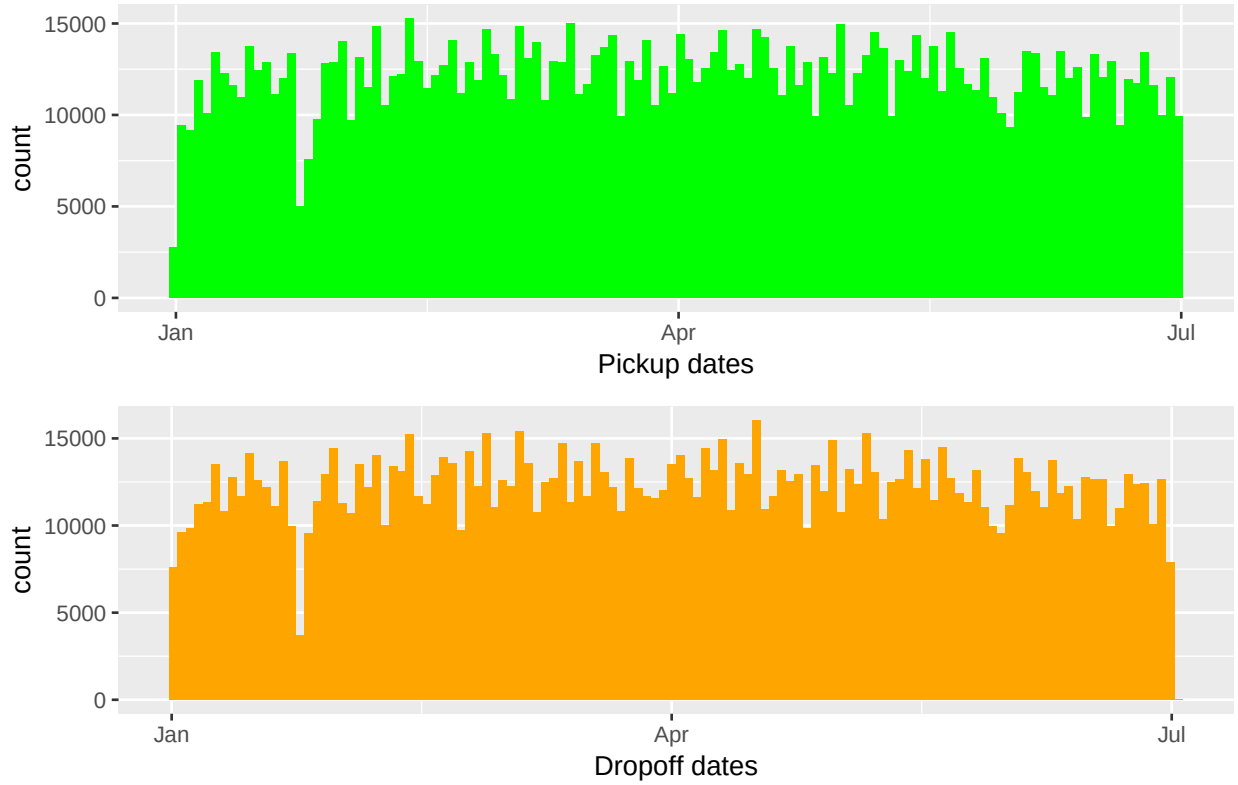


Figure 4: Distribution of pickup datetime and dropoff datetime over time

```r
# p1 <- 1; p2 <- 1 # clearing the stored graph objects in variables p1 and p2.
```

Both distributions are almost identical. Confirming the consistency in the data. However, we can see an interesting drop in the values around early February.

Let's see that part of the distribution in details.

```r
train %>% filter(pickup_datetime > ymd("2016-01-20") & pickup_datetime < ymd("2016-02-10")) %>%
    # values from January 20 to February 10.
ggplot(aes(pickup_datetime)) + # plot pickup_datetime
geom_histogram(fill = "grey60", bins = 120)
```

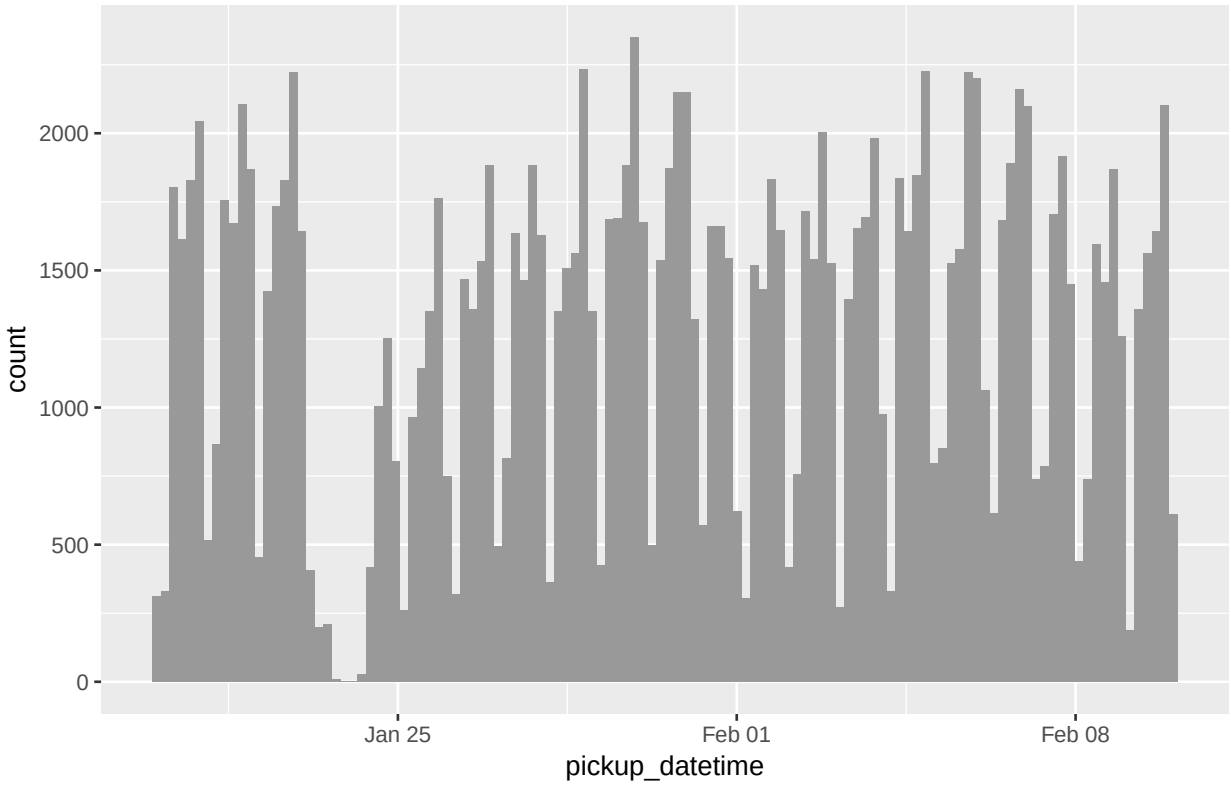


Figure 5: number of trips around the time when theres considerable drop in the distribution

```
# histogram with bin size of 120.
```

The drop is around January 24 to be more precise. When I googled "*New York January 24 2016*", the first result was this link. The data is consistent to the real world. It turns out on this day there was a blizzard in NewYork. On the wensite, it is mentioned that, Snowfall amounts ranged from over 2 feet from northeast New Jersey, New York City, and Long island to around a foot across parts of the Lower Hudson Valley and southern Connecticut. There was a tight gradient in amounts further inland where only around 1 to 2 inches of snow fell across northern Orange County while over a foot of snow fell in the southern portion of the county. Central Park, NY received 27.5" of snow, which is the largest snowstorm since records began in 1869.

Also, I am curious about the time of travel. We have information of time of the day as well. Let's see what is the most busy time for New Yorkers to use these two vendors offering taxi services.

```
train %>% # in train,
mutate(hpick = hour(pickup_datetime)) %>% # create the new variable hpick which store the hour value of
# variable.
group_by(hpick, vendor_id) %>% # group by entire data by hpick
count() %>% # report count
ggplot(aes(hpick, n, color = vendor_id)) + # pass this to ggplot
geom_point() + geom_line() + # show scatter plot and line chart as well.
labs(x = "Hour of the day", y = "Total number of pickups")
```

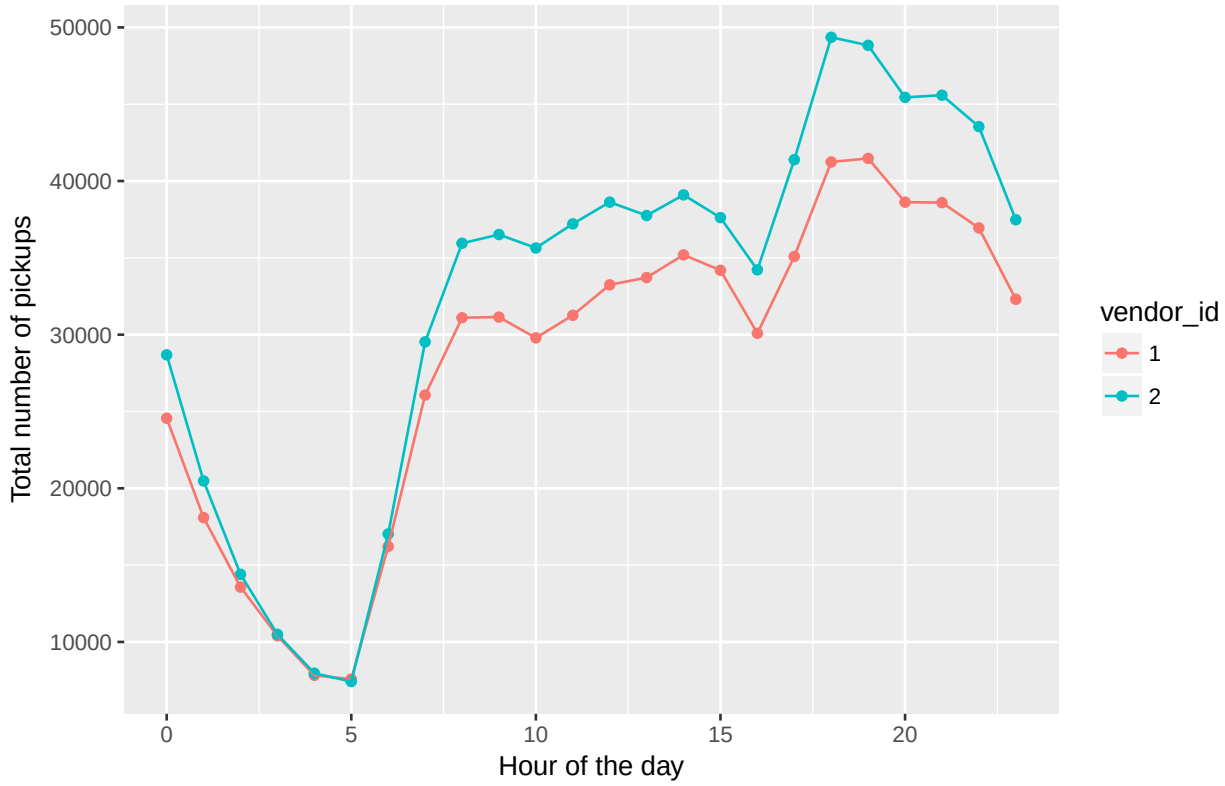


Figure 6: Distribution of number of trips over time of the day

```
# define labels.
```

It seems, after office hours are pretty busy time for taxi drivers. From this graph we can also conclude that there's similar pattern for both vendors. However, we know that the frequency of trips by vendor 2 is higher than vendor 1. Hence right now hypothesis can be formed that feature *vendor_id* can be statistically important in the predictive model.

Similarly, let's see the distribution over months of the year and day of the week using similar code.

```
p1 <- train %>%
  # in train,
  mutate(hpick = hour(pickup_datetime),
         # create variable hpick by derving hour value from variable pickup_datetime
         Month = factor(month(pickup_datetime, label = TRUE))) %>%
         #create another variable named month by derive month
         #from variable pickup_datetime and convert it to factor.
  group_by(hpick, Month) %>%
  #group by entire dataset using hpick and Month
  count() %>%
  #report count for e ach month
  ggplot(aes(hpick, n, color = Month)) +
  #plot this count using ggplot
  geom_line() +
  #line chart.
  labs(x = "Hour of the day", y = "count")
  #labels
```

```r
#similar to above code except few changes.
p2 <- train %>%
  #in train,
  mutate(hpick = hour(pickup_datetime),
         wday = factor(wday(pickup_datetime, label = TRUE))) %>%
         #the only change is derive day of the week
  group_by(hpick, wday) %>%
  #group By this data using hpick and wday
  count() %>%
  #report count - to be used for graph.
  ggplot(aes(hpick, n, color = wday)) +
  #using ggplot, plot hpick and count, seperate/aggregate data by wday
  geom_line() +
  #show line chart
  labs(x = "Hour of the day", y = "count")
  #labels for axes

#plot these two distribution in single graph object using multiplot
layout <- matrix(c(1,2),2,1,byrow=FALSE)
#define layout for this multiplot visual
multiplot(p1, p2, layout=layout)
```

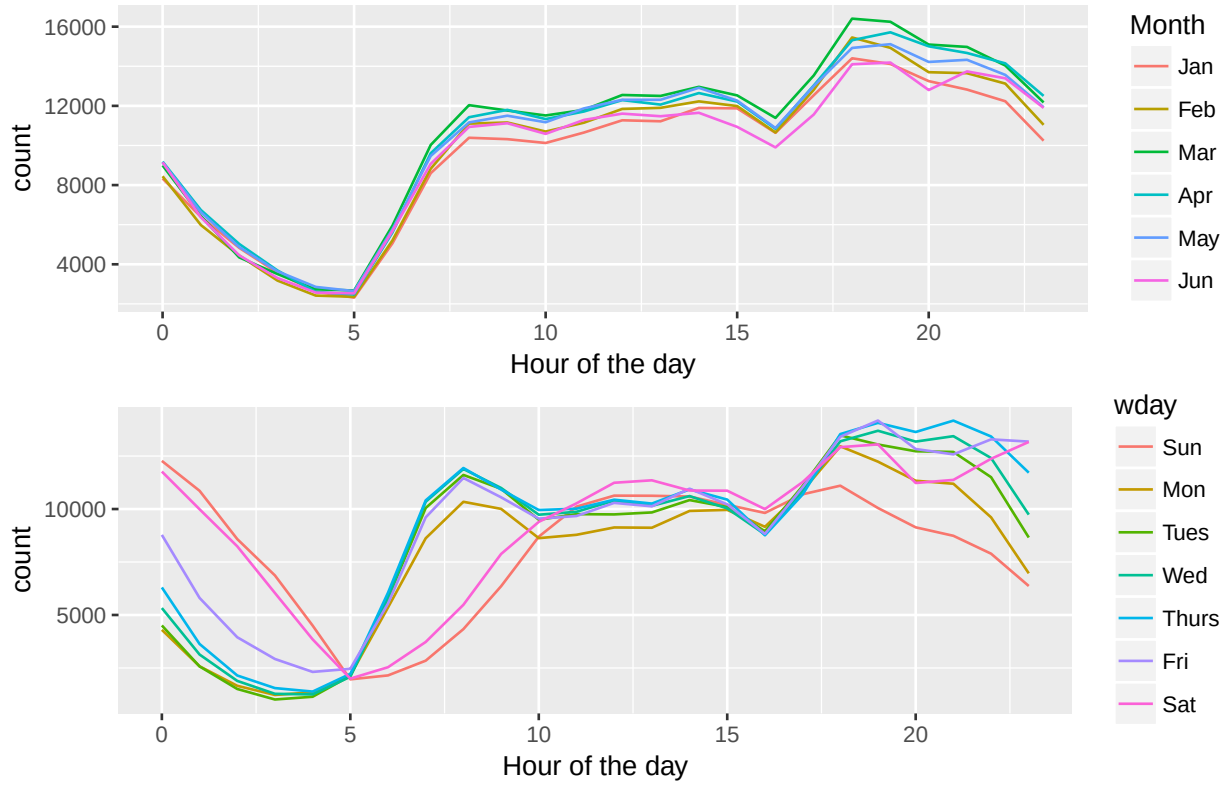


Figure 7: Distribution of trip counts over hour of the day grouped by months and days of week

```r
#plot using multiplot function.
p1 <- 1; p2 <- 1
```

```
#clearing the stored graph objects in variables p1 and p2.
```

January and June have fewer trips, whereas March and April are busier months. All the months follow almost similar pattern in picking up taxis at specific time of the day.

Comingto the secong graph, we can clearly see that, Saturday, Sunday, and partially Friday follow different trend than rest of the weekdays. On these three days, there is considerable higher counts of datapoint for late night and early morning.

On these days. during normal morning hours(5am - 10am), the count is significantly lower.

Also, let's observe the difference between frequecy of pickup counts in training and testing data set.

```
temp <- combine %>% # create a temporary dataframe from combine
mutate(date = date(ymd_hms(pickup_datetime))) %>% # mutate a new variable in combine named date which d
# existing variable pickup_datetime using ymd_hms()
group_by(date, dset) %>% # group by this data using variables date and dset.
count()
# report count

temp %>% # in this temp dataframe,
ggplot(aes(date, n/1000, color = dset)) + # plot the date count and seperate count by dset value.
geom_line(size = 1) + # use geom_line()
labs(x = "Time (Date)", y = "trips per day in thousands")
```

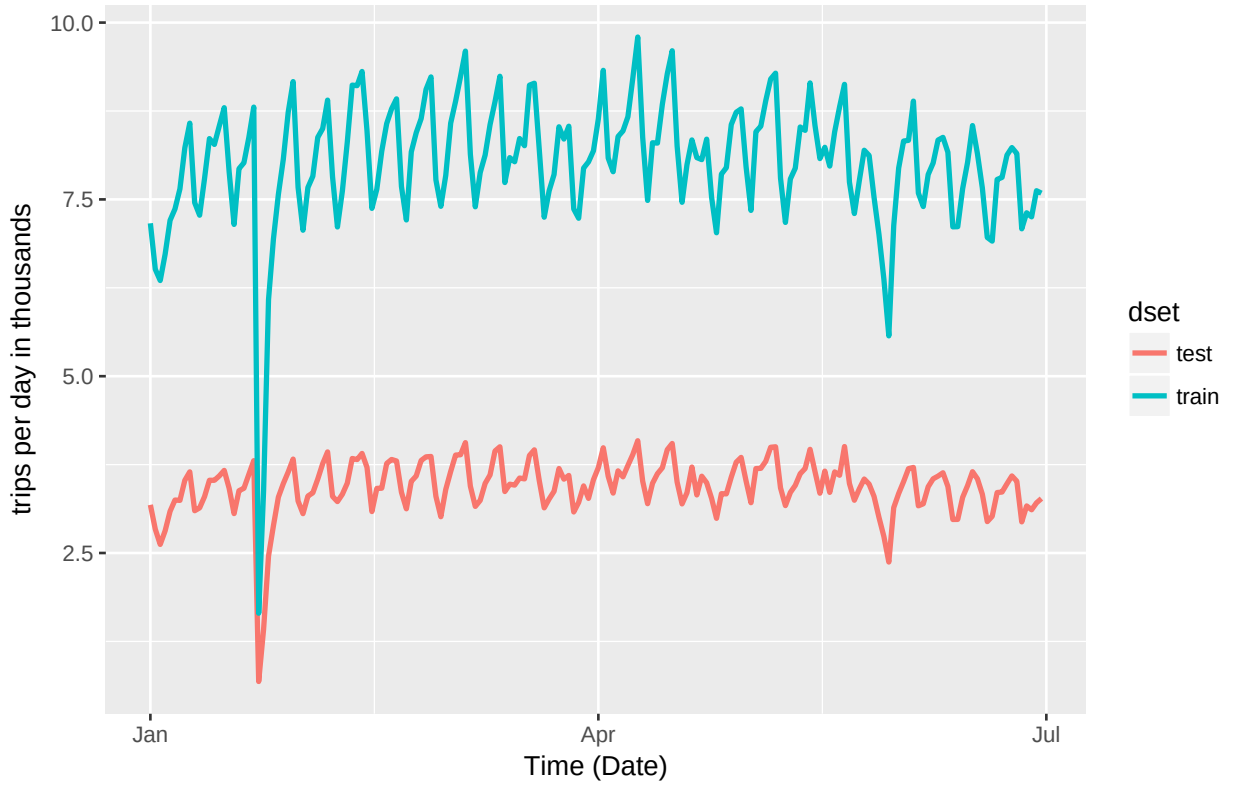


Figure 8: Frequency of pickup count in training and testing dataset

```
# labels
```

We can see that, there's almost similar pattern in number of counts of trip in training and test dataset over time. The only differenc is in magnitude. It works for us.

4.2.4 Location variables

Another important piece of information we have is location coordinates of pickup and drop off point. Let's use this information and plot the coordinates on the New York city map.

We will use *leaflet* library for this. I referred to this link for using this library.

```
# All 2 million datapoints would make graph too much crowded. For representation
# purpose, randomly 10000 datapoints are selected and plotte.

# subsetting 10000 datapoints at random as foo.

## set.seed(2) foo <- sample_n(train, 10000) #sample 10000 datapoints for this
## visual at random.

# Using subset With variables 'pickup_longitude and 'pickup_latitude' graphs are
# plotted.

## leaflet(data = foo) %>% in foo, addProviderTiles('Esri.NatGeoWorldMap') %>% add
## tile addCircleMarkers(~ pickup_longitude, ~pickup_latitude, radius = 1, color =
## 'blue', fillOpacity = 0.3)

# use pickup_longitude and pickup_latitude as markers.  Last three parameters
# define the circular datapoint in map.
```

We can see that the mostly datapoints are from main part of New York city, however, few points are crowded at JFK airport. We can see a datapoint in the middle of sea. That is weird. There must be more of these absurd datapoints in the entire dataset. We will take care of this issue in future.

It is difficult to see the spread of all 2 million datapoints on map due to system limits. The system crasahes is we try to plot all datapoints. We can have a way around this issue using simple histograms. Let's plot distribution of all the location based variables.

```
p1 <- train %>% # in train,
filter(pickup_longitude > -74.1 & pickup_longitude < -73.6) %>% # filter out extreme coordinates
ggplot(aes(pickup_longitude)) + # plot pickup_longitude
geom_histogram(fill = "green", bins = 40)
# histogram with bin size of 40.

p2 <- train %>% filter(dropoff_longitude > -74.1 & dropoff_longitude < -73.6) %>%
    # filter out extreme coordinates
ggplot(aes(dropoff_longitude)) + # plot dropoff_longitude
geom_histogram(fill = "blue", bins = 40)
# histogram with bin size of 40.

p3 <- train %>% filter(pickup_latitude > 40.5 & pickup_latitude < 40.8) %>% # filter out extreme coordi
ggplot(aes(pickup_latitude)) + # plot pickup_latitude
geom_histogram(fill = "green", bins = 40)
# histogram with bin size of 40.

p4 <- train %>% filter(dropoff_latitude > 40.5 & dropoff_latitude < 40.8) %>% # filter out extreme coor
ggplot(aes(dropoff_latitude)) + # plot dropoff_latitude
```

```
geom_histogram(fill = "blue", bins = 40)
# histogram with bin size of 40.

# define the layout of this visual. four graph objects. shown on the layout of 2
# by 2 matrix. i.e. 2 rows and 2 columns. total 4 positions.
layout <- matrix(c(1, 2, 3, 4), 2, 2, byrow = FALSE)
# use multiplot and above defined layout for rendering the visual.
multiplot(p1, p2, p3, p4, layout = layout)
```

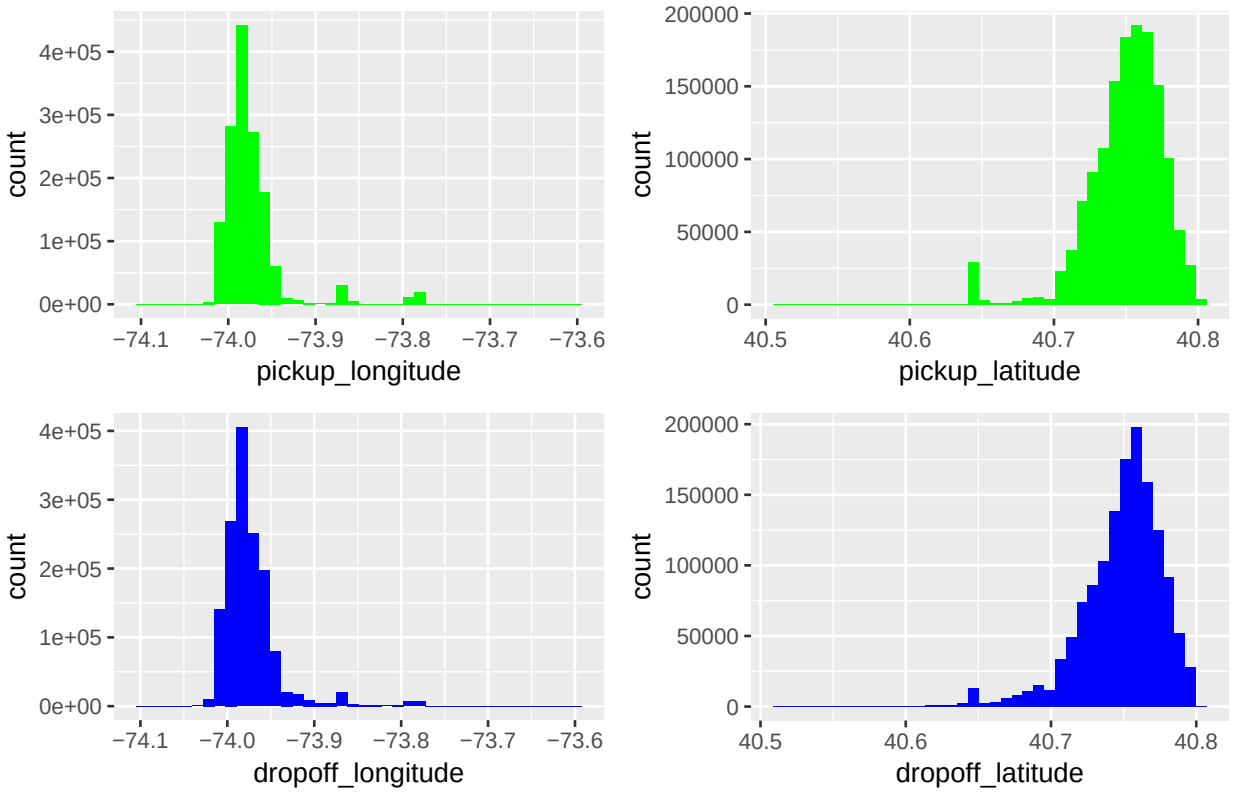


Figure 9: Distribution of location variables.

There's slight modification to the code as filter is applied. When the distribution was plotted using entire data, due to some extreme coordinates, the distribution plot was turning out to be a single column as the graph was trying to cover entire range of the datapoints. Hence after few trial and errors I came up with these limits so that wew can see different counts for considerable number of different bins. I am guessing that the small peaks after the fairly normal distribution's tails must be for airport location.

Also, it is good idea to see the difference in pickup coordinates in training and test datasets. This will help us in future to build some intuition for prediction model. If we build model based on majority of datapoints in one area of New York City and used that model to predict the *trip_duration* using datapoints whicch have the pickup locations in majority in different area, then our model will behave inadequately.

```
# create a temporary dataframe to hold the values
temp <- combine %>% # from combine
filter(pickup_longitude > -75 & pickup_longitude < -73) %>% # filter by pickup longitude and latitude a
# extreme locations)
filter(pickup_latitude > 40 & pickup_latitude < 42)
# filter by pickup latituded and latitude as we learned above (to avoid the
```

```
# extreme locations)

temp <- sample_n(temp, 5000)
# For conveniency in reading the map with adequate datapoints.  Too many
# datapoints would make the graph unreadable and script takes too long to run.

temp %>% ggplot(aes(pickup_longitude, pickup_latitude, color = dset)) + # use filtered longitude and la
# dset)
geom_point(size = 0.1, alpha = 0.5) + # using scatter plot show small dots for location
coord_cartesian(xlim = c(-74.02, -73.77), ylim = c(40.63, 40.84)) + # set limits to the graph axes.
facet_wrap(~dset) + # facet_wrap - create different grpah for different origin of datapoints.
theme(legend.position = "none")
```

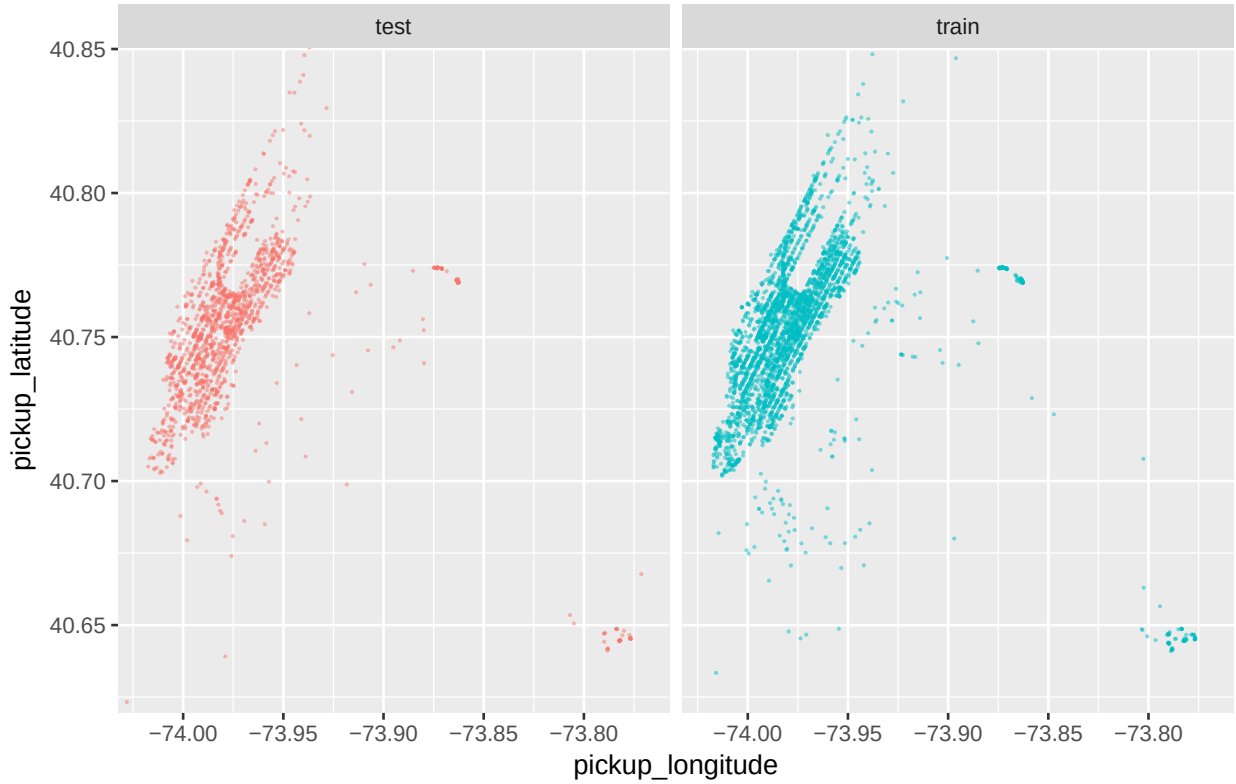


Figure 10: Pickup location in train and test dataset

```
# no need of legend. let's see the grapph ini maximum avaiable space.
```

We can conclude that the distriution of pickup locations cover the same area in majority of different areas across New York City.

5 Feature Engineering

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. Feature engineering is fundamental to the application of machine learning, and is both difficult and expensive. The need for manual feature engineering can be obviated by automated feature

learning. Feature engineering is an informal topic, but it is considered essential in applied machine learning.

Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.

In our example, we can think of many more possible variables that may affect the response variable as *trip_duration*. Common sense says the weather can be important factor. For a particular day the amount of rain, extreme low temperature, and amount of snow can affect the amount of traffic significantly which in turn affects the *trip_duration*.

Another thought says the routes distance must be a deciding factor in determining *trip_duration*. Number of intersections, stop signs, number of left sides turns contribute to the *trip_duration*.

5.1 Total Distance of the trip

For now, Let's calculate the total distance using pickup and dropoff location coordinates. We are using library called geosphere for this. More information is on this [link] (https://cran.r-project.org/web/packages/geosphere/index.html).

The idea is, we are trying to get direct distance using pickup and dropoff coordinates. This direct distance is from point A to point B irrespective of the routes. But this is the best we can do. When Geosphere library comes into picture, it calculates the distance using *DistCosine()* function. This takes the spherical shape of the earth into consideration while calculating distance.

```
pickup_coord <- train %>% dplyr::select(pickup_longitude, pickup_latitude)
# A list which contains picup coordinates
dropoff_coord <- train %>% dplyr::select(dropoff_longitude, dropoff_latitude)
# A list which contains dropoff coordinates
train$dist <- distCosine(pickup_coord, dropoff_coord)
# Cosine distance between both coordinates staured in a new variable called dist.
```

Let's plot the variable against *trip_duration* and see if distance increases as *trip_duration* increases. If there's any other trend in the data, then it will be an initeresting case to observe. We can figureout t he reasons later.

```
set.seed(1)
train %>% # In train
sample_n(10000) %>% # take a sample of 10000 dataoints
ggplot(aes(dist, trip_duration)) + # plot tripduration against dist
geom_point() + # show scatter plot
scale_x_log10() + # scale - log transform x axis to cover wide range of datapoints
scale_y_log10() + # Scale - log transform x axis to cover wide range of datapoints
labs(x = "Direct distance in meters", y = "Trip duration in seconds", title = "Log transformed")
```

```
## Warning: Transformation introduced infinite values in continuous x-axis
```

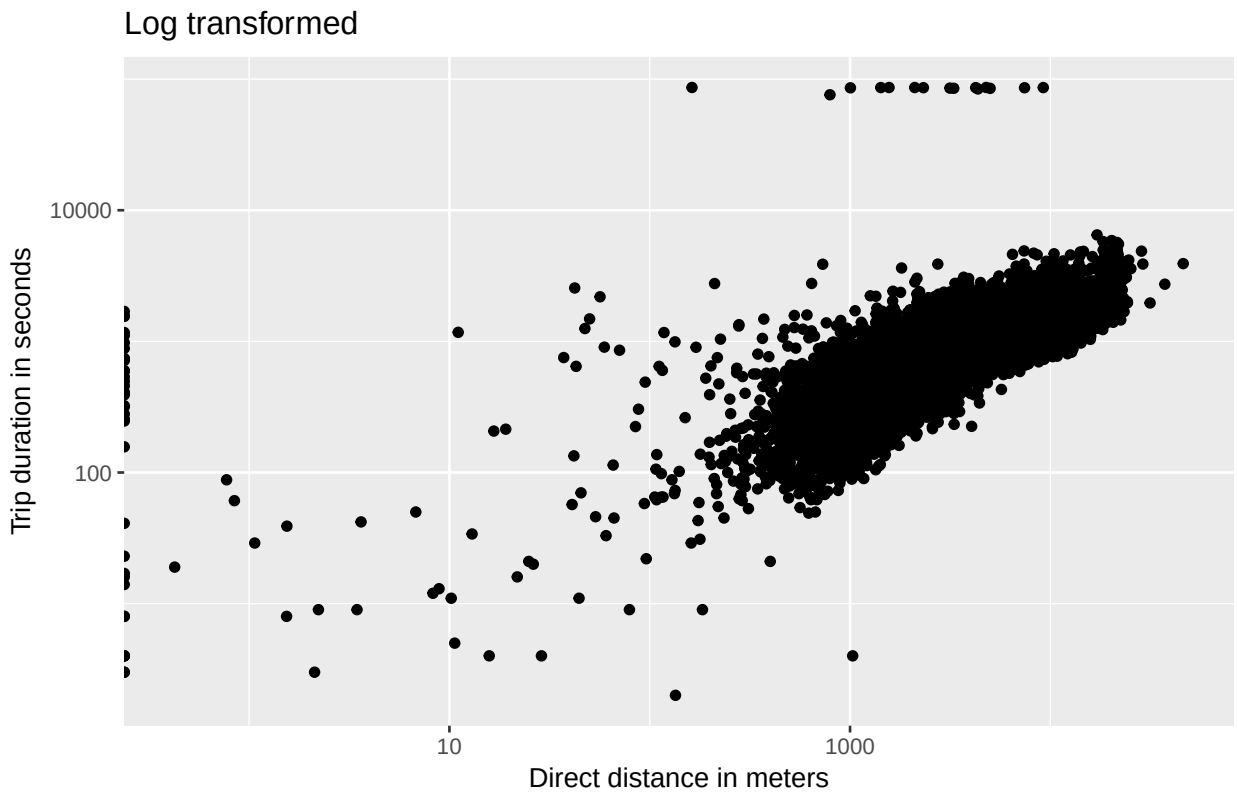


Figure 11: Scatter plot Distance against trip duration

```
# add labels
```

In the first trial, theere were extreme datapoints for both variables. Hence the graph was not displaying reasonable spread of datapoints so that we can identify the trends. Instead of applying filters, we log transformed the variables. And above graph is generated.

The error message shows that there must be few values in distance because of which log transformation went to infinity.

General trend is quite what we exepcted. However, by the looks of it, it seems, we there are many extreme outliers in the data.

5.2 Average Speed

We have *trip_duration* and we have *dist*. It's becoming that we will examine average speed. This speed can be plotted against time of the day. Let's see what we might get to observe.

```
train <- train %>% # In train,
  mutate(speed = dist/trip_duration*3.6,
         #create the variable speed by devding distance by trip_duration and multiply by 3.6 (in km/hr
         wday = wday(pickup_datetime, label = TRUE),
         #create variable called wday that stores theday of the week for a particular pickup_datetime
         wday = fct_relevel(wday, c("Mon", "Tues", "Wed", "Thurs", "Fri", "Sat", "Sun")),
         #change the datatype of wday to factor using fact_relevel
         hour = hour(pickup_datetime)
         # Hour of the day
```

```
        )

train %>%
  filter(speed > 2 & speed < 1e2) %>%
  #afterfew iterations, to have a reasonable distribution in graph, filters had to be applied.
  ggplot(aes(speed)) +
  #plot speed
  geom_histogram(fill = "blue", bins = 40) +
  #plot histogram with bin size of 40
  labs(x = "Average speed in Km/Hr")
```

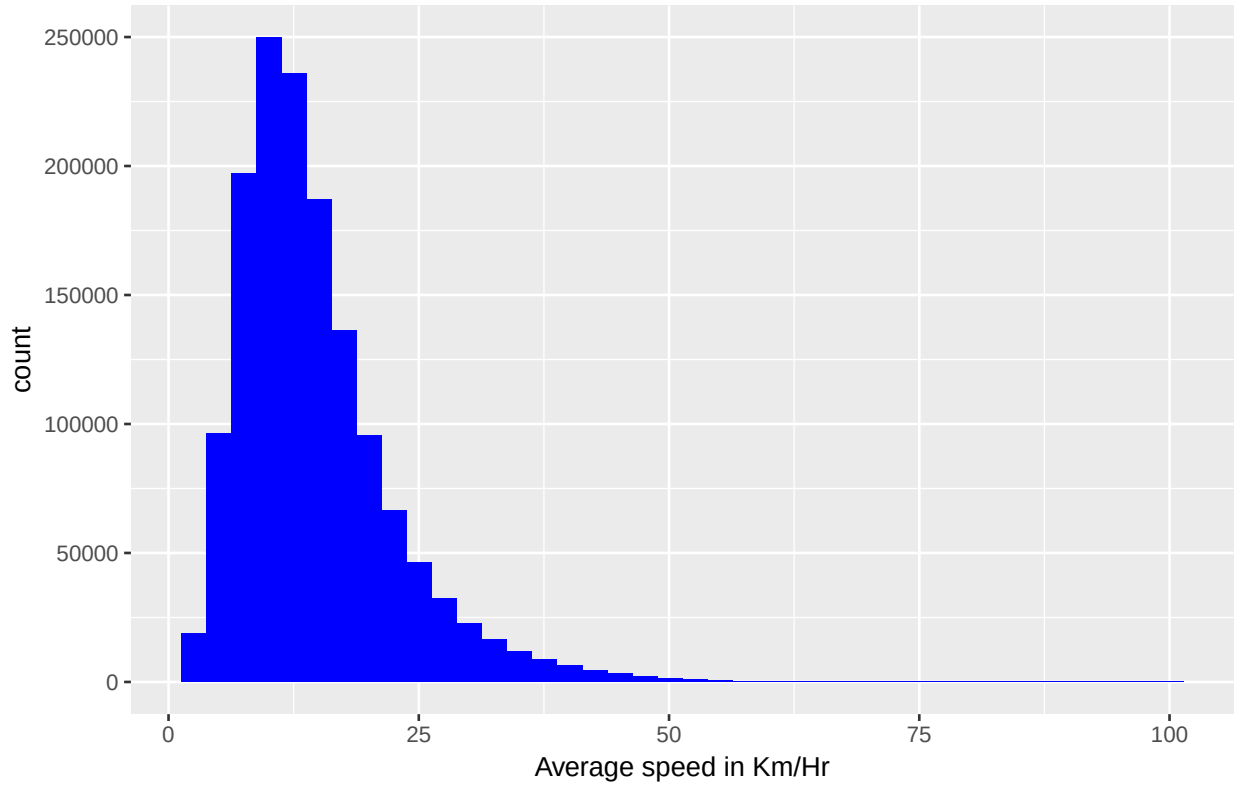


Figure 12: Distribution of Average speed in Km per Hr

```
  #X Axis label
```

The number of shorter trips are more in comaprison to the number of longer trips. Let's see the average speed in New York City over different times in the day. And let's group this plot by vendor_id.

```
p1 <- train %>% # in train,
group_by(wday, vendor_id) %>% # group_by wday and vendor_id
summarise(avg_speed = mean(speed)) %>% # summarize average speed for each group
ggplot(aes(wday, avg_speed, color = vendor_id)) + # using ggplot plot plot avg_speed against wday. Colo
geom_point(size = 3) + # Show scatter with size 3 of markers
labs(x = "Day of the week", y = "Average speed in Km/Hr")
# Labels to X axis and Y axis

p2 <- train %>% # in train,
```

```
group_by(hour, vendor_id) %>% # group_by hour and vendor_id
summarise(avg_speed = mean(speed)) %>% # summarize average speed for each group
ggplot(aes(hour, avg_speed, color = vendor_id)) + # using ggplot plot plot avg_speed against hour. Colo
geom_smooth(method = "loess", span = 1/2) + # Use Loess smooth curve
geom_point(size = 3) + # Show scatter as well
labs(x = "Hour of the day", y = "Average speed in Km/Hr") + # Labels to X axis and Y axis
theme(legend.position = "none")
# Do not consume space by showing legend

p3 <- train %>% # In train,
group_by(wday, hour) %>% # group_by wday and hour
summarise(avg_speed = mean(speed)) %>% # summarize average speed for each group
ggplot(aes(hour, wday, fill = avg_speed)) + # using ggplot plot plot avg_speed against hourand wday.
geom_tile() + # Use geom_tile forbetter representation
labs(x = "Hour of the day", y = "Day of the week") + # Labels to X axis and Y axis
scale_fill_distiller(palette = "Spectral")
# For geom_tile, using theme of palette as spectral for color distiller

layout <- matrix(c(1, 2, 3, 3), 2, 2, byrow = TRUE)
# create a matrix of layout as, [,1] [,2] [1,] 1 2 [2,] 3 3

# plot using multiplot with defined layout
multiplot(p1, p2, p3, layout = layout)
```

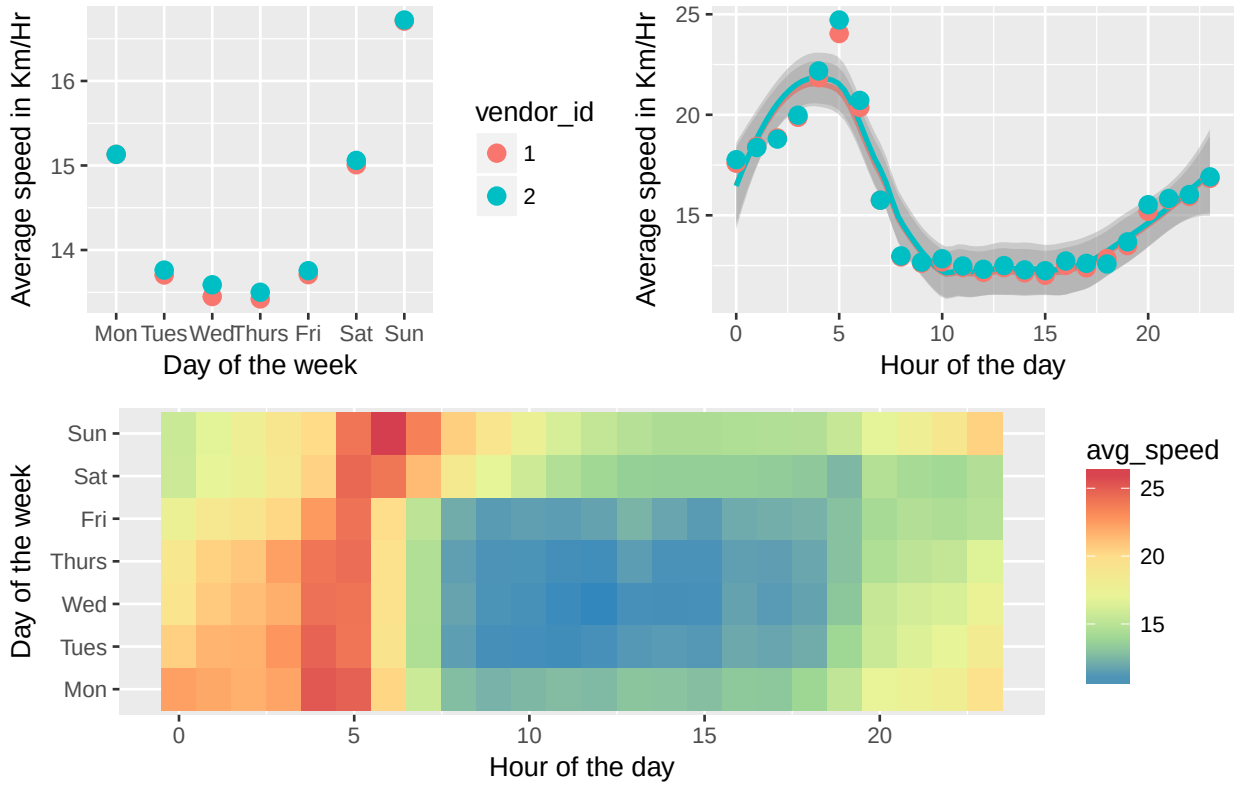


Figure 13: Average speed over time

- Taxi Drivers from both vendors show similar trend in speed.

- On weekends, The average speed is much higher throughout the day. Probably due to lesser traffic. Above 25 Km/Hr Hour.
- Early morning around 5 am is the time when taxis run at the highest speed in New York City.
- Monday till Friday during office hours i.e. 8 am till 6 pm, speed is considerably slower. Around 5 - 10 Km/Hr.
- Thinking of it, we can create a new variable called 'work' which will devide time as 8 am till 9 pm from Monday to Friday. (The Blue region)

```
train <- train %>%
  #in train,
  mutate(date = date(ymd_hms(pickup_datetime)),
         #extract date
         wday = wday(pickup_datetime, label = TRUE),
         #extract name of the day using wday() on pickup_datetime
         wday = fct_relevel(wday, c("Mon", "Tues", "Wed", "Thurs", "Fri", "Sat", "Sun")),
         #using fact_relevel, change the datatype as factor.
         hour = hour(pickup_datetime),
         #using hour() extract hour value of variable pickup_datetime
         work = (hour %in% seq(8,18)) & (wday %in% c("Mon","Tues","Wed","Thurs","Fri"))
         #create a flag and set it true if the hour values is between 8 and 18(6'o clock in evening)
         #and wday is other than weekend. Otherwise set work = False.
  )
```

5.3 Airport trips

Also from exploratory analysis for extreme *trip_duration*s mentioned in the data cleaning phase of this report (later part), we decided to extend feature engineering section for trips related to airports in the city. When we plotted location coordinates in the above part, we got to know that there are considerabl trips made to and from airport. And from later part of thereport we got to know that, these trips take considerable higher value in *trip_duration*. hence our intuition to create relavant features is justified.

To create this variable, general idea is, use the airports location coordinates. These are avaible on wikipedia page. We will use two airports as observed in majority from map with pickup coordinates.

The variable which we are creating will hold the value of cosine distance calculated by difference between pickup_coordinates and airport coordinates. This difference will be calculated for John F. Kennedy airport and LaGuardia airport.

```
#store the coordinates which represents the airport.
jfk_coord <- tibble(lon = -73.778889, lat = 40.639722)
la_guardia_coord <- tibble(lon = -73.872611, lat = 40.77725)

#create lists containg only locations from training dataset
pickup_loc <- train %>%
  #in train,
  dplyr::select(pickup_longitude, pickup_latitude)
  #select pickup longitude and latitude.

dropoff_loc <- train %>%
  #in train,
  dplyr::select(dropoff_longitude, dropoff_latitude)
  #select dropoff longitude and latitude.

#Create the new features, using distCosine()
```

```
train$jfk_dist_pick <- distCosine(pickup_loc, jfk_coord)
#this variable store the value of distance from JFK airport to pickup location.
train$jfk_dist_drop <- distCosine(dropoff_loc, jfk_coord)
#this variable store the value of distance from jfk airport to dropoff location.
train$lg_dist_pick <- distCosine(pickup_loc, la_guardia_coord)
#this variable store the value of distance from LaGuardia airport to pickup location.
train$lg_dist_drop <- distCosine(dropoff_loc, la_guardia_coord)
#this variable store the value of distance from LaGuardia airport to dropoff location.

#Let's crate the falgs and assign the value to a particular datapoint
#if the trip was to or from any of the airport.
#New variable jfk_trip (trip to JFK) and lg_trip (trip to LaGuardia) wil
#take the vallues true and false based on the values of above created features.

train <- train %>%
  mutate(jfk_trip = (jfk_dist_pick < 2200) | (jfk_dist_drop < 2200),
         #True if either of the jfk_dist_pick or jfk_dist_drop takes the
         #value less than 2200 meters.
         #The limit 1800 was decided after trail and error
         lg_trip = (lg_dist_pick < 2200) | (lg_dist_drop < 2200))
         #similarly for the lg_dist_pick or lg_dist_droop
```

5.4 Weather factor (External Dataset)

As we discussed above we know that weather factor has singificant effect on the *trip_duration*. Let's include the information related to weather. For this, we will use the external dataset called "weather_data_nyc_centralpark_2016". The csv file is obtained from this link. It contains further variables:

- **Date**
- **maximum temperature**
- **minimum temperature**
- **average temperature**
- **Precipitation**
- **snow fall**
- **snow depth**

Let's import the data and see the summary.

```
# import csv as tibble object
weather <- as.tibble(fread("C:/Users/Dell/OneDrive/UIC/Fall_2017/IDS 575/IDS575_pro1/weather_data_nyc_c

# see summary in presentable format using pander
pander(summary(weather), caption = "Weather data summary", digits = 4)
```

Table 10: Weather data summary (continued below)

date	maximum temperature	minimum temperature
Length:366	Min. :15.00	Min. :-1.00
Class :character	1st Qu.:50.00	1st Qu.:37.25
Mode :character	Median :64.50	Median :48.00
NA	Mean :64.63	Mean :49.81
NA	3rd Qu.:81.00	3rd Qu.:65.00

date	maximum temperature	minimum temperature
NA	Max. :96.00	Max. :81.00

average temperature	precipitation	snow fall	snow depth
Min. : 7.00	Length:366	Length:366	Length:366
1st Qu.:44.00	Class :character	Class :character	Class :character
Median :55.75	Mode :character	Mode :character	Mode :character
Mean :57.22	NA	NA	NA
3rd Qu.:73.50	NA	NA	NA
Max. :88.50	NA	NA	NA

```
# glmpse of data
glimpse(weather)
```

```
## Observations: 366
## Variables: 7
## $ date                  <chr> "1-1-2016", "2-1-2016", "3-1-2016", "4-1...
## $ `maximum temperature` <int> 42, 40, 45, 36, 29, 41, 46, 46, 47, 59, ...
## $ `minimum temperature` <int> 34, 32, 35, 14, 11, 25, 31, 31, 40, 40, ...
## $ `average temperature` <dbl> 38.0, 36.0, 40.0, 25.0, 20.0, 33.0, 38.5...
## $ precipitation         <chr> "0.00", "0.00", "0.00", "0.00", "0.00", ...
## $ `snow fall`           <chr> "0.0", "0.0", "0.0", "0.0", "0.0", "0.0"...
## $ `snow depth`          <chr> "0", "0", "0", "0", "0", "0", "0", "0", ...
```

Interestingly we observe letter "T" in variables *precipitation*, *snow fall*, and *snow depth*. From the datasource upon more investigating, this link concludes the meaning of "T" as tracecs. It means negligible value. we need to deal with this change of datatype in these variables.

Also,

```
weather <- weather %>%
  #in weather,
  mutate(date = dmy(date),
         #consider date as in datetime format using 'dmy'
         max_temp = `maximum temperature`,
         #rename maximum temperature
         min_temp = `minimum temperature`,
         #rename minimum temperature
         avg_temp = `average temperature`,
         #rename average temperature
         rain = as.numeric(ifelse(precipitation == "T", "0.01", precipitation)),
         #rename 'precipitation' as 'rain' and replace 'T' by negligible 0.01
         snow_fall = as.numeric(ifelse(`snow fall` == "T", "0.01", `snow fall`)),
         #rename 'snow fall' as 'snow_fall' and replace 'T' by negligible 0.01
         snow_depth = as.numeric(ifelse(`snow depth` == "T", "0.01", `snow depth`)),
         #rename 'snow depth' as 'snow_depth' and replace 'T' by negligible 0.01
         total_precip = snow_fall + rain,
         #create variable 'total_precip' which combines the values of amount of snow and amount of rain
         if_any = total_precip>0) %>%
```

```
  #create a flag if there was any weather factor on that day (OR of if_snow and _if_rain)
  dplyr::select(date, max_temp, min_temp, avg_temp, rain, snow_fall, snow_depth, total_precip, if_any)

#see glimpse of processed weatherr data
glimpse(weather)
```

```
## Observations: 366
## Variables: 9
## $ date         <date> 2016-01-01, 2016-01-02, 2016-01-03, 2016-01-04, ...
## $ max_temp     <int> 42, 40, 45, 36, 29, 41, 46, 46, 47, 59, 40, 44, 3...
## $ min_temp     <int> 34, 32, 35, 14, 11, 25, 31, 31, 40, 40, 26, 25, 2...
## $ avg_temp     <dbl> 38.0, 36.0, 40.0, 25.0, 20.0, 33.0, 38.5, 38.5, 4...
## $ rain         <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0...
## $ snow_fall    <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0...
## $ snow_depth   <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0...
## $ total_precip <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0...
## $ if_any       <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, ...
```

After data is pre processed, let's combine dataframes by key column 'date'. We will use 'left_join' function for this.

```
# extract date value from pickup_datetime variable in train. train <- train %>%
# mutate(date = date(ymd_hms(pickup_datetime)))

# combine over date
train <- left_join(train, weather, by = "date")
```

Let's visualize the trip durations and effect of weather.

```
# Let's see the relation between
p1 <- train %>% group_by(date) %>% summarise(trips = n(), total_precip = mean(total_precip)) %>%
    ggplot(aes(date, total_precip)) + geom_line(color = "red", size = 1) + labs(x = "",
    y = "Snowfall") + scale_y_sqrt() + scale_x_date(limits = ymd(c("2015-12-28",
    "2016-06-30")))

p2 <- train %>% group_by(date) %>% summarise(trips = n(), snow_depth = mean(snow_depth)) %>%
    ggplot(aes(date, snow_depth)) + geom_line(color = "red", size = 1) + labs(x = "",
    y = "Snow depth") + scale_y_sqrt() + scale_x_date(limits = ymd(c("2015-12-29",
    "2016-06-30")))

p3 <- train %>% group_by(date) %>% count() %>% ggplot(aes(date, n)) + geom_line(color = "red",
    size = 1) + labs(x = "", y = "trips per day")

layout <- matrix(c(1, 2, 3), 3, 1, byrow = FALSE)
multiplot(p1, p2, p3, layout = layout)
```

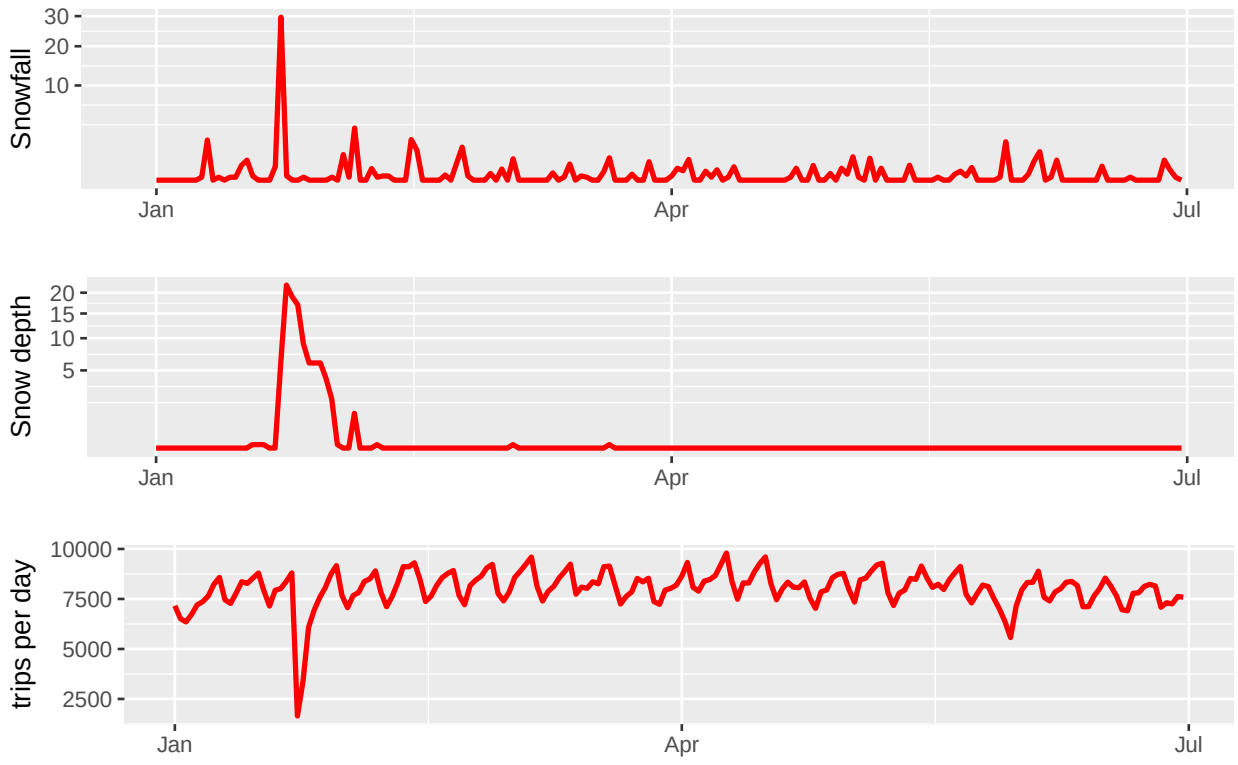


Figure 14: Relation between trip duration and weather

The above two graphs shows the amount of total precipitation (rain + snow) and snow depth. We can see that number of trips on the paprticular day are significantly affected by the weather factor. Whenever there were critical conditions weather wise, the number of trips dropped significantly. We will inculde this effect in our final model.

6 Data Cleaning

Let's proceed towards building the model. As from our exploratory data analysis, we discovered that, there are extreme values for few variables. They need to be taken care of before we build our model. We did not clean our data till now so that, we can get wholistic overview of our data.

6.1 Extreme values of trip_duration

During initial data visualization, we had observed few trips which were longer than a day. Let's use that code to see the dataframe with such extreme *trip_duration*.

6.1.1 trip_duration longer than a day.

```
outlies_more24 <- train %>% # in train,
filter(trip_duration > 24 * 3600)
```

```
# Filter data with trip_duration more than a day

# See this dataframe with only 4 variables of our interest.
pander((outlies_more24 %>% dplyr::select(pickup_datetime, dropoff_datetime, dist,
    speed)), caption = "trips with duration more than the day (Distance is in meters)",
    digits = 4)
```

Table 12: trips with duration more than the day (Distance is in meters)

pickup_datetime	dropoff_datetime	dist	speed
2016-01-05 00:19:42	2016-01-27 11:08:38	20171	0.03744
2016-02-13 22:38:00	2016-03-08 15:57:38	5989	0.01052
2016-01-05 06:14:15	2016-01-31 01:01:07	1637	0.002645
2016-02-13 22:46:52	2016-03-25 18:18:14	19923	0.02034

```
# digits=4 tries to override the number of columns shown in one table.
```

6.1.2 trip_duration < 24 Hrs & > 20 Hrs.

Also, let's see the data with *trip_duration* of 24 hours.

```
outlies_about24 <- train %>%
  # in train,
  filter(trip_duration <= 24*3600 & trip_duration > 20*3600)
  #filter data with trip_duration ranging from 10 hours to 24 hours.


#See this data using pander.
pander((outlies_about24 %>%
      #show this dataframe
      arrange(desc(dist)) %>%
      #arrange by descending order
      dplyr::select(pickup_datetime, dropoff_datetime, dist, speed) %>%
      #show pickup_datetime, dropoff_datetime, dist, speed variables
      head(10)),
      # And display first 10 datapoints
      caption = "trips with duration ranging from 10 hours to 24 hours (Distance is in meters)",
      digits = 4)
```

Table 13: trips with duration ranging from 10 hours to 24 hours (Distance is in meters)

pickup_datetime	dropoff_datetime	dist	speed
2016-06-04 13:54:29	2016-06-05 13:40:30	60666	2.553
2016-01-28 21:43:02	2016-01-29 21:33:30	42401	1.778
2016-03-14 22:46:05	2016-03-15 22:24:27	28116	1.189
2016-05-20 03:27:07	2016-05-21 00:09:11	24263	1.172
2016-06-18 17:41:47	2016-06-19 16:30:41	22808	0.9997
2016-06-01 19:52:42	2016-06-02 19:35:09	22759	0.96
2016-03-19 00:35:59	2016-03-19 23:54:45	22751	0.9759

pickup_datetime	dropoff_datetime	dist	speed
2016-03-31 13:01:29	2016-04-01 12:48:23	22639	0.952
2016-04-17 18:26:48	2016-04-18 17:27:14	22565	0.9808
2016-01-11 10:09:31	2016-01-12 10:05:27	22407	0.9363

Let's plot these data points as line segments from pickup location to dropoff location in New York City map.

```
# Combine both dataframes of outliers
outlies_total <- rbind(outlies_more24, outlies_about24)

# Get the New york city map as a polygon interesting observatiion to see 'map'.
# the data frame contains values of longitude and latitude of New York City
# perimeter.  connecting all these points will give us a shape of polygon for New
# City map.  map_data has already stored values of laongitude and latitude of
# major cities using parameters state and region more details
# http://homepage.divms.uiowa.edu/~luke/classes/STAT4580/maps.html
map <- as.tibble(map_data("state", region = "new york:manhattan"))

# for convenience of reading the map and faster script running, let's plot only
# 200 datapoints.
set.seed(1)
# set seed for reproducability of results
outlies_total <- outlies_total %>% sample_n(200)
# sybset dataframe

# save pickup location and dropoff location in two seperate dataframes later this
# will be used while calculcating path.
pickup_loc <- outlies_total %>% dplyr::select(lon = pickup_longitude, lat = pickup_latitude)
dropoff_loc <- outlies_total %>% dplyr::select(lon = dropoff_longitude, lat = dropoff_latitude)

# create a plot.
p1 <- ggplot() + geom_polygon(data = map, aes(x = long, y = lat), fill = "orange") +
    # layer of New York city polygon with orange colorusing map we created earliers.
geom_point(data = pickup_loc, aes(x = lon, y = lat), size = 1, color = "blue", alpha = 1) +
    # layers of pickup location as blue dots
geom_point(data = dropoff_loc, aes(x = lon, y = lat), size = 1, color = "red", alpha = 1)
# layers of dropoff location as red dots

# Now we need to draw line sengments connecting each pair of bue and red dots.
# Let's add layer for each pair of pickup and dropoff location to existing
# ggplot.  we will use for loop. for i from 1 to number of pairs we have to plot,
for (i in seq(1, nrow(pickup_loc))) {
    # create a tuple/dataframe which stores an i'th pair temporarily
    inter <- as.tibble(gcIntermediate(pickup_loc[i, ], dropoff_loc[i, ], n = 40,
        addStartEnd = TRUE))
    # using geom_line add a layer of line segment to existing graph
    p1 <- p1 + geom_line(data = inter, aes(x = lon, y = lat), color = "blue", alpha = 0.25)
    # Continue loop till it runs of datapoints.
}

# plot the graph.
p1 + ggtitle("Sample of Outlier trips in New York City")
```

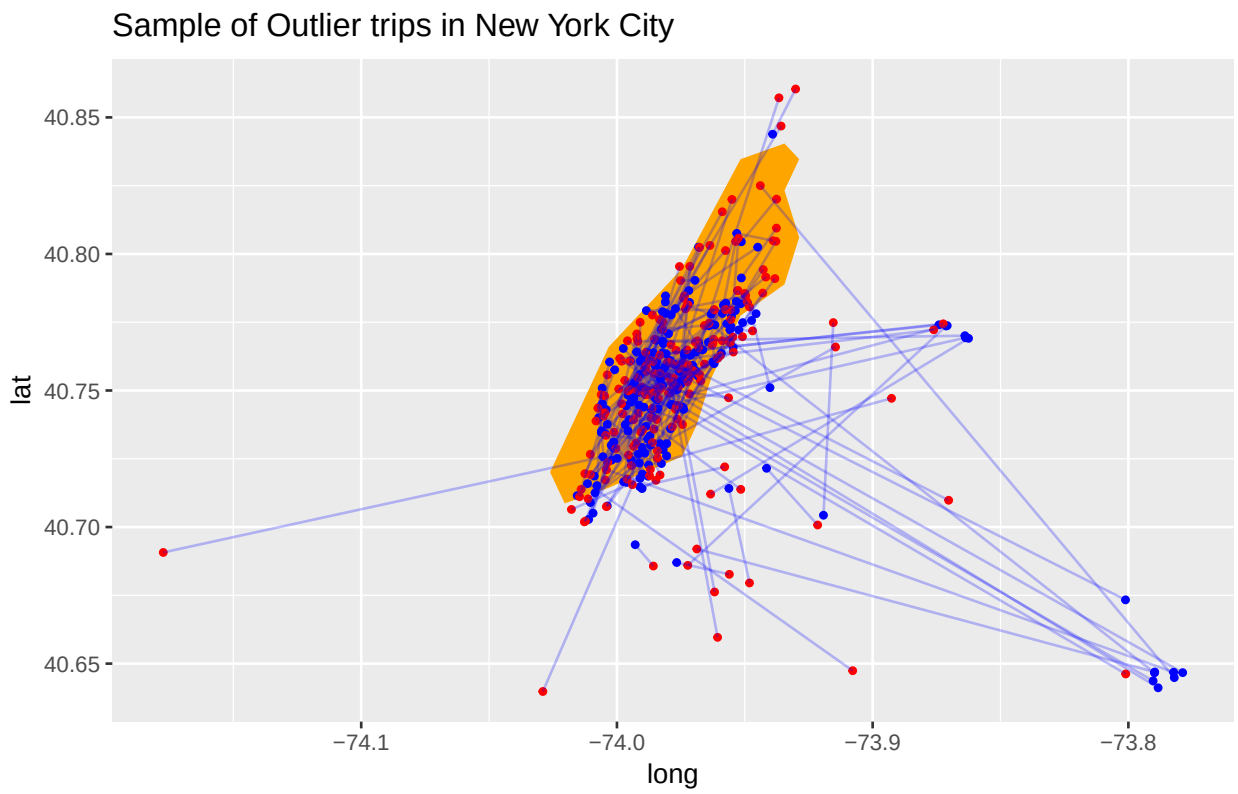


Figure 15: Outliers in trip duration on New York City Map

- Some datapoints suggest that these extreme *trip_duration*s were real as the path are too long.
- Remember the map of pickup location. We have airports on South East side of city. Intuitively, trips to or from any of the airports (most prominently JFK) are unlikely to be very short.
- The close distance of either pickup or dropoff to the airport could be a valuable predictor for longer *trip_duration*. This is something that we took from here to the feature engineering.
- We will remove datapoints with *trip_duration*s longer than 20 hours from the model.

6.1.3 trip_duration extreme short trip.

Till now we saw trips which had duration more than 20 hours. Let's see more details aboutt trips which were exceptionally shorter duration.

```
short_trips <- train %>%
  #in train,
  filter(trip_duration < 5*60)
  #filter by trip_duration less than 5 minutes.

# Let's print this data using pander
pander((
short_trips %>%
  arrange(dist) %>%
  #arrange in ascending order for variable dist
  dplyr::select(dist, pickup_datetime, dropoff_datetime, speed) %>%
```

```
  #show these 4 variables
  head(10)), caption = "short trip duration dataframe", digits = 4)
```

Table 14: short trip duration dataframe

dist	pickup_datetime	dropoff_datetime	speed
0	2016-02-29 18:39:12	2016-02-29 18:42:59	0
0	2016-01-27 22:29:31	2016-01-27 22:29:58	0
0	2016-01-22 16:13:01	2016-01-22 16:13:20	0
0	2016-01-18 15:24:43	2016-01-18 15:28:57	0
0	2016-05-04 22:28:43	2016-05-04 22:32:51	0
0	2016-05-23 14:20:49	2016-05-23 14:20:57	0
0	2016-04-18 12:55:14	2016-04-18 12:55:20	0
0	2016-01-21 12:07:45	2016-01-21 12:07:51	0
0	2016-04-27 15:14:54	2016-04-27 15:15:01	0
0	2016-04-11 15:30:02	2016-04-11 15:30:19	0

```
  #show head
```

Interestingly these trips have distance travel as zero. i.e. pickup and dropoff location coordinates are same. This suggests that the data contains cancelledd trips.

Let's see how many datapoints in total we have with same pickup and dropoff location.

```
# create a dtaframe with same location coordinates i.e. zero distance
distance_zero <- train %>% filter(near(dist, 0))

# show muber of datapoints. i.e. number of rows.
nrow(distance_zero)
```

```
## [1] 5897
```

Let's clean all these extreme datapoints and prepare the dataset on which model can be built.

```
train <- train %>%
  #in train,
  filter(trip_duration < 20*3600,
         #trip_durations less than 20 hours.
         dist > 0 | (near(dist, 0) & trip_duration < 60),
         #exclude datpoints with dist = 0 and
         #trip_durations < 60 seconds (minute long trips are unrealistic)
         jfk_dist_pick < 3e5 & jfk_dist_drop < 3e5,
         #Also, exlude datapoints which have pick up and drop off locations
         #more than 300 Kilometers. As we obderved with some data points.
         speed < 100)
         # exclude those datapoints which have average speed more than  100 km/Hr.
```

All these transofmration will be applied to the 'combine' dataframe which consists of training and testing dataset in future.

7 Fitting the model

7.1 Correlation

It is always good idea to see the correlation between different variables and between variables and response vavriables. The patterns in the dataset become visible in the correaltion plot.

```
train %>% #In train,
  dplyr::select(-id, -pickup_datetime, -dropoff_datetime, -jfk_dist_pick,
         -jfk_dist_drop, -lg_dist_pick, -lg_dist_drop,
         -date, -rain, -snow_fall, -min_temp, -max_temp) %>%
          # do not select these variables
  mutate(passenger_count = as.numeric(as.factor(passenger_count)),
         #change the datatype for these variables.
         vendor_id = as.numeric(vendor_id),
         #as spearman's corelation can process only numeric values.
         store_and_fwd_flag = as.numeric(as.factor(store_and_fwd_flag)),
         jfk_trip = as.numeric(as.factor(jfk_trip)),
         wday = as.numeric(wday),
         work = as.numeric(as.factor(work)),
         lg_trip = as.numeric(as.factor(lg_trip)),
         if_any = as.numeric(as.factor(if_any)),
         snow_depth = as.integer(snow_depth),
         total_precip = as.numeric(total_precip)) %>%
  dplyr::select(trip_duration, speed, everything()) %>%
  # show me first trip duration and speed and rest variables after
  cor(use="complete.obs", method = "spearman") %>%
  #spearman's corelation method.
  corrplot(type="lower", method="circle", diag=FALSE)
```

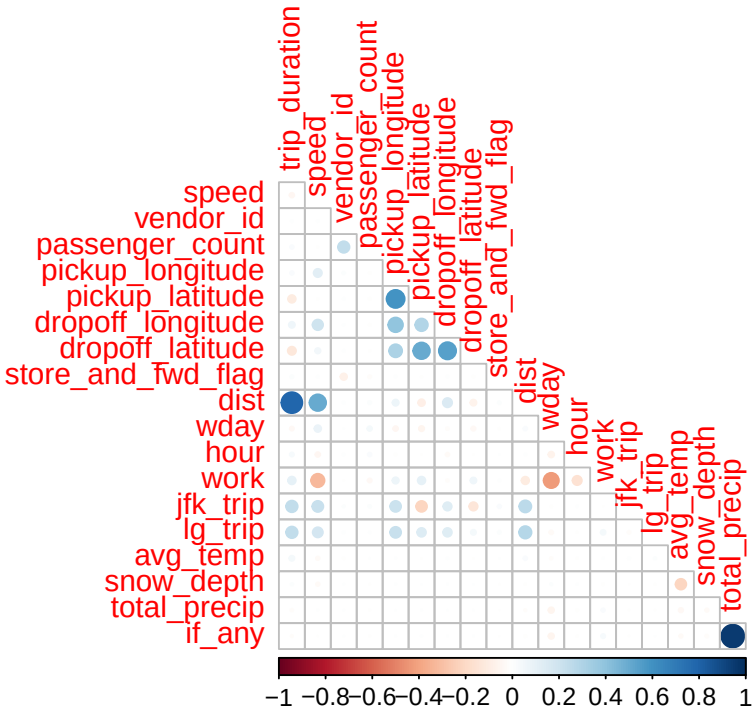


Figure 16: Corelation plot

```
#plot corrplot with lower diagonal style,show circular graph points.
```

From the ccorelationwe can takeaway further inferences,

- Varibale *dist* has the strongest corelation with the target variable *trip_duration*.
- As expected *pickup_latitude* and *pickup_longitude*, *dropoff_latitude* and *dropoff_longitude* are showing high corelation. Also, these loaction coordinates show some positive inter correation as well.
- *vendor_id* and *passanger_count* are showing moderate corelation.
- From feature engineering, the variables *jfk_trip* and *lg_trip* are showinig a small positive correlation with target varaiible *trip_duration*.
- Also, from feature engineering, the variable which we created to represent the time of the week for office hours, *work* shows moderate negative corelation with *speed* and *wday*. (it was derived from *wday*)
- Varibales related to airport trips show moderate positive corelation with the *speed* and *dist*.

7.2 dataframe prepeartion

Utill now we have done all the operations on training dataset for exploratory analysis perspective. We could have done it on all the datapoints i.e. dataframe *combine*. Let's apply all the above datacleaning and feature engineering operations to the *combine* dataframe. We will use this dataframe to build the model.

```
#airport coordinates again, just to be sure.
jfk_coord <- tibble(lon = -73.778889, lat = 40.639722)
la_guardia_coord <- tibble(lon = -73.872611, lat = 40.77725)

#derive distances using distCosine function.
pickup_loc <- combine %>% dplyr::select(pickup_longitude, pickup_latitude)
```

```r
#store the values of pickup coordinates
dropoff_loc <- combine %>% dplyr::select(dropoff_longitude, dropoff_latitude)
#store the values of dropoff coordinates
combine$dist <- distCosine(pickup_loc, dropoff_loc)
#create variable using distCosine as difference of above stored values.

combine$jfk_dist_pick <- distCosine(pickup_loc, jfk_coord)
#this variable store the value of distance from JFK airport to pickup location.
combine$jfk_dist_drop <- distCosine(dropoff_loc, jfk_coord)
#this variable store the value of distance from JFK airport to dropoff location.
combine$lg_dist_pick <- distCosine(pickup_loc, la_guardia_coord)
#this variable store the value of distance from LaGuardia airport to pickup location.
combine$lg_dist_drop <- distCosine(dropoff_loc, la_guardia_coord)
#this variable store the value of distance from LaGuardia airport to dropoff location.

# add dates
combine <- combine %>%
  #in combine
  mutate(pickup_datetime = ymd_hms(pickup_datetime),
         #change the type of datetime variable using ymd_hms()
         dropoff_datetime = ymd_hms(dropoff_datetime),
         #similar for dropoff variable.
         date = date(pickup_datetime)
         #create the new variable as only date value of pickup_datetime
  )


weather <- weather %>%
  dplyr::select(avg_temp, date, total_precip, snow_depth, if_any)
#select these variables from weather data

combine <- left_join(combine, weather, by = "date")
#combine weather data

# reformat to numerical and recode levels
combine <- combine %>%
  #in combine
  mutate(store_and_fwd_flag = as.integer(factor(store_and_fwd_flag)),
         #as factor store_and_fwd_flag
         vendor_id = as.integer(vendor_id),
         #vendor_id as integer
         month = as.integer(month(pickup_datetime)),
         #month from pick_up datetime and considering it as numeric value.
         wday = wday(pickup_datetime, label = TRUE),
         #weekday from pickup_datetime
         wday = as.integer(fct_relevel(wday, c("Sun", "Sat", "Mon", "Tues", "Wed", "Thurs", "Fri"))),
         #wday as factor
         hour = hour(pickup_datetime),
         #deriving hour from pickup_datetime
         work = as.integer( (hour %in% seq(8,18)) & (wday %in% c("Mon","Tues","Fri","Wed","Thurs")) ),
         #considering office time to craete the flag yes  or no. changing it to integer.
         jfk_trip = as.integer( (jfk_dist_pick < 1800) | (jfk_dist_drop < 1800) ),
         #trips to airport yes no flag converted to integer type data
```

```
        lg_trip = as.integer( (lg_dist_pick < 1800) | (lg_dist_drop < 1800) ),
        #similar for LaGuardia airport.
        if_any = as.integer(factor(if_any))
        #binary conversion of if_any
        )
```

Let's see the dataframe *combine* which we will use from now onwards.

```
glimpse(combine)
```

```
## Observations: 2,083,778
## Variables: 28
## $ id                 <chr> "id2875421", "id2377394", "id3858529", "id3...
## $ vendor_id          <int> 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2...
## $ pickup_datetime    <dttm> 2016-03-14 17:24:55, 2016-06-12 00:43:35, ...
## $ dropoff_datetime   <dttm> 2016-03-14 17:32:30, 2016-06-12 00:54:38, ...
## $ passenger_count    <int> 1, 1, 1, 1, 1, 6, 4, 1, 1, 1, 1, 4, 2, 1, 1...
## $ pickup_longitude   <dbl> -73.98215, -73.98042, -73.97903, -74.01004,...
## $ pickup_latitude    <dbl> 40.76794, 40.73856, 40.76394, 40.71997, 40....
## $ dropoff_longitude  <dbl> -73.96463, -73.99948, -74.00533, -74.01227,...
## $ dropoff_latitude   <dbl> 40.76560, 40.73115, 40.71009, 40.70672, 40....
## $ store_and_fwd_flag <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ trip_duration      <int> 455, 663, 2124, 429, 435, 443, 341, 1551, 2...
## $ dset               <fctr> train, train, train, train, train, train, ...
## $ dist               <dbl> 1500.1995, 1807.5298, 6392.2513, 1487.1625,...
## $ jfk_dist_pick      <dbl> 22315.02, 20258.98, 21828.54, 21461.51, 236...
## $ jfk_dist_drop      <dbl> 21025.4008, 21220.9775, 20660.3899, 21068.1...
## $ lg_dist_pick       <dbl> 9292.897, 10058.778, 9092.997, 13228.107, 8...
## $ lg_dist_drop       <dbl> 7865.248, 11865.578, 13461.012, 14155.920, ...
## $ date               <date> 2016-03-14, 2016-06-12, 2016-01-19, 2016-0...
## $ avg_temp           <dbl> 45.5, 72.5, 22.0, 39.0, 46.5, 33.5, 70.5, 6...
## $ total_precip       <dbl> 0.29, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0...
## $ snow_depth         <dbl> 0.00, 0.00, 0.01, 0.00, 0.00, 6.00, 0.00, 0...
## $ if_any             <int> 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1...
## $ month              <int> 3, 6, 1, 4, 3, 1, 6, 5, 5, 3, 5, 5, 2, 6, 5...
## $ wday               <int> 3, 1, 4, 5, 2, 2, 7, 2, 7, 6, 4, 1, 7, 5, 7...
## $ hour               <int> 17, 0, 11, 19, 13, 22, 22, 7, 23, 21, 22, 1...
## $ work               <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ jfk_trip           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ lg_trip            <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
```

```
pander(summary(combine), caption = "combine dataframe summary", digits = 4)
```

Table 15: combine dataframe summary (continued below)

id	vendor_id	pickup_datetime
Length:2083778	Min. :1.000	Min. :2016-01-01 00:00:17
Class :character	1st Qu.:1.000	1st Qu.:2016-02-17 17:48:02
Mode :character	Median :2.000	Median :2016-04-01 18:14:21
NA	Mean :1.535	Mean :2016-04-01 11:09:23
NA	3rd Qu.:2.000	3rd Qu.:2016-05-15 06:05:20
NA	Max. :2.000	Max. :2016-06-30 23:59:58
NA	NA	NA

Table 16: Table continues below

dropoff_datetime	passenger_count	pickup_longitude
Min. :2016-01-01 00:03:31	Min. :0.000	Min. :-121.93
1st Qu.:2016-02-17 17:05:32	1st Qu.:1.000	1st Qu.: -73.99
Median :2016-04-01 17:35:12	Median :1.000	Median : -73.98
Mean :2016-04-01 10:26:24	Mean :1.664	Mean : -73.97
3rd Qu.:2016-05-15 04:10:51	3rd Qu.:2.000	3rd Qu.: -73.97
Max. :2016-07-01 23:02:03	Max. :9.000	Max. : -61.34
NA's :625134	NA	NA

Table 17: Table continues below

pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag
Min. :34.36	Min. :-121.93	Min. :32.18	Min. :1.000
1st Qu.:40.74	1st Qu.: -73.99	1st Qu.:40.74	1st Qu.:1.000
Median :40.75	Median : -73.98	Median :40.75	Median :1.000
Mean :40.75	Mean : -73.97	Mean :40.75	Mean :1.006
3rd Qu.:40.77	3rd Qu.: -73.96	3rd Qu.:40.77	3rd Qu.:1.000
Max. :51.88	Max. : -61.34	Max. :48.86	Max. :2.000
NA	NA	NA	NA

Table 18: Table continues below

trip_duration	dset	dist	jfk_dist_pick
Min. : 1	test : 625134	Min. : 0	Min. : 138
1st Qu.: 397	train:1458644	1st Qu.: 1233	1st Qu.: 20647
Median : 662	NA	Median : 2096	Median : 21282
Mean : 959	NA	Mean : 3442	Mean : 20802
3rd Qu.: 1075	NA	3rd Qu.: 3882	3rd Qu.: 21962
Max. :3526282	NA	Max. :1242299	Max. :4128727
NA's :625134	NA	NA	NA

Table 19: Table continues below

jfk_dist_drop	lg_dist_pick	lg_dist_drop	date
Min. : 432	Min. : 174	Min. : 147	Min. :2016-01-01
1st Qu.: 20587	1st Qu.: 8464	1st Qu.: 8347	1st Qu.:2016-02-17
Median : 21272	Median : 9715	Median : 9603	Median :2016-04-01
Mean : 20966	Mean : 9733	Mean : 9768	Mean :2016-03-31
3rd Qu.: 22015	3rd Qu.: 11115	3rd Qu.: 11091	3rd Qu.:2016-05-15
Max. :4128726	Max. :4118057	Max. :4118057	Max. :2016-06-30
NA	NA	NA	NA

Table 20: Table continues below

avg_temp	total_precip	snow_depth	if_any
Min. : 7.00	Min. : 0.0000	Min. : 0.0000	Min. :1.000

avg_temp	total_precip	snow_depth	if_any
1st Qu.:39.00	1st Qu.: 0.0000	1st Qu.: 0.0000	1st Qu.:1.000
Median :51.50	Median : 0.0000	Median : 0.0000	Median :1.000
Mean :51.51	Mean : 0.1484	Mean : 0.4056	Mean :1.408
3rd Qu.:63.00	3rd Qu.: 0.0400	3rd Qu.: 0.0000	3rd Qu.:2.000
Max. :81.50	Max. :29.6100	Max. :22.0000	Max. :2.000
NA	NA	NA	NA

Table 21: Table continues below

month	wday	hour	work	jfk_trip
Min. :1.000	Min. :1.000	Min. : 0.00	Min. :0	Min. :0.00000
1st Qu.:2.000	1st Qu.:2.000	1st Qu.: 9.00	1st Qu.:0	1st Qu.:0.00000
Median :4.000	Median :4.000	Median :14.00	Median :0	Median :0.00000
Mean :3.518	Mean :4.071	Mean :13.61	Mean :0	Mean :0.02874
3rd Qu.:5.000	3rd Qu.:6.000	3rd Qu.:19.00	3rd Qu.:0	3rd Qu.:0.00000
Max. :6.000	Max. :7.000	Max. :23.00	Max. :0	Max. :1.00000
NA	NA	NA	NA	NA

lg_trip
Min. :0.00000
1st Qu.:0.00000
Median :0.00000
Mean :0.03746
3rd Qu.:0.00000
Max. :1.00000
NA

7.3 Defining role of features

We can consider all the input features we have in the *combine* dataset. However, when we did feature engineering, we derived some features based on existing features. These original and respective derived features will be definitely colinear in nature. Hence w e need to take care of this while selecting our predictors.

Also, with the help correlation matrix we potted earlier, we can excluded either of the features in a pair which has strong correlation. hence below, train_cols selects only such features which avoid the correlation.

Let's define few terminologies for features as which role they will be playing in building model.

```
train_cols <- c("vendor_id", "passenger_count", "pickup_longitude", "pickup_latitude",
    "store_and_fwd_flag", "dist", "jfk_trip", "lg_trip", "hour", "avg_temp", "total_precip",
    "snow_depth", "wday", "month")
# input features

y_col <- c("trip_duration")
# target feature

id_col <- c("id")
# identification feature
```

```
aux_cols <- c("dset")
# train et index

clean_cols <- c("jfk_dist_drop", "jfk_dist_pick")
# features by which we will filter the data

# extract test id column
test_id <- combine %>% # from combine
filter(dset == "test") %>% # select those datapoints which have dset as test
select_(.dots = id_col)
# Select columns by vector of names using dplyr

# all relevant columns for train/test
cols <- c(train_cols, y_col, aux_cols, clean_cols)
# use these columns
combine <- combine %>% # in combine
select_(.dots = cols)
# Select columns by vector of names using dplyr

# split train/test
train <- combine %>% # in combine
filter(dset == "train") %>% # filter by dset = train
select_(.dots = str_c("-", c(aux_cols)))
# donot select auxilary columns i.e. index

test <- combine %>% # similary for test. in combine,
filter(dset == "test") %>% # filter by dset = test
select_(.dots = str_c("-", c(aux_cols, clean_cols, y_col)))
# Select all except index column, cleaning features and response variable.
```

7.4 Transforming Response variable.

As from our intensive exploratory data analysis we came to know that the response variable has the wide range of values and those values realistic. Which means we can not neglect these values in our model. So if we consider these values as they are, then some datapoints with extreme *trip_duration* will introduce undesired bias. To handle this issue, the classic solutionis data transformation. Log_transformation will serve the purpose t bring the wide range of values on a level playing field.

One thing to keep in mind that there could be some datapoints with $trip\_duration = 0$. We have got this warning above during our exploration. So we will add 1 to each value to avoid this algebric problem.

Another thing to keep in mind realted to log transformation is, about loss function. Regular error metric 'Mean_squared_error' will no longer be applicable for calculating performance of the model. We have to consider 'root mean squarred logarithmic error'.

```
# in train, perform log transfomration and add 1.
train <- train %>% mutate(trip_duration = log(trip_duration + 1))
```

7.5 Validaton datasset creation

We will use validation dataset strategy for assessing the generality of the model over data. Caret offers inbuilt datapartition feature. Which is more convenient to use.

```r
# for reproduability
set.seed(1)

# using caretDataPartition with 80 percent split.
trainIndex <- createDataPartition(train$trip_duration, p = 0.8, list = FALSE, times = 1)

# split into validation and train using above created index.
train <- train[trainIndex, ]
valid <- train[-trainIndex, ]
```

7.6 Final cleaning the training dataset

With all the insights we gained throughout this project, we will apply cleaning using cleaning_features which we defined earlier. This stp is necessary as we are building model based on

```r
# From validation select everything except cleaning features.
valid <- valid %>% select_(.dots = str_c("-",c(clean_cols))) #exclude cleaning features from validation

train <- train %>% #in train,
  #exclude datpoints with *trip_duration* more than 18 hours
  #exclude datapoints with jfk_dist_pick and jfk_dist_drop values more than 200 Km.
  filter(trip_duration < 18*3600, jfk_dist_pick < 200000 & jfk_dist_drop < 200000) %>%
  #select everything except cleaning features
  select_(.dots = str_c("-",c(clean_cols)))
```

7.7 Simple Linear regression

7.7.1 Fit Linear regrssion model on train data

Let's fit the simplest model as linear regrssion. This will give us generall idea about statistical significance of the variables.

```r
lr <- lm(trip_duration ~ ., data = train)
# linear regression to predict trip_duration using all rest of the variables.
summary(lr)
```

```
##
## Call:
## lm(formula = trip_duration ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -26.1520  -0.3050   0.0418   0.3549   8.5041
##
## Coefficients:
##                      Estimate Std. Error  t value Pr(>|t|)
## (Intercept)        -7.427e+00  2.001e+00   -3.712 0.000205 ***
## vendor_id           1.909e-02  1.159e-03   16.473  < 2e-16 ***
## passenger_count     7.477e-03  4.382e-04   17.064  < 2e-16 ***
## pickup_longitude   -9.019e-01  2.144e-02  -42.071  < 2e-16 ***
## pickup_latitude    -1.317e+00  2.290e-02  -57.495  < 2e-16 ***
## store_and_fwd_flag -5.524e-03  7.475e-03   -0.739 0.459919
## dist                1.603e-04  2.026e-07  791.088  < 2e-16 ***
```

```
## jfk_trip           -1.190e+00  6.022e-03 -197.564  < 2e-16 ***
## lg_trip            -4.363e-02  3.619e-03  -12.055  < 2e-16 ***
## hour                6.361e-03  8.665e-05   73.406  < 2e-16 ***
## avg_temp           -1.961e-04  6.101e-05   -3.214 0.001311 **
## total_precip        2.244e-04  5.296e-04    0.424 0.671742
## snow_depth          1.273e-02  2.558e-04   49.739  < 2e-16 ***
## wday                2.818e-02  2.759e-04  102.129  < 2e-16 ***
## month               2.289e-02  5.818e-04   39.335  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5961 on 1166873 degrees of freedom
## Multiple R-squared:  0.4385, Adjusted R-squared:  0.4385
## F-statistic: 6.509e+04 on 14 and 1166873 DF,  p-value: < 2.2e-16
```

```
# see summary of this regression
```

Let's analyse the summary of linear regression.

7.7.2 Anlaysis of Linear regression

residuals Let's analyze the distribution of residuals using boxplot. Ideally it should be symmetricacl representing normal distribution of residuals.

```
# recall the code from 3rd assignment 1.3.2 B
boxplot(resid(lr))
# boxplot
abline(h = min(resid(lr)), col = "Blue")
# minimum value by blue line
abline(h = max(resid(lr)), col = "Yellow")
# maxresidual value by yellow line
abline(h = median(resid(lr)), col = "Green")
# median value by green line
abline(h = quantile(resid(lr), c(0.25, 0.75)), col = "Red")
```

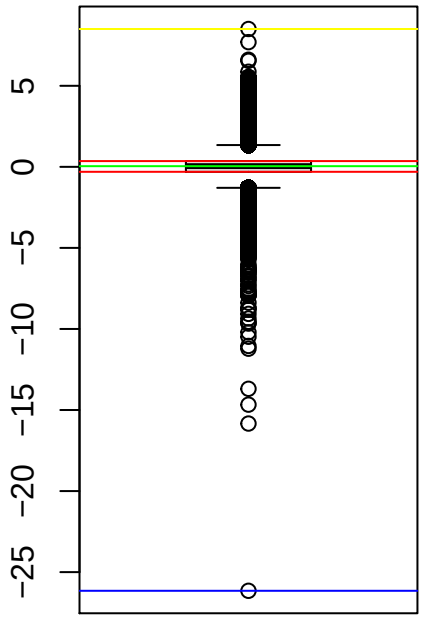


Figure 17: Distribution of residuals in linear regression

```
# quantiles boundries by red color
```

It is difficult to conclude that the residuals are symmetrically distributed. This is because of extrreme outliers. Length of whiskers shows some symmetry. Let's try to analyse other information in the summary.

β_j

From the coefficients summary all coefficients are shown most important for the models except coeficient of 'store_and_fwd_flag'. It is difficult to compare the relative importance of the features.

R square Adjusted R square and R square being around 43% show the poor fit of the model over the data. The residual standard error of 0.59 confirms the poor fit.

With such poor model, there's no point in predicting *trip_duration* for test data. We will try ensemble techniique to improve accuracy. We will fit gradient boosting model.

7.8 Parameters for xgboosting

eXtreme Gradient Boosting: From all the lectures, algorithms learned in the class and all the assignments, I experienced that ensemble techniques provides the best accuracy. The concept of popular vote with many models works very well for problem statement which includes diverse features.

In the mid report, one of the parameter I used for XGBoosting was objective as 'reg:linear'. I decided to go in more depth of the possible choices for this attribute along with different parameters. After trial and error, few of these parameters are finalized. Important details about the parameters are,

- **booster** [default=gbtree]
  - which booster to use, can be gbtree, gblinear or dart. gbtree and dart use tree based model while gblinear uses linear function.
- **eta** [default=0.3, alias: learning_rate]
  - step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative.
- **colsample_bylevel** [default=1]
  - subsample ratio of columns for each split, in each level.
  - range: (0,1]
- **subsample** [default=1]
  - subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collected half of the data instances to grow trees and this will prevent overfitting.
  - range: (0,1]
- **max_depth** [default=6]
  - maximum depth of a tree, increase this value will make the model more complex / likely to be overfitting. 0 indicates no limit, limit is required for depth-wise grow policy.
  - range: $[0, \infty]$

7.8.1 Attempt to determine the distribution

When we try to implement a statistical model, we assume the normal distribution of our data. But the real world data hardly ever follows perfectly bell shaped normal curve. If we ignore this imperfection and still try to test the accuracy of the fitted statistical model, it results into higher error. This section of this report addresses this problem. I tried best to interprete all the literature I could find relted to this problem in given time. Now zeroing down to the twp important parameters eval_metric and objective, I studied about possible choices,

7.8.1.1 objective [default=reg:linear]

- "reg:linear" –linear regression
- "reg:logistic" –logistic regression
- "binary:logistic" –logistic regression for binary classification, output probability
- max_delta_step is set to 0.7 by default in poisson regression (used to safeguard optimization)
- "multi:softmax" –set XGBoost to do multiclass classification using the softmax objective, you also need to set num_class(number of classes)
- "multi:softprob" –same as softmax, but output a vector of ndata * nclass, which can be further reshaped to ndata, nclass matrix. The result contains predicted probability of each data point belonging to each class.
- "reg:tweedie" –Tweedie regression with log-link. It might be useful, e.g., for modeling total loss in insurance, or for any outcome that might be Tweedie-distributed.

The idea behind the different possible choices is that, to choose the distribution which closely resembles with the data we are dealing with. Now before we choose the value for objective, let's try to examine the distribution of the trip_duration. I decided to carry out this step from further links, link_1, link_2, link_3. The package that is being used here is fitdistrplus. More details are on link.

Let's use the the function and see the distribution,

```
# http://civil.colorado.edu/~balajir/CVEN5454/R-sessions/sess2/intro2fitdistrplus.pdf
plotdist(train$trip_duration)
```

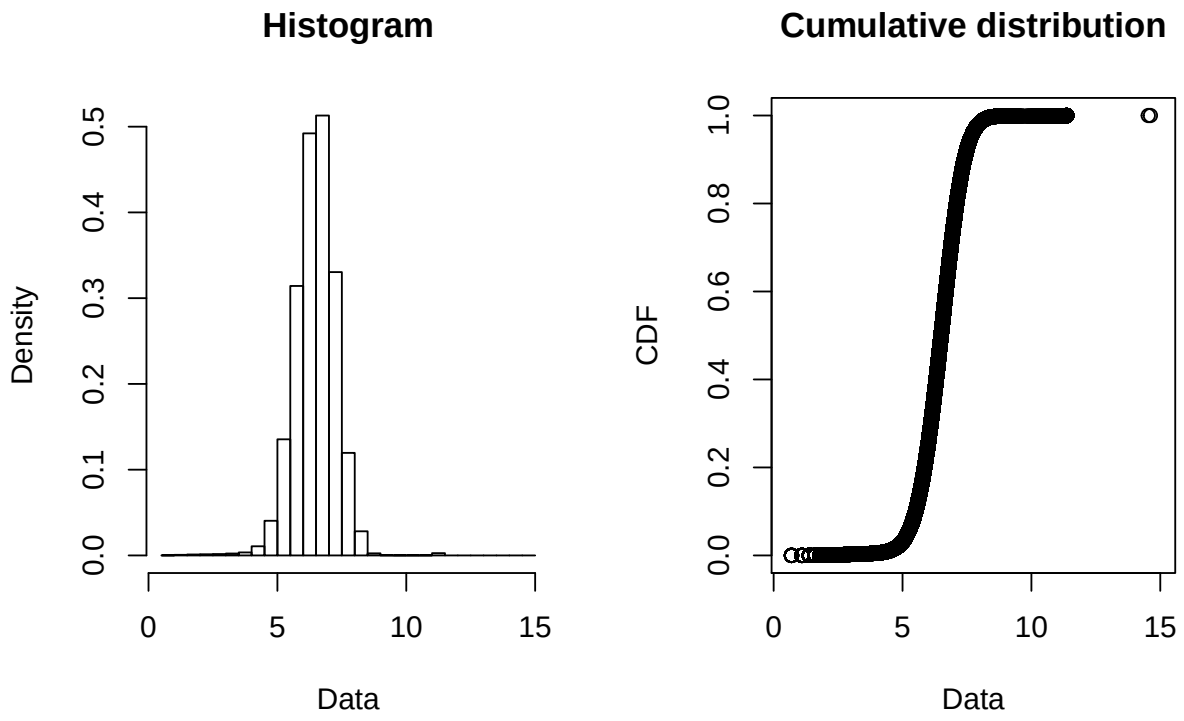


Figure 18: distribution of tripDuration using fitdistrplus

Data seems fairly symmetrical. It has tails almost symmetrical length. With few negligible spikes at both ends. It must be because of extreme trip duration we observed in the data and few trips with zero trip_duration. These trips are legit. We cannot exclude these trips considering outliers.

Initial intuition says that we are deaing with distribution which is more likely described using log scale. This is because considerably majority of datpoints have similar trip_duration (high pick of distribution). And tails are quite short. When there are such extremities in the data, we can verify the possibility of log transformation related distribution.

```
descdist(train$trip_duration)
```

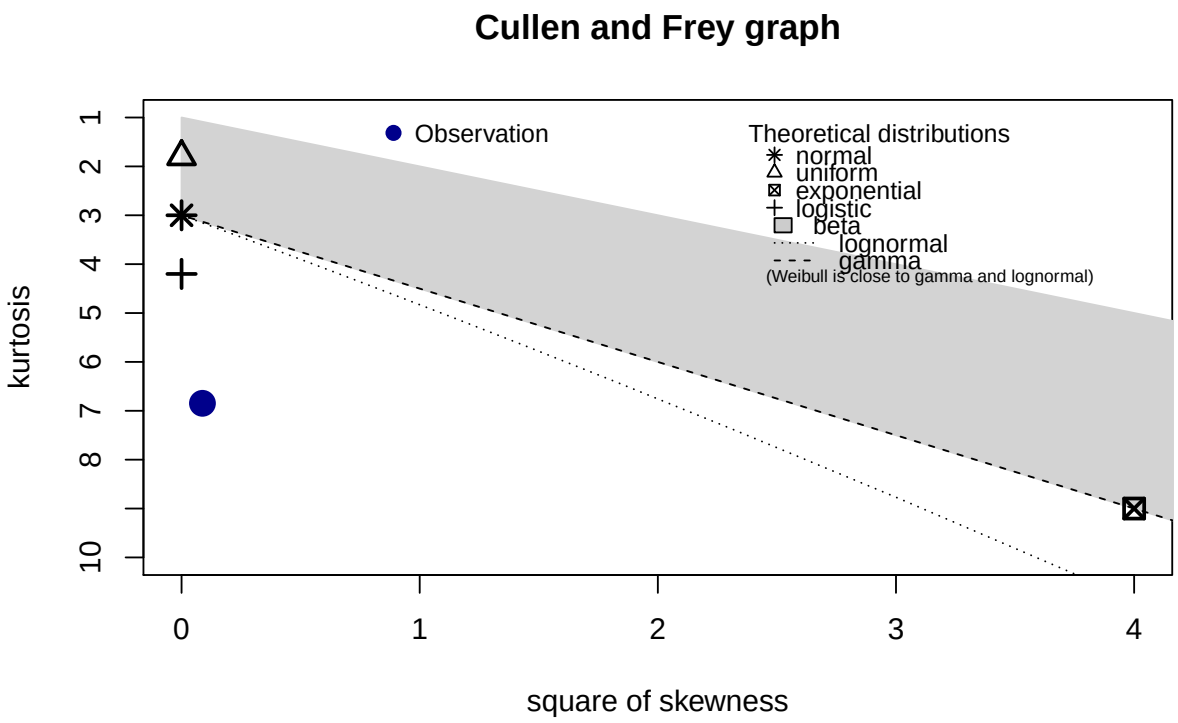


Figure 19: Cullen and Frey Graph for trip duration distribution

```
## summary statistics
## ------
## min:  0.6931472   max:  14.61644
## median:  6.496775
## mean:  6.466776
## estimated sd:  0.7954538
## estimated skewness:  -0.296164
## estimated kurtosis:  6.846708
```

Let's analyze the obtained information. From link, I conclude further things,

- from the summary information, we get mean, median, min, and max statistics, there's not much difference between median and mean.
- skewness is negative but close to zero. i.e. mean is less than median.
- From Cullen and Frey graph, we see that, the observation fairly symmetrical (square of skewness near zero).
- Also kurtosis (excess) (more information here), is 7. It shows more pointedness of data.
- Also, from graph, the observation has skewness and kurtosis resembling with lognormal distribution. According to information studied from link, let's bootstrap the samples and claim this assertion with some confidence.
- I learned that, even though the observavtion is in the region for *lognormal* distribution, we should not conclude the distribution. This above figure just confirms the characteristic about skewness and kurtosis of the data.
- We will use descdist() with boot = n for n bootstrap sample. data has almost 1 million samples. My system crashes for more than 100 samples. Let's see results.

```
descdist(train$trip_duration, boot = 100, discrete = FALSE, obs.pch = 3)
```

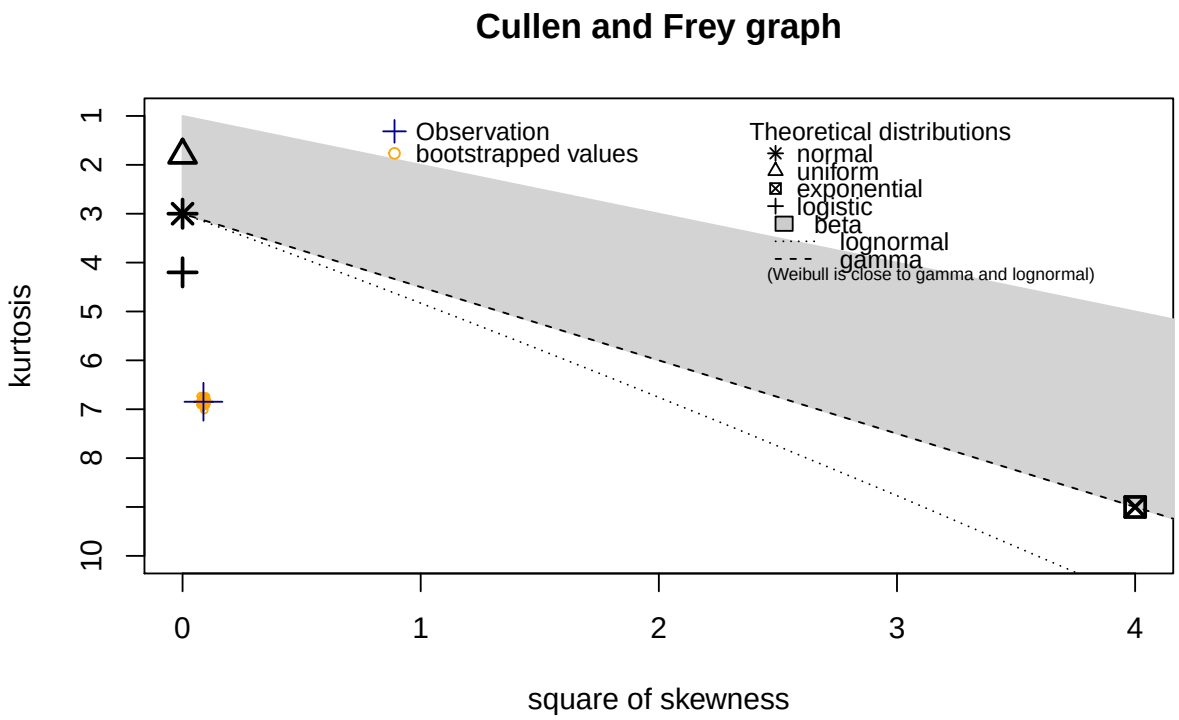


Figure 20: distribution for n bootstrap samples

```
## summary statistics
## ------
## min:  0.6931472   max:  14.61644
## median:  6.496775
## mean:  6.466776
## estimated sd:  0.7954538
## estimated skewness:  -0.296164
## estimated kurtosis:  6.846708
```

All bootstrap samples show close resemblance of skewness and kurtosis values with original data.

We can check the fit for the lognormal distribution. Upon learning more about methods by which can fit one or more parametric distributions may then be fitted to the data set, one at a time, using the function *fitdist*. This function uses the *maximum likelihood* method if the argument method="mle" (or if it is omitted) or the matching moments estimation if the argument method="mme". When fitting continuous distributions, Kolmogorov-Smirnov and Anderson-Darling statistics are computed and corresponding tests are performed when possible. Even if less appropriate for continuous distributions, the Chi-squared statistic is also computed when possible.

```
f1g <- fitdist(train$trip_duration, "lnorm", method = "mle")
plot(f1g)
```

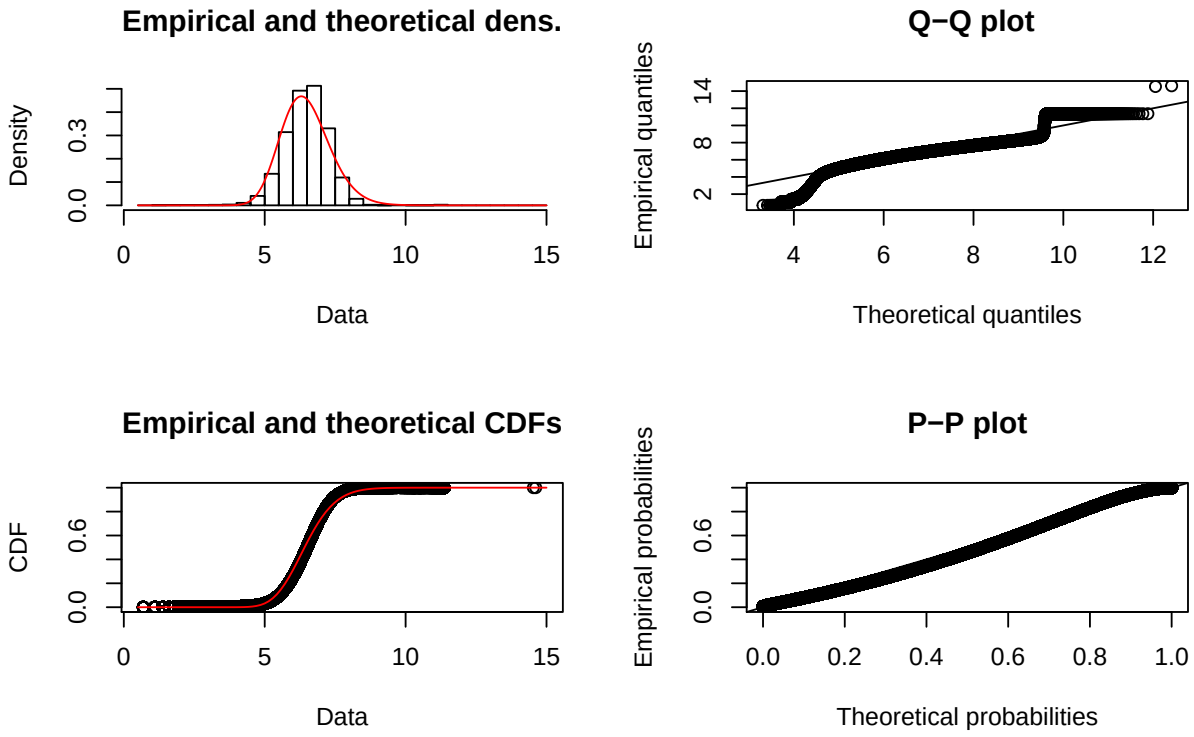


Figure 21: fitting log normal distribution

before we analyze this grpah in more details, let's also have the summary of the the above figure.

```
summary(f1g)
```

```
## Fitting of the distribution ' lnorm ' by maximum likelihood
## Parameters :
##          estimate   Std. Error
## meanlog 1.8583441 1.241919e-04
## sdlog   0.1341553 8.779496e-05
## Loglikelihood:  -1480227   AIC:  2960458   BIC:  2960482
## Correlation matrix:
##              meanlog        sdlog
## meanlog 1.000000e+00 1.586657e-12
## sdlog   1.586657e-12 1.000000e+00
```

Starting from the graph, to interprete the results, I referred to this link.

- As data is plotted against empirical and theorotical density of log normal distribution, we observe the close fit. One might say that, distribution is fairly log normal.
- However, qq plot tells us another story. It shows log normal distribution is not a good fit at all.
- We can verify this bad fit using summary. In the summary, we can get estimate for the parameters of log noraml distribution as meanlog and sdlog (we got using mximum liklihood method). Using these estimates, we can define parameters for qq plot.
- Below is the qqplot for the log normal distribution with the parameters given by maximum liklihood estimate. We plugged in meanlog and sdlog of the *fitdist()* object.

```r
set.seed(1)
data_list <- sample(train$trip_duration, 1e+06, replace = FALSE)
qqPlot(data_list, distribution = "lnorm", meanlog = f1g$estimate[1], sdlog = f1g$estimate[2])
```

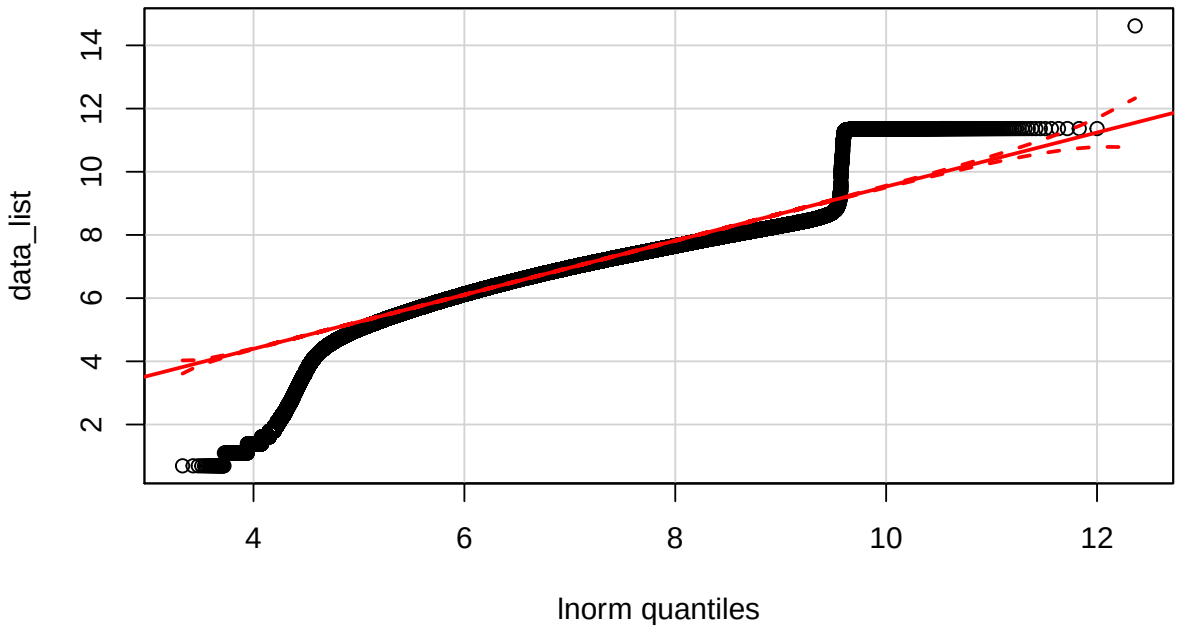


Figure 22: attempt to fit Log normal distribution for trip duration

So far we tried with conventional log normal distribution. We tried to fit this lognoraml distribution using the parameters we obtained from maximum liklihood estimation. But the distribution is not quite log normal. QQ plot clearly shows the difference. The possibility is that, the data could be mixture of different distributions.

At this time, I came across of two parameterized probability distributions to deal with this undefined distribution problem. One is, weibull distribution and another is tweedie distribution.

In the xgboosting model, I found **"reg:tweedie** as possible option for the parameter objejctive. So, I searched for and learned about tweedie first and further are my learnings. (from link)

In probability and statistics, the Tweedie distributions are a family of probability distributions which include the purely continuous normal and gamma distributions, the purely discrete scaled Poisson distribution, and the class of mixed compound Poisson gamma distributions which have positive mass at zero, but are otherwise continuous.

In simple words, it can be explained as generalised form of linear model family object with any power variance function and any power link. Here is the link. It takes two arguments, tweedie(var.power=0, link.power=1-var.power)

- var.power - index of power variance function
- link.power - index of power link function. link.power=0 produces a log-link. Defaults to the canonical link, which is 1-var.power.

This function provides access to a range of generalized linear model response distributions which are not otherwise provided by R, or any other package for that matter. It is also useful for accessing distribution/link combinations which are disallowed by the R glm function.

Further is summary of my learning about tweedie reg.

- For any random variable γ that obeys a Tweedie distribution, the variance var(Y) relates to the mean E(Y) by the power law, $Var(Y) = \alpha[E(Y)]^p$
- where a and p are positive constants.
- The Tweedie distributions include a number of familiar distributions as well as some unusual ones, each being specified by the domain of the index parameter. We have the,
- normal distribution, p = 0,
- Poisson distribution, p = 1,
- compound Poisson-gamma distribution, 1 < p < 2,
- gamma distribution, p = 2,
- positive stable distributions, 2 < p < 3,
- inverse Gaussian distribution, p = 3,
- positive stable distributions, p > 3, and
- extreme stable distributions, p = ∞.
- For 0 < p < 1 no Tweedie model exists.

Also for weibull distribution two parameters are shape and scale. We can find these parameters by 4 methods, using Maximum liklihood extimation, Using Less square fit, using median and quartiles, using mean and standard deviation. Let's use *fitdstr()*. I learned this from reading this page.

```
# 1. Estimate k and c by MLE

# https://stats.stackexchange.com/questions/60511/weibull-distribution-parameters-k-and-c-for-wind-spee
# http://blog.minitab.com/blog/meredith-griffith/identifying-the-distribution-of-your-data

weibull_fit <- fitdistr(train$trip_duration, densfun = "weibull", lower = 0)
k1 <- weibull_fit$estimate[1]
c1 <- weibull_fit$estimate[2]

# 2. Estimate k and c using the least square fit

n <- 100  # number of bins
breaks <- seq(0, max(train$trip_duration), length.out = n)

freqs <- as.vector(prop.table(table(cut(train$trip_duration, breaks = breaks))))
cum.freqs <- c(0, cumsum(freqs))

xi <- log(breaks)
yi <- log(-log(1 - cum.freqs))

# Fit the linear regression
least.squares <- lm(yi[is.finite(yi) & is.finite(xi)] ~ xi[is.finite(yi) & is.finite(xi)])
lin.mod.coef <- coefficients(least.squares)

k2 <- lin.mod.coef[2]
```

```r
c2 <- exp(-lin.mod.coef[1]/lin.mod.coef[2])

# 3. Estimate k and c using the median and quartiles

med <- median(train$trip_duration)
quarts <- quantile(train$trip_duration, c(0.25, 0.75))

k3 <- log(log(0.25)/log(0.75))/log(quarts[2]/quarts[1])
c3 <- med/log(2)^(1/k3)

# 4. Estimate k and c using mean and standard deviation.

k4 <- (sd(train$trip_duration)/mean(train$trip_duration))^(-1.086)
c4 <- mean(train$trip_duration)/(gamma(1 + 1/k4))
```

Acordingly we get,

- By maximum liklihood estimation, shape = 7.8655277 and scale = 6.802365.
- By the least square fit, shape = 5.4226591 and scale = 7.6441408.
- By the median and quartiles, shape = 10.2315821 and scale = 6.7337192.
- By mean and standard deviation, shape = 9.7350704 and scale = 6.8052744.

All methods provide the values of shape and scale of weibull distribution very close to each other. Let's use these parameters and see the qq plot for *weibull* distribution.

```r
qqPlot(train$trip_duration, distribution = "weibull", shape = k1, scale = c1)
```

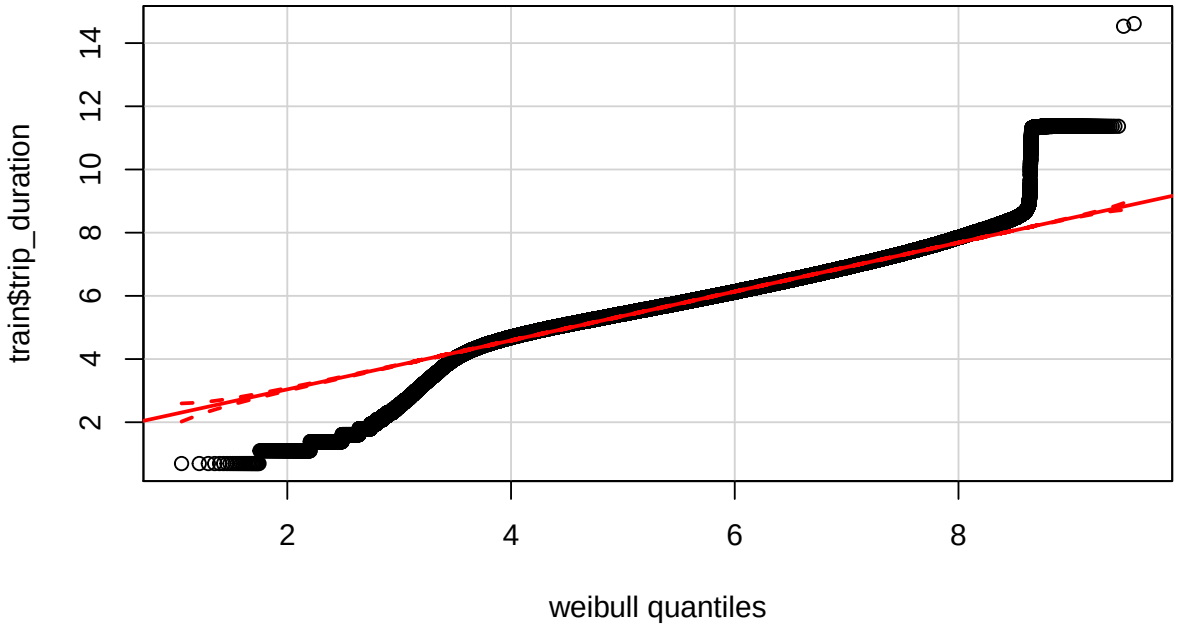


Figure 23: qqplot for weibull distribution using parameters

It seems, we need to optimize the shape and scale parameters. Recall the 4th assignment, where we implemented SVM using cv for different values of error and gamma. We can use the same code to identify best values of shape and scale.

Conclusion

Learned from here

- The pp-values of a Kolmovorov-Smirnov-Test (KS-Test) with estimated parameters will be quite wrong. So unfortunately, you can't just fit a distribution and then use the estimated parameters in a Kolmogorov-Smirnov-Test to test your sample. I excluded this part from the report.
- We have to keep in mind that our sample data obtained in real world will never follow a specific distribution exactly. So even if our pp-values from the KS-Test would be valid and $>0.05>0.05$, it would just mean that we can't rule out that our data follow this specific distribution.
- Another formulation would be that our sample is compatible with a certain distribution. But the answer to the question "Does my data follow the distribution xy exactly?" is always no.
- The goal here cannot be to determine with certainty what distribution your sample follows. The goal is to find *parsimonious approximate descriptions* of the data. Having a specific parametric distribution can be useful as a model of the data.
- This is because, It is impossible to have enough power to rule out a mixture: when the mixture is of two almost identical distributions it cannot be detected and when all but one component have very small proportions it cannot be detected, either. Typically (in the absence of a theory which might suggest a distributional form), one fits parametric distributions in order to achieve parsimonious approximate descriptions of data. Mixtures are none of those: they require too many parameters and are too flexible for the purpose.
- Well in this project, I decided to run regression based on tweedie distribution with default values of parameters to define distribution and estimate rmse.

7.8.1.2 eval_metric

Anotheer important parameter is evaluation of performance of the model. i.e. **eval_metric** [default according to objective] - evaluation metrics for validation data, a default metric will be assigned according to objective (rmse for regression, and error for classification, mean average precision for ranking) - User can add multiple evaluation metrics, for python user, remember to pass the metrics in as list of parameters pairs instead of map, so that latter 'eval_metric' won't override previous one - The choices are listed below: - "rmse": root mean square error - "mae": mean absolute error - "logloss": negative log-likelihood - "error": Binary classification error rate. It is calculated as #(wrong cases)/#(all cases). For the predictions, the evaluation will regard the instances with prediction value larger than 0.5 as positive instances, and the others as negative instances. - "merror": Multiclass classification error rate. It is calculated as #(wrong cases)/#(all cases). - "mlogloss": Multiclass logloss - "auc": Area under the curve for ranking evaluation. - "map":Mean average precision - "tweedie-nloglik": negative log-likelihood for Tweedie regression (at a specified value of the tweedie_variance_power parameter)

7.8.2 Optimized xgboost parameters

Among all these possible *objectives* and *eval_metric*, I started with the simplest *reg:linear* regression and *rmse*. Among all the possible choices for regression problem, objective **reg:treedie** was something new to me and caught my eye because of it's description. I learned about it and used it. Accuracy wise it did not perform that much well compared to simple linear reg (results for simple linear regression xgboost model were discussed in the project mid report.).

The detailed information for applying this model is referred from link_1 and link_2.

```
# https://www.rdocumentation.org/packages/xgboost/versions/0.6-4/topics/xgb.train
# http://xgboost.readthedocs.io/en/latest/parameter.html
# https://cran.r-project.org/web/packages/xgboost/vignettes/xgboostPresentation.html
# https://github.com/dmlc/xgboost/blob/master/doc/parameter.md

xgb_params <- list(seed = 1,
                   #for reproducablity
                   colsample_bytree = 0.7,
                   #variables per tree
                   eval_metric = "rmse",
                   #rmse evaluation
                   objective = "reg:tweedie",
                   #specify the learning task and the corresponding learning objective
                   tweedie_variance_power = 1.4,
                   #tweedie variance power
                   subsample = 0.7,
                   #data subset per tree
                   booster = "gbtree",
                   #gbtree booster
                   max_depth = 5,
                   #tree levels
                   eta = 0.3
                   #shrinkage
                   )

#Dense Matrix: R's dense matrix - xgb.DMatrix
data_train <- xgb.DMatrix(as.matrix(train %>% dplyr::select(-trip_duration)),label = train$trip_duration
#training dataframe except trip_duration
data_valid <- xgb.DMatrix(as.matrix(valid %>% dplyr::select(-trip_duration)),label = valid$trip_duration
#validation dataframe except trip_duration
data_test <- xgb.DMatrix(as.matrix(test))
#test dataframe as Dense matrix

#deciding what information should be printed
info_list <- list(train=data_train, valid=data_valid)
```

7.9 Building XGB Model

Using ll the above parameters, lets pass the data into xgb model and run it. We can observe the RMSE values.

```
set.seed(1)

grad_boost <- xgb.train(params = xgb_params, #pass the parameters list
                   data = data_train, #pass the data
                   nrounds = 500, #Number of iterations #500
                   print_every_n = 100, #print iteration for every 100th iteration.
                   watchlist = info_list) #show the information as requested.
```

```
## [1]  train-rmse:4.074299 valid-rmse:4.073619
## [101]    train-rmse:0.437994 valid-rmse:0.439328
## [201]    train-rmse:0.431678 valid-rmse:0.432859
## [301]    train-rmse:0.427062 valid-rmse:0.428558
```

```
## [401]    train-rmse:0.423026 valid-rmse:0.424454
## [500]    train-rmse:0.419711 valid-rmse:0.421060
```

Over the iterations, root mean square error for both training dataset and validation dataset seems to be decreasing. At 500th iteration, the train rmse is 0.405930 and test rmse is 0.406729.

Let's plot the evaluation log for thebuilt model. We can observe the rmse value drops as the model iterates.
~

```
# the results of each iteration in grad_boost model are stored in the return
# attribute named evaluation_log extract these values and store them in different
# lists
iter <- grad_boost$evaluation_log[, 1]  #nth iteration
rmse_train <- grad_boost$evaluation_log[, 2]  #rmse for train data
rmse_valid <- grad_boost$evaluation_log[, 3]  #rmse for validation data

# form dataframe of these three lists
plot_df <- data.frame(iter, rmse_train, rmse_valid)
# select every 20th row for reporting using seq function
plot_df <- plot_df[seq(1, nrow(plot_df), 15), ]

# plot using plot function
plot(x = plot_df[, 1], y = plot_df[, 2], type = "o", xlab = "iterations", ylab = "rmse",
    col = "blue")
lines(plot_df[, 3], col = "red", lty = 6)
legend(140, 4, legend = c("train_rmse", "valid_rmse"), col = c("blue", "red"), lty = c(1,
    1), cex = 0.8, title = "rmse type", text.font = 4, bg = "lightblue")
```

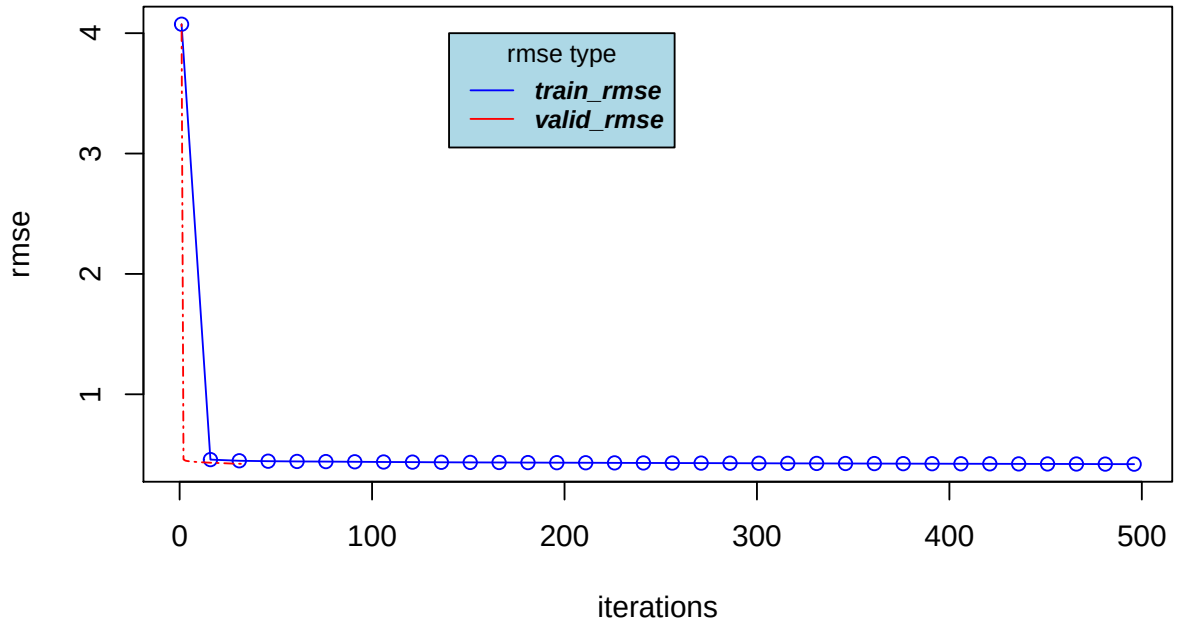


Figure 24: rmse iteration wise for xgboost model

7.10 Cross validation

Aftertraining our model with above models, let's use cross validation to improve model's performance. Interesting parameter to learn here is *early_stopping_rounds*. Which defines the criteria to stop iterations if the past n iterations fail to improve model's performance.

```
# xgb.cv is an inbuilt function for k fold cross validation.  parameters are set
# as further.
xgb_cv <- xgb.cv(xgb_params, data_train, early_stopping_rounds = 8, nfold = 4, nrounds = 500,
    print_every_n = 100)
```

```
## [1]  train-rmse:4.070875+0.001558    test-rmse:4.070880+0.003310
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 8 rounds.
##
## [101]    train-rmse:0.436625+0.000929    test-rmse:0.444643+0.001409
## Stopping. Best iteration:
## [132]    train-rmse:0.433924+0.000723    test-rmse:0.443918+0.001660
```

Note: For running this script fast enough, the iterations and folds are reduced.

We decided to see the train rmse and test rmse for every 100th iteration. We can see the rmse values decreasing over the period.

With k fold cross validaion, after 224th iteration the model was stopped due to early_stopping_rounds override. That is, after these many iterations, model failed to improve upon its accuracy for consecutive 8 times.

The best iteration number is 132. The results are,

-

Interesting observation was if the *nfolds* were increasaed, *early_stopping_rounds* would coclude the cross validation in around 100 iterations. If nfolds were kept as they are, the cross validation would run up to late 400 iterations before *early_stopping_rounds* would terminate the algorithm.

7.11 Feature Importance.

Recall the third assignment that we can see the importance of a particular feature in the entire model. Let's use that code and visualize the importance.

```
# Calculate the importance of features using xgb.importance(). selecting colnames
# except trip_duration.
importance <- as.tibble(xgb.importance(feature_names = colnames(train %>% dplyr::select(-trip_duration))
    model = grad_boost))

# report the values.
pander(importance, caption = "Feature importance")
```

Table 23: Feature importance

Feature	Gain	Cover	Frequency
dist	0.7357	0.5047	0.2208
pickup_longitude	0.08551	0.1579	0.2019
hour	0.05307	0.05427	0.1083
pickup_latitude	0.04831	0.145	0.1856
wday	0.0203	0.02539	0.05115

Feature	Gain	Cover	Frequency
lg_trip	0.01785	0.007563	0.005424
vendor_id	0.009528	0.002868	0.02366
avg_temp	0.008997	0.04422	0.08279
total_precip	0.005177	0.02259	0.04437
month	0.004875	0.009125	0.02305
passenger_count	0.004446	0.01524	0.03111
snow_depth	0.002902	0.00878	0.01296
jfk_trip	0.002888	0.001542	0.004821
store_and_fwd_flag	0.0004585	0.0008757	0.004068

Referring to link, xgb.importance() function summarizes three values for each predictor.

- **Gain** - Gain is the improvement in accuracy brought by a feature to the branches it is on. The idea is that before adding a new split on a feature X to the branch there was some wrongly classified elements, after adding the split on this feature, there are two new branches, and each of these branch is more accurate (one branch saying if your observation is on this branch then it should be classified as 1, and the other branch saying the exact opposite).
- **cover** - Cover measures the relative quantity of observations concerned by a feature.
- **Frequency** - Frequency is a simpler way to measure the Gain. It just counts the number of times a feature is used in all generated trees.

Takeaway is, the features have all these 3 metrics value in same order.

Let's plot these results in a visually representable way.

```
#
importance %>% ggplot(aes(reorder(Feature, Gain, FUN = max), Gain, fill = Feature)) +
    # plot reordered Feature against their gain values
geom_col() + # bar chart
labs(x = "Features", y = "Importance", size = 0.6) + # X axis and y axis labels
theme(axis.title.x = element_text(), axis.text.x = element_text(angle = 90, vjust = 0.5,
    size = 8)) + # for rotating x axis labels 90 degrees
theme(legend.position = "none")
```

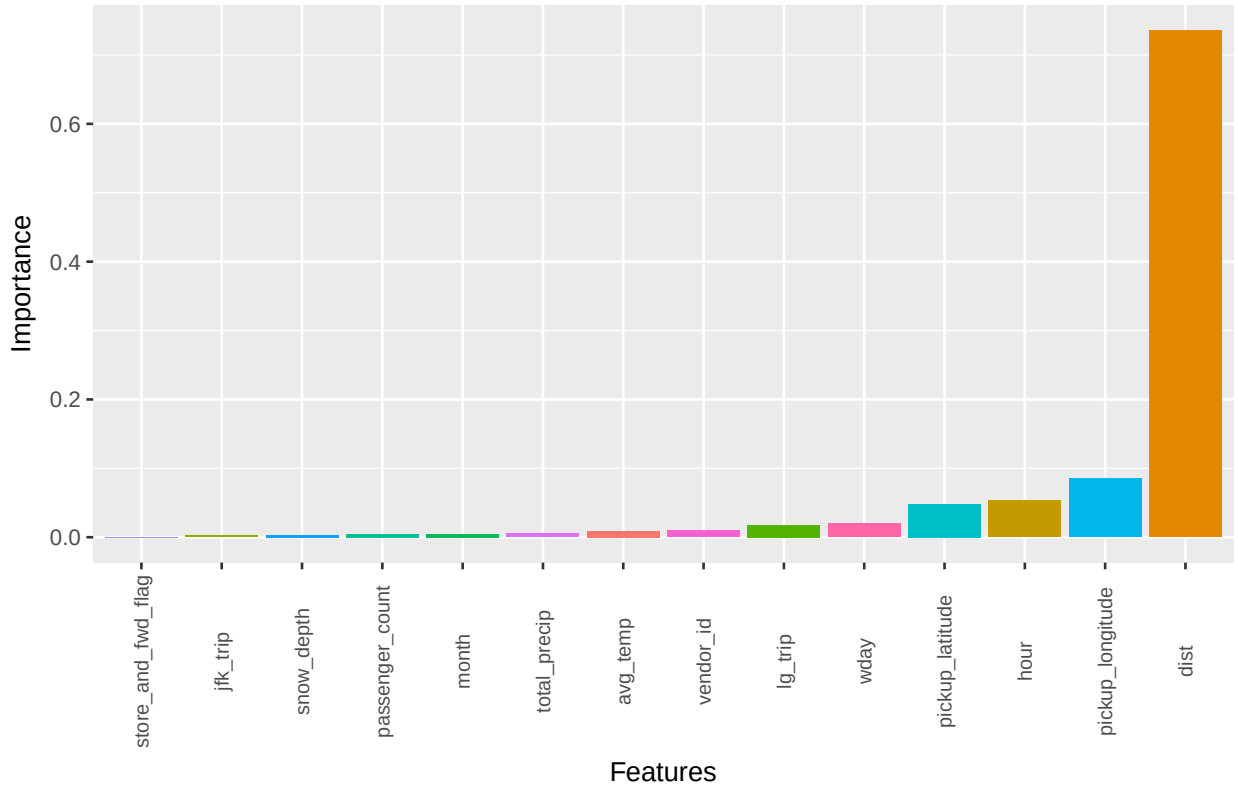


Figure 25: Feature importance

```
# legend not required
```

This graph sums up the feature importance. The information gain for each of the predictors is plotted. *dist* is the by far most important feature which we derived using feature engineering. pickup coordinates are among the most important features. Another feature which we derived *lg_dist_drop* is third important features. We learned that thinking out side the boxes helps in a data science problem.

7.12 Passing test data

Let's pass the testdata into built model and see the test accuracy.

```
# fit the testing data set into the model
test_prediction <- predict(grad_boost, data_test)


prediction <- test_id %>% # create the dataframe
mutate(trip_duration = exp(test_prediction) - 1)
# removing the 1 from log transformed value we added earlier

# If needed write the csv file for results pred %>% write_csv('submit.csv')
```

7.13 Visualizing results

```r
# Actual values from training data, log transformed
temp <- train %>% # in train
dplyr::select(trip_duration) %>% # select trip duration
mutate(dset = "train", trip_duration = exp(trip_duration) - 1)
# use exp (exponential) function for lof transformation and remove 1 we added
# earlier.

# predicted values for test data
bar <- prediction %>% mutate(dset = "predicted")
# use the dataframe prediction, the values for dset for these datapoints is
# 'predicted'

# http://ggplot2.tidyverse.org/reference/geom_density.html
bind_rows(temp, bar) %>% # similar to cbind. bind two dataframes side by side.
ggplot(aes(trip_duration, fill = dset)) + # plot trip duration and fill according to the datapoints sou
geom_density(alpha = 0.5) + # show density distribution
scale_x_log10()
```

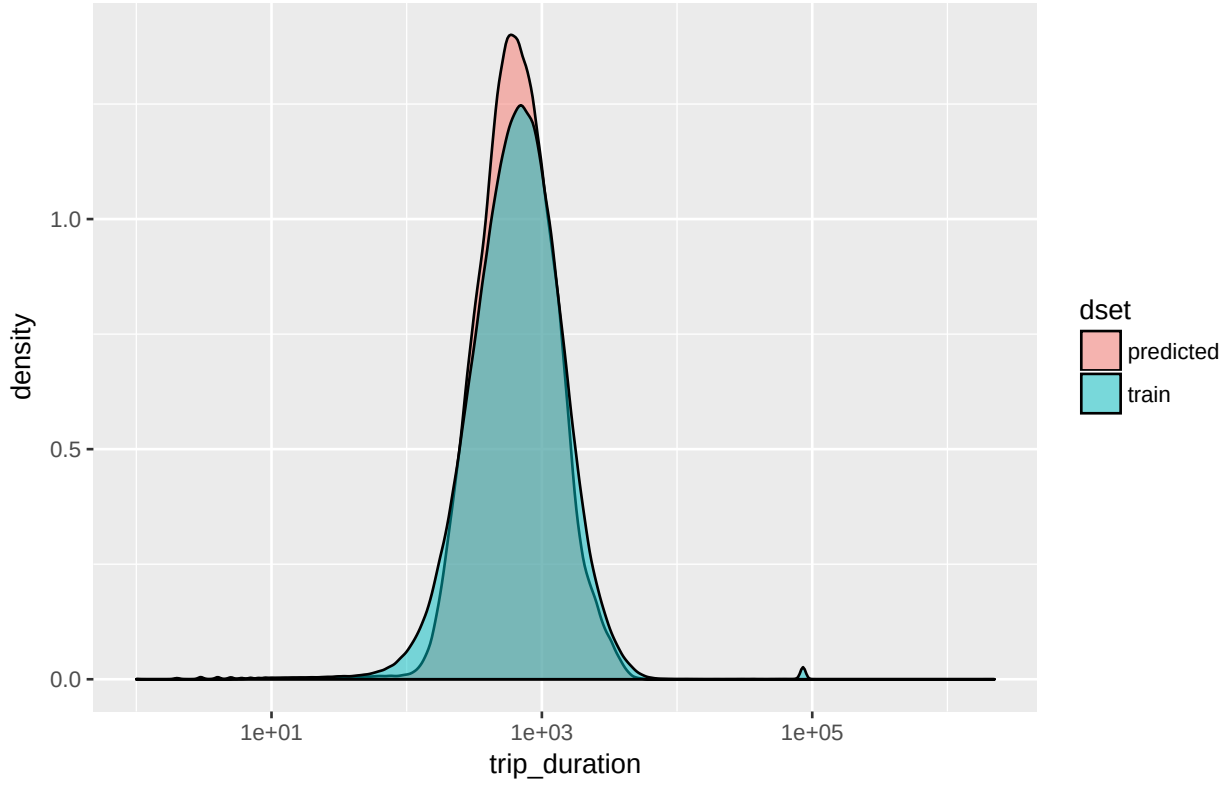


Figure 26: distribution of prediction and actual values of trip duration

```r
# log transform x axis to cover the extreme range of datapoints with more
# details.
```

geom_density() computes and draws kernel density estimate, which is a smoothed version of the histogram. This is a useful alternative to the histogram if for continuous data that comes from an underlying smooth distribution.

From results we can conclude that, density distribution of actual *trip_duration* in training dataset closely

resembles with the kernel density estimate for testing data. The prediction has excess Kurtosis than training distribution. There will always be more scope to improve.

Overall, with this resemblance, I coclude the project at this stage.

8 Future work

8.1 Optimizing Parameters of mixed distribution

We can attempt to define the distribution more accurately. We saw that qq plot shows misfit of the distribution at the tail regions.

8.2 Feature Engineering

There's always a scope for including different datasources that can affect on trip_duration. The inspiration for the majority of this report was this fascinating and intensive script. In this script, there's also another feature engineered which deals with actual new york city map for the fastest and shortest path. It takes into account OSRM (**O**pen **S**ource **R**outinig **M**achine using C++) project as well.

Also another possibile feature that is used in the original report is bearingwhich is built upon the idea of direction of travelling. It considers busy parts of New York City.

8.3 Different predictive models.

This report is primarily focused on exploratory data analysis and details of defining parameters for xgboosting. We can try variety of different statisticacl models as random Forest, Ridge regression, Lasso regression, SVM etc. as we have already implemented in other class assignments successfully. This is in future scope.

8.4 More on Ambitious Project

The Ambitiousproject includes studying the maths discussed in the paper and integrating it in code. At this moment it appears very difficult to me. I intend to work on it during winter break.

9 Presentation- Learnings

- rmarkdown proficiency exmples
  - fig_pos.tex
  - table of contents
  - chunk detailing
  - figure / table controls
  - in line math code
  - hyper links
  - exploring Latex engines
  - .pdf and .html examples
- Intensive exploratory data anlysis
- Feature engineering
  - dist cosine and average speed
  - week day (wday)
  - airport trip
  - weather (temperature, rain, snow)
- Identifying distribution
  - Confidence in attempting to fit a distribution
  - to interprete "*Cullen and Frey grapgh*" and *qq plot*.
  - Tweedie and weibull distribution parameters using CV.
- xgboost model and cross validation
- Results and conclusion
- Future work

10 Appendix: First Report

Expectations:

From this course project, I intend to get an opportunity to apply statistical prediction and classification models on the real-world problems which I am learning in the classroom. I was looking for a problem statement which will have a definite real-world problem statement supplemented by a dataset, sufficiently large enough to make some assumption with good confidence level and consisting of diverse variables. This will open-up Pandora's box of exploratory data analysis.

Problem Statement and dataset: I found a competition on Kaggle which brings the challenge to build a model that predicts the total ride duration of taxi trips in New York City. The primary dataset is one released by the NYC Taxi and Limousine Commission, which includes pickup time, geo-coordinates, number of passengers, and several other variables. Before I explain the reason why I chose this problem statement, let's have a look at the variables in the dataset.

- *id* - a unique identifier for each trip
- *vendor_id* - a code indicating the provider associated with the trip record
- *pickup_datetime* - date and time when the meter was engaged
- *dropoff_datetime* - date and time when the meter was disengaged
- *passenger_count* - the number of passengers in the vehicle (driver entered value)
- *pickup_longitude* - the longitude where the meter was engaged
- *pickup_latitude* - the latitude where the meter was engaged
- *dropoff_longitude* - the longitude where the meter was disengaged
- *dropoff_latitude* - the latitude where the meter was disengaged
- *store_and_fwd_flag* - This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip.
- *trip_duration* - duration of the trip in seconds

There are 1,458,644 observations in the training dataset while 625 in testing dataset.

Why I chose this problem statement: In the first look at the variables, I could think of many ideas to explore this dataset. To use the variables to derive useful information. For example, if the vendors affect the taxi service quality and consequently the trip duration, how pickup times, daily schedules and office hours affect the durations, if the more number of people in a trip tend to go some entertainment place far away from city etc. The added-bonus that comes with pick-up and drop-off location coordinates make me think of n number of exploratory hypothesis. For example, usually, pick up locations in the crowded part of the city could have longer trip duration due to inability to travel quickly to the nearby places due to the traffic. Another one could be, if the park places, or entertainment centers must have more number of trips with the number of customers as more than 1 etc.

Another aspect of choosing this problem statement was that I can think of applying a combination of regression analysis and classification techniques. With classification or clustering analysis for few variables (like pickup location) I can try to build the regression models for trip durations. This prediction model for classified trips (customers) can be compared over one big inclusive prediction model for accuracy.

With all these preliminary ideas and scope for exploration, this problem statement for the project sounded perfect to me.

Proposed statistical models: For prediction of trip durations, at this stage I can think of implementing linear and non-linear regression fits. This will control the model complexity. Also, soft subset selection techniques like ridge and lasso can also be implemented to weight the important features more. If it deemed possible, I will try clustering the pickup locations or number of passengers in a trip or time of the day to classify the available datapoints and build the subsets of predictions models for these classes. Ensemble techniques like random forest are also a good opportunity in this project.

Ambitious Project: Few months earlier, I came across the paper for real time trend detection using

segmental linear regression. The paper contains all the mathematics part of the idea. At that time, I found the idea quite interesting however, hard to implement in any programming language. I would like to take this course project opportunity to give it a try. Overall idea of the paper can be explained with the help of the further figure.

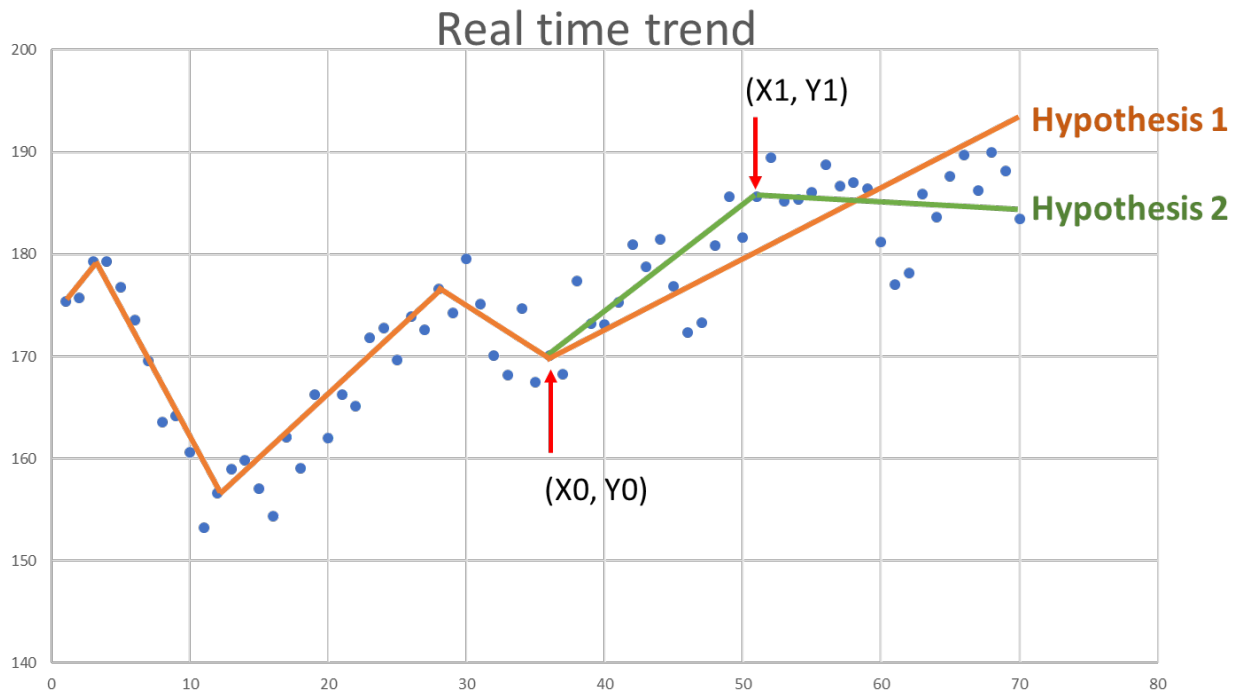


Figure 27: Representation of real time trend detection using segmental linear regression.

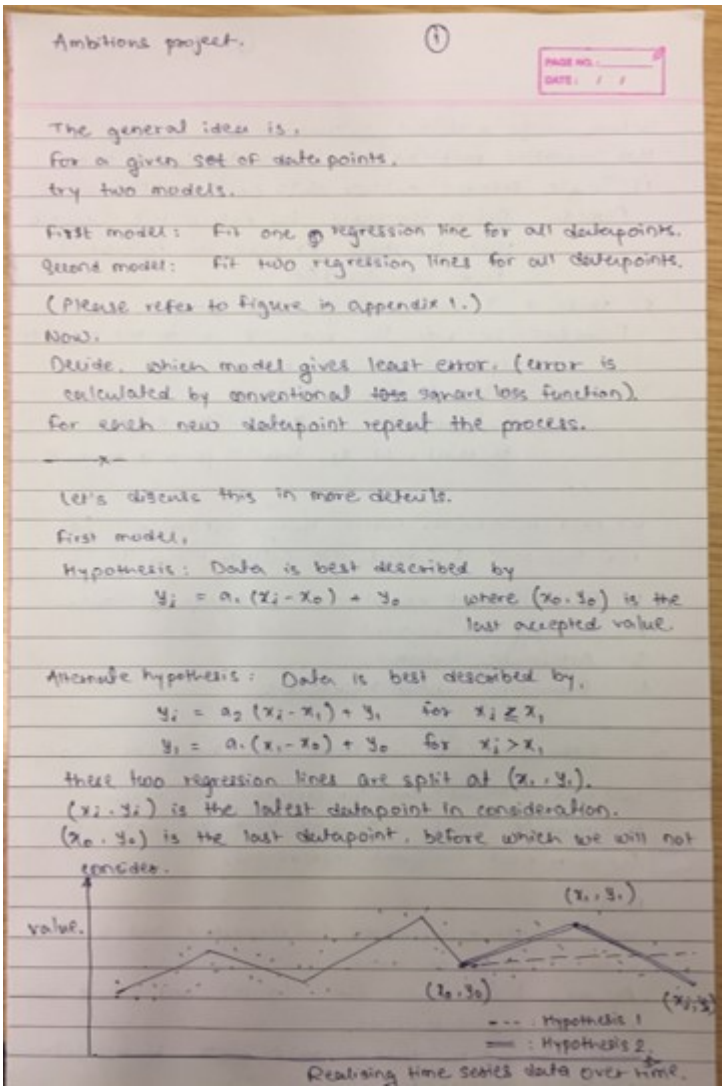
The general idea is, for every new data point realized, two regression analysis are carried out to represent the data in best possible way. Hypothesis one is, only one regression fit is performed for all available datapoints from the last accepted datapoint (X0, Y0). Hypothesis two is, two regression fits are performed for all the available datapoints - one from (X_0, Y_0) to (X_1, Y_1) and second from (X_1, Y_1) to the latest datapoint.

$Y_i = a_1(x_i - x_0) + y_0$ (Hypothesis 1) Or (Hypothesis 2, two parts) $Y_i = a_2(x_i - x_1) + y_1$ for $x_i >= x_1$, $Y_1 = a_1(x_i - x_0) + y_0$ for $x_i < x_1$,

Errors are calculated for both regression approaches using loss function. The approach with the least error is selected. And the (X_1, Y_1) is now new (X_0, Y_0).

This entire approach can be used as an alternative for conventional time series analysis (ARIMA / ARMA). This approach can also be used as a predictive model for future datapoints by extrapolating the end of the regression fit. However, at this stage, I am not sure that extrapolation can detect the change in the trend. Perhaps after few weeks, once the mathematics is understood, this question can be answered. The paper also claims that, shifts in the incoming new datapoints can be accommodated in the mathematical datapoints by including a new parameter for it.

11 Appendix 2: Ambitious Project



Ambitions project. ①

The general idea is,
for a given set of datapoints,
try two models.

First model: Fit one regression line for all datapoints.
Second model: Fit two regression lines for all datapoints.

(Please refer to figure in appendix 1.)

Now,
Decide, which model gives least error. (error is calculated by conventional ~~loss~~ square loss function)
for each new datapoint repeat the process.

---x---

Let's discuss this in more details.

First model,

Hypothesis: Data is best described by

$y_i = a_1 (x_i - x_0) + y_0$ where (x_0, y_0) is the last accepted value.

Alternate hypothesis: Data is best described by,

$y_i = a_2 (x_i - x_1) + y_1$ for $x_i \leq x_1$

$y_1 = a_1 (x_1 - x_0) + y_0$ for $x_i > x_1$

these two regression lines are split at (x_1, y_1).
(x_i, y_i) is the latest datapoint in consideration.
(x_0, y_0) is the last datapoint, before which we will not consider.

value.

(x_1, y_1)

(x_0, y_0)

(x_i, y_i)

--- : Hypothesis 1
=== : Hypothesis 2

Realising time series data over time.

Before we go in detail calculations,
two potential problems that can be faced are,

1) Single extreme outlier data point in time series.
Remedy: Set a threshold for extremities. Beyond that threshold, exclude the datapoint from consideration.

2) Shift in Y values after some point.
Remedy: Include the variable 'b' in the equation representing shift magnitude.
With this,
hypothesis 1: $y_i = a_1(x_i - x_0) + y_0 + b$
So that, at x_1, there is jump of b_2 in y_2.

—x—

We need to consider further things dynamically (as new datapoint is added into system every time).

1. Derivation of equations for change of slope.
2. Starting the segment.
3. Alternate hypothesis
4. Reasoning about hypotheses.

—x—

1. Derivation of equations for change of slope.

We need a mechanism, for preserving as much intermediate states as possible to make computations for each new datapoint incremental.

First,
we will derive equations for connected line segments with a change of slope.
Assume data starts from (0, 0), and break at x_1 with no datapoints.
And there are n_1 datapoints from x_1 to the latest realized datapoint.

To maintain these assumptions, as we move from segment to segment, we will translate axes appropriately.

As starting point is $(0,0)$. under this assumption, line segment becomes.

$$\begin{cases} y = a_1 x_i & \text{if } x_i \le x_1 \\ \text{and} \quad y = a_2 (x_i - x_1) + a_1 x_1 & \text{if otherwise.} \end{cases}$$

The problem is to determine slopes a_1 and a_2.

We have objective function as to minimize square loss. i.e

$$\frac{\partial}{\partial a_1} \sum_{x_i} (y_i - \hat{y})^2 = 0 \qquad \frac{\partial}{\partial a_2} \sum_{x_i} (y_i - \hat{y})^2 = 0.$$

where $\hat{y}$ is linear estimate of values.

← Substituting value of $\hat{y}$ in above two equations

2.1) $$\frac{\partial}{\partial a_1} \left(\sum_{x_i \le x_1} (y_i - a_1 x_i)^2 + \sum_{x_i > x_1} (y_i - a_2 x_i + a_2 x_1 - a_1 x_1)^2 \right) = 0$$

Similarly, w.r.t. a_2.

2.2) $$\frac{\partial}{\partial a_2} \left(\sum_{x_i \le x_1} (y_i - a_1 x_i)^2 + \sum_{x_i > x_1} (y_i - a_2 x_i + a_2 x_1 - a_1 x_1)^2 \right) = 0$$

Solving 2.1,

$$-2 \sum_{x_i \le x_1} ((y_i - a_1 x_i) x_i) - 2 x_1 \sum_{x_i > x_1} (y_i + a_2 x_1 - a_1 x_1 - a_2 x_i) = 0$$

$$\sum_{x_i \le x_1} (y_i x_i - a_1 x_i^2) + x_1 \sum_{x_i > x_1} (y_i - a_2 x_i) + n_1 a_2 x_1^2 - n_1 a_1 x_1^2 = 0$$

consider substitutes,

$$S_{xy_0} = \sum_{x_i \le x_1} y_i x_i \qquad S_{xx_0} = \sum_{x_i \le x_1} x_i^2 \qquad S_{yy_0} = \sum_{x_i \le x_1} y_i^2$$

$$S_{x_0} = \sum_{x_i \le x_1} x_i \qquad S_{y_0} = \sum_{x_i \le x_1} y_i \qquad S_{xy_1} = \sum_{x_i > x_1} y_i x_i$$

$$S_{xx_1} = \sum_{x_i > x_1} x_i^2 \qquad S_{yy_1} = \sum_{x_i > x_1} y_i^2 \qquad S_{x_1} = \sum_{x_i > x_1} x_i$$

$S_{y_1} = \sum_{x_i > x_1} y_i$. (4)

Then (3) becomes,

$S_{xy_0} - a_1 S_{xx_0} + x_1 S_{y_1} - a_2 x_1 S_{x_1} + n_1 a_2 x_1^2 - n_1 a_1 x_1^2 = 0$

Arranging these terms for a_1 and a_2,

2.3) $a_1 (S_{xx_0} + n_1 x_1^2) + a_2 x_1 (S_{x_1} - n_1 x_1) = S_{xy_0} + x_1 S_{y_1}$

Now, solving 2.2,

$-2 \sum_{x_i > x_1} ((y_i - a_2 x_i + a_2 x_1 - a_1 x_1)(x_i - x_1)) = 0$

$\sum_{x_i > x_1} (x_i y_i - a_2 x_1^2 + 2 a_2 x_1 x_i - a_1 x_1 x_i - a_2 x_1^2 + a_1 x_1^2) = 0.$

Substituting replacement terms,

2.4) $a_1 x_1 (S_{x_1} - n_1 x_1) + a_2 (S_{xx_1} - x_1 (2 S_{x_1} - n_1 x_1)) = S_{xy_1} - x_1 S_{y_1}$

eqn (2.3) & (2.4) are simultaneous equations for 2 unknowns a_1 & a_2.

In the system of simultaneous equations, of

$a_1 b_1 + a_2 c_1 = d_1$ - eqn (1)

$a_1 b_2 + a_2 c_2 = d_2$ - eqn (2)

then

$$a_1 = \frac{c_2 d_1 - c_1 d_2}{b_1 c_2 - b_2 c_1}$$

& $$a_2 = \frac{b_1 d_2 - b_2 d_1}{b_1 c_2 - b_2 c_1}$$

Using this knowledge, the denominator $(b_1 c_2 - b_2 c_1)$

$(S_{xx_0} + n_1 x_1^2)(S_{xx_1} - 2 x_1 S_{x_1} + n_1 x_1^2) - \{x_1^2 (S_{x_1} - n_1 x_1)^2$.

Numerator of a_1,

$(S_{xx_1} - 2 x_1 S_{x_1} + n_1 x_1^2)(S_{xy_0} + x_1 S_{y_1}) - x_1 (S_{x_1} - n_1 x_1)(S_{xy_1} - x_1 S_{y_1})$

numerator of a_2 :

$$(S_{xx_0} + n_1 x_1^2)(S_{xy_1} - x_1 S_{y_1}) - (x_1(S_{x_1} - n_1 x_1))(S_{xy_0} + x_1 S_{y_1})$$

Now, thinking for each incoming new datapoint,

$$\sum_{x_i > x_1} y_i x_i\,(i) = \sum_{x_i > x_1} y_i x_i\,(i-1) + x(i)y(i)$$

for each new i.

i.e. $S_{xy_1}(i) = S_{xy_1}(i-1) + x(i)y(i)$ and so forth.

The basis for slope actually changing is unexplained variation in the y values.

i.e. $\sum_{x_i} (y_i - \hat{y})^2$.

Substituting $\hat{y}$, we already have,

2.5) $$\sum_{x_i \le x_1} (y_i - a_1 x_i)^2 + \sum_{x_i > x_1} (y_i - a_2 x_i + a_2 x_i - a_1 x_i)^2$$

multiplying & substituting $q = x_1(a_2 - a_1)$,

$$S_{y y_0} - 2a_1 S_{x y_0} + a_1^2 S_{xx_0} + S_{yy_1} - 2a_2 S_{xy_1} + 2qy_1 + a_2^2 S_{xx_1} - 2qa_2 S_{x_1} + n_1 q^2$$

I did not understand the further part in the paper. I am working on it. I will present further learning in the next report.