TAXI TAXI - A beginner approach on Exploratory Analysis

Suresh
Date- 2018-01-23


```
## [1] 56000
```

Introduction

This is an basic Exploratory Data Analysis for the NYC Taxi Ride Duration competition.

New York City taxi rides paint a vibrant picture of life in the city. The millions of rides taken each month can provide insight into traffic patterns, road blockage, or large-scale events that attract many New Yorkers. With ridesharing apps gaining popularity, it is increasingly important for taxi companies to provide visibility to their estimated fare and ride duration, since the competing apps provide these metrics upfront. Predicting fare and duration of a ride can help passengers decide when is the optimal time to start their commute.

The primary goal of this project is to predict trip duration of NYC Taxis based on features like trip coordinates, duration date and time. Training dataset has close to 1.5 Million and 630k records in test dataset. Each row contains one taxi trip.

Load required Packages

- Predominantly we have used dplyr, ggplot2 and lubridate packges

```r
library(dplyr)
library(tidyr)
library(lubridate)
library(ggplot2)
library(ggmap)
library(stringr)
library(scales)
library(gridExtra)
library(corrplot)
library(RColorBrewer)
library(geosphere)
library(tibble)
library(forcats)
library(xgboost)
library(caret)
library(leaflet)
```

```r
library(maps)
library(readr)
```

Load Data

We are using Read CSV function to read test, train and submission file data into R.

```r
taxi <- read.csv("train.csv")
test <- read.csv("test.csv")
submit <- read.csv("sample_submission.csv")
```

File Structure & Integrity

In this section we are checking the summary of Taxi Records and also check if any BLANKS or NA values present.

```r
glimpse(taxi)
```

```
## Observations: 1,458,644
## Variables: 11
## $ id                 <fctr> id2875421, id2377394, id3858529, id3504673...
## $ vendor_id          <int> 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2...
## $ pickup_datetime    <fctr> 2016-03-14 17:24:55, 2016-06-12 00:43:35, ...
## $ dropoff_datetime   <fctr> 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 <fctr> N, N, N, N, N, N, N, N, N, N, N, N, N, N, ...
## $ trip_duration      <int> 455, 663, 2124, 429, 435, 443, 341, 1551, 2...
```

```r
taxi <- data.frame(taxi)
test <- data.frame(test)
submit <- data.frame(submit)
```

We find below observations on Taxi Records

- Trip *Id* is unique identification of a trip
- *vendor_id* field has only 2 values "1" or "2" asuming two taxi companies
- *pickup/dropoff_datetime* - holds date and time of pickup and dropoff. we need to mutate the fields to get date and time seperately.
- *pickup/dropoff_longitute/latitute* - hold values of geographical coordinates where the meter was activate/deactivated.
- *store_and_fwd_flag* is a flag that indicates whether the trip data was sent immediately to the vendor ("N") or held in the memory of the taxi because there was no connection to the server ("Y"). Maybe there could be a correlation with certain geographical areas with bad reception?
- *trip_duration* hold the duration in seconds and its our target prediction of ths project.
- Please note Test data will not have actual trip duration data. We have to submit the data in Kaggle to know the model score.

Lets check the missing values of Taxi and Test Records

```r
sum(is.na(taxi))
```

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

```r
sum(is.na(test))
```

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

We have received Zero count. So we dont have any missing values in Dataset.

In preparation for our eventual modelling analysis we combine the Test and Train(Taxi) records into a single one.

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

Reformating data for our analysis

For our analysis, we will date fields from characters into date objects. We also change vendor_id as a factor. This makes it easier to visualise relationships that involve these features.

```r
taxi <- taxi %>% mutate(
  pickup_datetime = ymd_hms(pickup_datetime),
  dropoff_datetime = ymd_hms(dropoff_datetime),
  vendor_id = factor(vendor_id),
  passenger_count = factor(passenger_count)
)
```

Consistency check

This code is to check *trip_durations* are consistent with the intervals between *pickup_datetime* and *dropoff_datetime*.

Actual count of Taxi file is 1458644. Count of below check should the same else the records are inconsistent.

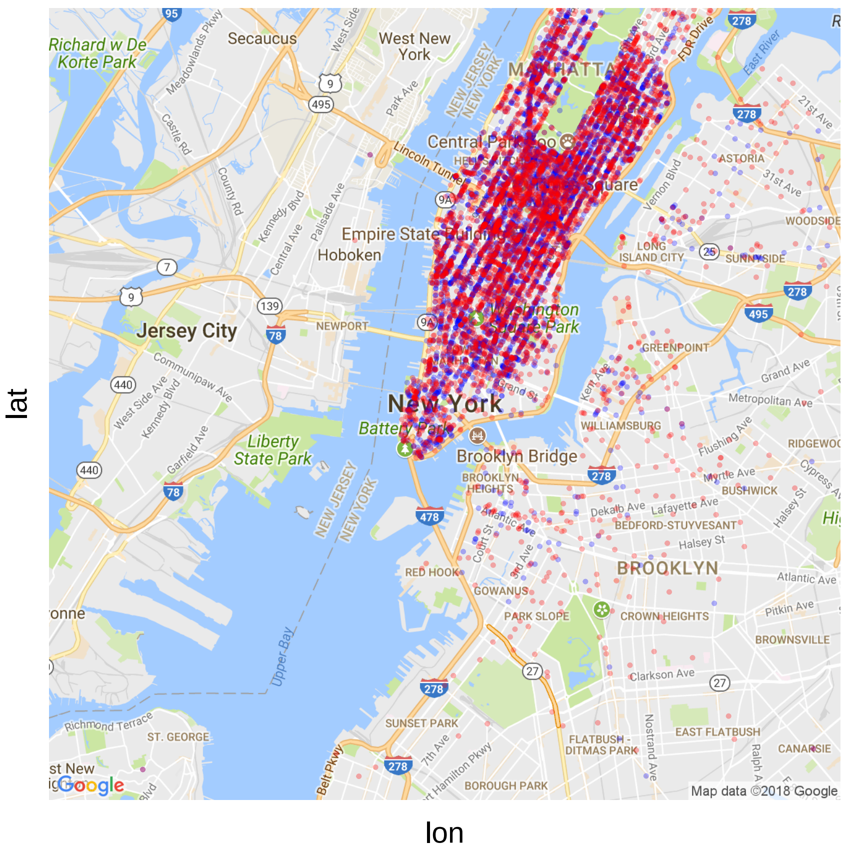
```r
taxi %>%
  mutate(check = abs(int_length(interval(dropoff_datetime, pickup_datetime)) + trip_duration) > 0) %>%
  select(check, pickup_datetime, dropoff_datetime, trip_duration) %>%
  group_by(check) %>%
  count()
```

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

Feature Visualizations

We are starting with NYC Map to check where the most of pickup and dropoff are happening. We took 5000 samples from Taxi file and plotted. It turns out that almost all of our trips were in fact taking place in Manhattan only.

```
my_map <- get_map(location = "New York City", zoom = 12, maptype = "roadmap", source = "google", color =
set.seed(1234)
tax_samp <- sample_n(taxi, 5000)
ggmap(my_map) +
  geom_point(data = tax_samp, aes(x = pickup_longitude, y = pickup_latitude), size = 0.3, alpha = 0.3, c
  geom_point(data = tax_samp, aes(x = dropoff_longitude, y = dropoff_latitude), size = 0.3, alpha = 0.3
  theme(axis.ticks = element_blank(), axis.text = element_blank())
```



Below analysis to check if any abnormal trip duration exists in our data.

```
taxi %>%
  arrange(desc(trip_duration)) %>%
  select(trip_duration, pickup_datetime, dropoff_datetime) %>%
  head(5)
```

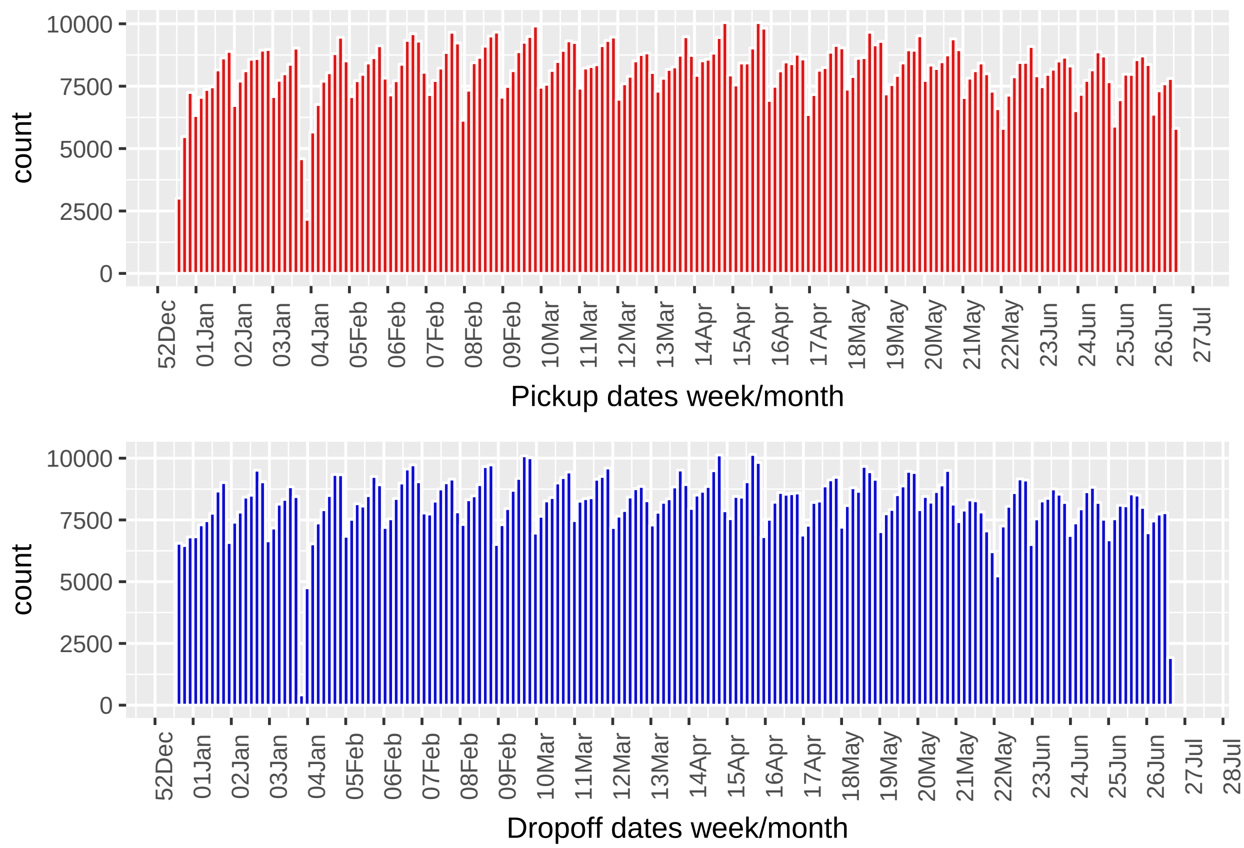
```
##   trip_duration     pickup_datetime    dropoff_datetime
## 1       3526282 2016-02-13 22:46:52 2016-03-25 18:18:14
## 2       2227612 2016-01-05 06:14:15 2016-01-31 01:01:07
## 3       2049578 2016-02-13 22:38:00 2016-03-08 15:57:38
## 4       1939736 2016-01-05 00:19:42 2016-01-27 11:08:38
## 5         86392 2016-02-15 23:18:06 2016-02-16 23:17:58
```

```r
taxi %>%
  arrange(desc(trip_duration)) %>%
  select(trip_duration, pickup_datetime, dropoff_datetime) %>%
  tail(5)
```

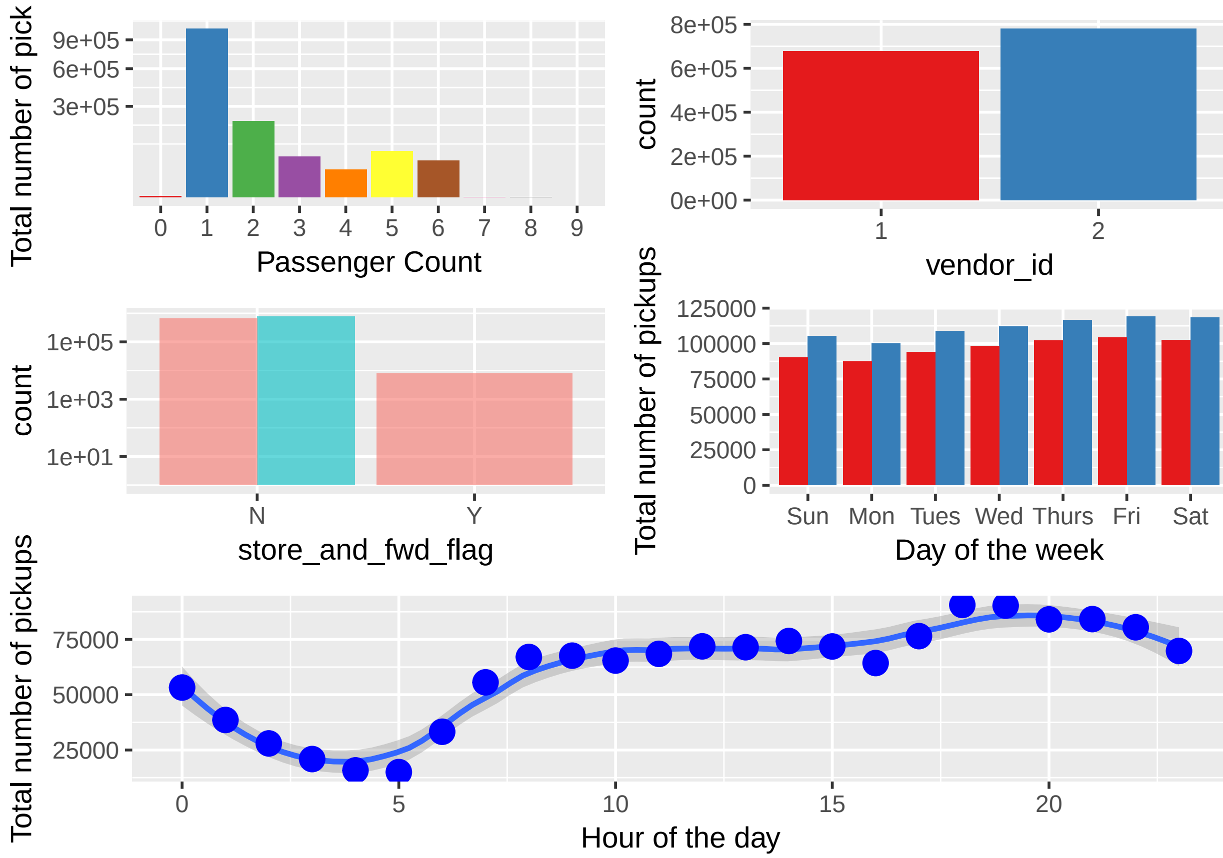
```
##         trip_duration     pickup_datetime    dropoff_datetime
## 1458640             1 2016-02-22 00:40:25 2016-02-22 00:40:26
## 1458641             1 2016-03-12 02:15:31 2016-03-12 02:15:32
## 1458642             1 2016-01-14 12:33:28 2016-01-14 12:33:29
## 1458643             1 2016-02-06 13:40:27 2016-02-06 13:40:28
## 1458644             1 2016-01-03 16:55:44 2016-01-03 16:55:45
```

Lets check the distributions of pickup_datetime and dropoff_datetime by year.

```r
p1 <- taxi %>%
  ggplot(aes(x = pickup_datetime)) +
  geom_histogram(fill = "red", colour = "white", bins = 180) +
  scale_x_datetime(date_breaks = "1 week", date_labels = "%W%b") +
  labs(x = "Pickup dates week/month") +
  theme(axis.text.x = element_text(angle = 90))

p2 <- taxi %>%
  ggplot(aes(dropoff_datetime)) +
  geom_histogram(fill = "blue", colour = "white", bins = 180) +
  scale_x_datetime(date_breaks = "1 week", date_labels = "%W%b") +
  labs(x = "Dropoff dates week/month") +
  theme(axis.text.x = element_text(angle = 90))

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



In the below plot we are checking passenger count, vendor_id, total number of pickups on hour/day distribution.



- We found some abnormal trip with zero passenger and more 7 passengers

- We find an interesting pattern with Monday being the quietest day and Friday very busy.

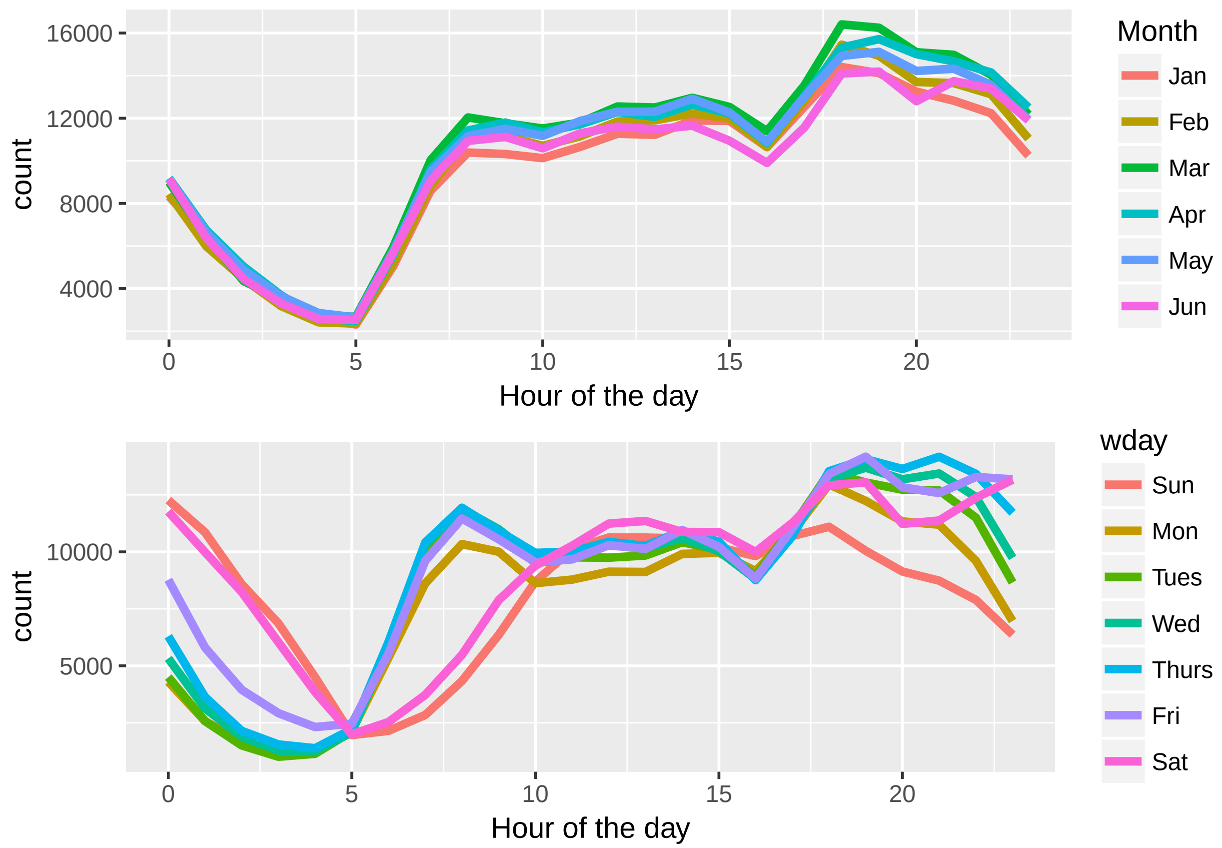
- we find evening hours are busiest hours of the day.

Now lets check how the trends in different vizualization.

- We find Jan and June has less number of trips
- We find During weekends early morning are busy

```
p1 <- taxi %>%
  mutate(
    hpick = hour(pickup_datetime),
    Month = factor(month(pickup_datetime, label = TRUE))
  ) %>%
  group_by(hpick, Month) %>%
  count() %>%
  ggplot(aes(hpick, n, color = Month)) +
  geom_line(size = 1.5) +
  labs(x = "Hour of the day", y = "count")

p2 <- taxi %>%
  mutate(
    hpick = hour(pickup_datetime),
    wday = factor(wday(pickup_datetime, label = TRUE))
  ) %>%
  group_by(hpick, wday) %>%
```

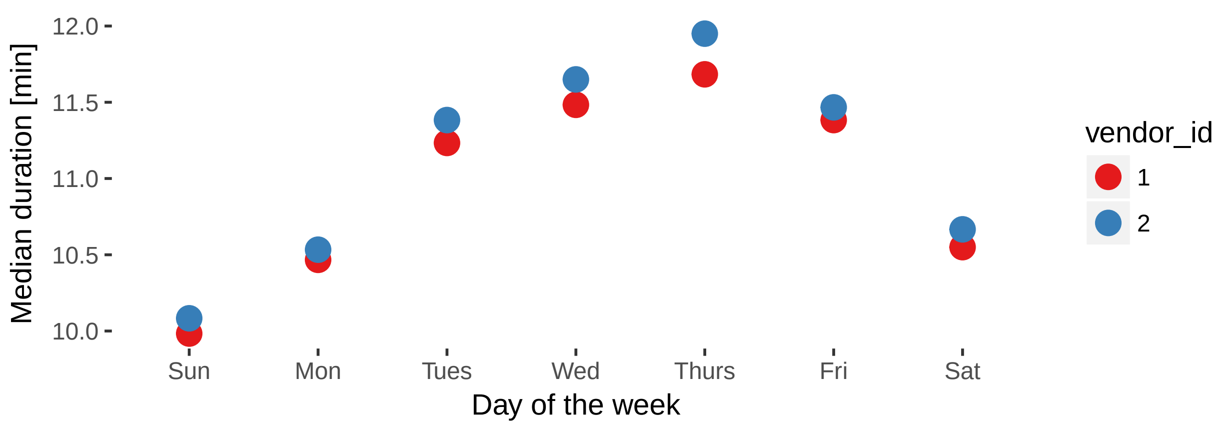
```
  count() %>%
  ggplot(aes(hpick, n, color = wday)) +
  geom_line(size = 1.5) +
  labs(x = "Hour of the day", y = "count")

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

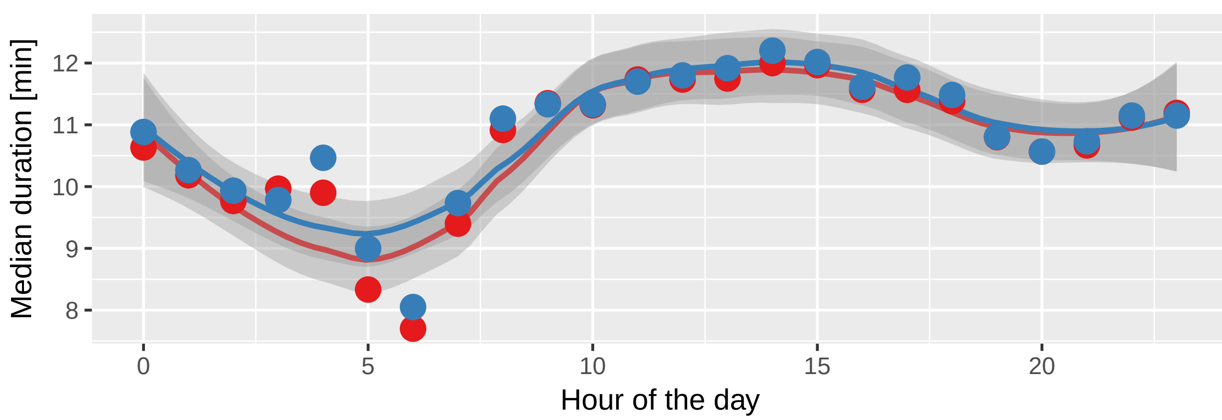


In the next slides we trying find the relation between the trip duration and picking up data & time. This will help us identify how strongly correlated to add them in the model.

```
p1 <- taxi %>%
  mutate(wday = wday(pickup_datetime, label = TRUE)) %>%
  group_by(wday, vendor_id) %>%
  summarise(median_duration = median(trip_duration) / 60) %>%
  ggplot(aes(wday, median_duration, color = vendor_id)) +
  geom_point(size = 4) +
  scale_colour_brewer(palette = "Set1") +
  labs(x = "Day of the week", y = "Median duration [min]") +
  theme(panel.background = element_rect(fill = "white"))

p2 <- taxi %>%
  mutate(pickt = hour(pickup_datetime)) %>%
  group_by(pickt, vendor_id) %>%
  summarise(median_duration = median(trip_duration) / 60) %>%
```

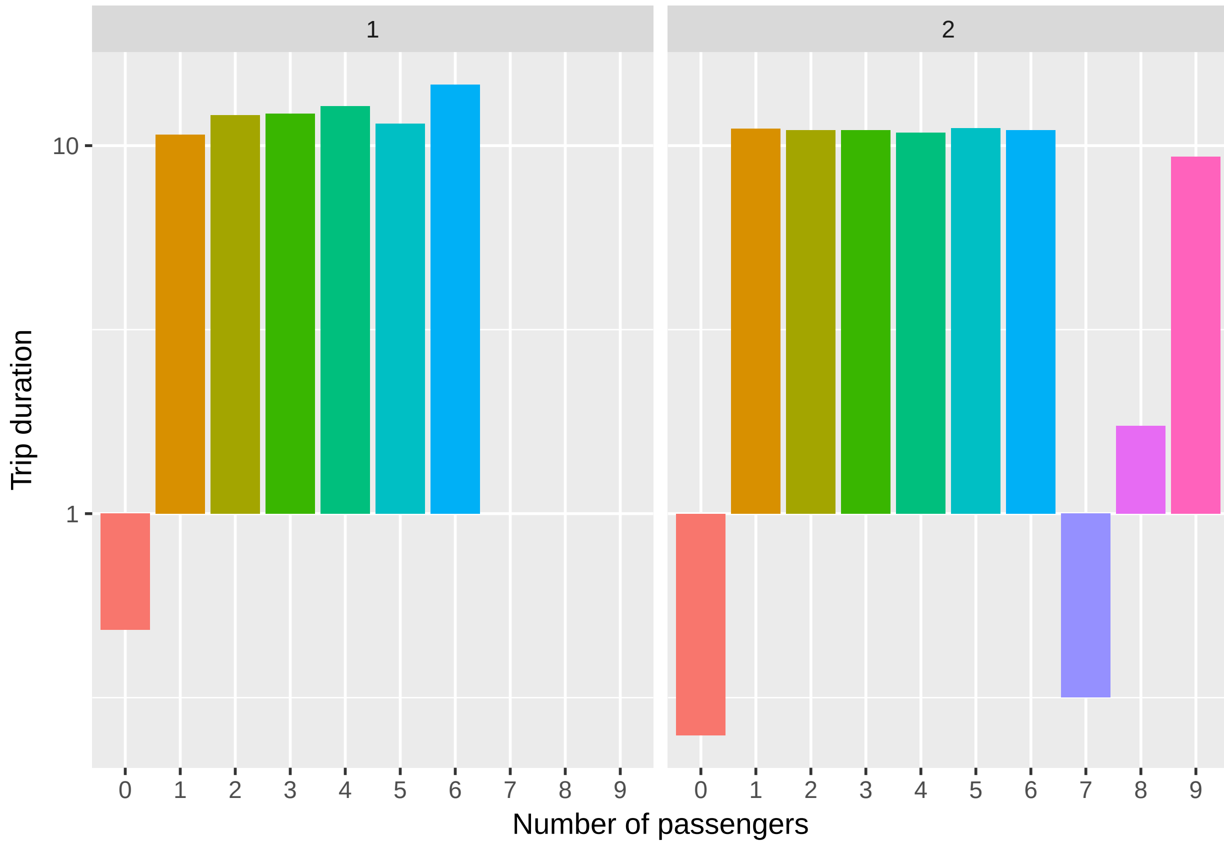
```r
  ggplot(aes(pickt, median_duration, color = vendor_id)) +
  geom_smooth(method = "loess", span = 1 / 2) +
  geom_point(size = 4) +
  scale_colour_brewer(palette = "Set1") +
  labs(x = "Hour of the day", y = "Median duration [min]") +
  theme(legend.position = "none")

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





In the our next slide, we are checking for any correlation between passenger count and trip duration.

```r
taxi %>%
  group_by(passenger_count, vendor_id) %>%
  summarise(median_duration = median(trip_duration) / 60) %>%
  ggplot(aes(passenger_count, median_duration, fill = passenger_count)) +
  geom_bar(stat = "identity") +
  scale_y_log10() +
  theme(legend.position = "none") +
  facet_wrap(~ vendor_id) +
  labs(y = "Trip duration", x = "Number of passengers")
```

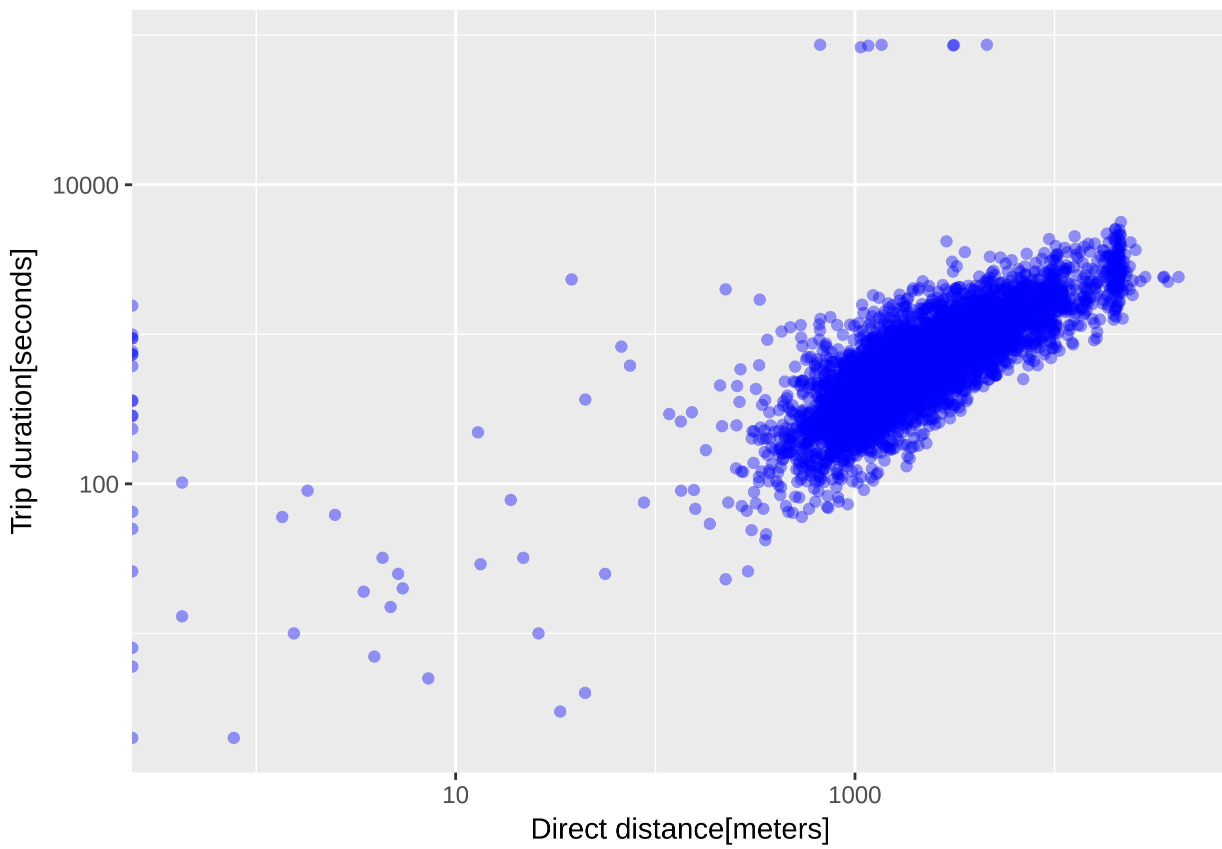


Relation between Time and Direct distance

In this section we are trying to find out the relation between drip duration and direct distance. To derive direct distance, we are using "Geosphere" package. Also, we are trying to find out any significance counts trips made out of Manhattan. Two major airports attract more taxi rides from city. We need to find how significant they are for our modelling.

Lets plot relationship trip duration and distance

```
set.seed(1234)
taxi %>%
  sample_n(5000) %>%
  ggplot(aes(dist, trip_duration)) +
  geom_point(color = "blue", alpha = 0.4) +
  scale_x_log10() +
  scale_y_log10() +
  labs(x = "Direct distance[meters]", y = "Trip duration[seconds]")
```

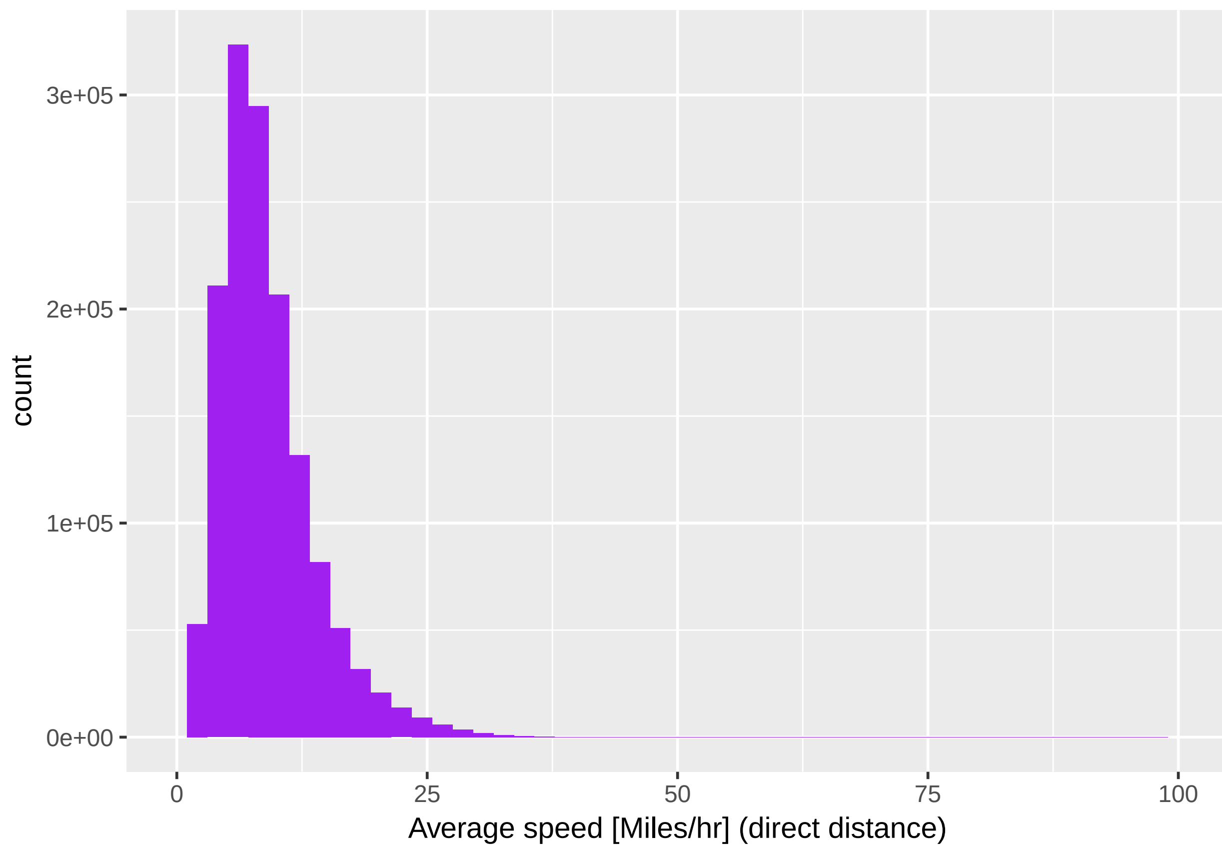


- Its clear observation, that distance increases trip duration.

Lets visualize how speed new york taxis travelling during peak hours and weekends. In order to find bogus values in the datasets, we can find extreme speed records and eliminate them

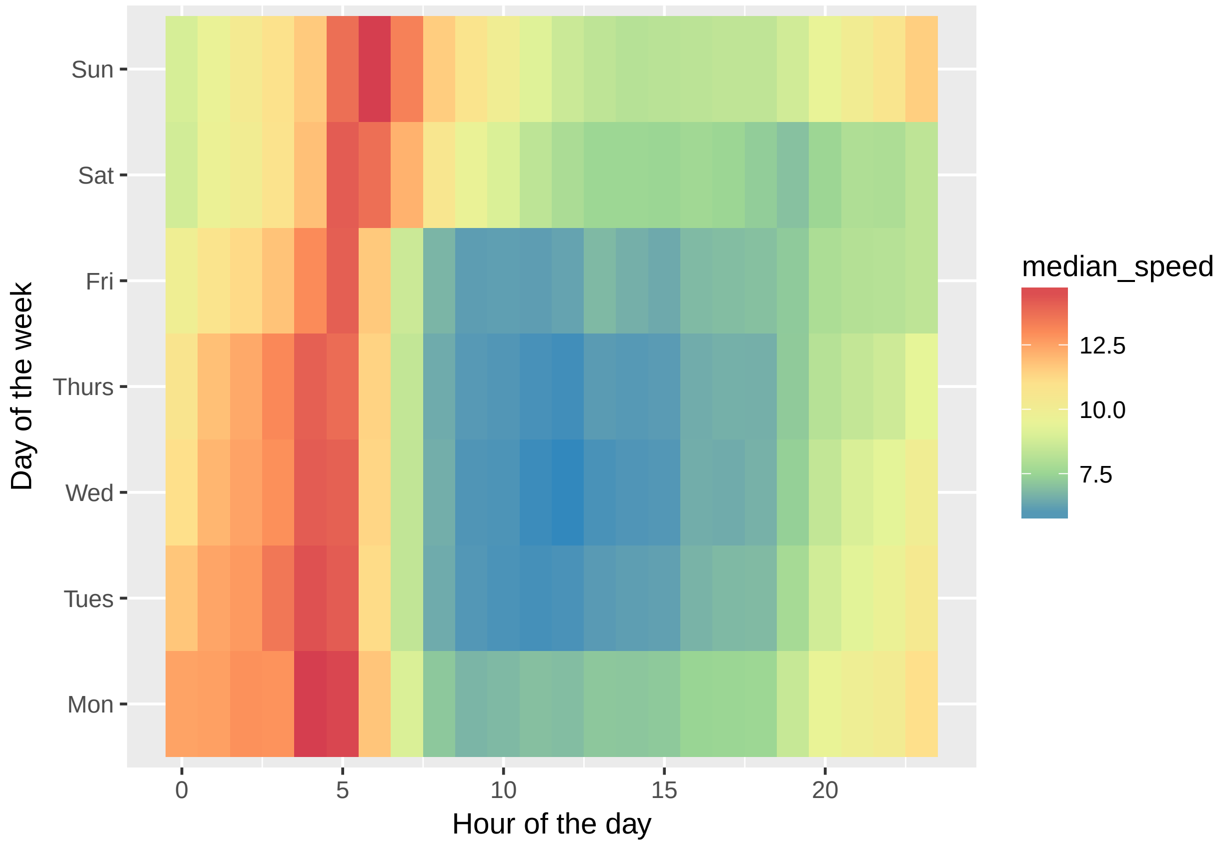
```
taxi %>%
  ggplot(aes(speed)) +
  geom_histogram(fill = "purple", bins = 50) +
  scale_x_continuous(limits = c(0, 100)) +
  labs(x = "Average speed [Miles/hr] (direct distance)")
```

```
## Warning: Removed 89 rows containing non-finite values (stat_bin).
```



Plotting speed on different times using heat map.

```
P1 <- taxi %>%
  group_by(wday, hour) %>%
  summarise(median_speed = median(speed)) %>%
  ggplot(aes(hour, wday, fill = median_speed)) +
  geom_tile() +
  labs(x = "Hour of the day", y = "Day of the week") +
  scale_fill_distiller(palette = "Spectral")
layout <- matrix(c(1, 1), 1, 1, byrow = TRUE)
multiplot(P1, layout = layout)
```

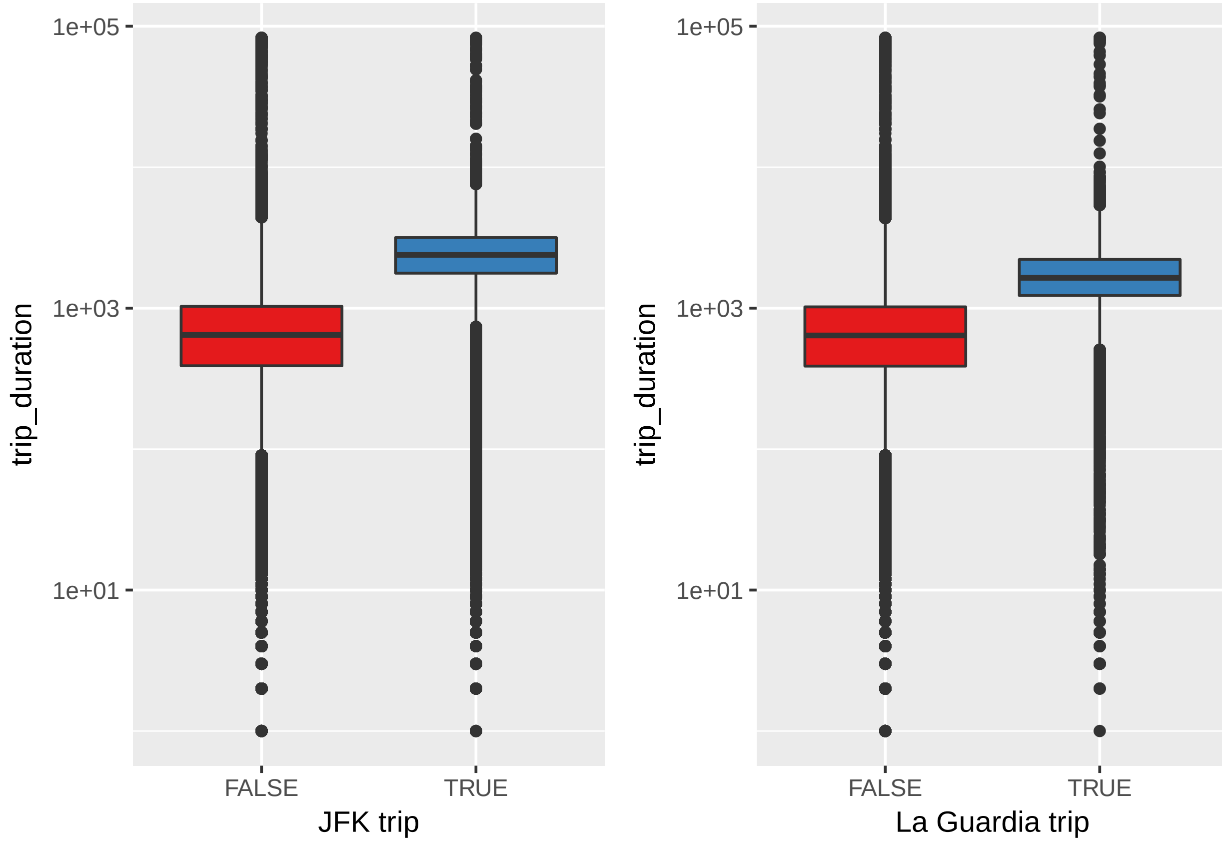


Airport trips has significant number of counts. Since airports are usually not in the city center it is reasonable to assume that the pickup/dropoff distance from the airport could be a useful predictor for longer trip_duration. We have derived a trip is JFK/La Gaurdia based on distance between pickup/dropoff location and airport coordinates. If its near to them, then we assume they are airport trips.

Let's visualize the how significantly trip duration increased for the airport trips. Based on below plot, its evident aiport trips are longer than regular trip to diff parts of city.

```
p1 <- taxi %>%
  filter(trip_duration < 23 * 3600) %>%
  ggplot(aes(jfk_trip, trip_duration, fill = jfk_trip)) +
  geom_boxplot() +
  scale_y_log10() +
  scale_fill_brewer(palette = "Set1") +
  theme(legend.position = "none") +
  labs(x = "JFK trip")

p2 <- taxi %>%
  filter(trip_duration < 23 * 3600) %>%
  ggplot(aes(lg_trip, trip_duration, fill = lg_trip)) +
  geom_boxplot() +
  scale_y_log10() +
  scale_fill_brewer(palette = "Set1") +
  theme(legend.position = "none") +
  labs(x = "La Guardia trip")

layout <- matrix(c(1, 2), 1, 2, byrow = FALSE)
```

```r
multiplot(p1, p2, layout = layout)
```



Data Cleaning.

The aim here is to remove trips that have improbable features, such as extreme trip durations or very low average speed.

** Filter trips more than 24 hours.

```r
long_hrtrips <- taxi %>%
  filter(trip_duration > 24 * 3600)

long_hrtrips %>%
  arrange(desc(dist)) %>%
  select(pickup_datetime, dropoff_datetime, speed) %>%
  head(05)
```

```
##       pickup_datetime    dropoff_datetime       speed
## 1 2016-01-05 00:19:42 2016-01-27 11:08:38 0.023252072
## 2 2016-02-13 22:46:52 2016-03-25 18:18:14 0.012633059
## 3 2016-02-13 22:38:00 2016-03-08 15:57:38 0.006533943
## 4 2016-01-05 06:14:15 2016-01-31 01:01:07 0.001643123
```

** Filter trips shorter than a few minutes

```r
min_trips <- taxi %>%
  filter(trip_duration < 5 * 60)
```

```r
min_trips %>%
  arrange(dist) %>%
  select(dist, pickup_datetime, dropoff_datetime, speed) %>%
  head(05)
```

```
##   dist     pickup_datetime    dropoff_datetime speed
## 1    0 2016-02-29 18:39:12 2016-02-29 18:42:59     0
## 2    0 2016-01-27 22:29:31 2016-01-27 22:29:58     0
## 3    0 2016-01-22 16:13:01 2016-01-22 16:13:20     0
## 4    0 2016-01-18 15:24:43 2016-01-18 15:28:57     0
## 5    0 2016-05-04 22:28:43 2016-05-04 22:32:51     0
```

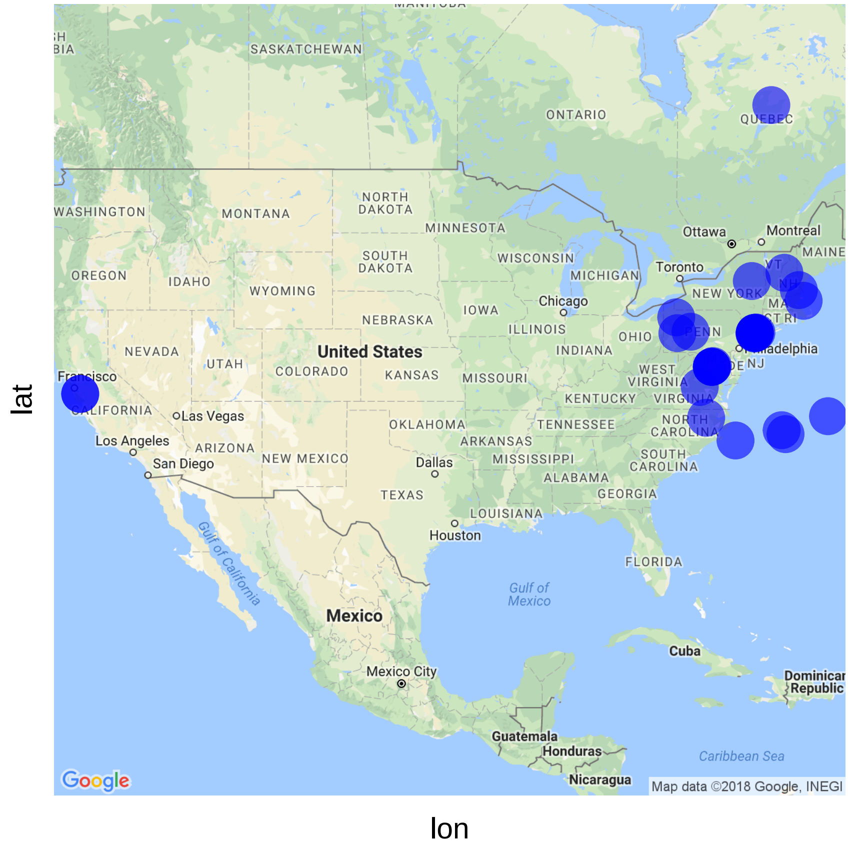
** Find trip with zero miles

```r
zero_dist <- taxi %>%
  filter(near(dist, 0))
nrow(zero_dist)
```

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

** Find trips away from New York

```r
long_dist <- taxi %>%
  filter((jfk_dist_pick > 3e5) | (jfk_dist_drop > 3e5))
long_dist <- data.frame(long_dist)
long_dist_coord <- long_dist %>%
  select(lon = pickup_longitude, lat = pickup_latitude)

my_map1 <- get_map(location = "United States", zoom = 4, maptype = "roadmap", source = "google", color =
ggmap(my_map1) +
  geom_point(data = long_dist, aes(x = pickup_longitude, y = pickup_latitude), size = 6, alpha = 0.6, co
  theme(axis.ticks = element_blank(), axis.text = element_blank())
```



Filter all bogus data from taxi dataset

```
taxi <- taxi %>%
  filter(
    trip_duration < 22 * 3600,
    dist > 0 | (near(dist, 0) & trip_duration < 60),
    jfk_dist_pick < 3e5 & jfk_dist_drop < 3e5,
    trip_duration > 10,
    speed < 100
  )
```

External data

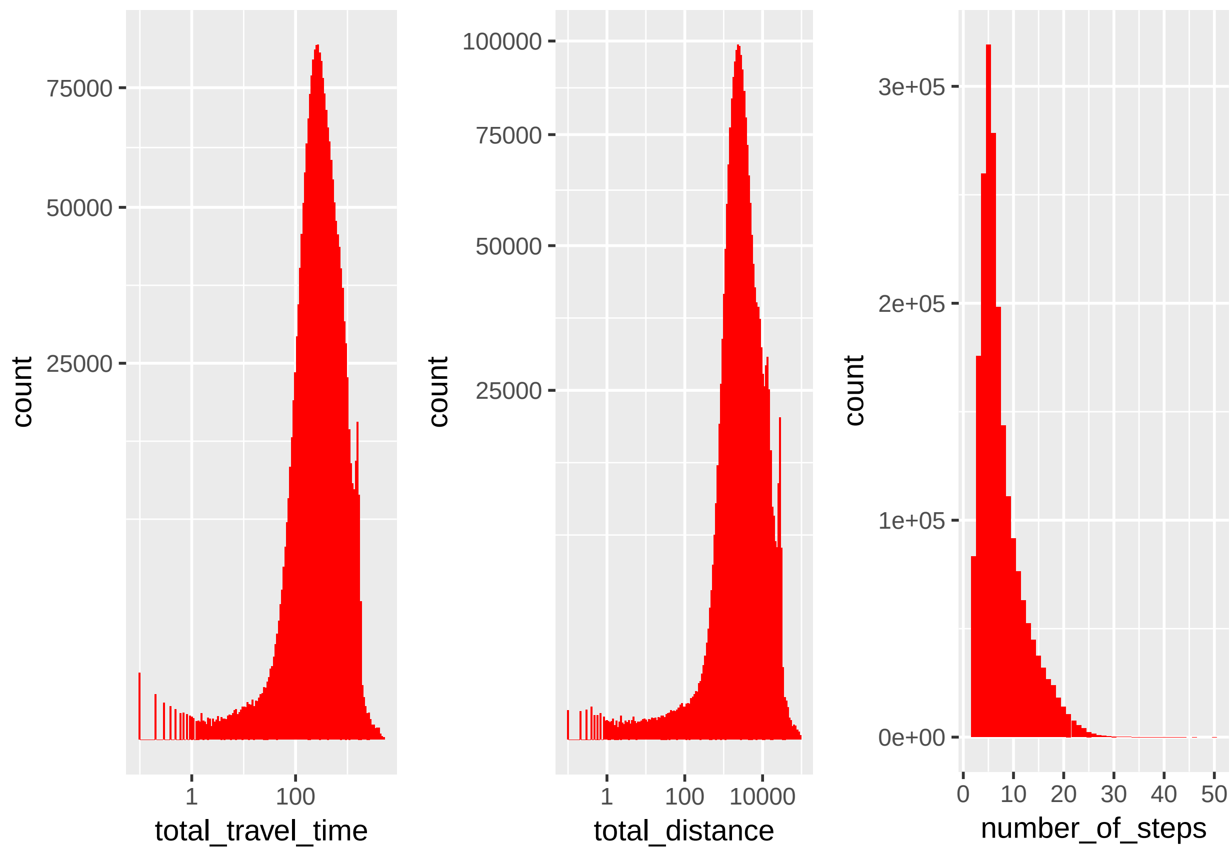
We are adding Open Source Routing Machine, OSRM data for each trip. This data is provided by oscarleo and we can download data from here and includes the pickup/dropoff streets and total distance/duration between these two points together with a sequence of travels steps such as turns or entering a highway.

```
fast1 <- read.csv("fastest_train_part_1.csv")
fast2 <- read.csv("fastest_train_part_2.csv")
ftest <- read.csv("fastest_routes_test.csv")
fast1 <- data.frame(fast1)
fast2 <- data.frame(fast2)
ftest <- data.frame(ftest)

fastest_route <- bind_rows(fast1, fast2, ftest)
glimpse(fastest_route)
```

```
## Observations: 2,083,777
## Variables: 12
## $ id                   <chr> "id2875421", "id2377394", "id3504673", "i...
## $ starting_street      <chr> "Columbus Circle", "2nd Avenue", "Greenwi...
## $ end_street           <chr> "East 65th Street", "Washington Square We...
## $ total_distance       <dbl> 2009.1, 2513.2, 1779.4, 1614.9, 1393.5, 1...
## $ total_travel_time    <dbl> 164.9, 332.0, 235.8, 140.1, 189.4, 138.8,...
## $ number_of_steps      <int> 5, 6, 4, 5, 5, 5, 2, 13, 6, 9, 4, 4, 10, ...
## $ street_for_each_step <chr> "Columbus Circle|Central Park West|65th S...
## $ distance_per_step    <chr> "0|576.4|885.6|547.1|0", "877.3|836.5|496...
## $ travel_time_per_step <chr> "0|61.1|60.1|43.7|0", "111.7|109|69.9|25....
## $ step_maneuvers       <chr> "depart|rotary|turn|new name|arrive", "de...
## $ step_direction       <chr> "left|straight|right|straight|arrive", "n...
## $ step_location_list   <chr> "-73.982316,40.767869|-73.981997,40.76768...
```

Lets visualize few of the important data points in Historgram to see the distribution.

```r
p1 <- fastest_route %>%
  ggplot(aes(total_travel_time)) +
  geom_histogram(bins = 150, fill = "red") +
  scale_x_log10() +
  scale_y_sqrt()

p2 <- fastest_route %>%
  ggplot(aes(total_distance)) +
  geom_histogram(bins = 150, fill = "red") +
  scale_x_log10() +
  scale_y_sqrt()

p3 <- fastest_route %>%
  ggplot(aes(number_of_steps)) +
  geom_bar(fill = "red")
layout <- matrix(c(1, 2, 3), 1, 3, byrow = TRUE)
multiplot(p1, p2, p3, layout = layout)
```



Joining OSRM file and TAXI file for further analysis.

In a first look at the joined data, we will mainly focus on the total_distance and total_travel_time, when combined with our original training data set:

```
osrm <- fastest_route %>%
  select(
    id, total_distance, total_travel_time, number_of_steps,
    step_direction, step_maneuvers
  ) %>%
  mutate(
    fastest_speed = total_distance / total_travel_time * 2.236,
    left_turns = str_count(step_direction, "left"),
    right_turns = str_count(step_direction, "right"),
    turns = str_count(step_maneuvers, "turn")
  ) %>%
  select(-step_direction, -step_maneuvers)

taxi <- left_join(taxi, osrm, by = "id")
```

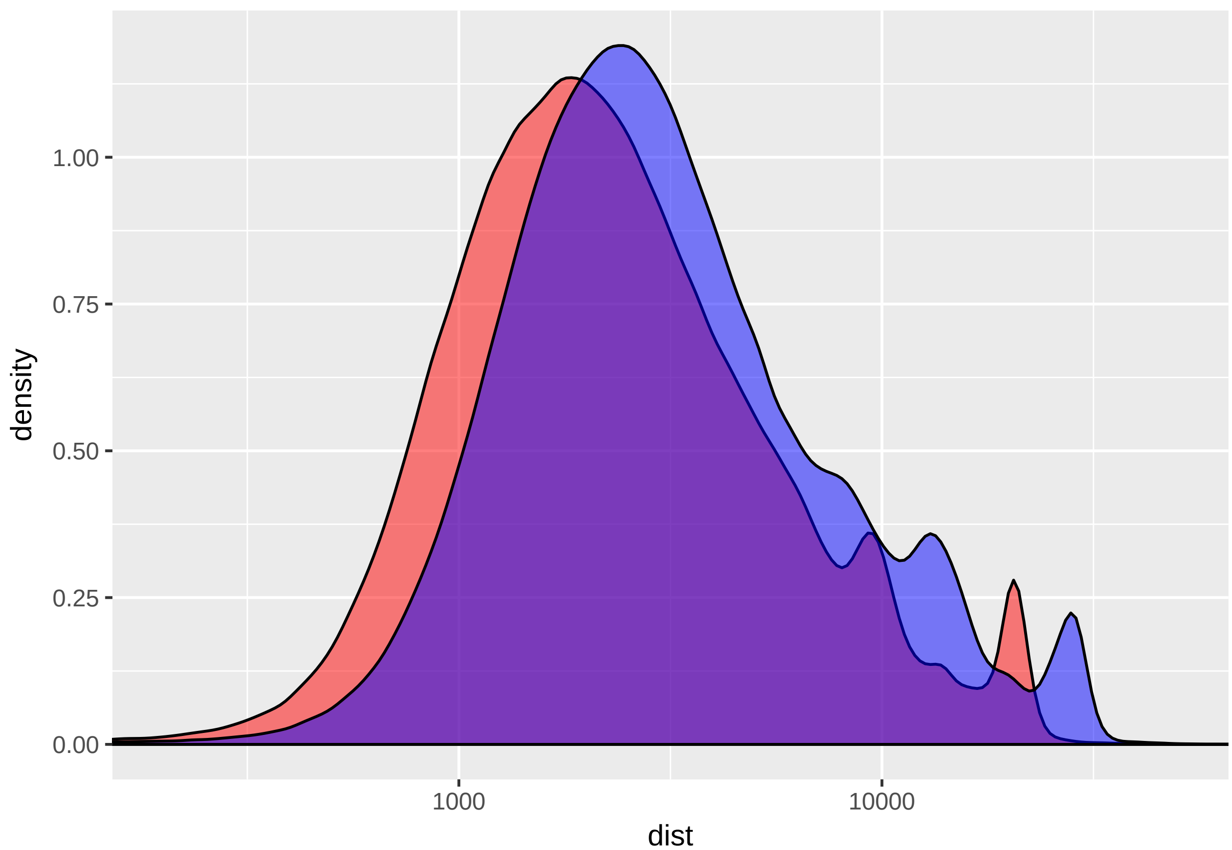
```
## Warning: Column `id` joining factor and character vector, coercing into
## character vector
```

In order to consider fastest route file for modeling, we need to find out total_travel_time in OSRM vs drip_duration recorded in taxi file.

```r
p1 <- taxi %>%
  ggplot(aes(trip_duration)) +
  geom_density(fill = "red", alpha = 0.5) +
  geom_density(aes(total_travel_time), fill = "blue", alpha = 0.5) +
  scale_x_log10() +
  coord_cartesian(xlim = c(5e1, 8e3))
```

Check for trip distance aswell.

```r
taxi %>%
  ggplot(aes(dist)) +
  geom_density(fill = "red", alpha = 0.5) +
  geom_density(aes(total_distance), fill = "blue", alpha = 0.5) +
  scale_x_log10() +
  coord_cartesian(xlim = c(2e2, 5e4))
```



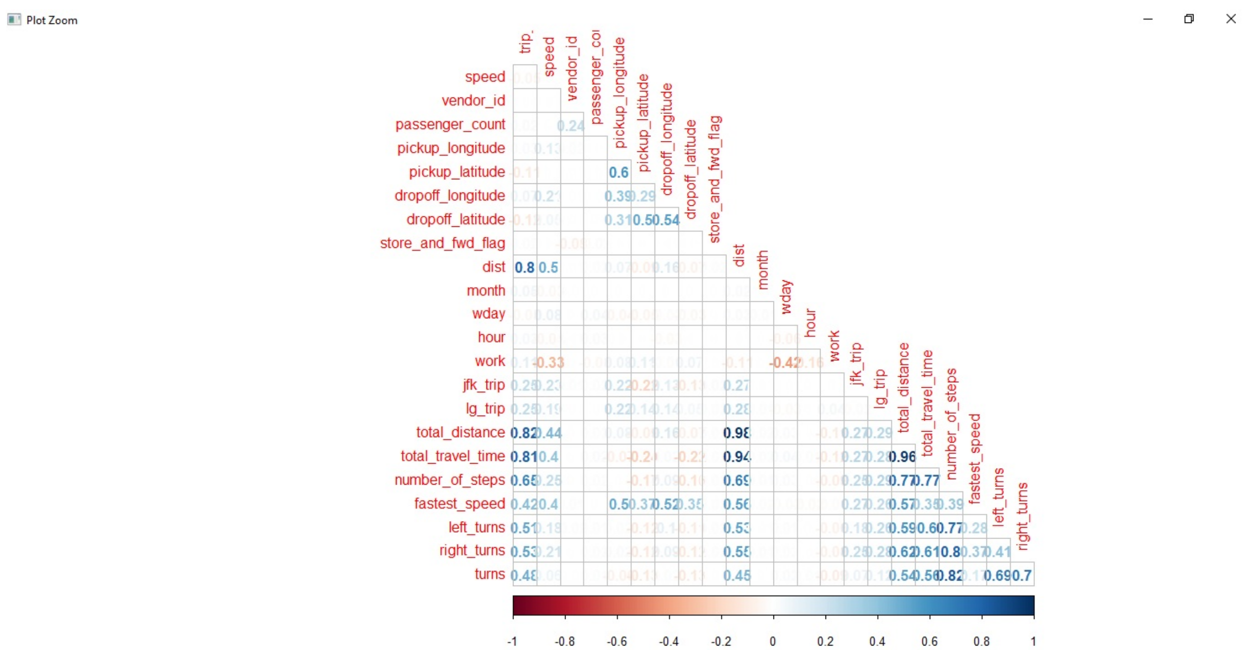
Correlation Matrix

Lets vizualize the relations between our variables using correlation matrix. We are using corrplot package for plotting.

NOTE: To resolve memory issue, we have run the corrplot outside markdown and attached screenshot below.

```r
# taxi %>%
#   select(
#     -id, -pickup_datetime, -dropoff_datetime, -jfk_dist_pick,
#     -jfk_dist_drop, -lg_dist_pick, -lg_dist_drop, -date
```

```
#    ) %>%
#    mutate(
#      passenger_count = as.integer(passenger_count),
#      vendor_id = as.integer(vendor_id),
#      store_and_fwd_flag = as.integer(as.factor(store_and_fwd_flag)),
#      jfk_trip = as.integer(jfk_trip),
#      wday = as.integer(wday),
#      month = as.integer(month),
#      work = as.integer(work),
#      lg_trip = as.integer(lg_trip)
#    ) %>%
#    select(trip_duration, speed, everything()) %>%
#    cor(use = "complete.obs", method = "spearman") %>%
#    corrplot(type = "lower", method = "number", diag = FALSE)
```



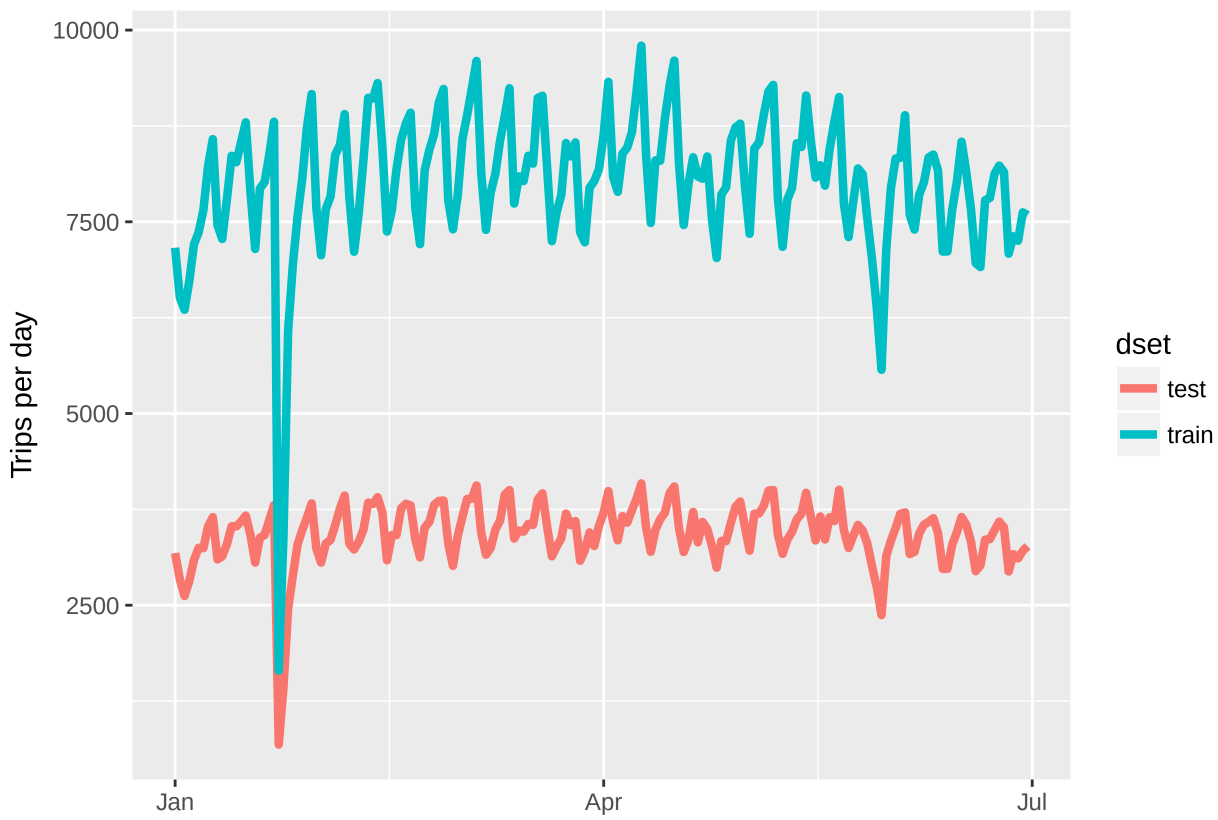
- The strongest correlations with the trip_duration are seen for the direct distance as well as the total_distance and total_travel_time derived from the OSRM data.
- Number of turns, presumably mostly via the number_of_steps, are having an impact on the trip_duration.
- Another effect on the trip_duration can be seen for our engineered features jfk_trip and lg_trip indicating journeys to either airport.
- Features like store_and_fwd_flag, vendor_id, passenger_count, speed are having little to no correlation, So we are removing from our model.

Model Selection and Prediction

Next is our most important step to find best method to predict the duration. We gonna test model our data with XGBoost.

We will start with XGBoost model now. Inorder to make sure we are training a feature which is relevent to test data. Plotting Test and Training Data overlap to see they are relevent.

```r
Temp1 <- combine %>%
  mutate(date = date(ymd_hms(pickup_datetime))) %>%
  group_by(date, dset) %>%
  count()
Temp1 %>%
  ggplot(aes(date, n, color = dset)) +
  geom_line(size = 1.5) +
  labs(x = "", y = "Trips per day")
```



*** Data formatting

Here we will format the selected features to turn them into integer columns, since many classifiers cannot deal with categorical values. We have formated Taxi data already. However, we need to redo same formating with combined records which intially joined both train and test data.

```r
# airport coordinates again
jfk_coord <- tibble(lon = -73.778889, lat = 40.639722)
la_guardia_coord <- tibble(lon = -73.872611, lat = 40.77725)

# derive distances
pick_coord <- combine %>%
  select(pickup_longitude, pickup_latitude)
drop_coord <- combine %>%
  select(dropoff_longitude, dropoff_latitude)
combine$dist <- distCosine(pick_coord, drop_coord)
combine$bearing <- bearing(pick_coord, drop_coord)
```

```r
combine$jfk_dist_pick <- distCosine(pick_coord, jfk_coord)
combine$jfk_dist_drop <- distCosine(drop_coord, jfk_coord)
combine$lg_dist_pick <- distCosine(pick_coord, la_guardia_coord)
combine$lg_dist_drop <- distCosine(drop_coord, la_guardia_coord)

combine <- combine %>%
  mutate(
    pickup_datetime = ymd_hms(pickup_datetime),
    dropoff_datetime = ymd_hms(dropoff_datetime),
    date = date(pickup_datetime)
  )
# join OSRM data with our data.
combine <- left_join(combine, osrm, by = "id")
```

Reformat data in to integer format.

```r
combine <- combine %>%
  mutate(
    store_and_fwd_flag = as.integer(factor(store_and_fwd_flag)),
    vendor_id = as.integer(vendor_id),
    month = as.integer(month(pickup_datetime)),
    wday = wday(pickup_datetime, label = TRUE),
    wday = as.integer(fct_relevel(wday, c("Sun", "Sat", "Mon", "Tues", "Wed", "Thurs", "Fri"))),
    hour = hour(pickup_datetime),
    work = as.integer((hour %in% seq(8, 18)) & (wday %in% c("Mon", "Tues", "Fri", "Wed", "Thurs"))),
    jfk_trip = as.integer((jfk_dist_pick < 2e3) | (jfk_dist_drop < 2e3)),
    lg_trip = as.integer((lg_dist_pick < 2e3) | (lg_dist_drop < 2e3))
  )
```

Just to make sure our data is in proper format.

```r
glimpse(combine)
```

```
## Observations: 2,083,778
## Variables: 32
## $ 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,...
## $ bearing            <dbl> 99.932546, -117.063997, -159.608029, -172.7...
## $ 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...
## $ total_distance     <dbl> 2009.1, 2513.2, 11060.8, 1779.4, 1614.9, 13...
```

```
## $ total_travel_time  <dbl> 164.9, 332.0, 767.6, 235.8, 140.1, 189.4, 1...
## $ number_of_steps    <int> 5, 6, 16, 4, 5, 5, 5, 17, 2, 13, 6, 9, 4, 4...
## $ fastest_speed      <dbl> 27.24286, 16.92625, 32.21984, 16.87336, 25....
## $ left_turns         <int> 1, 2, 5, 2, 2, 1, 1, 4, 0, 7, 2, 6, 1, 1, 4...
## $ right_turns        <int> 1, 2, 7, 1, 2, 3, 3, 9, 1, 5, 2, 2, 1, 1, 3...
## $ turns              <int> 1, 2, 9, 1, 3, 3, 2, 6, 0, 9, 3, 5, 2, 2, 5...
## $ 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...
```

Next, we are selecting Feature to be in the model. I have run the feature selection outside this document, to select optimal features which gives less RMSLE values. Based on that below are the selected Features will be used across all algorithms. Our evaluation metic is Root Mean Squared Logarithmic Error. In order to easily simulate the evaluation metric in our model fitting we replace the trip_duration with its logarithm. We adding +1 to duration to avoid infinity value incase of zeros.

```r
# predictor variables
train_cols <- c(
  "total_travel_time", "total_distance", "hour", "dist",
  "vendor_id", "jfk_trip", "lg_trip", "wday", "month",
  "pickup_longitude", "pickup_latitude", "lg_dist_drop", "jfk_dist_drop"
)
# target variable
y_col <- c("trip_duration")
# id feature
id_col <- c("id")
# auxilliary varaible
aux_cols <- c("dset")
# cleaning variable
clean_cols <- c("jfk_dist_drop", "jfk_dist_pick")
# ---------------------------------

# General extraction
# ---------------------------------
# extract test id column
test_id <- combine %>%
  filter(dset == "test") %>%
  select_(.dots = id_col)

# all relevant columns for train/test
cols <- c(train_cols, y_col, aux_cols, clean_cols)
combine <- combine %>%
  select_(.dots = cols)

# split train/test
train <- combine %>%
  filter(dset == "train") %>%
  select_(.dots = str_c("-", c(aux_cols)))
test <- combine %>%
  filter(dset == "test") %>%
  select_(.dots = str_c("-", c(aux_cols, clean_cols, y_col)))
```

```
# convert trip_duration to log

train <- train %>%
  mutate(trip_duration = log(trip_duration + 1))
```

To assess the mode performance we will run a cross-validation step and also split our training data into a *train* vs *validation* data set. Thereby, the model performance can be evaluated on a sample that the algorithm has not seen. We split our data into 80/20 fractions using a tool from the caret package.

```
set.seed(4321)
trainIndex <- createDataPartition(train$trip_duration, p = 0.8, list = FALSE, times = 1)

train <- train[trainIndex, ]
valid <- train[-trainIndex, ]
```

Next we clean the date before we start processng the date. We will do celaning only on Training dataset inorder to identify overfitting if we see any variance with Validation dataset.

```
valid <- valid %>%
  select_(.dots = str_c("-", c(clean_cols)))

train <- train %>%
  filter(
    trip_duration < 24 * 3600,
    jfk_dist_pick < 3e5 & jfk_dist_drop < 3e5
  ) %>%
  select_(.dots = str_c("-", c(clean_cols)))
```

XGBoost parameters and fitting

In order for *XGBoost* to properly ingest our data samples we need to re-format them slightly:

```
# convert to XGB matrix
temptrn <- train %>% select(-trip_duration)
tempval <- valid %>% select(-trip_duration)

dtrain <- xgb.DMatrix(as.matrix(temptrn), label = train$trip_duration)
dvalid <- xgb.DMatrix(as.matrix(tempval), label = valid$trip_duration)
dtest <- xgb.DMatrix(as.matrix(test))
```

Now we define the meta-parameters that govern how *XGBoost* operates. See here for more details. The *watchlist* parameter tells the algorithm to keep an eye on both the *training* and *validation* sample metrics.

```
xgb_params <- list(
  colsample_bytree = 0.7, # variables per tree
  subsample = 0.7, # data subset per tree
  booster = "gbtree",
  max_depth = 5, # tree levels
  eta = 0.3, # shrinkage
  eval_metric = "rmse",
  objective = "reg:linear",
  seed = 4321
)

watchlist <- list(train = dtrain, valid = dvalid)
```

And here we *train* our classifier, i.e. fit it to the *training* data. To ensure reproducability we set an R seed here.

NOTE: Recuing number of rounds to 60 due to memory and cpu issues. Actual submission model was trined with 100 rounds.

```
set.seed(4321)
gb_dt <- xgb.train(
  params = xgb_params,
  data = dtrain,
  print_every_n = 10,
  watchlist = watchlist,
  nrounds = 60
)
```

```
## [1]  train-rmse:4.227819 valid-rmse:4.226773
## [11] train-rmse:0.452179 valid-rmse:0.450131
## [21] train-rmse:0.424156 valid-rmse:0.422387
## [31] train-rmse:0.419012 valid-rmse:0.417150
## [41] train-rmse:0.415007 valid-rmse:0.413054
## [51] train-rmse:0.412323 valid-rmse:0.410407
## [60] train-rmse:0.410390 valid-rmse:0.408646
```

After the fitting we are running a 5-fold cross-validation (CV) to estimate our model's performance.

NOTE: Reducing number of rounds to 60 due to memory and cpu issue.

```
xgb_cv <- xgb.cv(xgb_params, dtrain, early_stopping_rounds = 10, nfold = 5, nrounds = 60)
```

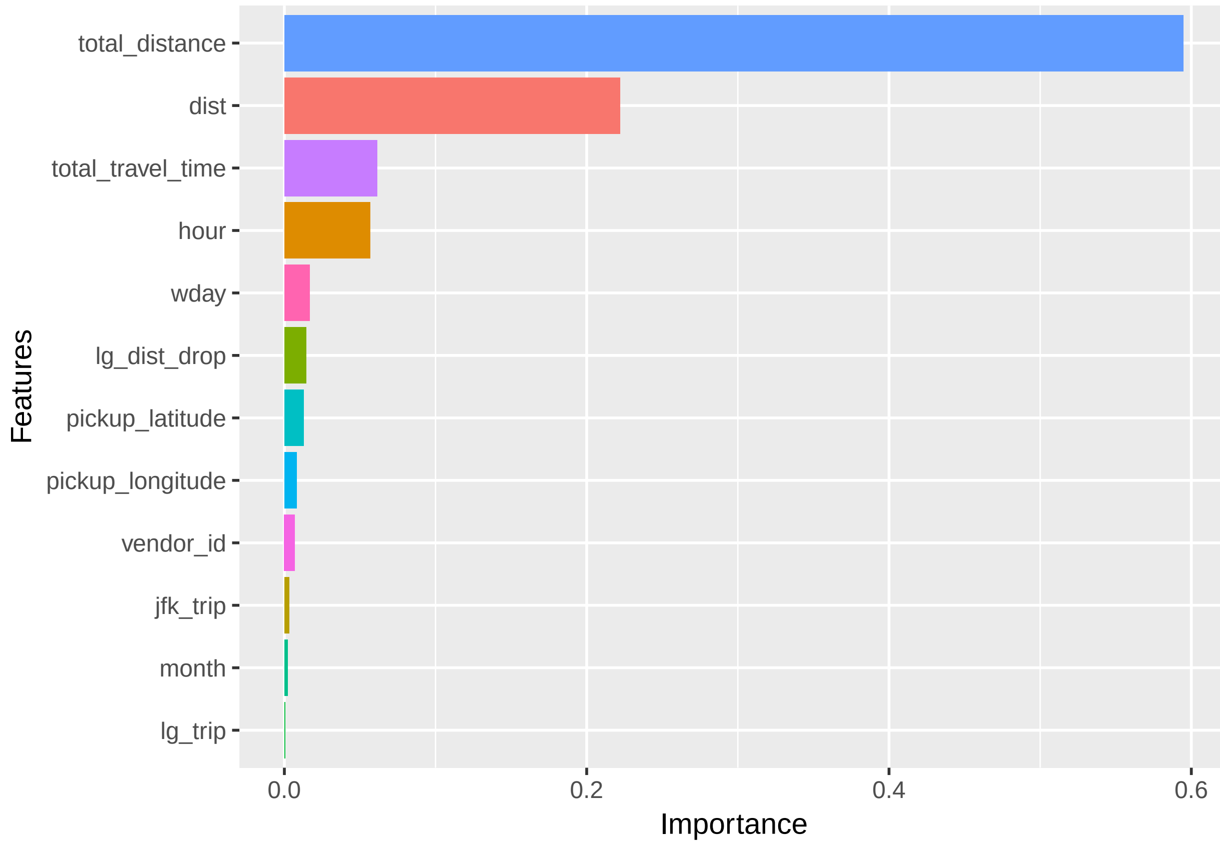
```
## [1]  train-rmse:4.228088+0.000448    test-rmse:4.228042+0.000605
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 10 rounds.
##
## [2]  train-rmse:2.978558+0.000502    test-rmse:2.978528+0.000325
## [3]  train-rmse:2.110823+0.000571    test-rmse:2.110840+0.000416
## [4]  train-rmse:1.512594+0.000554    test-rmse:1.512690+0.000706
## [5]  train-rmse:1.106134+0.000775    test-rmse:1.106351+0.000909
## [6]  train-rmse:0.837714+0.001575    test-rmse:0.838003+0.001365
## [7]  train-rmse:0.665955+0.001648    test-rmse:0.666381+0.001234
## [8]  train-rmse:0.562006+0.001620    test-rmse:0.562574+0.001420
## [9]  train-rmse:0.502488+0.002147    test-rmse:0.503211+0.000745
## [10] train-rmse:0.469604+0.002474    test-rmse:0.470459+0.000939
## [11] train-rmse:0.451871+0.002886    test-rmse:0.452830+0.001049
## [12] train-rmse:0.441905+0.002895    test-rmse:0.442953+0.001047
## [13] train-rmse:0.435746+0.002778    test-rmse:0.436824+0.000960
## [14] train-rmse:0.432275+0.002539    test-rmse:0.433440+0.000654
## [15] train-rmse:0.429903+0.002085    test-rmse:0.431206+0.000748
## [16] train-rmse:0.428347+0.002179    test-rmse:0.429711+0.000936
## [17] train-rmse:0.426969+0.002212    test-rmse:0.428429+0.000609
## [18] train-rmse:0.425685+0.001893    test-rmse:0.427165+0.001033
## [19] train-rmse:0.424519+0.001874    test-rmse:0.426130+0.001045
## [20] train-rmse:0.423698+0.001916    test-rmse:0.425439+0.000944
## [21] train-rmse:0.423002+0.001858    test-rmse:0.424838+0.000910
## [22] train-rmse:0.422236+0.002028    test-rmse:0.424121+0.000905
## [23] train-rmse:0.421357+0.001729    test-rmse:0.423298+0.001082
## [24] train-rmse:0.420745+0.001622    test-rmse:0.422845+0.001156
## [25] train-rmse:0.420236+0.001608    test-rmse:0.422425+0.001125
```

```
## [26] train-rmse:0.419639+0.001624    test-rmse:0.421963+0.001071
## [27] train-rmse:0.419143+0.001694    test-rmse:0.421562+0.000956
## [28] train-rmse:0.418691+0.001582    test-rmse:0.421192+0.001053
## [29] train-rmse:0.418234+0.001491    test-rmse:0.420855+0.001051
## [30] train-rmse:0.417704+0.001583    test-rmse:0.420397+0.000935
## [31] train-rmse:0.417276+0.001350    test-rmse:0.420046+0.001177
## [32] train-rmse:0.416826+0.001169    test-rmse:0.419647+0.001367
## [33] train-rmse:0.416448+0.001122    test-rmse:0.419366+0.001420
## [34] train-rmse:0.416042+0.001245    test-rmse:0.419049+0.001386
## [35] train-rmse:0.415729+0.001253    test-rmse:0.418823+0.001413
## [36] train-rmse:0.415359+0.001135    test-rmse:0.418539+0.001527
## [37] train-rmse:0.414956+0.001070    test-rmse:0.418218+0.001606
## [38] train-rmse:0.414631+0.001186    test-rmse:0.418031+0.001639
## [39] train-rmse:0.414343+0.001134    test-rmse:0.417834+0.001712
## [40] train-rmse:0.413955+0.001121    test-rmse:0.417476+0.001913
## [41] train-rmse:0.413637+0.001072    test-rmse:0.417206+0.001945
## [42] train-rmse:0.413364+0.001022    test-rmse:0.416983+0.002073
## [43] train-rmse:0.412983+0.001032    test-rmse:0.416651+0.002011
## [44] train-rmse:0.412799+0.001030    test-rmse:0.416581+0.001997
## [45] train-rmse:0.412531+0.000973    test-rmse:0.416370+0.002049
## [46] train-rmse:0.412327+0.000917    test-rmse:0.416268+0.002109
## [47] train-rmse:0.412038+0.000913    test-rmse:0.416041+0.002067
## [48] train-rmse:0.411735+0.000850    test-rmse:0.415810+0.002168
## [49] train-rmse:0.411526+0.000861    test-rmse:0.415734+0.002218
## [50] train-rmse:0.411242+0.000837    test-rmse:0.415535+0.002139
## [51] train-rmse:0.410965+0.000792    test-rmse:0.415343+0.002299
## [52] train-rmse:0.410757+0.000811    test-rmse:0.415251+0.002299
## [53] train-rmse:0.410489+0.000738    test-rmse:0.415105+0.002466
## [54] train-rmse:0.410319+0.000706    test-rmse:0.414979+0.002458
## [55] train-rmse:0.410111+0.000703    test-rmse:0.414841+0.002535
## [56] train-rmse:0.409880+0.000651    test-rmse:0.414702+0.002583
## [57] train-rmse:0.409630+0.000749    test-rmse:0.414544+0.002567
## [58] train-rmse:0.409362+0.000725    test-rmse:0.414347+0.002478
## [59] train-rmse:0.409096+0.000856    test-rmse:0.414166+0.002393
## [60] train-rmse:0.408956+0.000831    test-rmse:0.414050+0.002417
```

Feature importance

After training we will check which features are the most important for our model. This can provide the starting point for an iterative process where we identify, step by step, the significant features for a model. Here we will simply visualise these features:

```r
imp_matrix <- as.tibble(xgb.importance(feature_names = colnames(train %>% select(-trip_duration)), model

imp_matrix %>%
  ggplot(aes(reorder(Feature, Gain, FUN = max), Gain, fill = Feature)) +
  geom_col() +
  coord_flip() +
  theme(legend.position = "none") +
  labs(x = "Features", y = "Importance")
```



Write the XGBoost prediction to csv file and later will submit in Kaggle to know the actual score. Please refer below screenshot to see submission score.

```
test_preds <- predict(gb_dt, dtest)
pred <- test_id %>%
  mutate(trip_duration = exp(test_preds) - 1)

pred %>% write_csv("submit_XGBOOST.csv")
```

We have to compare the prediction file trip ids are matching with submit file Id. Also we have to checking the

```
identical(dim(submit), dim(pred))
```

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

```
glimpse(submit)
```

```
## Observations: 625,134
## Variables: 2
## $ id            <fctr> id3004672, id3505355, id1217141, id2150126, id1...
## $ trip_duration <int> 959, 959, 959, 959, 959, 959, 959, 959, 959, 959...
```

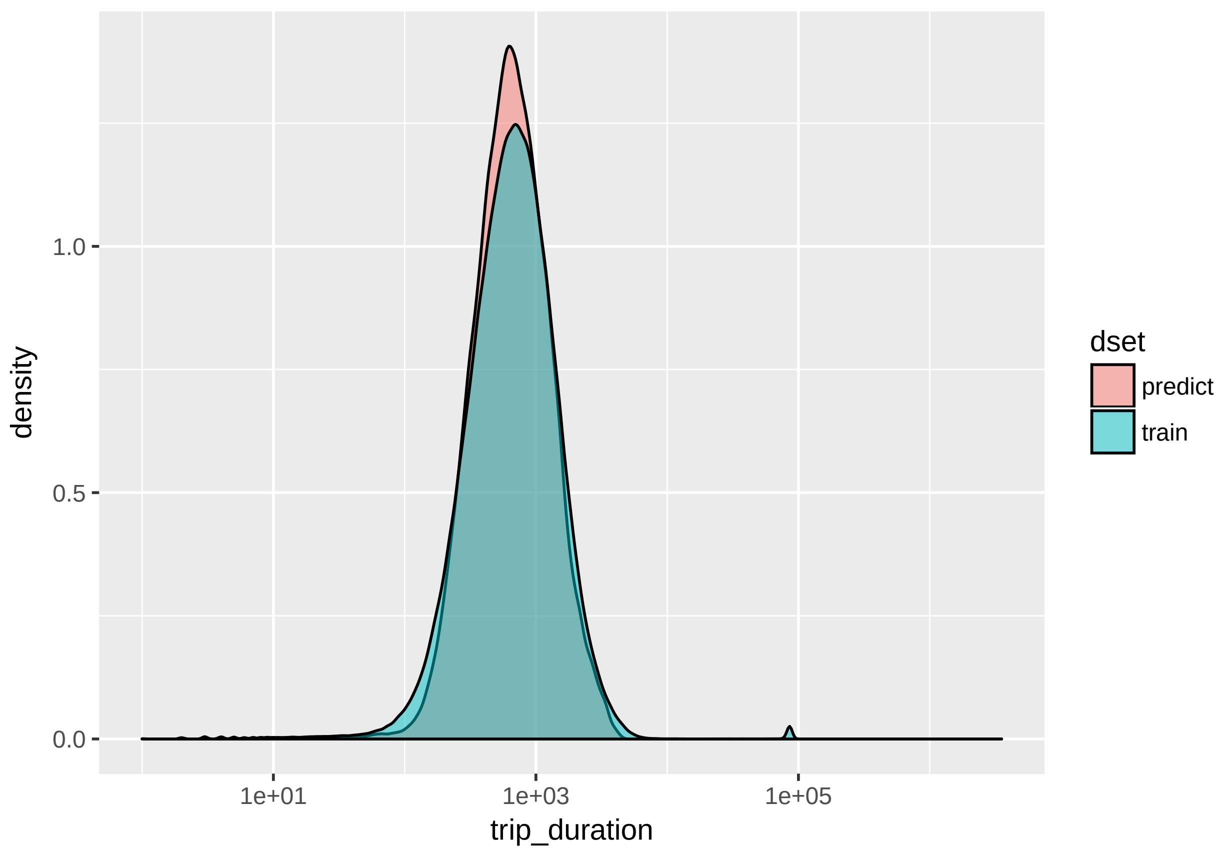
```
glimpse(pred)
```

```
## Observations: 625,134
## Variables: 2
## $ id            <chr> "id3004672", "id3505355", "id1217141", "id215012...
## $ trip_duration <dbl> 793.4018, 482.9604, 353.4382, 1071.8338, 262.098...
```

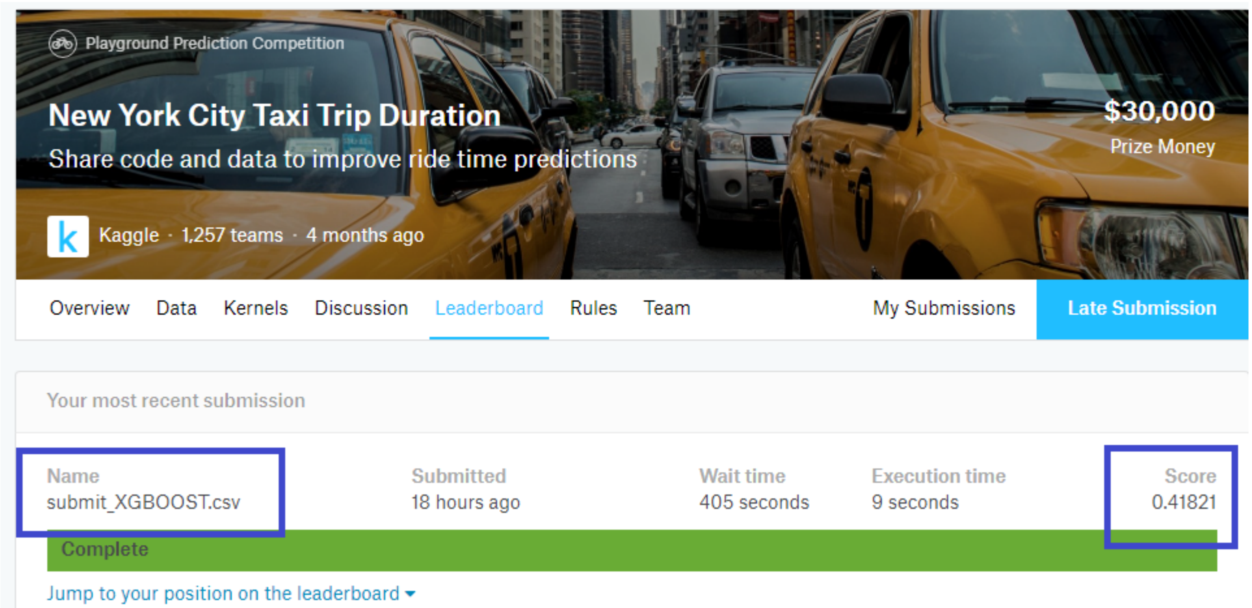
```r
bind_cols(submit, pred) %>% head(5)
```

```
##          id trip_duration       id1 trip_duration1
## 1 id3004672           959 id3004672       793.4018
## 2 id3505355           959 id3505355       482.9604
## 3 id1217141           959 id1217141       353.4382
## 4 id2150126           959 id2150126      1071.8338
## 5 id1598245           959 id1598245       262.0988
```

```r
trntemp <- train %>%
  select(trip_duration) %>%
  mutate(
    dset = "train",
    trip_duration = exp(trip_duration) - 1
  )
predtmp <- pred %>%
  mutate(dset = "predict")

bind_rows(trntemp, predtmp) %>%
  ggplot(aes(trip_duration, fill = dset)) +
  geom_density(alpha = 0.5) +
  scale_x_log10()
```



Final Score calcualted by Kaggle



References:

- EDA - NYC TAXI EDA The fast & the curious LINK
- EDA - From EDA to the Top (LB 0.367) LINK
- Mapping techniques LINK
- GGPLOT LINK
- Machine Learning LINK
- XGBOOST Basics LINK