UNIVERSITÉ
JEAN MONNET
SAINT-ÉTIENNE

Data Mining Project

New York City Taxi Trip Duration

From Kaggle's Competition

TRAN Tuan-Anh

Jean Monnet University - MLDM

07-03-2018

Table of Contents

I. Problem Understanding

1. Introduction

This is a data mining project in University Jean Monnet, ST Etienne, France. An important part of this project is to find “amazing knowledges” (interesting, unexpected, or valuable structures) that are embedded in a large dataset. The subject of project is open, so I choose this Kaggle’s competition to pratice my skills, and to have some ideas about real-life problems in Data Science. At the moment I write this notebook, the competition has been closed already. All information about the competition, you can find at Kaggle.

This project aims to build a simple *XGBoost model* that is able to predict the total ride duration of taxi trips in New York City. We have 2 files `.csv` to train and test the model.

The source codes and strategy I used in this project is from the awesome EDA of Heads or Tails in Kaggle’s forum. However, instead of going into details of everything, I just explore the important aspects. Besides that, I use only the dataset given by Kaggle and don’t add any external data to build my model.

Because of the large size of dataset, I could not put it in github. However, you can find the code R on my github.

2. Load Data

We can use ***tibble*** library to speed up loading data:

```
train <- as.tibble(fread('./data/train.csv'))
test <- as.tibble(fread('./data/test.csv'))
sample_submit <- as.tibble(fread('./data/sample_submission.csv'))
```

Data’s features:

```
## 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...
```

We aware that:

- `vendor_id` is only 1 or 2, so maybe this is 2 different taxi companies
- `pickup` and `dropoff` describe the time and the coordinates where the meter engage and disengage
- `store_and_fwd_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 a trip in second

3. Missing values

It is important to know if the data miss values or not. To check this, we can use function `is.na()`:

```
sum(is.na(train))

## [1] 0

sum(is.na(test))

## [1] 0
```

We can see that there is no missing values in our data!

4. Combining train and test

For categorical encoding, all categories might be labelled differently if done in two separate operations. That's why we need to combine sets to maintain consistency between them.

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

5. Reformating features

To prepare for the data visualization, we need to turn date characters to date object and encode the categorical data to ***factors.***

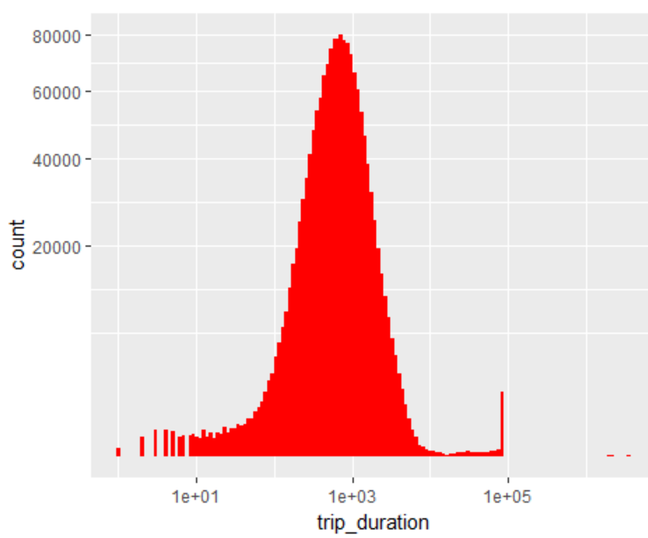
```
train <- train %>%
  mutate(pickup_datetime = ymd_hms(pickup_datetime),
         dropoff_datetime = ymd_hms(dropoff_datetime),
         vendor_id = factor(vendor_id),
         passenger_count = factor(passenger_count))
```

II. Data Understanding

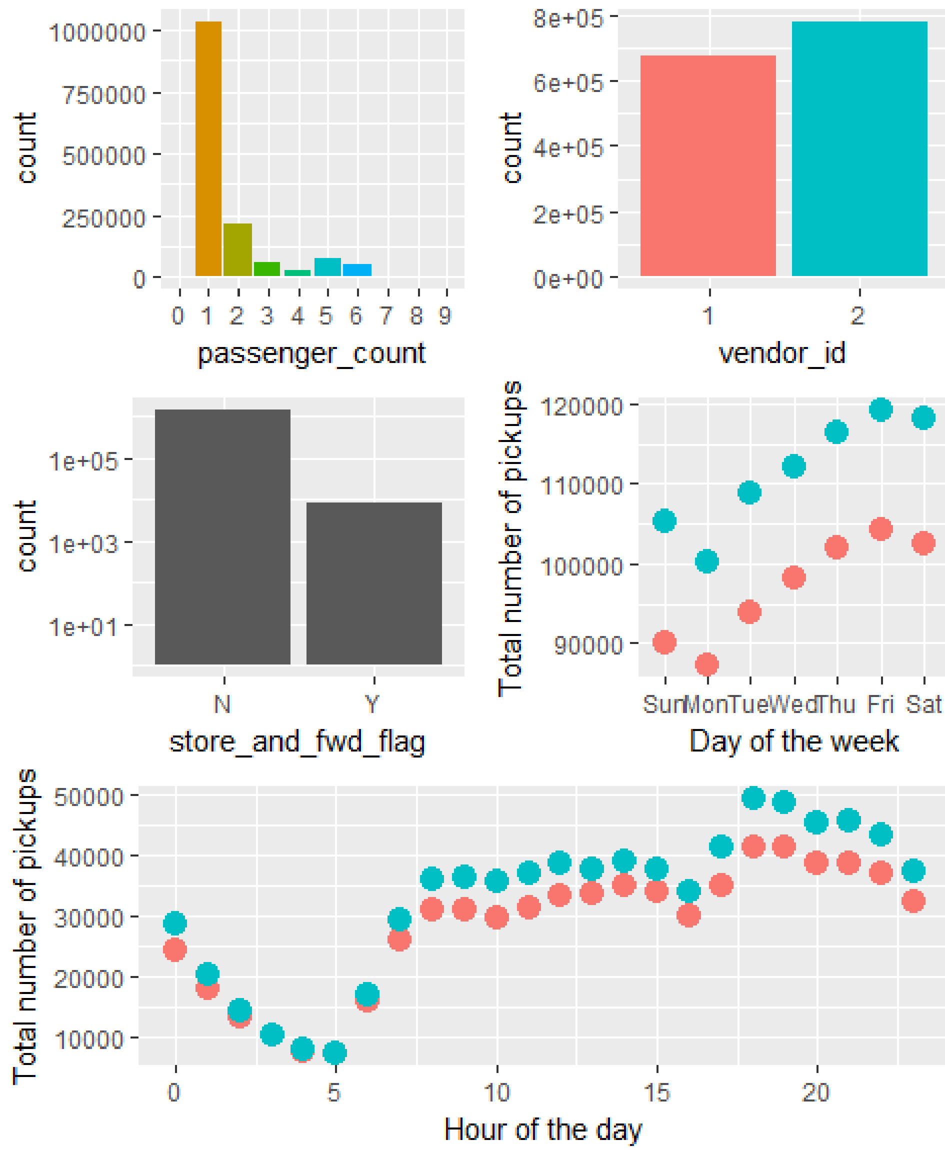
1. Visualisations

To understand the features of our data, now we take a look at where taxi driver pickups their clients in NewYork city by using `leaflet` package:

We found that most of trips locate in one part of NewYork, and there are also many trips from/to the JFK airport and La Guardia airport. Now we'll see the distribution of trips by its duration:

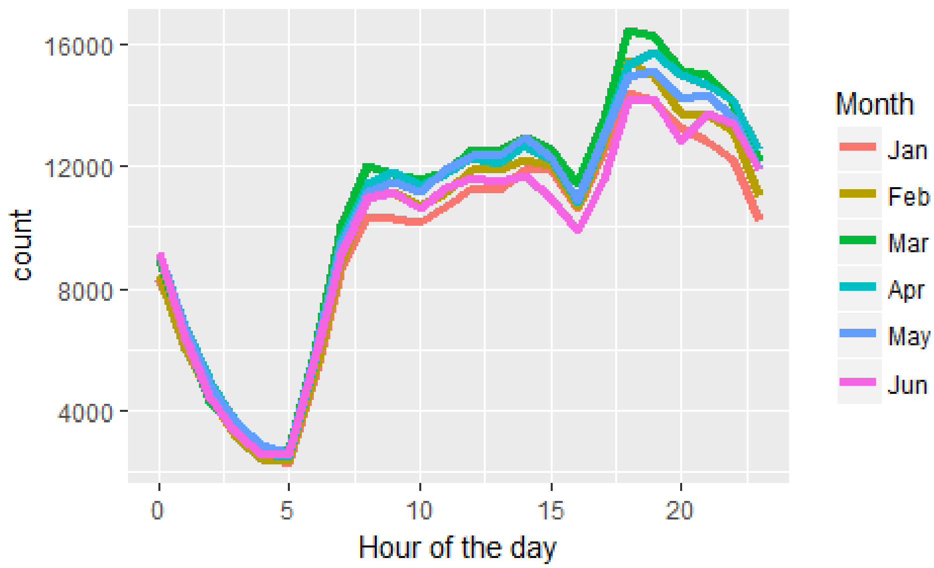


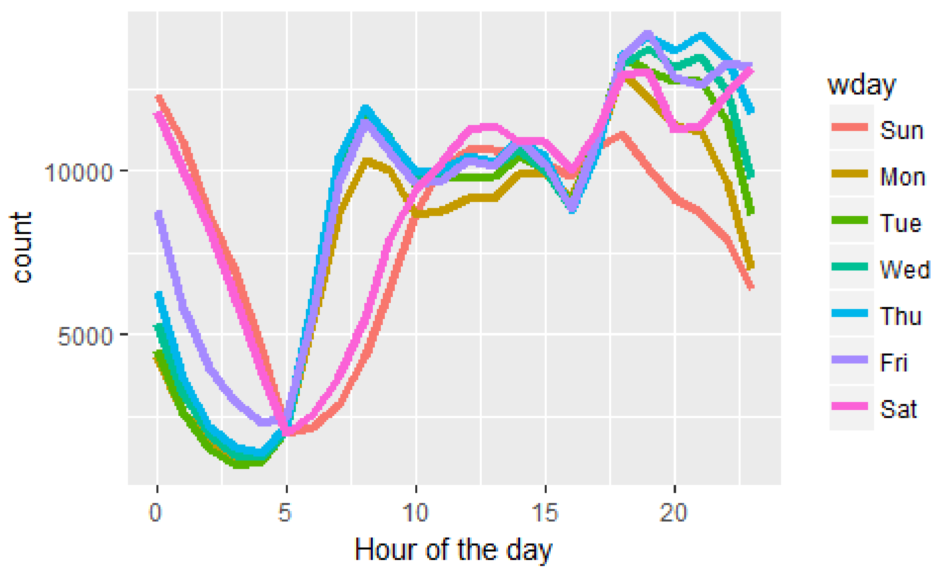
Most of trips is about 1000s, which means ~17min.



We find that:

- Most of rides has only 1 passenger.
- The groups of 5 and 6 passengers are more frequent than 3 and 4.
- Vendor 2 sale more tickets than vendor 1 all day of the week and at the end of the week (Fri, Sat) they sell many more tickets than on Monday.
- The number of trips is stable in the morning and increases at the rush hour in the evening and drop until 5am.
- About 50% of the trip data is not transmitted to the vendors immediately.





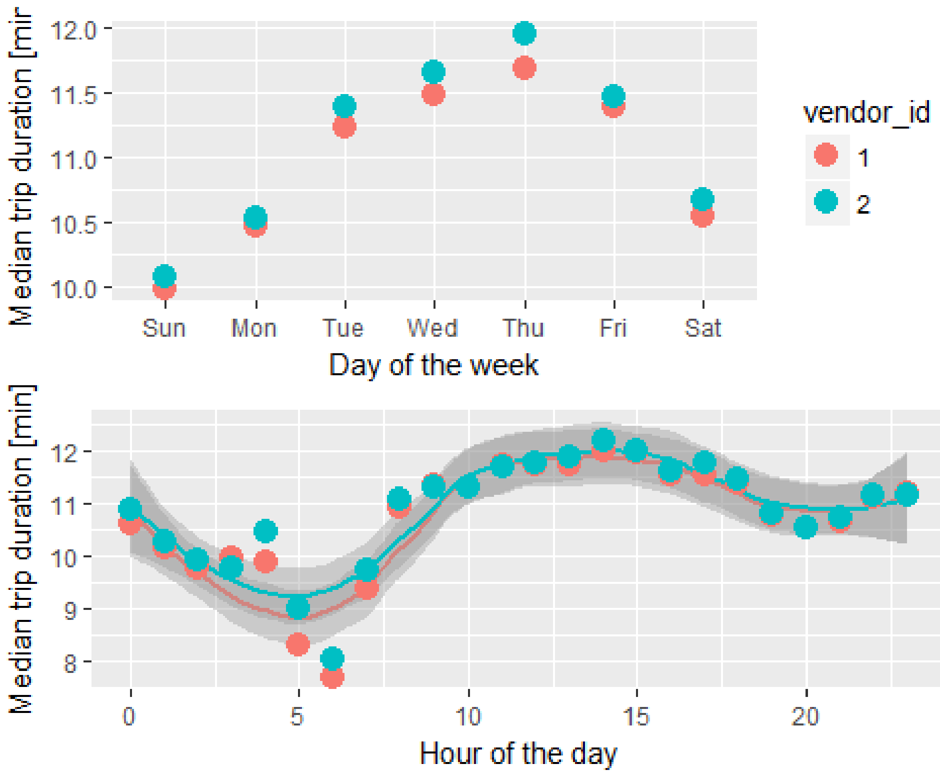
We also find that June has fewer trips than Mars and at the weekend, people have tendency to go out for night parties so there are more trips in Saturday and Sunday early morning than other days.

2. Feature relations

a. Pickup date/time vs `trip_duration`

In this section, we'll try to answer the questions following:

- How does the variation in trip numbers throughout the day and the week affect the average trip duration?
- Do quieter days and hours lead to faster trips?

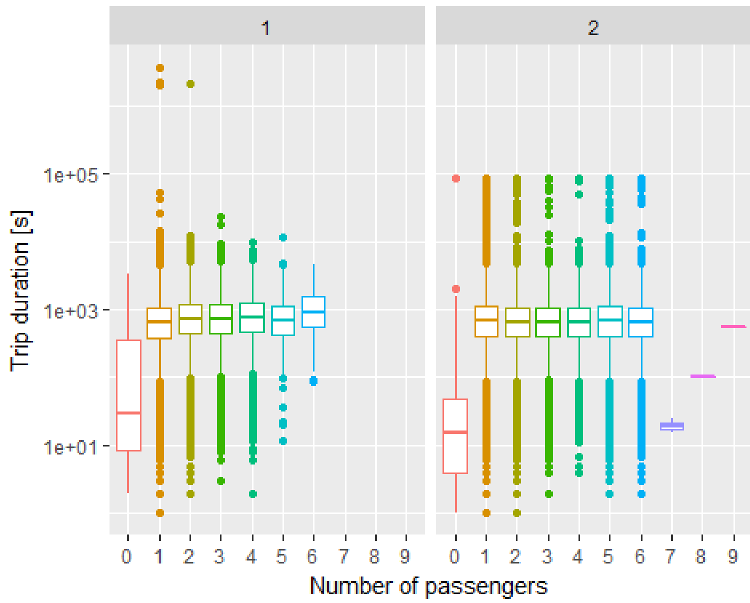


We find:

- There is indeed a similar pattern as for the business of the day of the week. Vendor 2, the one with the more frequent trips, also has consistently higher trip durations than vendor 1. Therefore, **it will be worth adding the *vendor_id* feature to a model to test its predictive importance.**
- Over the course of a typical day we find a peak in the early afternoon and dips around 5-6am and 8pm. **The weekday and hour of a trip appear to be important features for predicting its duration and should be included in a successful model.**

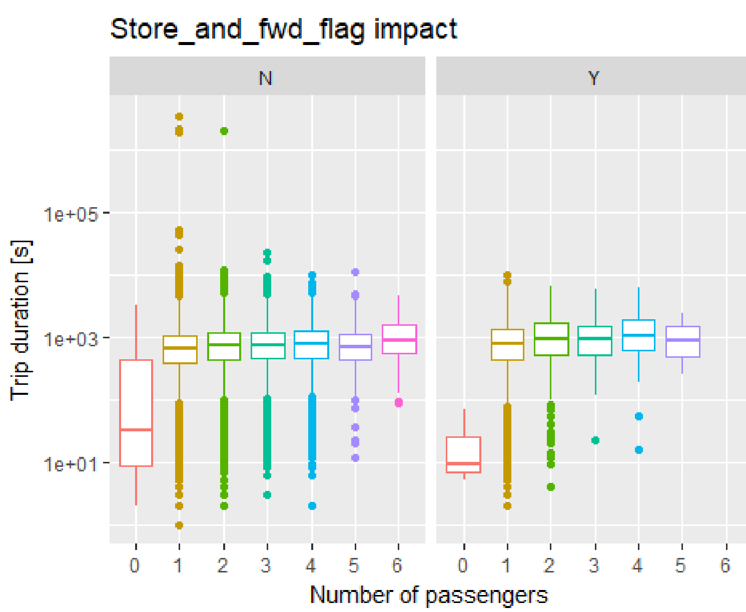
b. Passenger count and Vendor vs `trip_duration`

Now we want to know the impact of number of passengers on the trip duration. We can use boxplots to figure out this issue.



We find that between 1 and 6 passengers, the trip duration is very similar in both vendors.

c. Store and Forward vs trip_duration



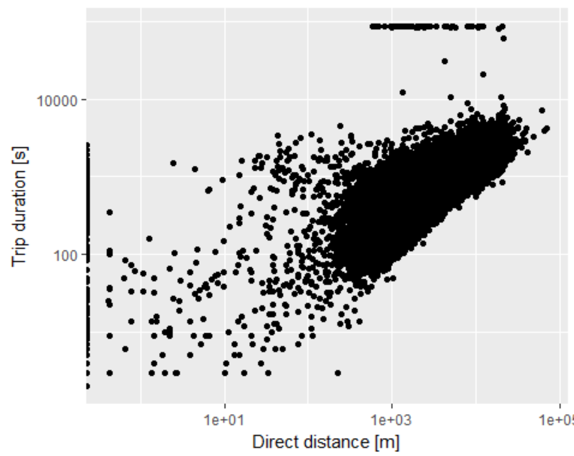
We find that there is no overwhelming differences between the stored and non-stored trips. The stored ones might be slightly longer, though, and don't include any of the suspiciously long trips.

3. Feature engineering

In this section we build new features from the existing ones, trying to find better predictors for our target variable. The new temporal features (date, month, wday, hour) are derived from the pickup_datetime. We got the JFK and La Guardia airport coordinates from Wikipedia.

a. Direct distance of the trip

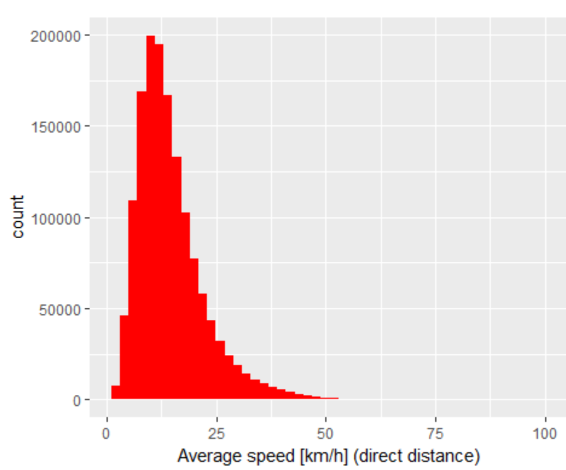
By calculating the distance between pickup and drop point, we have the minimum possible travel distance. To compute these distances, we can use the **distCosine** function of the `geosphere` package.



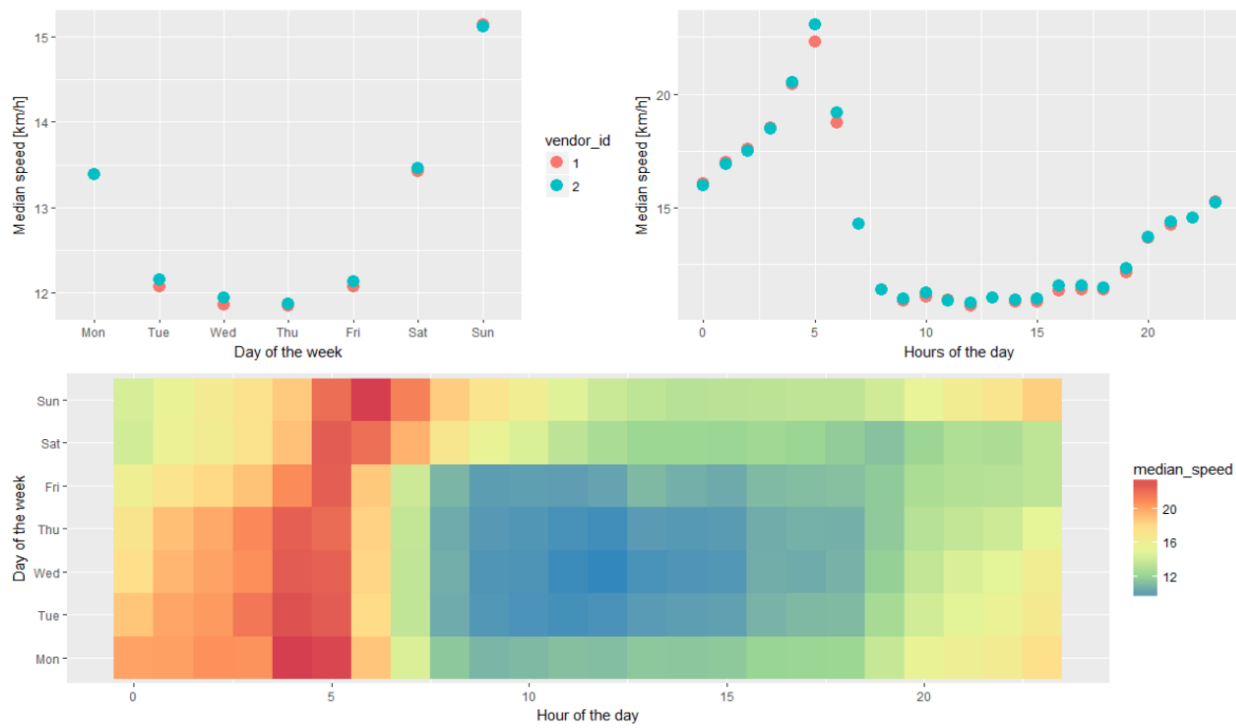
We find that in general, the `trip_duration` is proportionnal to the distance of travels. However, the 24-hour trips look suspicious and there are number of trips with very short distances, down to 1 meter, but with a large range of apparent `trip_durations`.

b. Travel speed

We can easily compute the speed during taxi trips, it's not used as a predictor for our model. However it might be useful to clean up our data and find other features.



The average speed is about 15 km/h, we can guess that New York is a crowed city with many traffic jams every day. In a similar way as the average duration per day and hour we can also investigate the average speed for these time bins:



We find that:

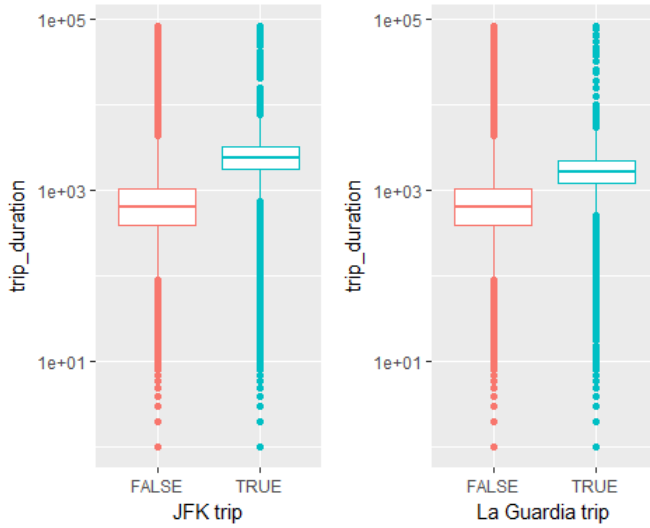
- Taxis travel faster on the weekend and on Monday than the rest of the week.
- In the early morning, taxis' speed is higher than in working hours.
- The heatmap in the lower panel visualizes how these trends combine to create a "low-speed-zone" in the middle of the day and week. Based on this, we create a new feature work, which we define as working time (8am-6pm on Mon-Fri).

c. Airport distance

Since airports are usually not in the city centre it is reasonable to assume that the pickup/dropoff distance from the airport could be a useful predictor for longer `trip_durations`.

In Feature Engineering section, we already defined the coordinates of the two airports and compute the corresponding distances. We can also define a JFK/La Guardia trip as having a pickup or dropoff distance of less than 2 km from the corresponding airport.

Now, what are the trip_durations of these journeys?



We noticed that the `trip_duration` to the airports is always longer than normal trips, or our hypothesis was correct.

III. Data Preparation

We will remove trips that have improbable features, such as extreme trip durations or very low average speed.

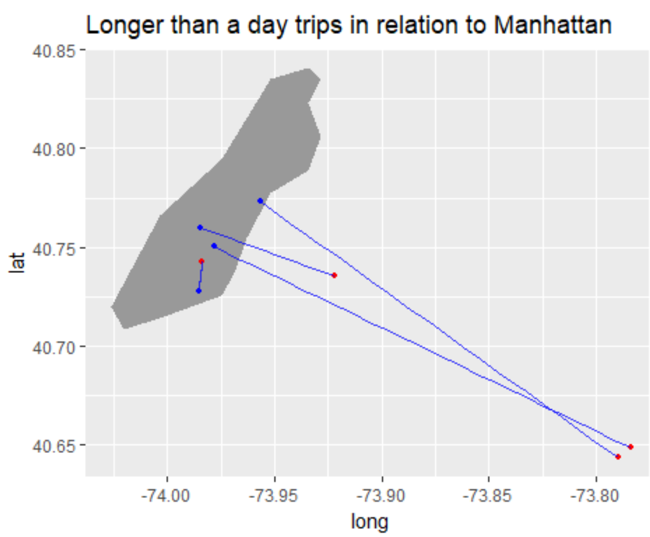
1. Extreme trip durations

Now we'll see the distances of the trips that took a day or longer. Here we make use of the maps package to draw an outline of Manhattan, then overlay the pickup coordinates in red, and the dropoff coordinates in blue.

a. Longer than a day

These are few trips that need more than 1 day to complete:

```
## # A tibble: 4 x 3
##   pickup_datetime     dropoff_datetime      speed
##   <dttm>              <dttm>                <dbl>
## 1 2016-01-05 00:19:42 2016-01-27 11:08:38 0.0374
## 2 2016-02-13 22:38:00 2016-03-08 15:57:38 0.0105
## 3 2016-01-05 06:14:15 2016-01-31 01:01:07 0.00265
## 4 2016-02-13 22:46:52 2016-03-25 18:18:14 0.0203
```



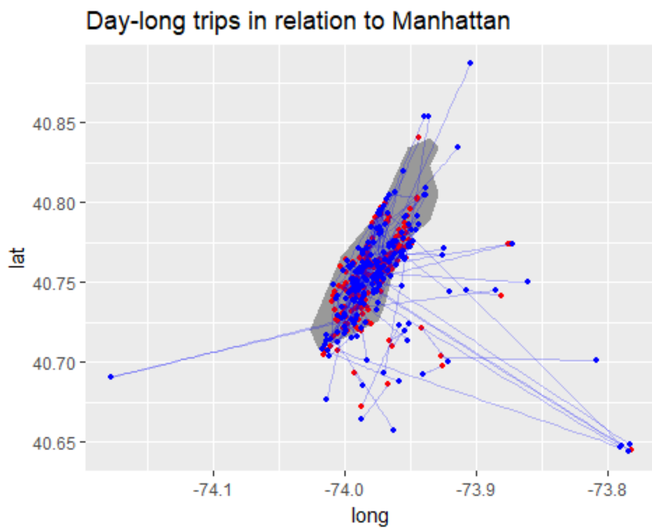
These values should be removed from the training data set for continued exploration and modelling.

b. Close to 24 hours

It's weird if there exist any trip lasting about 24h consecutively without any break. Now, we'll take a look at these trips whose duration is between 22h and 24h.

```
## # A tibble: 5 x 4
##    dist pickup_datetime     dropoff_datetime    speed
##   <dbl> <dttm>              <dttm>              <dbl>
## 1 60666 2016-06-04 13:54:29 2016-06-05 13:40:30 2.55
## 2 42401 2016-01-28 21:43:02 2016-01-29 21:33:30 1.78
## 3 28116 2016-03-14 22:46:05 2016-03-15 22:24:27 1.19
## 4 22808 2016-06-18 17:41:47 2016-06-19 16:30:41 1.000
## 5 22759 2016-06-01 19:52:42 2016-06-02 19:35:09 0.960
```

What do these trips look like on the map?



We find:

- There are two major groups: within Manhattan and between Manhattan and the airport.
- There exist a few long trips from other places that might cause the duration more than 22h.

We will remove `trip_durations` longer than 22 hours from the exploration and possibly from the modelling.

c. Shorter than a few minutes

On the other side, the trips lasting for a couple of minutes are absolutly abnormal.

```
## # A tibble: 5 x 4
##    dist pickup_datetime     dropoff_datetime    speed
##   <dbl> <dttm>              <dttm>              <dbl>
## 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
```

We notice that there are also many zero-distance trips:

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

Now, let's see those trips:

```
## # A tibble: 5 x 4
##   trip_duration pickup_datetime     dropoff_datetime    vendor_id
##           <int> <dttm>              <dttm>              <fct>
## 1         86352 2016-06-05 01:09:39 2016-06-06 01:08:51 2
## 2         85333 2016-01-01 15:27:28 2016-01-02 15:09:41 2
## 3         78288 2016-05-18 13:40:45 2016-05-19 11:25:33 2
## 4          5929 2016-06-08 16:47:44 2016-06-08 18:26:33 2
## 5          4683 2016-05-25 17:36:49 2016-05-25 18:54:52 2
```

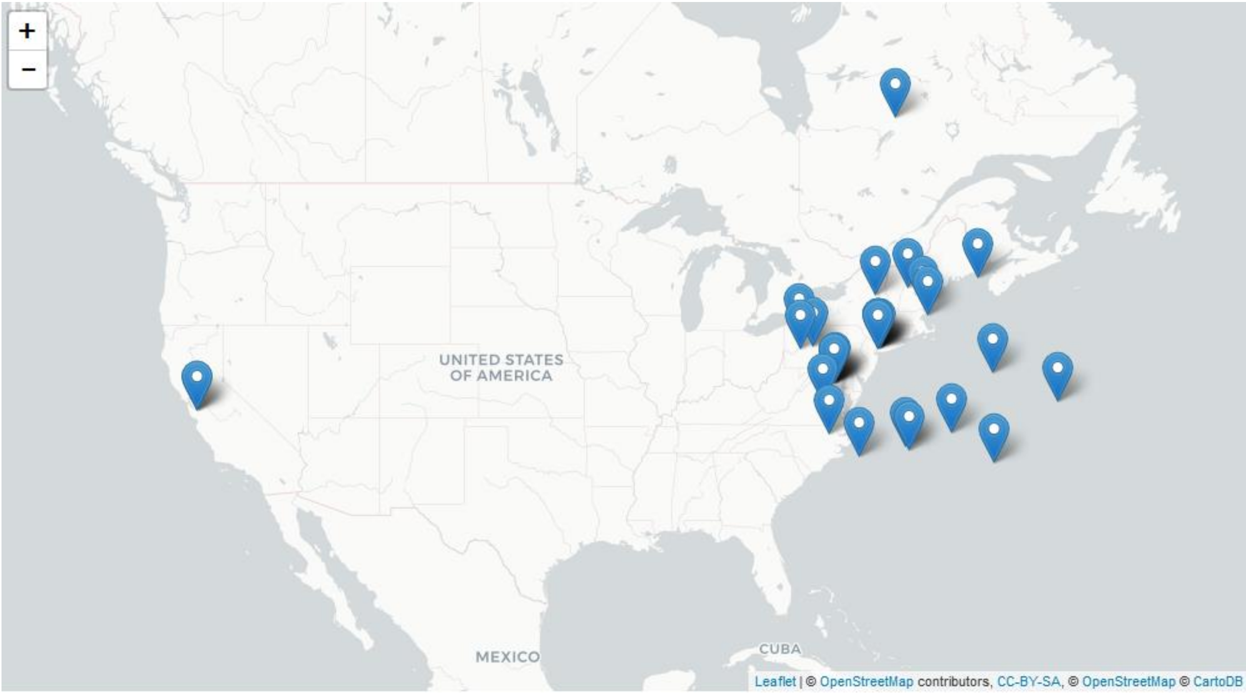
Both phenomena might still be somehow possible. For the first one, assuming that someone got into a taxi but then changed their mind before the taxi could move. For the second one, they might get into a taxi that's stuck in a traffic jam or maybe they go out to somewhere, then changed their mind and come back the starting point, or maybe they go out to pick up someone else, etc… **Therefore, I will not remove these information from dataset.**

2. Strange trips

There are some unbelievable information with pickup or dropoff locations more than 300 km away from NYC (JFK airport)

```
## # A tibble: 31 x 6
##    id        jfk_dist_pick jfk_dist_drop      dist trip_duration     speed
##    <chr>             <dbl>         <dbl>     <dbl>         <int>     <dbl>
##  1 id2854272       4128727       4128719      14.8           499    0.107
##  2 id3777240       4128721       4128726      21.8          1105    0.0711
##  3 id2306955       1253574         21484 1242299             792 5647
##  4 id1974018       1115646       1115646       0             369    0
##  5 id0267429        988748        988748       0             961    0
##  6 id0838705        695767        481141  215468            1131  686
##  7 id0205460        684133        684133       0             329    0
##  8 id0978162        674251        941618  315117             875 1296
##  9 id3525158        665877        665877       0             385    0
## 10 id1510552        642663        472020  892212             611 5257
## # ... with 21 more rows
```

Now let's see where these trips happen:



We can see that many trips has been taken even on the ocean☺. So we must remove these trips from training set.

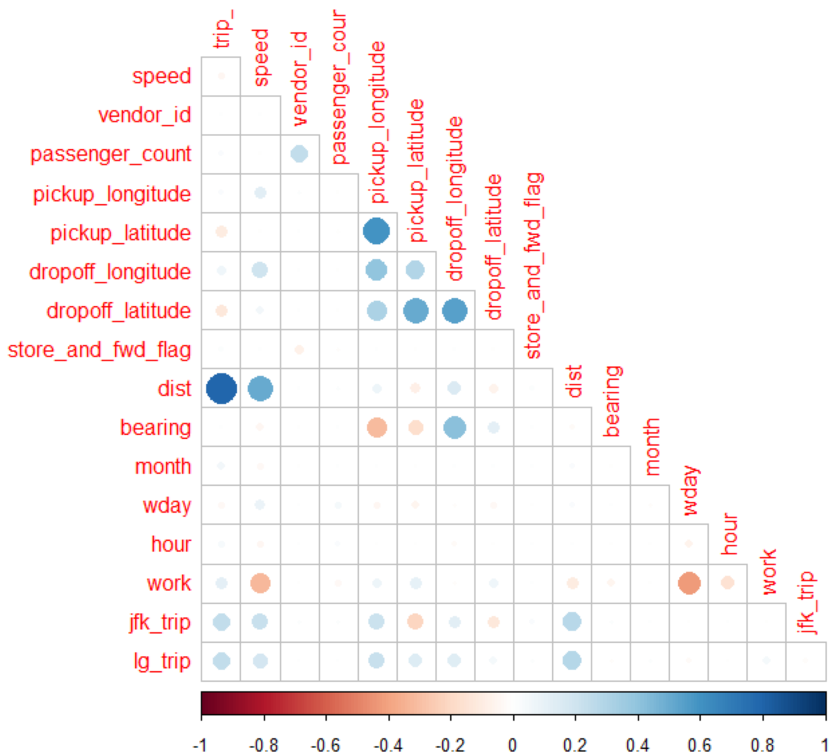
3. Final cleaning

Here we apply the cleaning filters that are discussed above. This code block is likely to expand as the analysis progresses.

```
train <- train %>%
  filter(trip_duration < 22*3600,
         jfk_dist_pick < 3e5 & jfk_dist_drop < 3e5,
         trip_duration > 10)
```

4. Correlations overview

Before starting the modelling, we need to visualize the relations between our parameters using a *correlation matrix*.



We find:

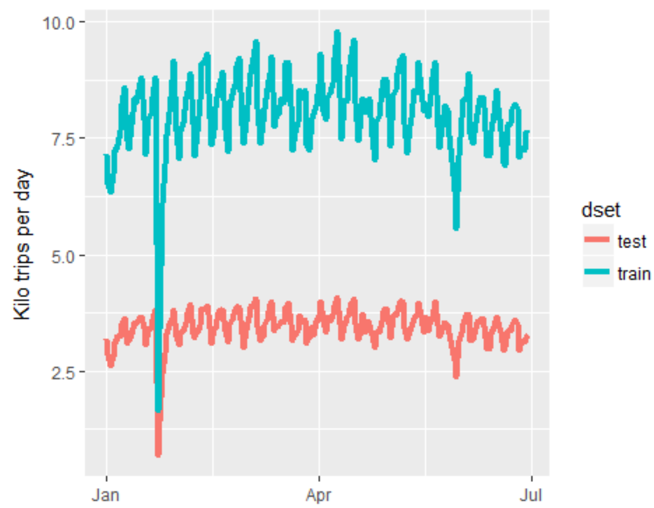
- The strongest correlations with the *trip_duration* are seen for the direct *dist*ance.
- Another effect on the *trip_duration* can be seen for our engineered features *jfk_trip* and *lg_trip*; indicating journeys to either airport. A similar statement is true for the average *speed* and airport travel.
- The pickup and dropoff coordinates are correlated.
- *Work* is correlated with *wday* and *speed.* It makes sense since we analyzed the factors influencing the speed before.

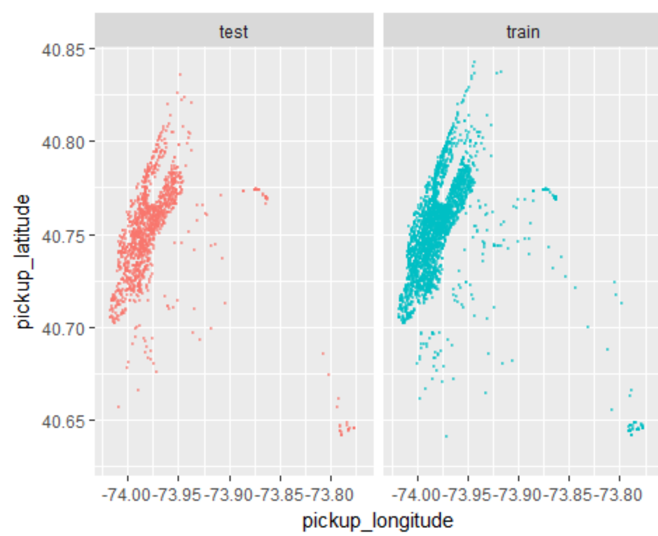
IV. Modeling

1. Preparations

a. Train vs *test* overlap

In order to make sure that we are really training on features that are relevant to our *test* data set we will now briefly compare the temporal and spatial properties of the *train* and *test* data:





We find that our *train* and *test* data sets do indeed cover the same time range and geographical area.

b. Data formatting

Here we will format the selected features to turn them into integer columns, since many classifiers cannot deal with categorical values.

Consistency check:

```
## Observations: 2,083,778
## Variables: 25
## $ 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               <fct> train, train, train, train, train, train, t...
## $ 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...
## $ 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...
```

The only non-numerical features are *id*, *pickup_datetime*, *dropoff_datetime*, and *date*, which will remove in any case, together with *dset* which we will use now to separate the *train* vs *test* again.

c. Feature selection, metric adjustment, validation split, and careful cleaning

Not all features in our data set will be useful. Here we only include meaningful variables and remove for instance the *id* feature.

We could include all features but we have engineered a couple of features from existing ones (such as *work*). Besides, we have many strongly correlated features which don't add much new information. Therefore, adding all features can cause significant *collinearity*, which will make it more difficult to interpret the result of our model in terms of the impact of individual features.

```r
# predictor features
train_cols <- c( "hour", "dist",
                "vendor_id", "jfk_trip", "lg_trip", "wday", "month",
                "pickup_longitude", "pickup_latitude", "bearing", "lg_dist_dr
op")
```

For this taxi challenge, the evaluation metric is RMSLE, the 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. (The `+ 1` is added to avoid an undefined `log(0)` and we need to remember to remove this 1 second for the prediction file).

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

Now, we will split our training data into a *train* vs *validation* data set with 80/20 fractions using a tool from the caret package.

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

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

V. Evaluation

1. XGBoost parameters and fitting

In order to predict taxi duration we're going to build a *XGBoost - eXtreme Gradient Boosting* model.

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

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

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

Now we define the meta-parameters :

```
xgb_params <- list(colsample_bytree = 0.8, #variables per tree
                   subsample = 0.8, #data subset per tree
                   booster = "gbtree",
                   max_depth = 8, #tree levels
                   eta = 0.2, #shrinkage
                   eval_metric = "rmse",
                   objective = "reg:linear",
                   seed = 4321
                   )

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

And here we *train* our classifier by using the *training* data set. To make it execute quickly, I put only 50 sample rounds.

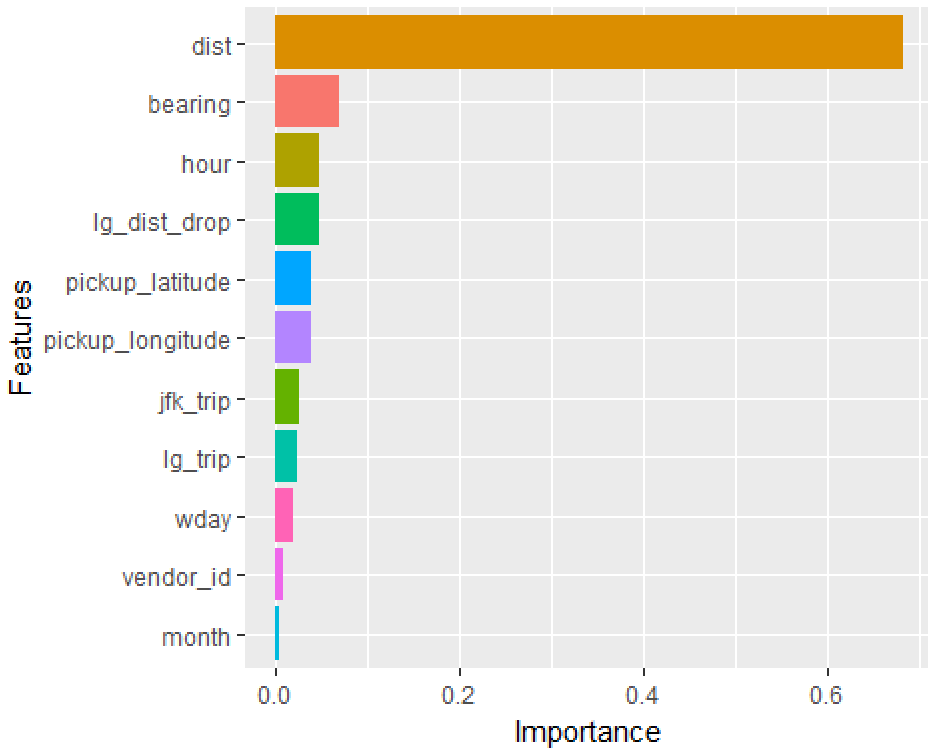
```
set.seed(4321)
gb_dt <- xgb.train(params = xgb_params,
                   data = dtrain,
                   print_every_n = 5,
                   watchlist = watchlist,
                   nrounds = 50)

## [1]  train-rmse:4.824728 valid-rmse:4.823612
## [6]  train-rmse:1.652136 valid-rmse:1.650897
## [11] train-rmse:0.681390 valid-rmse:0.679992
## [16] train-rmse:0.454404 valid-rmse:0.452948
## [21] train-rmse:0.413124 valid-rmse:0.411574
## [26] train-rmse:0.403995 valid-rmse:0.402412
## [31] train-rmse:0.398931 valid-rmse:0.397393
```

```
## [36] train-rmse:0.395693 valid-rmse:0.394067
## [41] train-rmse:0.393189 valid-rmse:0.391465
## [46] train-rmse:0.390890 valid-rmse:0.389319
## [50] train-rmse:0.389467 valid-rmse:0.387887
```

2. Feature importance

Now we can check which features are the most important for our model:



We find that ***dist*** feature is much more important than the others!

3. Prediction and submission file

After building our model, we can use it to predict the test dataset and save the result into submission file in order to get score on Kaggle.

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

pred %>% write_csv('submit.csv')
```

My first score is 0.40973. After modifying the parameters of my model by adding more trees and reducing *eta*, I get a little better score 0.39889. I think we can improve this by using an external data to get more relevant features or maybe use another strategy to explore data.