Problem Statement

At some point or the other almost each one of us has used an Ola or Uber for taking a ride.

Ride hailing services are services that use online-enabled platforms to connect between passengers and local drivers using their personal vehicles. In most cases they are a comfortable method for door-to-door transport. Usually they are cheaper than using licensed taxicabs. Examples of ride hailing services include Uber and Lyft.

To improve the efficiency of taxi dispatching systems for such services, it is important to be able to predict how long a driver will have his taxi occupied. If a dispatcher knew approximately when a taxi driver would be ending their current ride, they would be better able to identify which driver to assign to each pickup request.

In this competition, we are challenged to build a model that predicts the total ride duration of taxi trips in New York City.

1. Exploratory Data Analysis

Let's check the data files! According to the data description we should find the following columns:

- **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** - (target) duration of the trip in seconds

Here, we have 2 variables dropoff_datetime and store_and_fwd_flag which are not available before the trip starts and hence will not be used as features to the model.

1.1 Load Libraries

```
%matplotlib inline
import numpy as np
import pandas as pd
from datetime import timedelta
import datetime as dt
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

Load Data

```
df = pd.read_csv('nyc_taxi_final.zip')
```

File structure and content

```
print('We have {} rows.'.format(df.shape[0]))
print('We have {} columns'.format(df.shape[1]))
df.iloc[1,:]
```

```
We have 729322 rows.
We have 11 columns
```

```
id                              id0889885
vendor_id                               1
pickup_datetime       2016-03-11 23:35:37
dropoff_datetime      2016-03-11 23:53:57
passenger_count                         2
pickup_longitude                 -73.9883
pickup_latitude                   40.7317
dropoff_longitude                -73.9948
dropoff_latitude                  40.6949
store_and_fwd_flag                      N
trip_duration                        1100
Name: 1, dtype: object
```

At first glance, we can see the types of each variable and what they look like.

Missing Values

Knowing about missing values is important because they indicate how much we don't know about our data. Making inferences based on just a few cases is often unwise. In addition, many modelling procedures break down when missing values are involved and the corresponding rows will either have to be removed completely or the values need to be estimated somehow.

```
np.sum(pd.isnull(df))
```

```
id                    0
vendor_id             0
pickup_datetime       0
dropoff_datetime      0
passenger_count       0
pickup_longitude      0
pickup_latitude       0
```

```
dropoff_longitude     0
dropoff_latitude      0
store_and_fwd_flag    0
trip_duration         0
dtype: int64
```

Fortunately, in this dataset we do not have any missing values which is great.

Reformatting features & Checking consistency

There are a variety of features within the dataset and it is important to convert them into the right format such that we can analyse them easily. This would include converting datetime features and string features.

Also, one important thing is never to take assumptions without backing it with data. Here, as you can see the trip duration can also be calculated pick up and drop off datetime. We will check whether the given duration is consistent with the calculated trip duration

```
# converting strings to datetime features
df['pickup_datetime'] = pd.to_datetime(df.pickup_datetime)
df['dropoff_datetime'] = pd.to_datetime(df.dropoff_datetime)

# Converting yes/no flag to 1 and 0
df['store_and_fwd_flag'] = 1 * (df.store_and_fwd_flag.values == 'Y')

df['check_trip_duration'] = (df['dropoff_datetime'] -
df['pickup_datetime']).map(lambda x: x.total_seconds())

duration_difference = df[np.abs(df['check_trip_duration'].values  -
df['trip_duration'].values) > 1]
duration_difference.shape
```

```
(0, 12)
```

This implies that there is no inconsistency in data wrt the drop location and trip duration

Target Exploration

In this section we will take a look at the trip duration which is the target variable. It is crucial to understand it in detail as this is what we are trying to predict accurately.

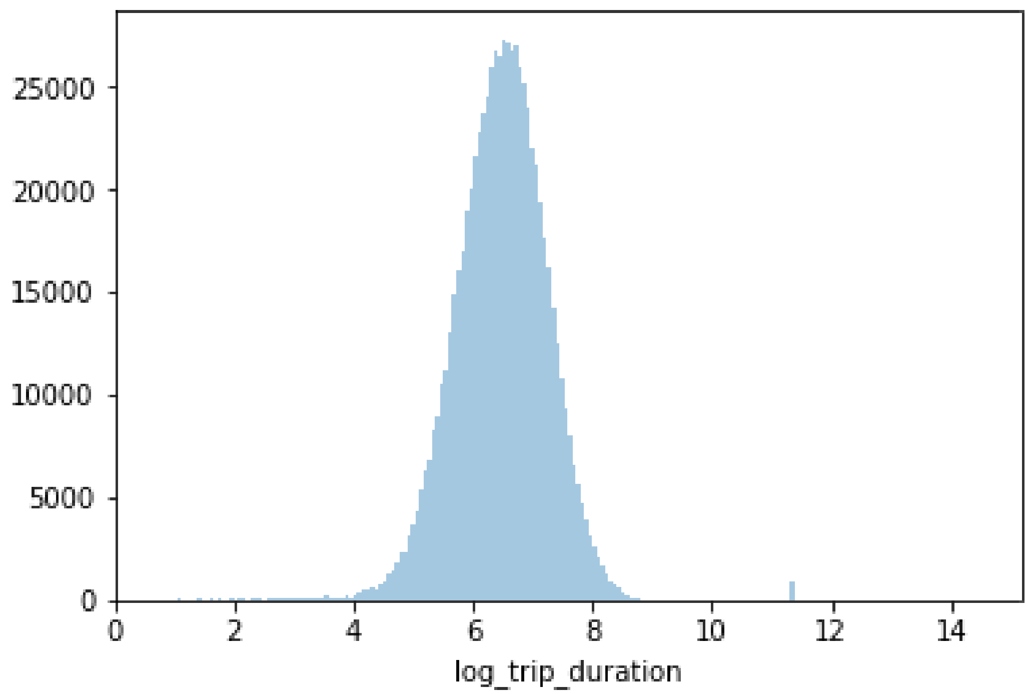
```
df['trip_duration'].describe()/3600 # Trip duration in hours
```

```
count    202.589444
mean       0.264508
std        1.073507
min        0.000278
25%        0.110278
50%        0.184167
75%        0.298611
```

```
max        538.815556
Name: trip_duration, dtype: float64
```

Woah! There is a trip with duration of 979 hours. This is a huge outlier and might create problems at the prediction stage. One idea is to log transform the trip duration before prediction to visualise it better.

```
df['log_trip_duration'] = np.log(df['trip_duration'].values + 1)
sns.distplot(df['log_trip_duration'], kde = False, bins = 200)
plt.show()
```



We find:

1. The majority of rides follow a rather smooth distribution that looks almost log-normal with a peak just around exp(6.5) i.e. about 17 minutes.
2. There are several suspiciously short rides with less than 10 seconds duration.
3. As discussed earlier, there are a few huge outliers near 12.

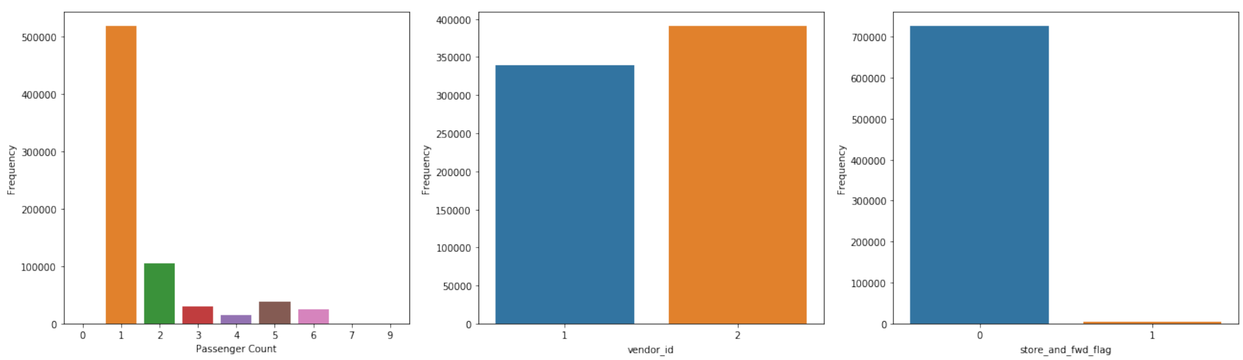
Univariate Visualization

First of all, let us look at some of the binary features. Looking at each feature might uncover some insight that might be useful at later modelling stages

```
# Binary Features
plt.figure(figsize=(22, 6))
#fig, axs = plt.subplot(ncols=2)
```

```
# Passenger Count
plt.subplot(131)
sns.countplot(df['passenger_count'])
plt.xlabel('Passenger Count')
plt.ylabel('Frequency')


# vendor_id
plt.subplot(132)
sns.countplot(df['vendor_id'])
plt.xlabel('vendor_id')
plt.ylabel('Frequency')

# store_and_fwd_flag
plt.subplot(133)
sns.countplot(df['store_and_fwd_flag'])
plt.xlabel('store_and_fwd_flag')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```



Observations:

1. Most of the trips involve only 1 passenger. There are trips with 7-9 passengers but they are very low in number.
2. Vendor 2 has more number of trips as compared to vendor 1
3. The store_and_fwd_flag values, indicating whether the trip data was sent immediately to the vendor ("0") or held in the memory of the taxi because there was no connection to the server ("1"), show that there was almost no storing taking place

Now, we will delve into the datetime features to understand the trend of number of hourly/monthly/daily taxi trips

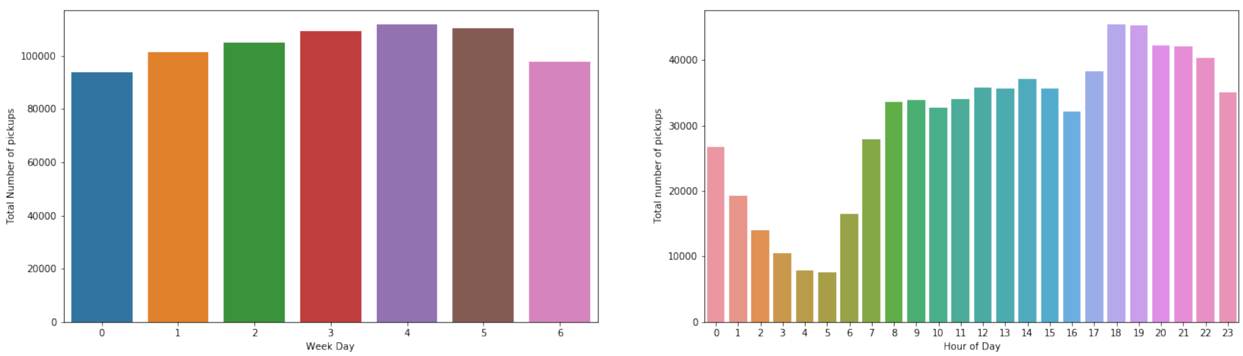
```
df['pickup_datetime'].min(), df['pickup_datetime'].max()
```

```
(Timestamp('2016-01-01 00:01:14'), Timestamp('2016-06-30 23:59:37'))
```

Clearly, These trips are for first 6 months of 2016. To look at trends, we first need to extract week days and hour of day from the pickup date.

```python
df['day_of_week'] = df['pickup_datetime'].dt.weekday
df['hour_of_day'] = df['pickup_datetime'].dt.hour


# Datetime features
plt.figure(figsize=(22, 6))

# Passenger Count
plt.subplot(121)
sns.countplot(df['day_of_week'])
plt.xlabel('Week Day')
plt.ylabel('Total Number of pickups')

# vendor_id
plt.subplot(122)
sns.countplot(df['hour_of_day'])
plt.xlabel('Hour of Day')
plt.ylabel('Total number of pickups')
```

```
Text(0, 0.5, 'Total number of pickups')
```



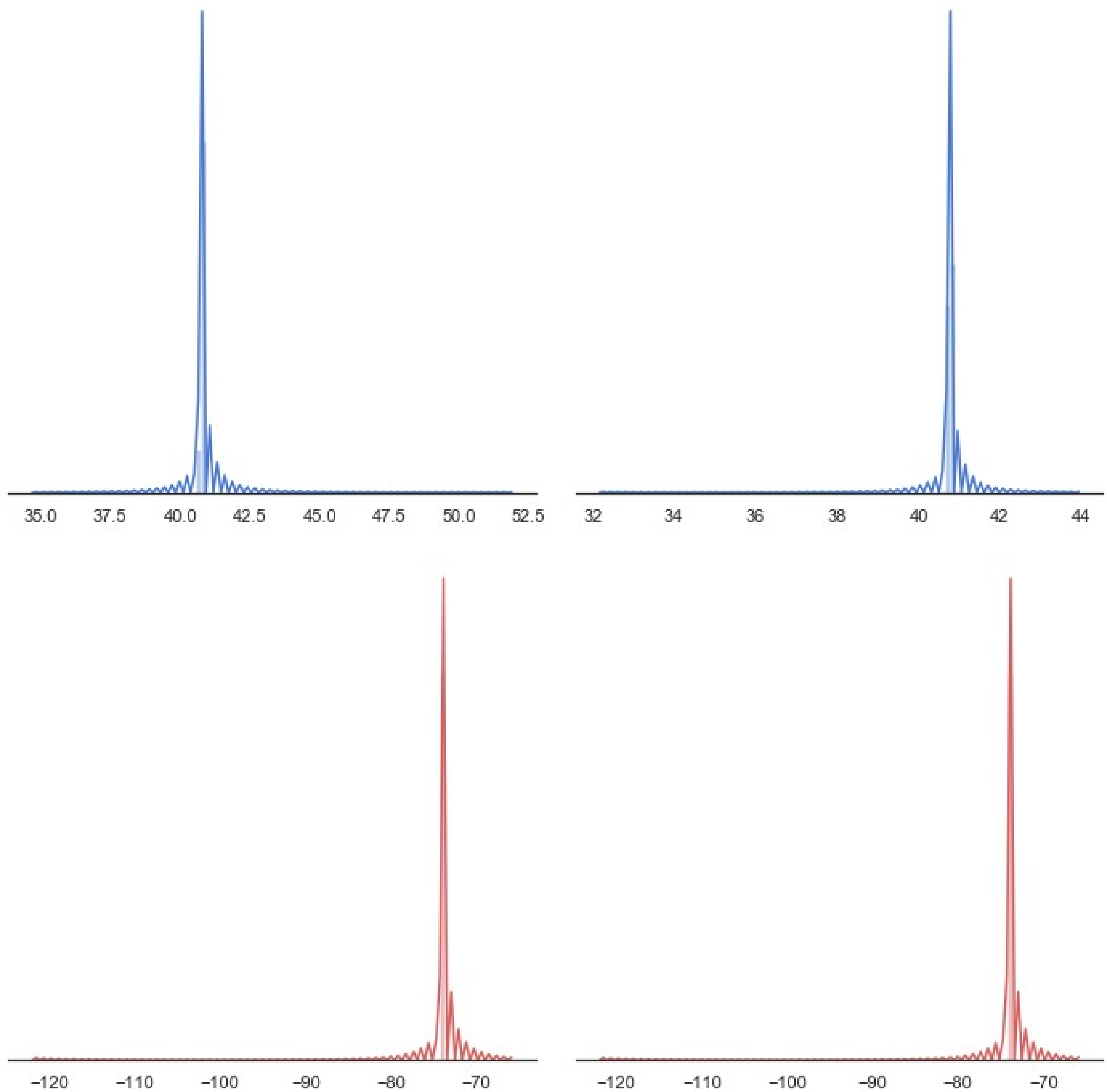
- Number of pickups for weekends is much lower than week days with a peak on Thursday (4). Note that here weekday is a decimal number, where 0 is Sunday and 6 is Saturday.
- Number of pickups as expected is highest in late evenings. However, it is much lower during the morning peak hours.

Lattitude & Longitude

Lets look at the geospatial or location features to check consistency. They should not vary much as we are only considering trips within New York city.

```python
sns.set(style="white", palette="muted", color_codes=True)
f, axes = plt.subplots(2,2,figsize=(10, 10), sharex=False, sharey = 
False)
sns.despine(left=True)
sns.distplot(df['pickup_latitude'].values, label = 
'pickup_latitude',color="b",bins = 100, ax=axes[0,0])
sns.distplot(df['pickup_longitude'].values, label = 
```

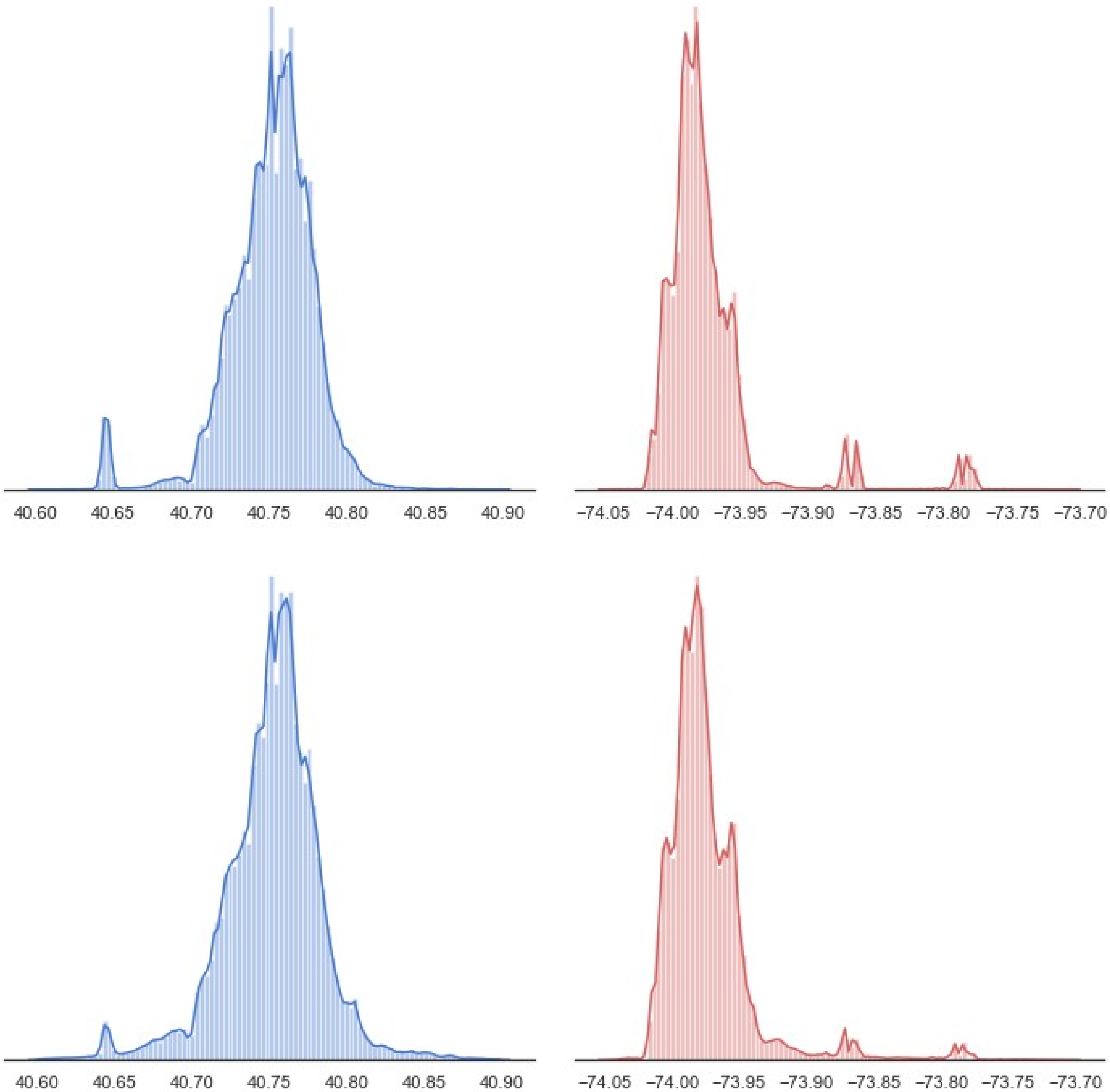
```
'pickup_longitude',color="r",bins =100, ax=axes[1,0])
sns.distplot(df['dropoff_latitude'].values, label =
'dropoff_latitude',color="b",bins =100, ax=axes[0,1])
sns.distplot(df['dropoff_longitude'].values, label =
'dropoff_longitude',color="r",bins =100, ax=axes[1,1])
plt.setp(axes, yticks=[])
plt.tight_layout()
plt.show()
```



Findings - (Here, red represents pickup and dropoff Longitudes & blue represents pickup & dropoff lattitudes)

1. From the plot above it is clear that pick and drop latitude are centered around 40 to 41, and longitude are situated around -74 to -73.
2. Some extreme co-ordinates has squeezed the plot such that we see a spike here
3. A good idea is to remove these outliers and look at the distribution more closely

```
df = df.loc[(df.pickup_latitude > 40.6) & (df.pickup_latitude < 40.9)]
df = df.loc[(df.dropoff_latitude>40.6) & (df.dropoff_latitude < 40.9)]
df = df.loc[(df.dropoff_longitude > -74.05) & (df.dropoff_longitude <
-73.7)]
df = df.loc[(df.pickup_longitude > -74.05) & (df.pickup_longitude < -
73.7)]
df_data_new = df.copy()
sns.set(style="white", palette="muted", color_codes=True)
f, axes = plt.subplots(2,2,figsize=(10, 10), sharex=False, sharey =
False)#
sns.despine(left=True)
sns.distplot(df_data_new['pickup_latitude'].values, label =
'pickup_latitude',color="b",bins = 100, ax=axes[0,0])
sns.distplot(df_data_new['pickup_longitude'].values, label =
'pickup_longitude',color="r",bins =100, ax=axes[0,1])
sns.distplot(df_data_new['dropoff_latitude'].values, label =
'dropoff_latitude',color="b",bins =100, ax=axes[1, 0])
sns.distplot(df_data_new['dropoff_longitude'].values, label =
'dropoff_longitude',color="r",bins =100, ax=axes[1, 1])
plt.setp(axes, yticks=[])
plt.tight_layout()

plt.show()
```



- We have a much better view of the distribution of coordinates instead of spikes. And we see that most trips are concentrated between these lat long only with a few significant clusters.
- These clusters are represented by the numerous peaks in the lattitude and longitude histograms

Bivariate Relations with Target

Now that we have gone through all the basic features one by one. Let us start looking at their relation with the target. This will help us in selecting and extracting features at the modelling stage.

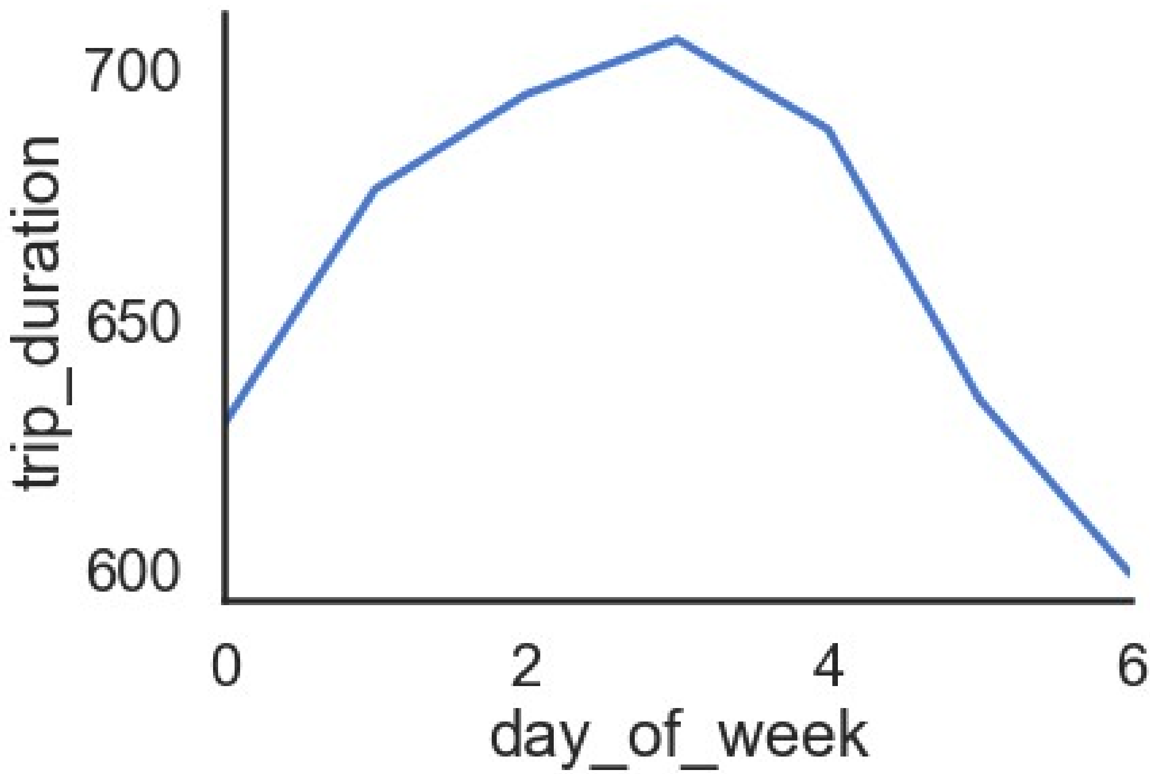
```
df.columns
```

```
Index(['id', 'vendor_id', 'pickup_datetime', 'dropoff_datetime',
       'passenger_count', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag',
       'trip_duration', 'check_trip_duration', 'log_trip_duration',
       'day_of_week', 'hour_of_day'],
      dtype='object')
```

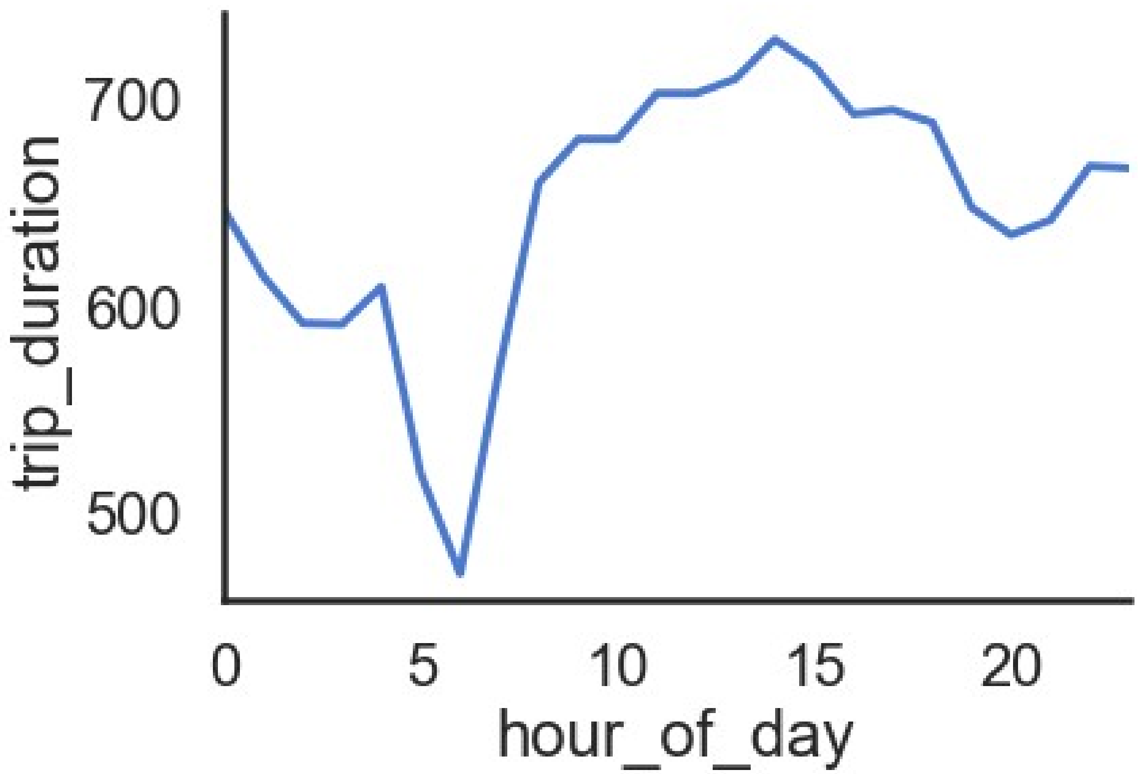
Trip Duration vs Weekday

For different week days, the trip durations can vary as different week days might have different traffic densities especially the weekends might have a much different patterns as compared to working days. Weekday is taken as a decimal number, where 0 - Sunday and 6 is Saturday.

```
summary_wdays_avg_duration = pd.DataFrame(df.groupby(['day_of_week'])
['trip_duration'].median())
summary_wdays_avg_duration.reset_index(inplace = True)
summary_wdays_avg_duration['unit']=1

sns.set(style="white", palette="muted", color_codes=True)
sns.set_context("poster")
sns.tsplot(data=summary_wdays_avg_duration, time="day_of_week", unit =
"unit", value="trip_duration")
sns.despine(bottom = False)
```




```
summary_hourly_avg_duration = pd.DataFrame(df.groupby(['hour_of_day'])
['trip_duration'].median())
summary_hourly_avg_duration.reset_index(inplace = True)
summary_hourly_avg_duration['unit']=1

sns.set(style="white", palette="muted", color_codes=True)
sns.set_context("poster")
sns.tsplot(data=summary_hourly_avg_duration, time="hour_of_day", unit
= "unit", value="trip_duration")
sns.despine(bottom = False)
```

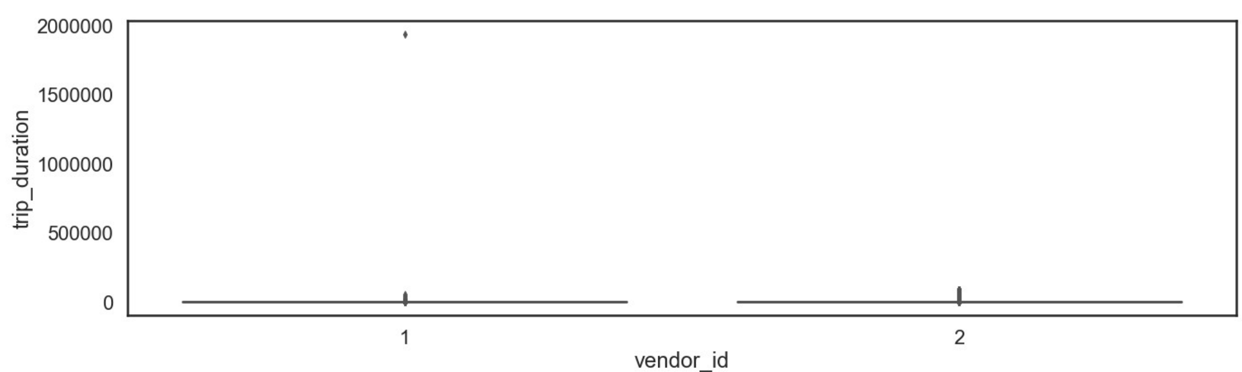


- Trip durations are definitely shorter for late night and early morning hours that can be attributed to low traffic density
- It follows a similar pattern when compared to number of pickups indicating a correlation between number of pickups and trip duration

vendor_id vs Trip Duration

Let's check how the trip duration varies for different vendors.

```
plt.figure(figsize=(22, 6))
sns.boxplot(x="vendor_id", y="trip_duration", data=df)
plt.show()
```



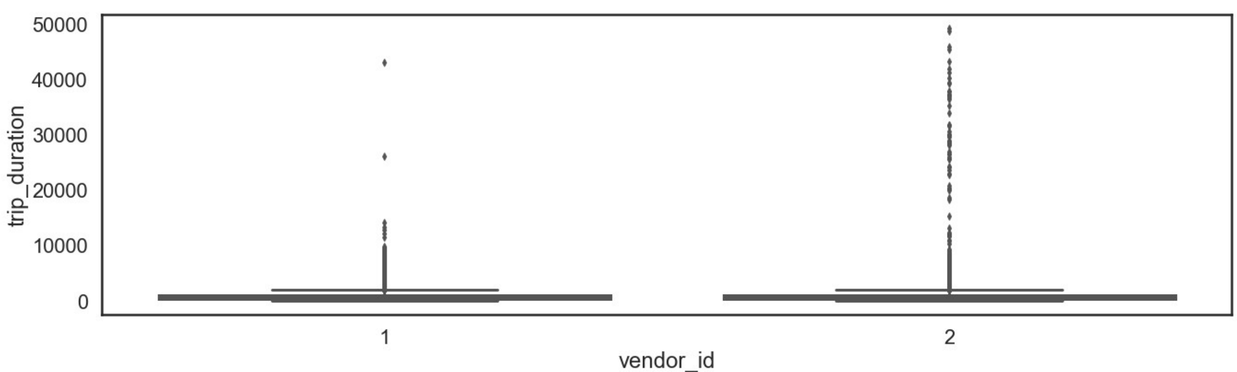
Woah! This did not came out as expected. The only thing I can see from this boxplot is that for vendor 2, there are a number of outliers exceeding 24 hours while vendor 1 does not have such long trips.

There could be 2 solutions to this:

1. Remove the huge outliers and plot again
2. Look at median trip duration for both vendors on hourly basis

Let's try the first technique now and check trips below 50000 seconds only

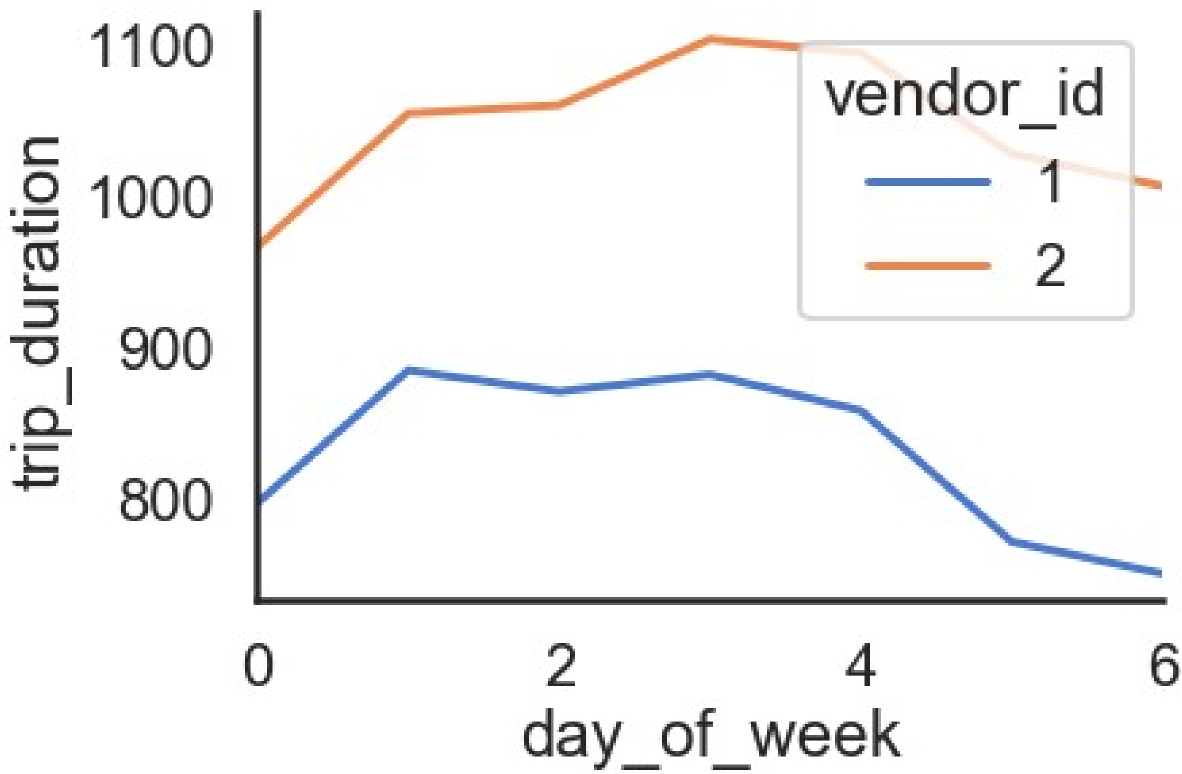
```
plt.figure(figsize=(22, 6))
df_sub = df[df['trip_duration'] < 50000]
sns.boxplot(x="vendor_id", y="trip_duration", data=df_sub)
plt.show()
```



As you can see, we were in a false perception earlier that vendor 1 had more outliers. Since the median is just around 600 seconds, we observe that vendor 2 has many more outliers as compared to vendor 1. Next, to confirm this, we will quickly look at the mean wrt day of week for both vendors using tsplot (time series plot) from seaborn.

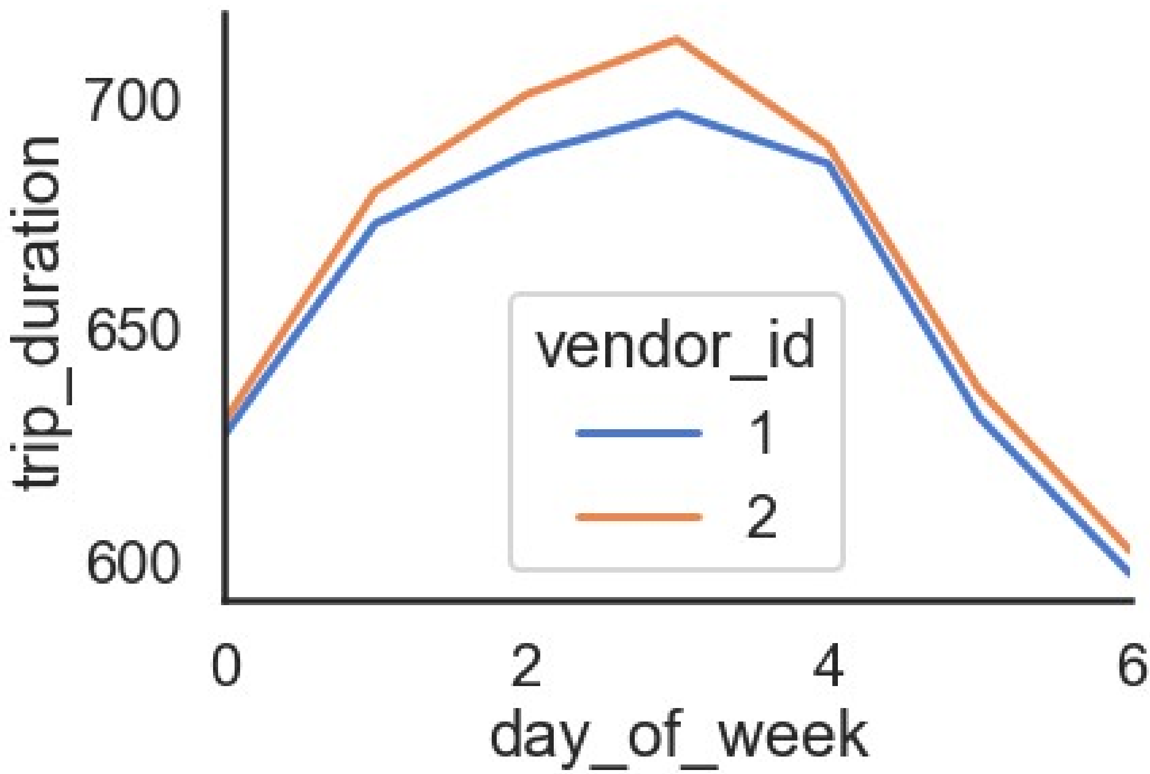
Mean Trip Duration Vendor Wise

```
summary_wdays_avg_duration =
pd.DataFrame(df.groupby(['vendor_id','day_of_week'])
['trip_duration'].mean())
summary_wdays_avg_duration.reset_index(inplace = True)
summary_wdays_avg_duration['unit']=1

sns.set(style="white", palette="muted", color_codes=True)
sns.set_context("poster")
sns.tsplot(data=summary_wdays_avg_duration, time="day_of_week", unit =
"unit", condition="vendor_id", value="trip_duration")
sns.despine(bottom = False)
```



Median Trip Duration Vendor Wise

```
summary_wdays_avg_duration =
pd.DataFrame(df.groupby(['vendor_id','day_of_week'])
['trip_duration'].median())
summary_wdays_avg_duration.reset_index(inplace = True)
summary_wdays_avg_duration['unit']=1

sns.set(style="white", palette="muted", color_codes=True)
sns.set_context("poster")
sns.tsplot(data=summary_wdays_avg_duration, time="day_of_week", unit =
"unit", condition="vendor_id", value="trip_duration")
sns.despine(bottom = False)
```



Median trip duration does not vary much as can be seen from the above plot for different vendors. It emphasises the importance of looking at the correct measure for central tendency for analysis.

Trip Duration vs Passenger Count

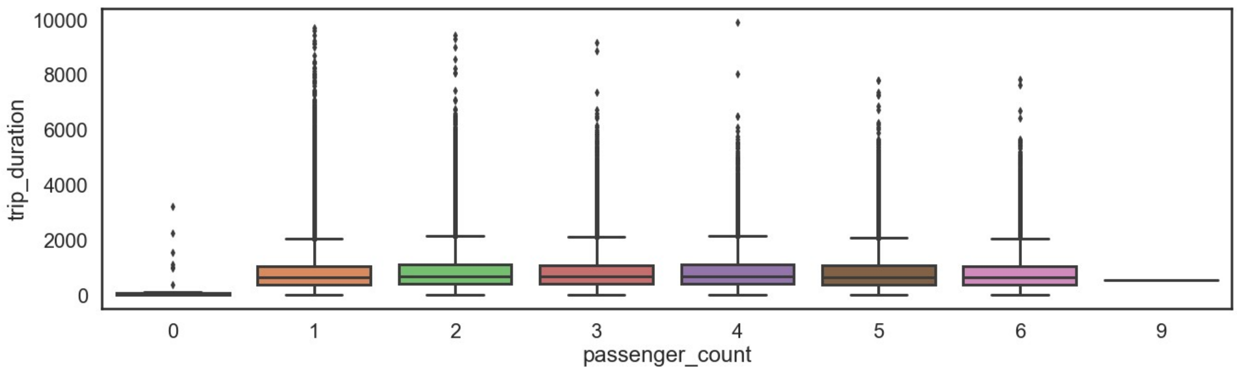
Again as we are aware, there are a large number of outliers for trip duration and we will not be able to observe the differences. For this, we have taken a cutoff of 10000 seconds and used a boxplot.

```
df.passenger_count.value_counts()
```

```
1    515243
2    104576
5     38776
3     29561
6     24035
4     13972
0        31
9         1
Name: passenger_count, dtype: int64
```

```
df.passenger_count.value_counts()
plt.figure(figsize=(22, 6))
df_sub = df[df['trip_duration'] < 10000]
```

```
sns.boxplot(x="passenger_count", y="trip_duration", data=df_sub)
plt.show()
```



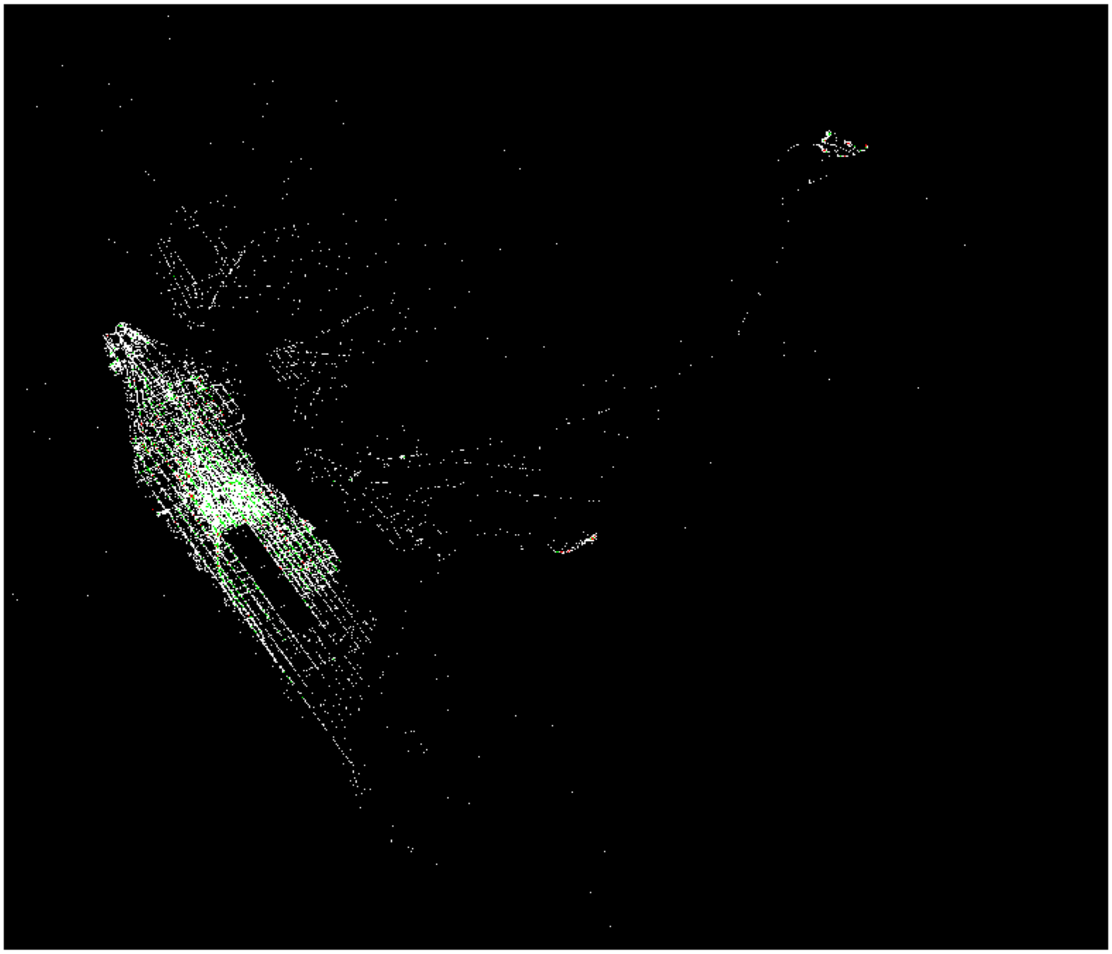
- The boxplot clearly shows that there not much of a difference in distribution for the most frequently occuring passenger count values - 1, 2, 3.
- Another key observation is that the number of outliers are reduced for higher passenger counts but that only comes down to the individual frequencies of each passenger count.

Visualise most frequently occuring Pickup points on the lattitude-longitude Map

Here, we try to visualise the most frequently occuring pickup points on the map and check how it is distributed spatially.

```
rgb = np.zeros((3000, 3500, 3), dtype=np.uint8)
rgb[..., 0] = 0
rgb[..., 1] = 0
rgb[..., 2] = 0
df_data_new['pick_lat_new'] = list(map(int, (df['pickup_latitude'] -
(40.6000))*10000))
df_data_new['drop_lat_new'] = list(map(int, (df['dropoff_latitude'] -
(40.6000))*10000))
df_data_new['pick_lon_new'] = list(map(int, (df['pickup_longitude'] -
(-74.050))*10000))
df_data_new['drop_lon_new'] = list(map(int,(df['dropoff_longitude'] -
(-74.050))*10000))

summary_plot = pd.DataFrame(df_data_new.groupby(['pick_lat_new',
'pick_lon_new'])['id'].count())

summary_plot.reset_index(inplace = True)
summary_plot.head(120)
lat_list = summary_plot['pick_lat_new'].unique()
for i in lat_list:
    lon_list = summary_plot.loc[summary_plot['pick_lat_new']==i]
```

```
['pick_lon_new'].tolist()
    unit = summary_plot.loc[summary_plot['pick_lat_new']==i]
['id'].tolist()
    for j in lon_list:
        a = unit[lon_list.index(j)]
        if (a//25) >0:
            rgb[i][j][0] = 255
            rgb[i,j, 1] = 0
            rgb[i,j, 2] = 0
        elif (a//10)>0:
            rgb[i,j, 0] = 0
            rgb[i,j, 1] = 255
            rgb[i,j, 2] = 0
        else:
            rgb[i,j, 0] = 255
            rgb[i,j, 1] = 255
            rgb[i,j, 2] = 255
fig, ax = plt.subplots(nrows=1,ncols=1,figsize=(14,20))
ax.imshow(rgb, cmap = 'hot')
ax.set_axis_off()
```



Findings - From the heatmap kind of image above -

- White points - 1-10 trips have white as pickup point
- Green points - 10-25 trips have green as pickup point
- Red points - More than 25 trips have red as pickup point

As expected there are a few small clusters for hot pickup points as displayed by red in the above plot. Most pickup points have less than 10 trips and distributed all over the city.

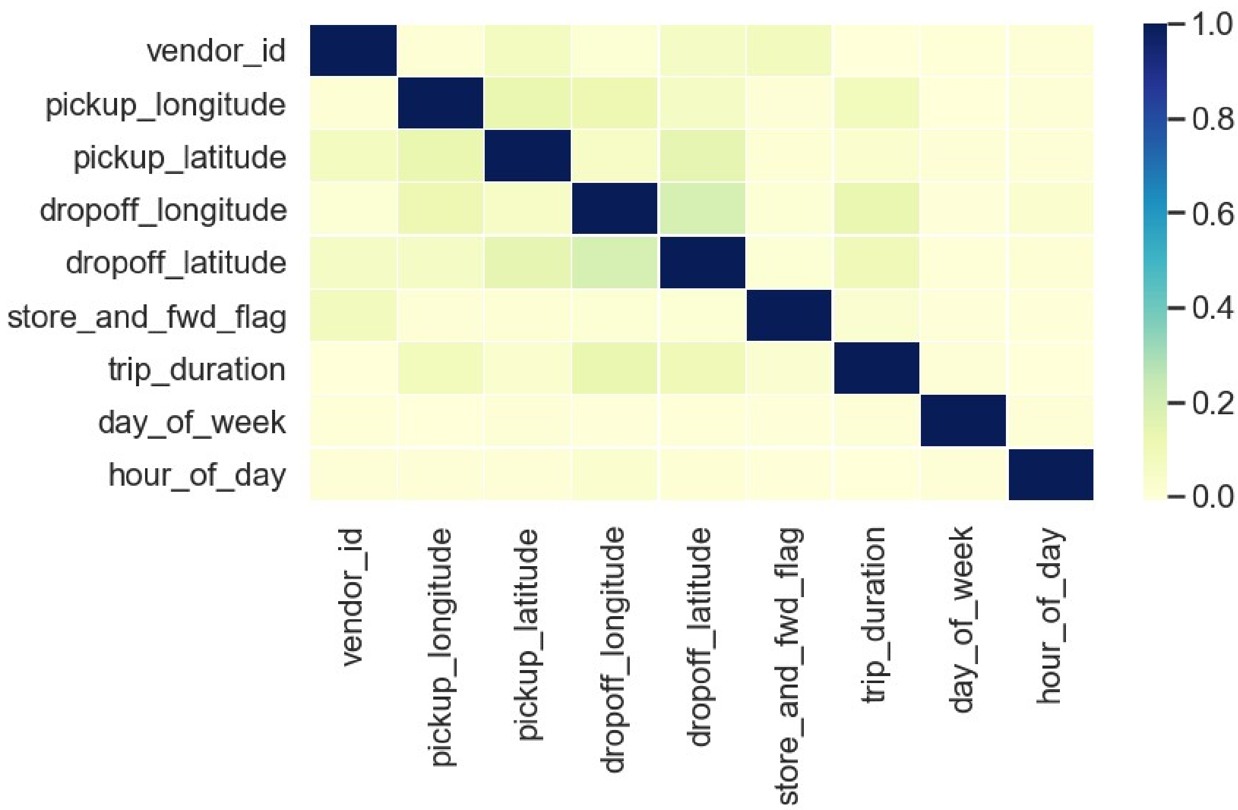
If you go and have a look at an actual map of New York City, red and green points are mostly concentrated around the Manhatten Area

Correlation Heatmap

Let us quickly look at the correlation heatmap to check the correlations amongst all features.

```
plt.figure(figsize=(12, 6))
df = df.drop(['id', 'pickup_datetime', 'dropoff_datetime',
        'passenger_count','check_trip_duration','log_trip_duration'],
```

```
        axis=1)
corr = df.apply(lambda x: pd.factorize(x)[0]).corr()
ax = sns.heatmap(corr, xticklabels=corr.columns,
yticklabels=corr.columns,
                 linewidths=.2, cmap="YlGnBu")
```



Conclusions

1. The majority of rides follow a rather smooth distribution that looks almost log-normal with a peak just around exp(6.5) i.e. about 17 minutes.
2. There are several suspiciously short rides with less than 10 seconds duration.
3. As discussed earlier, there are a few huge outliers near 12.
4. Most of the trips involve only 1 passenger. There are trips with 7-9 passengers but they are very low in number.
5. Vendor 2 has more number of trips as compared to vendor 1
6. Number of pickups for weekends is much lower than week days with a peak on Thursday (4). Note that here weekday is a decimal number, where 0 is Sunday and 6 is Saturday.
7. Number of pickups as expected is highest in late evenings. However, it is much lower during the morning peak hours.
8. We see that most trips are concentrated between these lat long only with a few significant clusters. These clusters are represented by the numerous peaks in the lattitude and longitude histograms
9. Trip durations are definitely shorter for late night and early morning hours that can be attributed to low traffic density

10. It follows a similar pattern when compared to number of pickups indicating a correlation between number of pickups and trip duration
11. Median trip duration does not vary much as can be seen from the above plot for different vendors.
12. The boxplot clearly shows that there not much of a difference in distribution for the most frequently occuring passenger count values - 1, 2, 3.
13. Another key observation is that the number of outliers are reduced for higher passenger counts but that only comes down to the individual frequencies of each passenger count.
14. From the correlation heatmap we see that the lattitude and longitude features have higher correlation with the target as compared to the other features.

NYC Taxi Trip Duration Feature Engineering & Model Building

Now, that we have seen the Exploration and have a good understanding of data shape and structure. We also looked at firstly the basic models such as decision tree and linear regression and later on ensemble methods such as random forest and XGBoost (Gradient Boosting).

But the model is as good as the training data. Can we engineer new features to improve performance? Let's find out

Data Dictionary

It is always a good idea to have the data dictionary handy.

- **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

Here dropoff_datetime and trip_duration are only available for the train set as that represents the target

Load Libraries

We will load libraries required to build models and validation sets

```
%matplotlib inline
import numpy as np
import pandas as pd
import datetime as dt
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings('ignore')

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsRegressor
```

Load Data

```
df = pd.read_csv('nyc_taxi_final.zip')
```

Preprocessing & Feature Extraction

As is clear from the previous modules, we can only feed numeric features as input to our models. So our next task is to convert the features in numeric form. It is time to jump into getting our data ready for feeding into the model but before that it is important to use the variables to do some feature engineering as t

Some of my ideas to create new variables and the reasons are as follows

- Difference between pickup and dropoff latitude - will give an idea about the distance covered which could be predictive
- Difference between pickup and dropoff longitude - same reason as above
- Haversine distance between pickup and dropoff co-ordinates - to capture the actual distance travelled
- Pickup minute - since pickup hour is an important variable, the minute of pickup might well have been predictive
- Pickup day of year - same reason as above

DateTime Conversion

The datetime features from csv files are read as strings and in order to easily extract features like day of week, month, year etc. we need to convert it into datetime format of python.

```
# converting strings to datetime features
df['pickup_datetime'] = pd.to_datetime(df.pickup_datetime)
df['dropoff_datetime'] = pd.to_datetime(df.dropoff_datetime)
```

```
# Log transform the Y values
df_y = np.log1p(df['trip_duration'])

# Add some datetime features
df.loc[:, 'pickup_weekday'] = df['pickup_datetime'].dt.weekday
df.loc[:, 'pickup_hour_weekofyear'] =
df['pickup_datetime'].dt.weekofyear
df.loc[:, 'pickup_hour'] = df['pickup_datetime'].dt.hour
df.loc[:, 'pickup_minute'] = df['pickup_datetime'].dt.minute
df.loc[:, 'pickup_dt'] = (df['pickup_datetime'] -
df['pickup_datetime'].min()).dt.total_seconds()
df.loc[:, 'pickup_week_hour'] = df['pickup_weekday'] * 24 +
df['pickup_hour']
```

Distance Features

As discussed earlier, distance features must be important and must be included here

Eucledian Distance

Let's Calculate the Eucledian distance between pickup and drop off location to get some idea on how far the pickup and dropoff points are since this would definitely impact the trip duration even though we know that cars can't fly

```
#displacement
y_dist= df['pickup_longitude'] - df['dropoff_longitude']
x_dist = df['pickup_latitude'] - df['dropoff_latitude']

#square distance
df['dist_sq'] = (y_dist ** 2) + (x_dist ** 2)

#distance
df['dist_sqrt'] = df['dist_sq'] ** 0.5
```

Haversine Distance

Let's calculate the distance (km) between pickup and dropoff points. The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. We will also calculate the approximate angle at which the dropoff location lies wrt the pickup location. pd.DataFrame.apply() would be too slow so the haversine function is rewritten to handle arrays.

Haversine direction represents the information of angle of the line connecting the dropoff and pickup point over the surface of earth wrt equator.

```
def haversine_array(lat1, lng1, lat2, lng2):
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    AVG_EARTH_RADIUS = 6371  # in km
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) *
np.sin(lng * 0.5) ** 2
    h = 2 * AVG_EARTH_RADIUS * np.arcsin(np.sqrt(d))
    return h

def direction_array(lat1, lng1, lat2, lng2):
    AVG_EARTH_RADIUS = 6371  # in km
    lng_delta_rad = np.radians(lng2 - lng1)
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2) *
np.cos(lng_delta_rad)
    return np.degrees(np.arctan2(y, x))


df['haversine_distance'] =
haversine_array(df['pickup_latitude'].values,

df['pickup_longitude'].values,

df['dropoff_latitude'].values,

df['dropoff_longitude'].values)


df['direction'] = direction_array(df['pickup_latitude'].values,

df['pickup_longitude'].values,

df['dropoff_latitude'].values,

df['dropoff_longitude'].values)
```

Fastest route by road

Sometimes, adding external information can be crucial for improving the model. Here we will use data extracted from The Open Source Routing Machine or OSRM for each trip in our original dataset. OSRM is a C++ implementation of a high-performance routing engine for shortest paths in road networks. This will give us a very good estimate of distances between pickup and dropoff Points

Source: http://project-osrm.org/

```
fr1 = pd.read_csv('osrm/fastest_routes_train_part_1.zip',
                  usecols=['id', 'total_distance',
```

```
'total_travel_time'])
fr2 = pd.read_csv('osrm/fastest_routes_train_part_2.zip',
                  usecols=['id', 'total_distance',
'total_travel_time'])

df_street_info = pd.concat((fr1, fr2))
df = df.merge(df_street_info, how='left', on='id')

df_street_info.head()
```

	id	total_distance	total_travel_time
0	id2875421	2009.1	164.9
1	id2377394	2513.2	332.0
2	id3504673	1779.4	235.8
3	id2181028	1614.9	140.1
4	id0801584	1393.5	189.4

Binning

The lattitude and longitude could be a bit noisy and it might be a good idea to bin them and create new features after rounding their values.

```
### Binned Coordinates ###
df['pickup_latitude_round3'] = np.round(df['pickup_latitude'],3)
df['pickup_longitude_round3'] = np.round(df['pickup_longitude'],3)

df['dropoff_latitude_round3'] = np.round(df['dropoff_latitude'],3)
df['dropoff_longitude_round3'] = np.round(df['dropoff_longitude'],3)
```

Other Features

One Hot Encoding

Here, Vendor ID can be converted to one hot encoding or frequency encoding since in the raw data it has values 1 and 2 without any inherent order.

```
df.vendor_id.value_counts()

2    390481
1    338841
Name: vendor_id, dtype: int64
```

Now, there is not much difference in the frequencies of both and that might not make for an important feature. so we will just convert it to 0 and 1 by subtracting 1 from it

```
df['vendor_id'] = df['vendor_id'] - 1

np.sum(pd.isnull(df))
```

```
id                             0
vendor_id                      0
pickup_datetime                0
dropoff_datetime               0
passenger_count                0
pickup_longitude               0
pickup_latitude                0
dropoff_longitude              0
dropoff_latitude               0
store_and_fwd_flag             0
trip_duration                  0
pickup_weekday                 0
pickup_hour_weekofyear         0
pickup_hour                    0
pickup_minute                  0
pickup_dt                      0
pickup_week_hour               0
dist_sq                        0
dist_sqrt                      0
haversine_distance             0
direction                      0
total_distance                 1
total_travel_time              1
pickup_latitude_round3         0
pickup_longitude_round3        0
dropoff_latitude_round3        0
dropoff_longitude_round3       0
dtype: int64

# For a route, the total distance and travel time are not available.
Let's impute that with 0
df.fillna(0, inplace = True)
```

Before we go on to build a model, we must drop the variables that should not be fed as features to the algorithms. We will drop

- id - Uniquely represents a sample in the train set
- pickup_datetime - Since we have extracted the datetime features, there is no need to keep the datetime column
- dropoff_datetime - If this is used to create features, it would be a leakage and we will get perfect model performance. Why? The time gap between dropoff_datetime and pickup_datetime is essentially what we are trying to predict
- trip_duration - This is the target variable so needs to be dropped
- store_and_fwd_flag - This variable is not available before the start of the trip and should not be used for modelling.

```
df = df.drop(['id', 'pickup_datetime', 'dropoff_datetime',
'trip_duration','store_and_fwd_flag'], axis=1)
```

Model Building

Now, before we go on to build the model, let us look at the dataset.

```
df.head()

   vendor_id  passenger_count  pickup_longitude  pickup_latitude  \
0          1                1        -73.953918        40.778873
1          0                2        -73.988312        40.731743
2          1                2        -73.997314        40.721458
3          1                6        -73.961670        40.759720
4          0                1        -74.017120        40.708469

   dropoff_longitude  dropoff_latitude  pickup_weekday  \
0         -73.963875         40.771164               0
1         -73.994751         40.694931               4
2         -73.948029         40.774918               6
3         -73.956779         40.780628               1
4         -73.988182         40.740631               2

   pickup_hour_weekofyear  pickup_hour  pickup_minute  \
0                       9           16             40
1                      10           23             35
2                       7           17             59
3                       1            9             44
4                       7            6             42

             ...              dist_sq  dist_sqrt
haversine_distance  \
0            ...             0.000159   0.012592            1.199073

1            ...             0.001397   0.037371            4.129111

2            ...             0.005287   0.072712            7.250753

3            ...             0.000461   0.021473            2.361097

4            ...             0.001872   0.043264            4.328534


    direction  total_distance  total_travel_time
pickup_latitude_round3  \
0 -135.634530          1630.9              172.5
40.779
1 -172.445217          5428.7              581.8
40.732
2   34.916093          9327.8              748.9
40.721
3   10.043567          8022.7              612.2
40.760
```

```
4   34.280582          5468.7              645.0
40.708

   pickup_longitude_round3  dropoff_latitude_round3
dropoff_longitude_round3
0                     -73.954                   40.771
-73.964
1                     -73.988                   40.695
-73.995
2                     -73.997                   40.775
-73.948
3                     -73.962                   40.781
-73.957
4                     -74.017                   40.741
-73.988

[5 rows x 22 columns]
```

We have all numerical data types in our dataset now. Time to delve into model building. A very simple baseline could just be the mean of the values in the train set. Let's check the performance on that.

Defining Metric

```
from sklearn.metrics import mean_squared_error
from math import sqrt
```

Test Train Split

We have all numbers in our dataset now. Time to delve into model building. But before that, we need to finalise a validation strategy to create the train and test sets. Here, we will do a random split and keep one third of the data in test set and remaining two third of data in the train set

```
#Splitting the data into Train and Validation set
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(df,df_y,test_size=1/3,
random_state=0)
```

Mean Prediction

Before we go on to try any machine learning model, let us look at the performance of a basic model that just says the mean of trip duration in the train set is the prediction for all the trips in the test set.

```
mean_pred = np.repeat(ytrain.mean(),len(ytest))

sqrt(mean_squared_error(ytest, mean_pred))

0.7986672307875027
```

Cross validation

Cross Validation is one of the most important concepts in any type of data modelling. It simply says, try to leave a sample on which you do not train the model and test the model on this sample before finalizing the model.

we divide the entire population into k equal samples. Now we train models on k-1 samples and validate on 1 sample. Then, at the second iteration we train the model with a different sample held as validation.

In k iterations, we have basically built model on each sample and held each of them as validation. This is a way to reduce the selection bias and reduce the variance in prediction power.

```
def cv_score(ml_model, rstate = 11,cols = df.columns):
    i = 1
    cv_scores = []
    df1 = df.copy()
    df1 = df[cols]

    kf = KFold(n_splits=5,random_state=rstate,shuffle=True)
    for train_index,test_index in kf.split(df1,df_y):
        print('\n{} of kfold {}'.format(i,kf.n_splits))
        xtr,xvl = df1.loc[train_index],df1.loc[test_index]
        ytr,yvl = df_y[train_index],df_y[test_index]

        model = ml_model
        model.fit(xtr, ytr)
        train_val = model.predict(xtr)
        pred_val = model.predict(xvl)
        rmse_score_train = sqrt(mean_squared_error(ytr, train_val))
        rmse_score = sqrt(mean_squared_error(yvl, pred_val))
        sufix = ""
        msg = ""
        #msg += "Train RMSE: {:.5f} ".format(rmse_score_train)
        msg += "Valid RMSE: {:.5f}".format(rmse_score)
        print("{}".format(msg))
        # Save scores
        cv_scores.append(rmse_score)
        i+=1
    return cv_scores
```

Linear Regression

Lets begin by using the simplest regression algorithm Linear regression to check the performance.

```
linreg_scores = cv_score(LinearRegression())
```

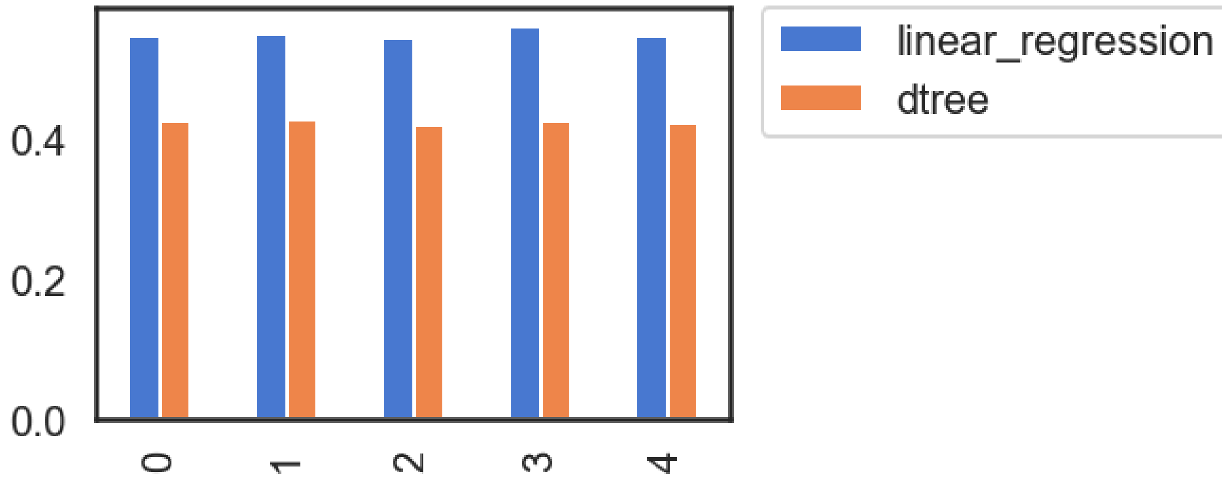
```
1 of kfold 5
```

```
Valid RMSE: 0.54881

2 of kfold 5
Valid RMSE: 0.54975

3 of kfold 5
Valid RMSE: 0.54520

4 of kfold 5
Valid RMSE: 0.56114

5 of kfold 5
Valid RMSE: 0.54677
```

We can already see that the performance of even linear regression has improved a lot. This demonstrates the power of feature engineering. Let's try decision tree once more and check performance

Decision Tree

```
dtree_scores = cv_score(DecisionTreeRegressor(min_samples_leaf=25,
min_samples_split=25))


1 of kfold 5
Valid RMSE: 0.42677

2 of kfold 5
Valid RMSE: 0.42935

3 of kfold 5
Valid RMSE: 0.41977

4 of kfold 5
Valid RMSE: 0.42661

5 of kfold 5
Valid RMSE: 0.42153

results_df = pd.DataFrame({'linear_regression':linreg_scores, 'dtree':
dtree_scores})

results_df.plot(y=["linear_regression", "dtree"], kind="bar",
legend=False)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



Woah! That's a lot of improvement. The reason for this could be the non linear relationship between the trip duration values and the location coordinates of pickup and dropoff points.

Decision Tree Visualization

```
from sklearn import tree

dtree = DecisionTreeRegressor(min_samples_leaf=25,
min_samples_split=25)
dtree.fit(xtrain, ytrain)

DecisionTreeRegressor(criterion='mse', max_depth=None,
max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=25,
           min_samples_split=25, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')

decision_tree =
tree.export_graphviz(dtree,out_file='tree.dot',feature_names=xtrain.co
lumns,max_depth=2,filled=True)
!dot -Tpng tree.dot -o tree.png
```

Now, let us view the decision tree till depth 2 and find out the features at the root and the first node.

As is clear from the above decision tree the extra features added are adding a lot of value to our decision tree learning indicating that the additional features carry good value and are very important to the model.

Looking at this, seems like a good idea to try more advanced decision tree based techniques which we will look at in the next section.