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

In [90]:
```
%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

In [91]:
```
df = pd.read_csv('nyc_taxi_trip_duration.csv')
```

File structure and content

In [92]:
```
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
```

Out[92]:
```
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.

In [93]:
```
np.sum(pd.isnull(df))
```

Out[93]:
```
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

In [94]:
```
# 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
```

Out[94]:
```
(0, 12)
```

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

Target Exploration

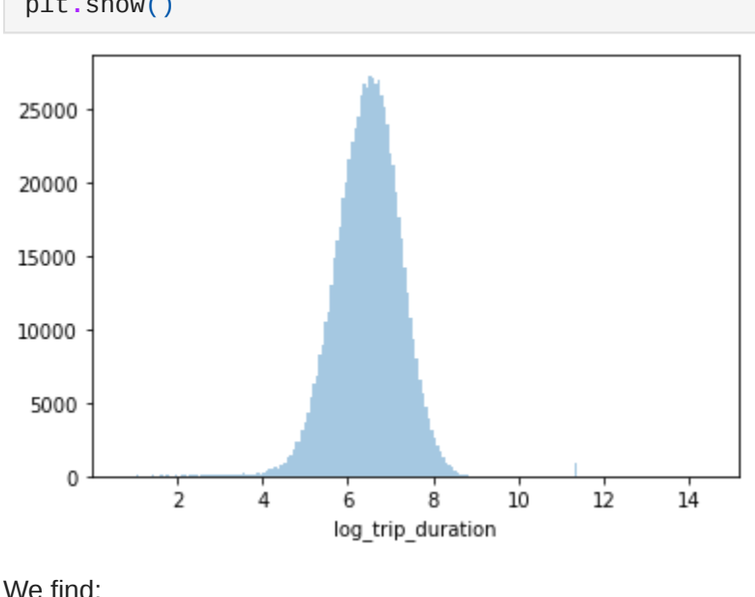
In this section we will take a look at the trip duration which is the target variable. It is crucial to understand it in detail as this is what we are trying to predict accurately.

In [95]:
```
df['trip_duration'].describe()/3600 # Trip duration in hours
```

Out[95]:
```
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.

In [96]:
```
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

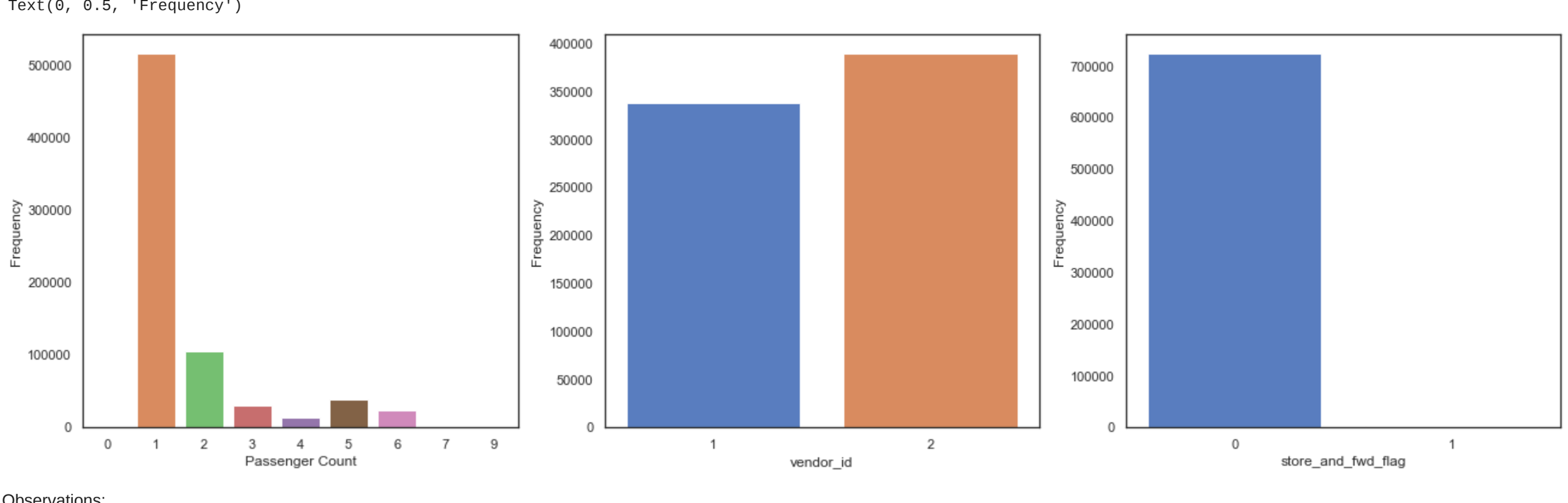
In [63]:
```
# Binary Features
plt.figure(figsize=(22, 6))
#fig, axs = plt.subplot(ncols=2)

# 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')
```

Out[63]:
```
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

In [97]:
```
df['pickup_datetime'].min(), df['pickup_datetime'].max()
```

Out[97]:
```
(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.

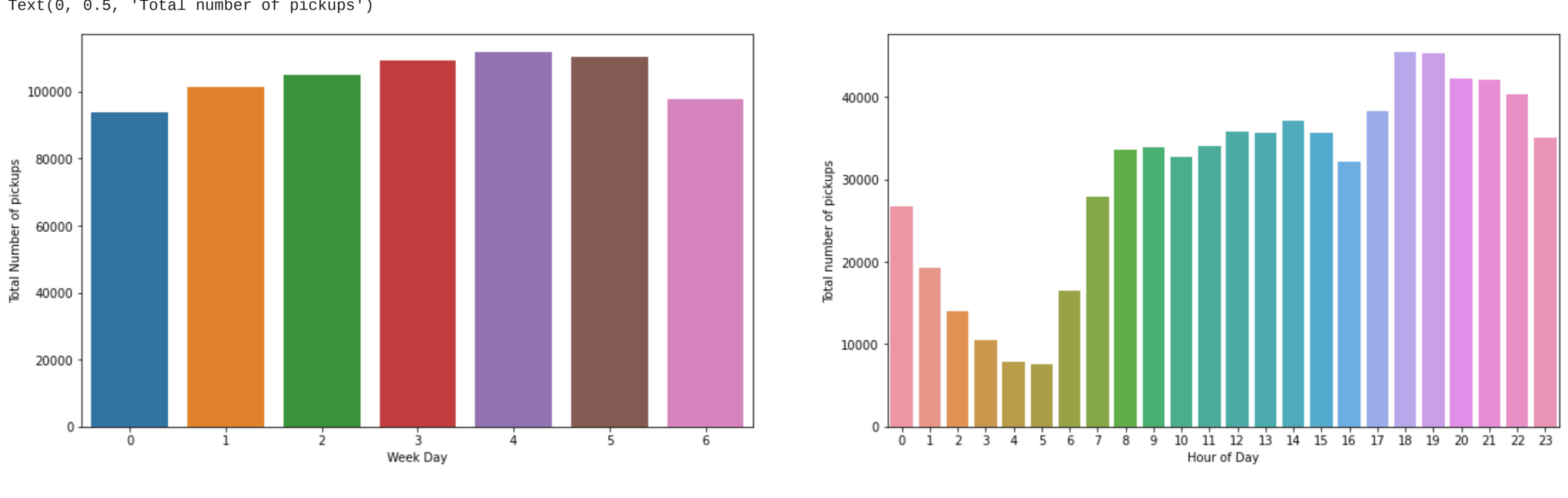
In [98]:
```
df['day_of_week'] = df['pickup_datetime'].dt.weekday
df['hour_of_day'] = df['pickup_datetime'].dt.hour
```

In [99]:
```
# 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')
```

Out[99]:
```
Text(0, 0.5, 'Total number of pickups')
```

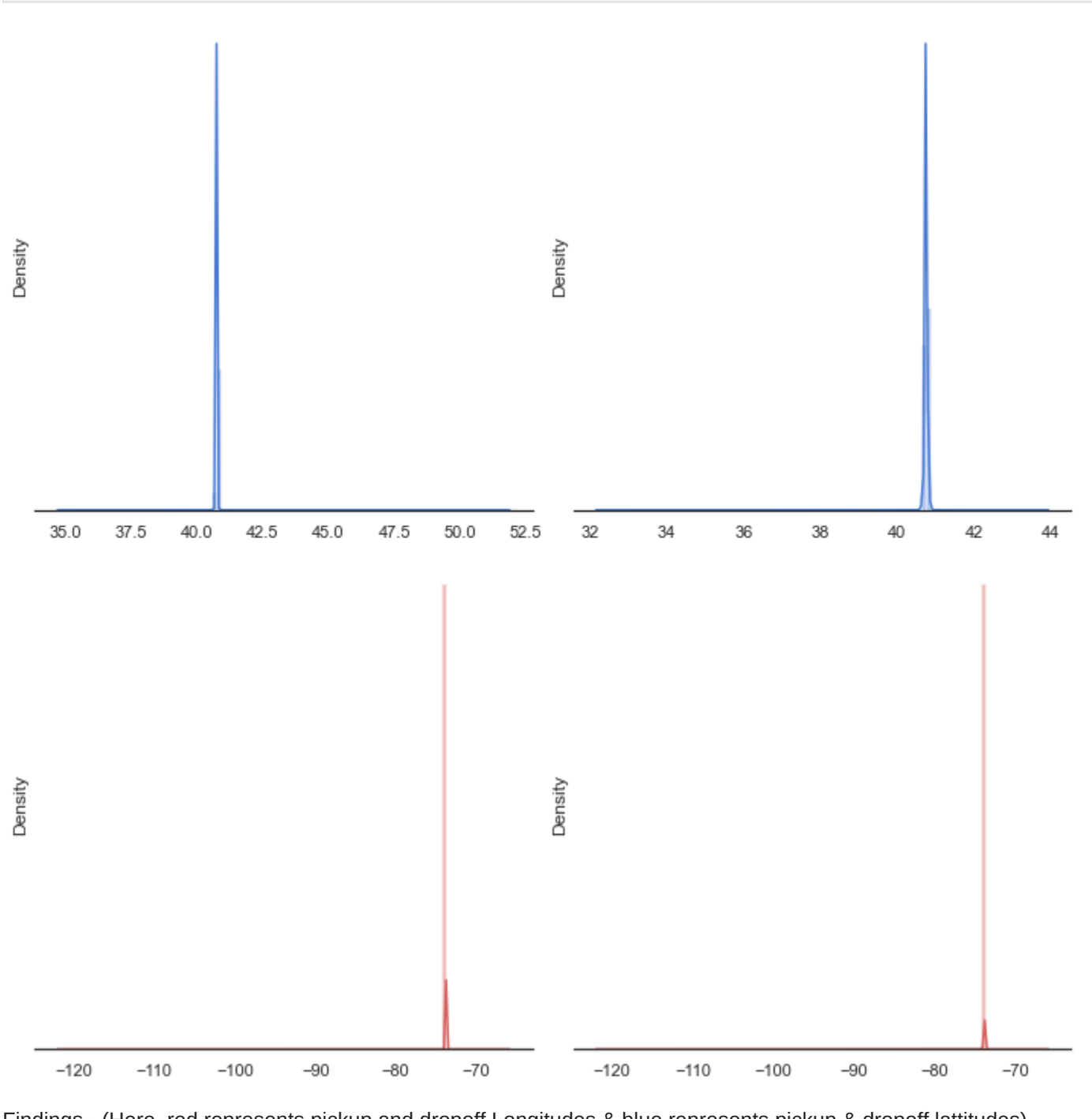


- Number of pickups for weekends is much lower than week days with a peak on Thursday (4). Note that here weekday is a decimal number, where 0 is Sunday and 6 is Saturday.
- Number of pickups as expected is highest in late evenings. However, it is much lower during the morning peak hours.

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.

In [67]:
```
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()
```

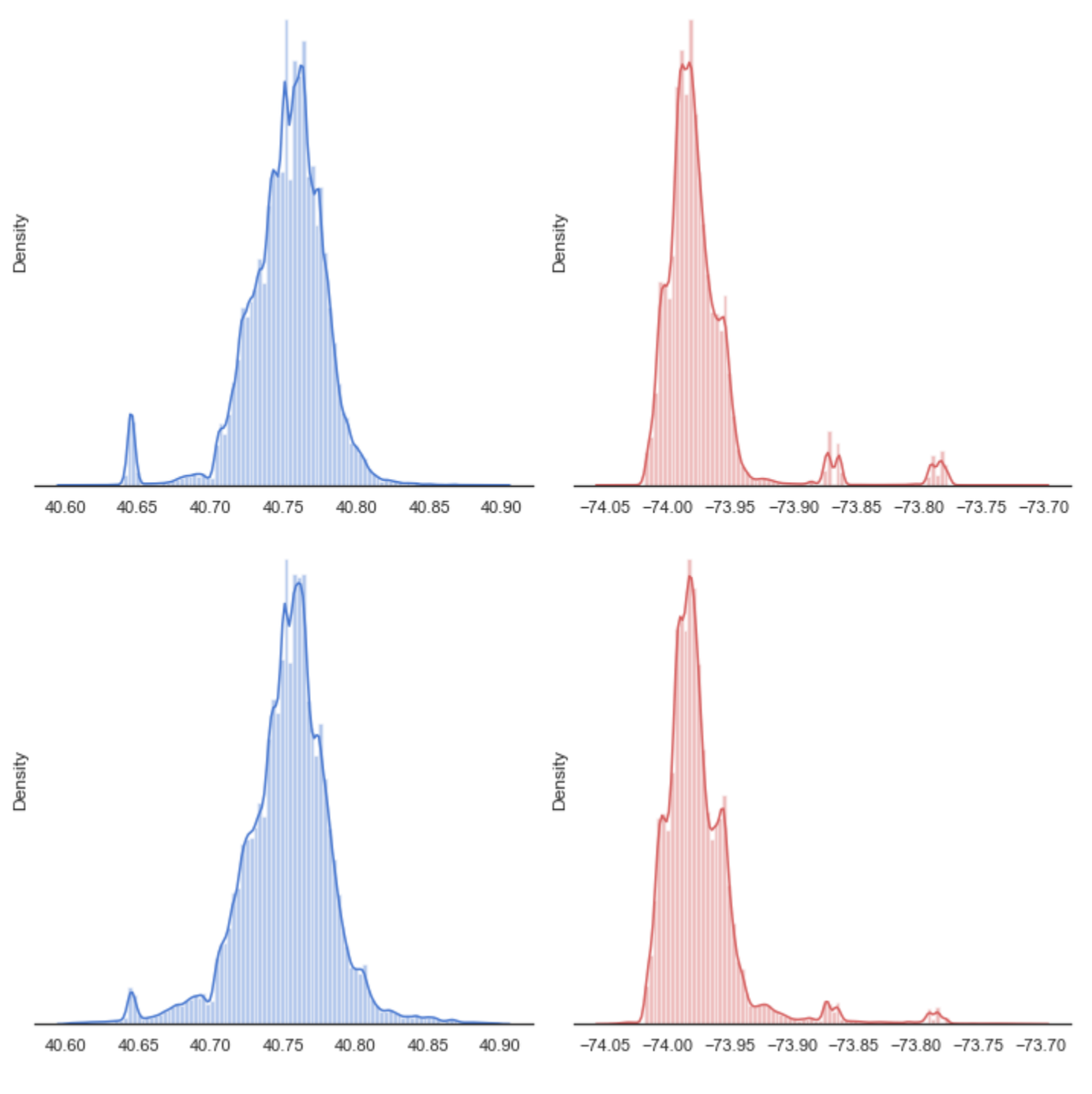


Findings - (Here, red represents pickup and dropoff Longitudes & blue represents pickup & dropoff lattitudes)

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

In [68]:
```
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.

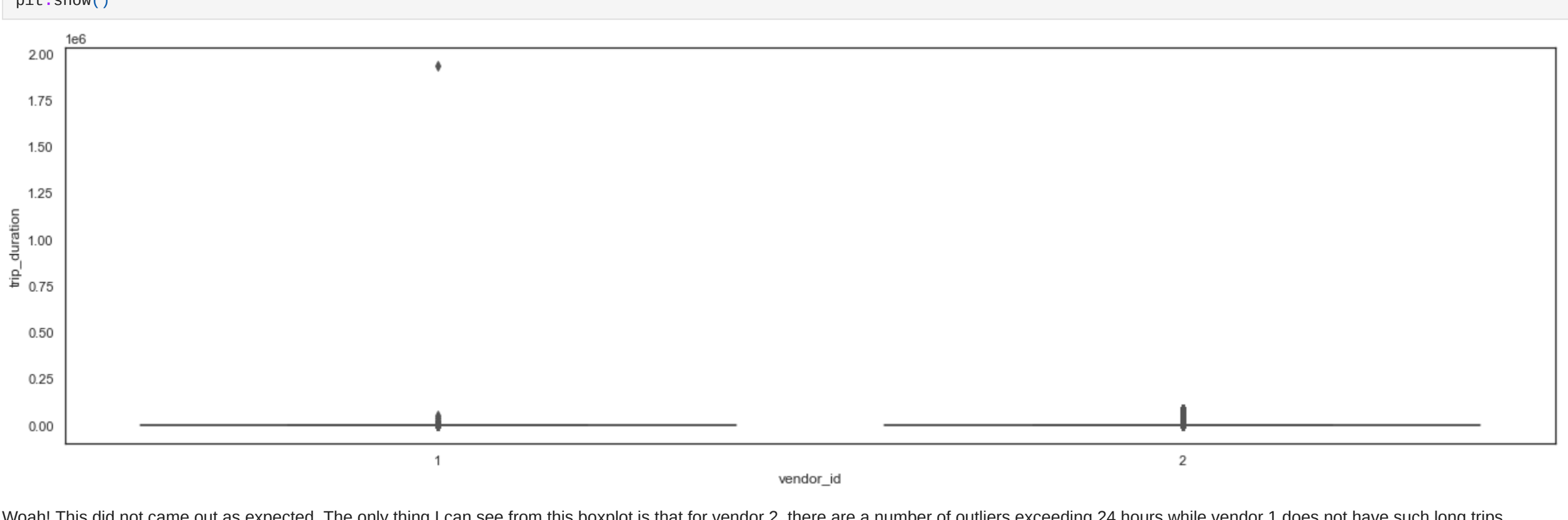
In [69]:
```
df.columns
```

Out[69]:
```
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')
```

vendor_id vs Trip Duration

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

In [70]:
```
plt.figure(figsize=(22, 6))
sns.boxplot(x="vendor_id", y="trip_duration", data=df)
plt.show()
```



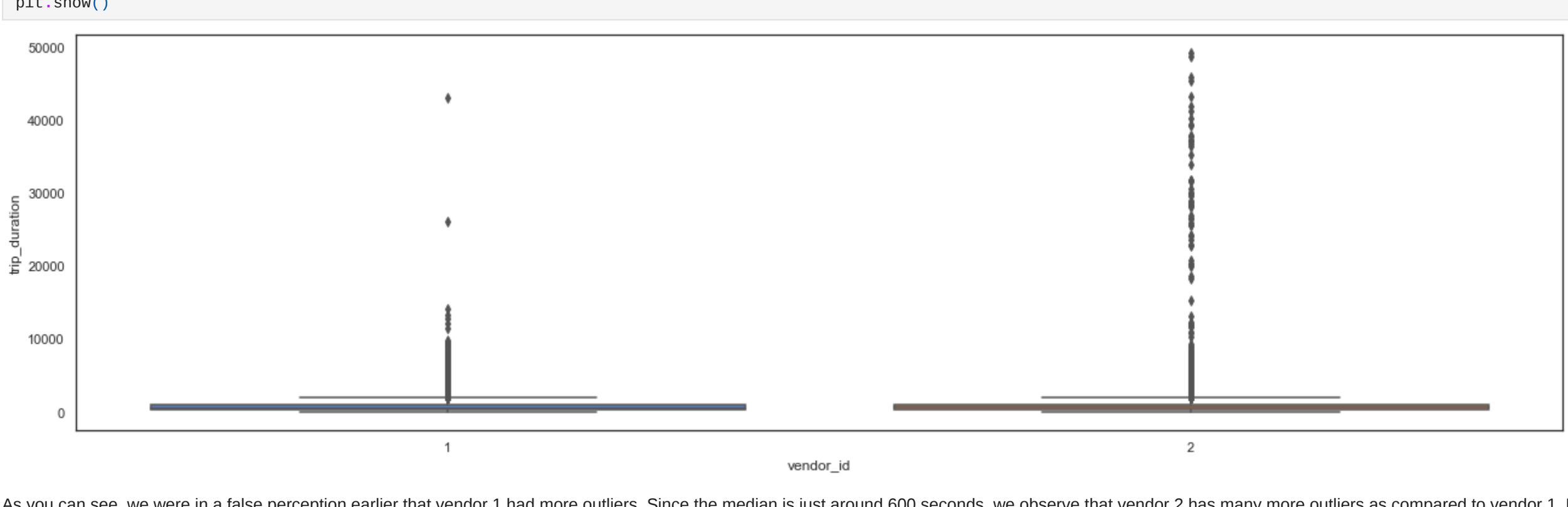
Woah! This did not came out as expected. The only thing I can see from this boxplot is that for vendor 2, there are a number of outliers exceeding 24 hours while vendor 1 does not have such long trips.

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

In [71]:
```
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.

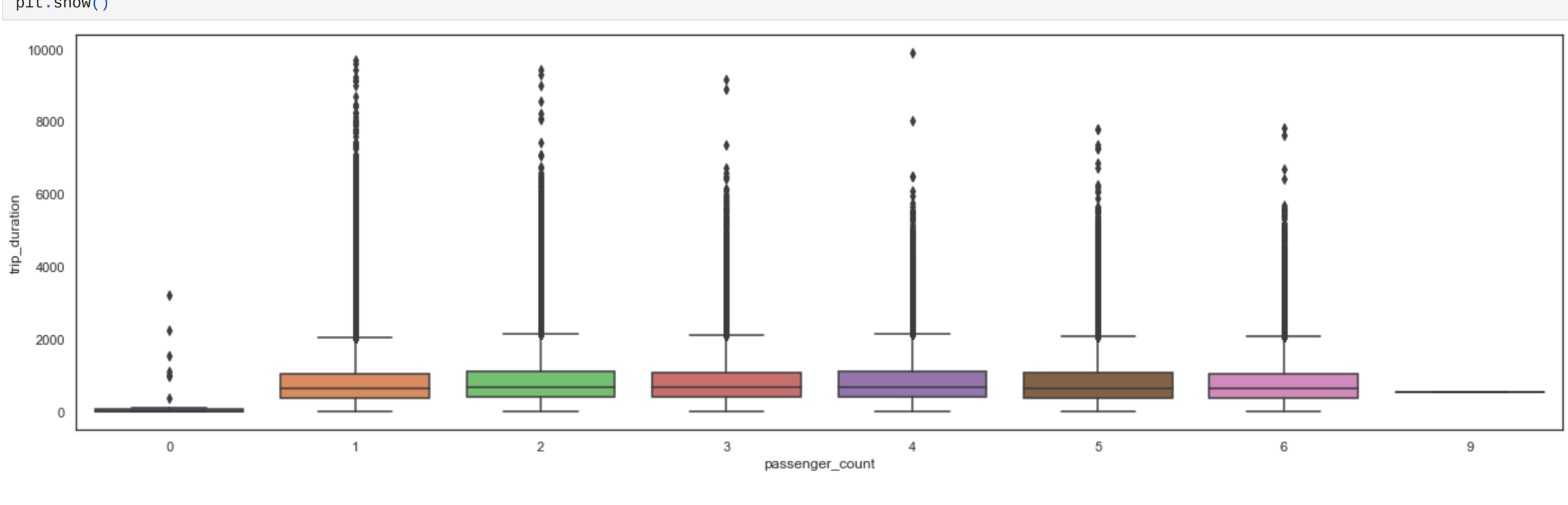
Trip Duration vs Passenger Count

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.

In [72]:
```
df.passenger_count.value_counts()
```

Out[72]:
```
1    515243
2    104576
5     38776
3     29561
6     24035
4     13972
0        31
9         1
Name: passenger_count, dtype: int64
```

In [73]:
```
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.

In [74]:
```
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()
```

In [75]:
```
df.to_csv('updated_data.csv', index=False)
```

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

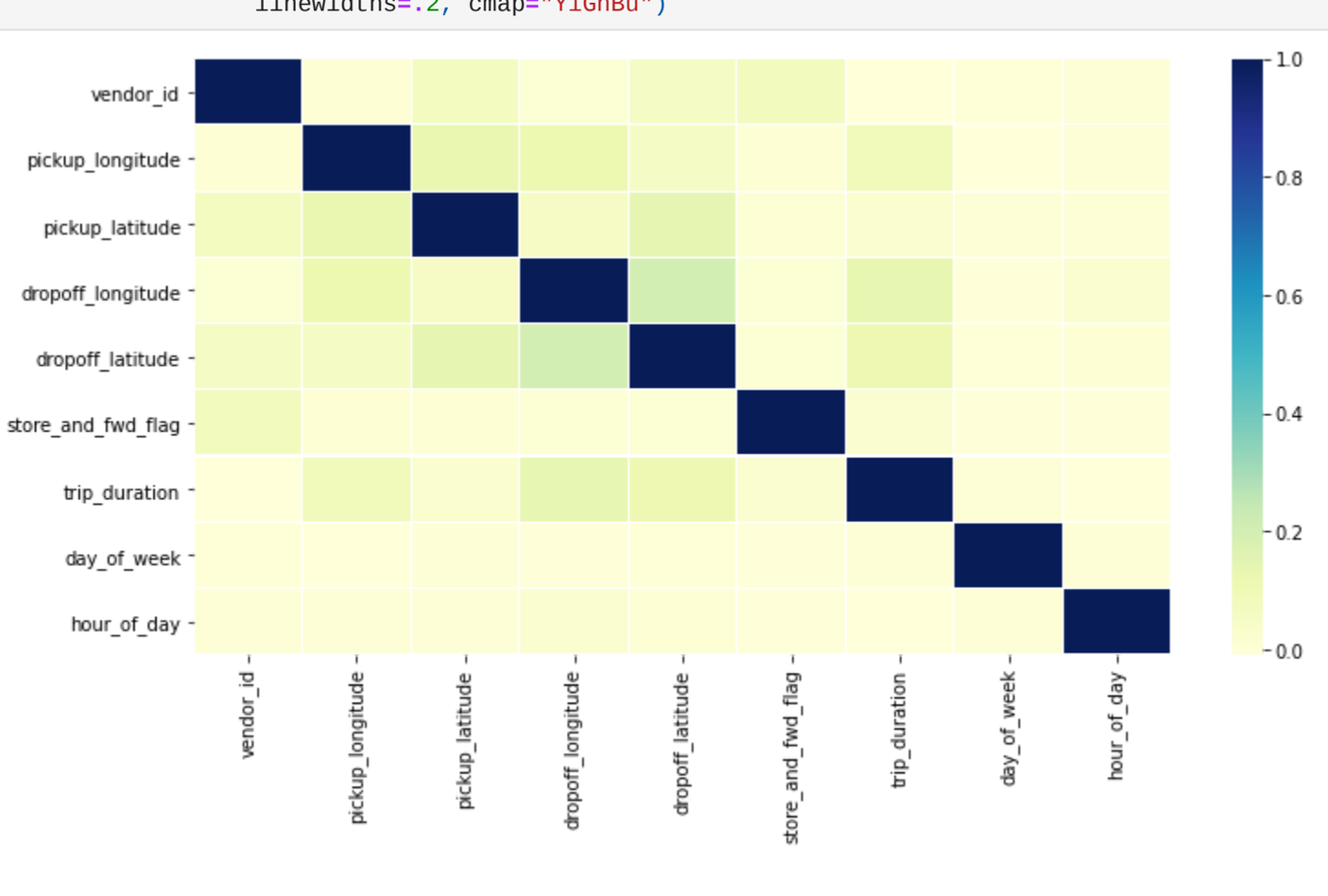
As expected there are a few small clusters for hot pickup points as displayed by red in the above plot. Most pickup points have less than 10 trips and distributed all over the city.

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.

In [100…
```
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.

In [71]:
```
df = pd.read_csv('updated_data.csv')
```

Calulating distance

In [72]:
```
df.head()
```

Out[72]:

	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
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	-73.963875	40.771164	0	400	400.0	5.993961	
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	-73.994751	40.694931	0	1100	1100.0	7.003974	
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	-73.948029	40.774918	0	1635	1635.0	7.400010	
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	-73.956779	40.780628	0	1141	1141.0	7.040536	
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	-73.988182	40.740631	0	848	848.0	6.744059	

In [73]:
```
def distance(lat1, lon1, lat2, lon2):
    p = 0.017453292519943295 #PL/188
    a= 0.5 - np.cos((lat2-lat1)*p)/2 + np.cos(lat1*p) *np.cos(lat2 *p)* (1- np.cos((lon2 - lon1)* p)) / 2
    return 0.6213712 *12742 * np.arcsin(np.sqrt(a))


df['distance_miles']= distance(df.pickup_latitude, df.pickup_longitude, df.dropoff_latitude, df.dropoff_longitude)
```

In [74]:
```
df = df.drop(['id', 'pickup_datetime', 'dropoff_datetime'
      ,'log_trip_duration','pickup_longitude','check_trip_duration','pickup_latitude','dropoff_longitude','dropoff_latitude'],
       axis=1)
```

In [75]:
```
df=pd.get_dummies(df)
df.head()
```

Out[75]:

	vendor_id	passenger_count	store_and_fwd_flag	trip_duration	day_of_week	hour_of_day	distance_miles
0	2	1	0	400	0	16	0.745069
1	1	2	0	1100	4	23	2.565711
2	2	2	0	1635	6	17	4.505409
3	2	6	0	1141	1	9	1.467118
4	1	1	0	848	2	6	2.689626

In [76]:
```
from sklearn.model_selection import train_test_split
x=df.drop(['trip_duration'],axis=1)
y=df[['trip_duration']]
train_x,test_x,train_y,test_y=train_test_split(x,y,random_state = 96)
```

In [77]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
x_train_scaled=scaler.fit_transform(train_x)
train_x=pd.DataFrame(x_train_scaled)
y_train_scaled=scaler.fit_transform(train_y)
train_y=pd.DataFrame(y_train_scaled)
```

In [78]:
```
print("shape of X_train : ",train_x.shape)
print("shape of X_test : ",test_x. shape)
print ("shape of y_train: ",train_y.shape)
print("Shape of y_test: ",test_y.shape)
```

```
shape of X_train :  (544646, 6)
shape of X_test :  (181549, 6)
shape of y_train:  (544646, 1)
Shape of y_test:  (181549, 1)
```

Implementing KNN Regresson

In [80]:
```
from sklearn import neighbors
from sklearn.metrics import mean_squared_error as mse
from math import sqrt
```

In [89]:
```
reg = neighbors.KNeighborsRegressor(n_neighbors = 9)

# Fitting the model
reg.fit(train_x, train_y)
# Predicting over the Train Set and calculating F1
train_predict_knn=reg.predict(train_x)
train_k_knn= sqrt(mse(train_predict_knn, train_y))
test_predict_knn = reg.predict(test_x)
test_k_knn = sqrt(mse(test_predict_knn, test_y))
print('Train RMSE    ', train_k_knn )
print('Test RMSE    ',test_k_knn )
```

```
Train RMSE     0.033093007117506894
Test RMSE     5664.908143786523
```

Implementing Linear Regression

In [63]:
```
from sklearn.linear_model import LinearRegression
```

In [64]:
```
lr=LinearRegression(normalize=True)
```

In [85]:
```
lr.fit(train_x,train_y)
train_pred_lr = lr.predict(train_x)
train_k_lr=sqrt(mse(train_y,train_pred_lr))
test_pred_lr=lr.predict(test_x)
test_k_lr=sqrt(mse(test_y,test_pred_lr))
print('Train RMSE    ', train_k_lr)
print('Test RMSE    ', test_k_lr)
```

```
Train RMSE     0.03519760272258626
Test RMSE     5664.877427108321
```

Decision Tree Model

In [86]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [87]:
```
regressor=DecisionTreeRegressor()
```

In [88]:
```
regressor.fit(train_x,train_y)
train_pred_dt = regressor.predict(train_x)
train_k_dt=mse(train_y,train_pred_dt)
test_pred_dt=regressor.predict(test_x)
test_k_dt=mse(test_y,test_pred_dt)
print('Train RMSE    ', train_k_lr )
print('Test RMSE    ', test_k_lr )
```

```
Train RMSE     0.03519760272258626
Test RMSE     5664.877427108321
```

In [ ]: