In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
df = pd.read_csv('nyc_taxi_trip_duration Dataset.csv')
df.head()
```

Out[2]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	-73.963875	40.771164	N	400
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	-73.994751	40.694931	N	1100
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	-73.948029	40.774918	N	1635
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	-73.956779	40.780628	N	1141
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	-73.988182	40.740631	N	848

In [3]:
```
# Our target variable is trip duration variable. Keeping it in mind lets start our EDA and make a suitable dataset for our models.
```

In [4]:
```
df.describe()
```

Out[4]:

	vendor_id	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration
count	729322.000000	729322.000000	729322.000000	729322.000000	729322.000000	729322.000000	7.293220e+05
mean	1.535403	1.662055	-73.973513	40.750919	-73.973422	40.751775	9.522291e+02
std	0.498745	1.312446	0.069754	0.033594	0.069588	0.036037	3.864626e+03
min	1.000000	0.000000	-121.933342	34.712234	-121.933304	32.181141	1.000000e+00
25%	1.000000	1.000000	-73.991859	40.737335	-73.991318	40.735931	3.970000e+02
50%	2.000000	1.000000	-73.981758	40.754070	-73.979759	40.754509	6.630000e+02
75%	2.000000	2.000000	-73.967361	40.768314	-73.963036	40.769741	1.075000e+03
max	2.000000	9.000000	-65.897385	51.881084	-65.897385	43.921028	1.939736e+06

In [5]:
```
df.isna().sum()
```

Out[5]:
```
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
```

In [6]:
```
# there are no null values in our dataset.
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 729322 entries, 0 to 729321
Data columns (total 11 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   id                  729322 non-null  object 
 1   vendor_id           729322 non-null  int64  
 2   pickup_datetime     729322 non-null  object 
 3   dropoff_datetime    729322 non-null  object 
 4   passenger_count     729322 non-null  int64  
 5   pickup_longitude    729322 non-null  float64
 6   pickup_latitude     729322 non-null  float64
 7   dropoff_longitude   729322 non-null  float64
 8   dropoff_latitude    729322 non-null  float64
 9   store_and_fwd_flag  729322 non-null  object 
 10  trip_duration       729322 non-null  int64  
dtypes: float64(4), int64(3), object(4)
memory usage: 61.2+ MB
```

In [8]:
```
df['trip_duration'].median() , df['trip_duration'].mean()
```

Out[8]:
```
(663.0, 952.2291333594764)
```

In [9]:
```
# median is less than mean..this means that there are larger number of smaller values but some larger outliers present in the far end.
```

In [10]:
```
Vendor_id = pd.get_dummies(df['vendor_id'] , prefix = 'vendor_id')
df = pd.concat([df , Vendor_id] , axis = 1)
```

In [11]:
```
# Creating dummy variables for vendor Id. Similarly doing so for store and fwd flag variable.
```

In [12]:
```
Store_and_fwd_Flag = pd.get_dummies(df['store_and_fwd_flag'] , prefix = 'store_and_fwd_flag')
df = pd.concat([df , Store_and_fwd_Flag] , axis = 1)
```

In [13]:
```
df.head()
```

Out[13]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration	vendor_id_1	vendor_id_2	store_and_fwd_flag
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	-73.963875	40.771164	N	400	0	1	
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	-73.994751	40.694931	N	1100	1	0	
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	-73.948029	40.774918	N	1635	0	1	
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	-73.956779	40.780628	N	1141	0	1	
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	-73.988182	40.740631	N	848	1	0	

In [14]:
```
# Now working with datetime features to extract features from pickup_datetime and dropoff_datetime variables.
```

In [15]:
```
import datetime

df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])
df['dropoff_datetime'] = pd.to_datetime(df['dropoff_datetime'])
```

In [16]:
```
df['weekday'] = df['pickup_datetime'].dt.weekday
df['hour_of_day'] = df['pickup_datetime'].dt.hour
df['month'] = df['pickup_datetime'].dt.month
```

In [17]:
```
df.head()
```

Out[17]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration	vendor_id_1	vendor_id_2	store_and_fwd_flag
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	-73.963875	40.771164	N	400	0	1	
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	-73.994751	40.694931	N	1100	1	0	
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	-73.948029	40.774918	N	1635	0	1	
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	-73.956779	40.780628	N	1141	0	1	
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	-73.988182	40.740631	N	848	1	0	

In [18]:
```
# We have extracted the necessary features that might help us in our model.
# Here weekday starts from 0 - Monday to 6 - Sunday
# Similarly for month 1 - January and 12 - December.
```

In [19]:
```
df.drop(['id' , 'vendor_id' , 'pickup_datetime' , 'dropoff_datetime' , 'store_and_fwd_flag'] , axis = 1 , inplace = True)
```

In [20]:
```
df.head()
```

Out[20]:

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration	vendor_id_1	vendor_id_2	store_and_fwd_flag_N	store_and_fwd_flag_Y	weekday	hour_of_day	month
0	1	-73.953918	40.778873	-73.963875	40.771164	400	0	1	1	0	0	16	2
1	2	-73.988312	40.731743	-73.994751	40.694931	1100	1	0	1	0	4	23	3
2	2	-73.997314	40.721458	-73.948029	40.774918	1635	0	1	1	0	6	17	2
3	6	-73.961670	40.759720	-73.956779	40.780628	1141	0	1	1	0	1	9	1
4	1	-74.017120	40.708469	-73.988182	40.740631	848	1	0	1	0	2	6	2

In [21]:
```
df.to_csv('cleaned_df.csv' , index = False)
```

In [22]:
```
x = df.drop(['trip_duration'] , axis = 1)
y = df['trip_duration']

x.shape , y.shape
```

Out[22]:
```
((729322, 12), (729322,))
```

In [23]:
```
from sklearn.model_selection import train_test_split
train_x , test_x , train_y , test_y = train_test_split(x , y , test_size = 0.25 , random_state = 100)
```

In [24]:
```
# Importing the random forest regressor module
```

In [25]:
```
from sklearn.ensemble import RandomForestRegressor
clf = RandomForestRegressor(random_state = 100 , n_estimators = 100 , n_jobs = 1)
```

In [26]:
```
from pprint import pprint
print('Parameters currently in use:\n')
pprint(clf.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': 1,
 'oob_score': False,
 'random_state': 100,
 'verbose': 0,
 'warm_start': False}
```

In [27]:
```
clf.fit(train_x , train_y)
```

Out[27]:
```
RandomForestRegressor(n_jobs=1, random_state=100)
```

In [28]:
```
clf.score(train_x , train_y)
```

Out[28]:
```
0.8485352228031487
```

In [29]:
```
clf.score(test_x , test_y)
```

Out[29]:
```
-0.026220339786820812
```

In [ ]:

In [30]:
```
# Okk we are getting a very worse score for our model.
# we will try to improve it later.
```

In [ ]:

In [31]:
```
# Now scaling and preprocessing the data for knn and linear regression
```

In [32]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

In [33]:
```
x = pd.DataFrame(x_scaled , columns = x.columns)
x.head()
```

Out[33]:

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	vendor_id_1	vendor_id_2	store_and_fwd_flag_N	store_and_fwd_flag_Y	weekday	hour_of_day	month
0	-0.504444	0.280911	0.832127	0.137198	0.538014	-0.931533	0.931533	0.074634	-0.074634	-1.560057	0.373006	-0.903461
1	0.257493	-0.212156	-0.570815	-0.306500	-1.577382	1.073500	-1.073500	0.074634	-0.074634	0.486536	1.466269	-0.308456
2	0.257493	-0.341220	-0.876953	0.364913	0.642175	-0.931533	0.931533	0.074634	-0.074634	1.509832	0.529187	-0.903461
3	3.305240	0.169785	0.261980	0.239160	0.800639	-0.931533	0.931533	0.074634	-0.074634	-1.048408	-0.720257	-1.498465
4	-0.504444	-0.625160	-1.263600	-0.212103	-0.309245	1.073500	-1.073500	0.074634	-0.074634	-0.536760	-1.188799	-0.903461

In [34]:
```
train_x , test_x , train_y , test_y = train_test_split(x , y , test_size = 0.25 , random_state = 100)
```

In [35]:
```
from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn.metrics import mean_absolute_error as mae
```

In [36]:
```
knn = KNN(n_neighbors = 380)
knn.fit(train_x , train_y)
```

Out[36]:
```
KNeighborsRegressor(n_neighbors=380)
```

In [37]:
```
# predicting over train data
#train_pred = knn.predict(train_x)
#k = mse(train_pred , train_y)
#print(k)
```

In [38]:
```
# predicting over test data
```

In [39]:
```
test_pred  = knn.predict(test_x)
k =mae(test_pred , test_y)
print(k)
```

```
437.0923462794242
```

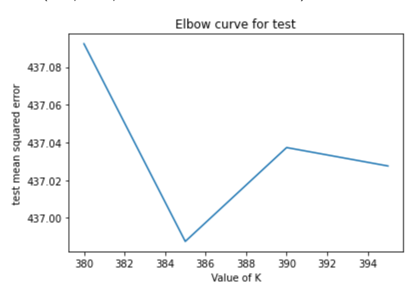
In [40]:
```
# creating an elbow curve

def elbow(K):
    test_mae = []

    for i in K:
        # instance of knn
        knn = KNN(n_neighbors = i)
        knn.fit(train_x , train_y)
        tmp = knn.predict(test_x)
        tmp = mae(tmp , test_y)
        test_mae.append(tmp)

    return test_mae
```

In [41]:
```
k = range(380,400,5)
```

In [42]:
```
test = elbow(k)
```

In [43]:
```
#plotting the curves
plt.plot(k , test)
plt.xlabel('Value of K')
plt.ylabel('test mean squared error')
plt.title('Elbow curve for test')
```

Out[43]:
```
Text(0.5, 1.0, 'Elbow curve for test')
```



In [44]:
```
# we can see from the plot that minimum value of k happens to be at 390.
# calculating the mean error at 390
```

In [45]:
```
knn = KNN(n_neighbors = 390)
knn.fit(train_x , train_y)
```

Out[45]:
```
KNeighborsRegressor(n_neighbors=390)
```

In [46]:
```
test_pred = knn.predict(test_x)
k = mae(test_pred , test_y)
print('mean test error score' , k)
```

```
mean test error score 437.03731116514086
```

In [ ]:

In [47]:
```
# This completes the knn model.
```

In [ ]:

In [48]:
```
# Building a basic regression model for our set.
```

In [49]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

In [50]:
```
lr.fit(train_x , train_y)
```

Out[50]:
```
LinearRegression()
```

In [51]:
```
train_pred = lr.predict(train_x)
k = mae(train_pred , train_y)
print('mean absolute error for train data is' , k)
```

```
mean absolute error for train data is 603.0866947181472
```

In [52]:
```
test_pred = lr.predict(test_x)
k = mae(test_pred , test_y)
print('mean absolute error for test data is' , k)
```

```
mean absolute error for test data is 588.890966773992
```

In [ ]:

In [53]:
```
# Regularising our model.
# Regularising with lasso regression
```

In [54]:
```
from sklearn import linear_model
lasso_reg = linear_model.Lasso(alpha = 0.01 , normalize = True , max_iter = 1000 , tol = 0.00001)
lasso_reg.fit(train_x , train_y)
```

Out[54]:
```
Lasso(alpha=0.01, normalize=True, tol=1e-05)
```

In [55]:
```
train_pred = lasso_reg.predict(train_x)
mean_train_error = mae(train_pred , train_y)
print('mean train error in case of lasso regression is' , mean_train_error)
```

```
mean train error in case of lasso regression is 602.7664418786728
```

In [56]:
```
test_pred = lasso_reg.predict(test_x)
mean_test_error = mae(test_pred , test_y)
print('mean test error in case of lasso regression is' , mean_test_error)
```

```
mean test error in case of lasso regression is 588.5287028506159
```

In [ ]:

In [57]:
```
# Regularising with ridge regression
```

In [58]:
```
from sklearn.linear_model import Ridge
ridge_reg = Ridge(alpha = 0.01 , normalize = True , max_iter = 1000 , tol = 0.00001)
ridge_reg.fit(train_x , train_y)
```

Out[58]:
```
Ridge(alpha=0.01, max_iter=1000, normalize=True, tol=1e-05)
```

In [59]:
```
train_pred = ridge_reg.predict(train_x)
mean_train_error = mae(train_pred , train_y)
print('mean train error in case of ridge regression is' , mean_train_error)
```

```
mean train error in case of ridge regression is 603.0867664004068
```

In [60]:
```
test_pred = ridge_reg.predict(test_x)
mean_test_error = mae(test_pred , test_y)
print('mean test error in case of ridge regression is' , mean_test_error)
```

```
mean test error in case of ridge regression is 588.8825527015712
```

In [ ]:

In [1]:
```
# Gradient Boosting Algorithm
```

In [61]:
```
from sklearn.ensemble import GradientBoostingRegressor
reg = GradientBoostingRegressor(random_state = 100)
```

In [63]:
```
reg.fit(train_x , train_y)
```

Out[63]:
```
GradientBoostingRegressor(random_state=100)
```

In [64]:
```
reg.score(test_x , test_y)
```

Out[64]:
```
0.026251521815612522
```

In [65]:
```
from pprint import pprint
print('Parameters currently in use:\n')
pprint(reg.get_params())
```

```
Parameters currently in use:

{'alpha': 0.9,
 'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'ls',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'random_state': 100,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

In [71]:
```
reg = GradientBoostingRegressor(random_state = 100 , alpha = 0.01 , n_estimators = 500)
```

In [72]:
```
reg.fit(train_x , train_y)
```

Out[72]:
```
GradientBoostingRegressor(alpha=0.01, n_estimators=500, random_state=100)
```

In [73]:
```
reg.score(test_x , test_y)
```

Out[73]:
```
0.028228928675545273
```

In [ ]: