Building Basic predictive models over the NYC Taxi Trip Dataset.

Exploratory Data Analysis

In [128]:
```
# Importing libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor

import warnings
warnings.filterwarnings('ignore')
```

In [129]:
```
#importing the Dataset
df = pd.read_csv('nyc_taxi_trip_duration.csv')
```

In [130]:
```
print("No. of rows: ", df.shape[0])
print("No. of columns: ", df.shape[1])
```

```
No. of rows:  729322
No. of columns:  11
```

In [131]:
```
df.head()
```

Out[131]:

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

In [132]:
```
df.tail()
```

Out[132]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	c
729317	id3905982	2	2016-05-21 13:29:38	2016-05-21 13:34:34	2	-73.965919	40.789780	-73.952637	
729318	id0102861	1	2016-02-22 00:43:11	2016-02-22 00:48:26	1	-73.996666	40.737434	-74.001320	
729319	id0439699	1	2016-04-15 18:56:48	2016-04-15 19:08:01	1	-73.997849	40.761696	-74.001488	
729320	id2078912	1	2016-06-19 09:50:47	2016-06-19 09:58:14	1	-74.006706	40.708244	-74.013550	
729321	id1053441	2	2016-01-01 17:24:16	2016-01-01 17:44:40	4	-74.003342	40.743839	-73.945847	

In [133]:
```
df.isnull().sum()
```

Out[133]:
```
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 [134]:
```
df.dtypes
```

Out[134]:
```
id                     object
vendor_id               int64
pickup_datetime        object
dropoff_datetime       object
passenger_count         int64
pickup_longitude      float64
pickup_latitude       float64
dropoff_longitude     float64
dropoff_latitude      float64
store_and_fwd_flag     object
trip_duration           int64
dtype: object
```

In [135]:
```
#Transforming pick_up and drop_off date time into a datetime object
df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'], format= '%Y-%m-%d %H:%M:%S')
df['dropoff_datetime'] = pd.to_datetime(df['dropoff_datetime'], format='%Y-%m-%d %H:%M:%S')
```

In [136]:
```
#Transforming vendor_id and store_and_fwd to categorical data type
df['vendor_id'] = df['vendor_id'].astype('category')
df['store_and_fwd_flag'] = df['store_and_fwd_flag'].astype('category')
```

In [137]:
```
# Converting yes/no flag to 1 and 0 and transforming it into categorical data type
df['store_and_fwd_flag'] = 1 * (df.store_and_fwd_flag.values == 'Y')
df['store_and_fwd_flag'] = df['store_and_fwd_flag'].astype('category')
```

In [138]:
```
#Checking the data types again
df.dtypes
```

Out[138]:
```
id                              object
vendor_id                     category
pickup_datetime         datetime64[ns]
dropoff_datetime        datetime64[ns]
passenger_count                  int64
pickup_longitude               float64
pickup_latitude                float64
dropoff_longitude              float64
dropoff_latitude               float64
store_and_fwd_flag            category
trip_duration                    int64
dtype: object
```

In [139]:
```
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[139]:
```
(0, 12)
```

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

In [140]:
```
print("Startdate: ", df['pickup_datetime'].min())
print("Enddate: ", df['pickup_datetime'].max())
```

```
Startdate:  2016-01-01 00:01:14
Enddate:  2016-06-30 23:59:37
```

The trip duration data is collected from the time period of first 6 months from the year 2016

In [141]:
```
# extracting more features from the datetime variable
# For pick_up
df['pickup_day']=df['pickup_datetime'].dt.day
df['pickup_hour'] = df['pickup_datetime'].dt.hour
df['pickup_weekday'] = df['pickup_datetime'].dt.weekday
# for Drop_off
df['dropoff_day'] = df['dropoff_datetime'].dt.day
df['dropoff_hour'] = df['dropoff_datetime'].dt.hour
df['dropoff_weekday'] = df['dropoff_datetime'].dt.weekday
```

In [142]:
```
df.head()
```

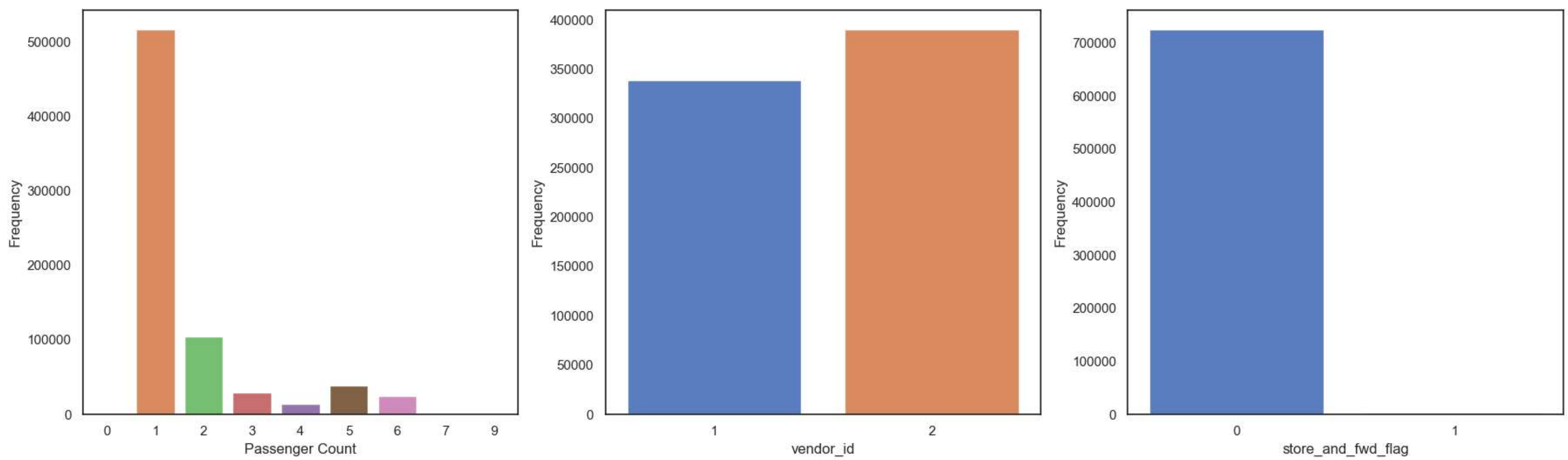
Out[142]:

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

Univariate Visualization

In [143]:
```
# 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[143]:
```
Text(0, 0.5, 'Frequency')
```



Most of the trips involve only 1 passenger.

Vendor 2 has more trips, compared to vendor 1.

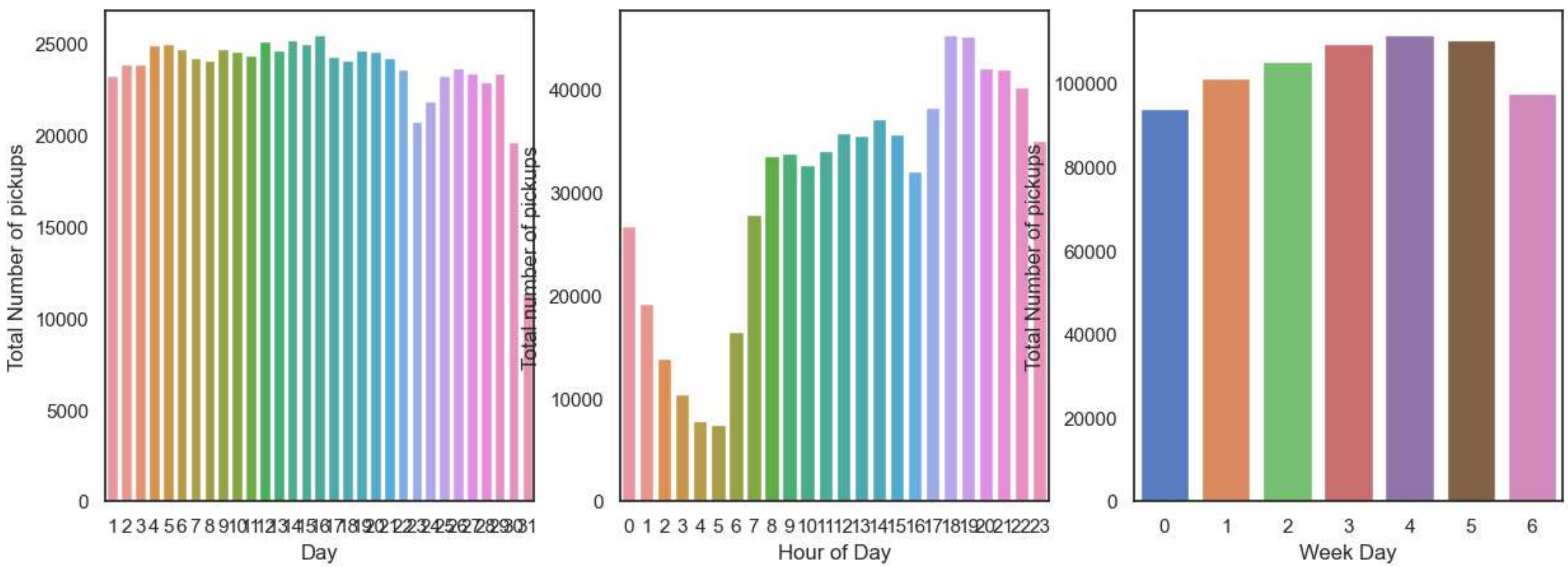
The value with 1 is very low in the store_and_fwd_flag variable. This suggests that almost no storing took place.

In [144]:

```python
# Datetime features
plt.figure(figsize=(20, 5))

# Passenger Count
plt.subplot(141)
sns.countplot(df['pickup_day'])
plt.xlabel('Day')
plt.ylabel('Total Number of pickups')

# vendor_id
plt.subplot(142)
sns.countplot(df['pickup_hour'])
plt.xlabel('Hour of Day')
plt.ylabel('Total number of pickups')

# Passenger Count
plt.subplot(143)
sns.countplot(df['pickup_weekday'])
plt.xlabel('Week Day')
plt.ylabel('Total Number of pickups')
```

Out[144]:

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



Trips are very low in early morning, while very high in the late evening hour in the day.

Trip is on peak on Thursday(4).

Trips are very low in early morning, while very high in the late evening hour in the day.

Feature engineering

In [145]:
```
df['passenger_count'].value_counts()
```

Out[145]:
```
1    517415
2    105097
5     38926
3     29692
6     24107
4     14050
0        33
7         1
9         1
Name: passenger_count, dtype: int64
```

Here as we can see that 0 and 9 are very less in number so we will remove it.

In [146]:
```
df=df[df['passenger_count']!=0]
df=df[df['passenger_count']<=6]
```

In [147]:
```
# checking
df['passenger_count'].value_counts()
```

Out[147]:
```
1    517415
2    105097
5     38926
3     29692
6     24107
4     14050
Name: passenger_count, dtype: int64
```

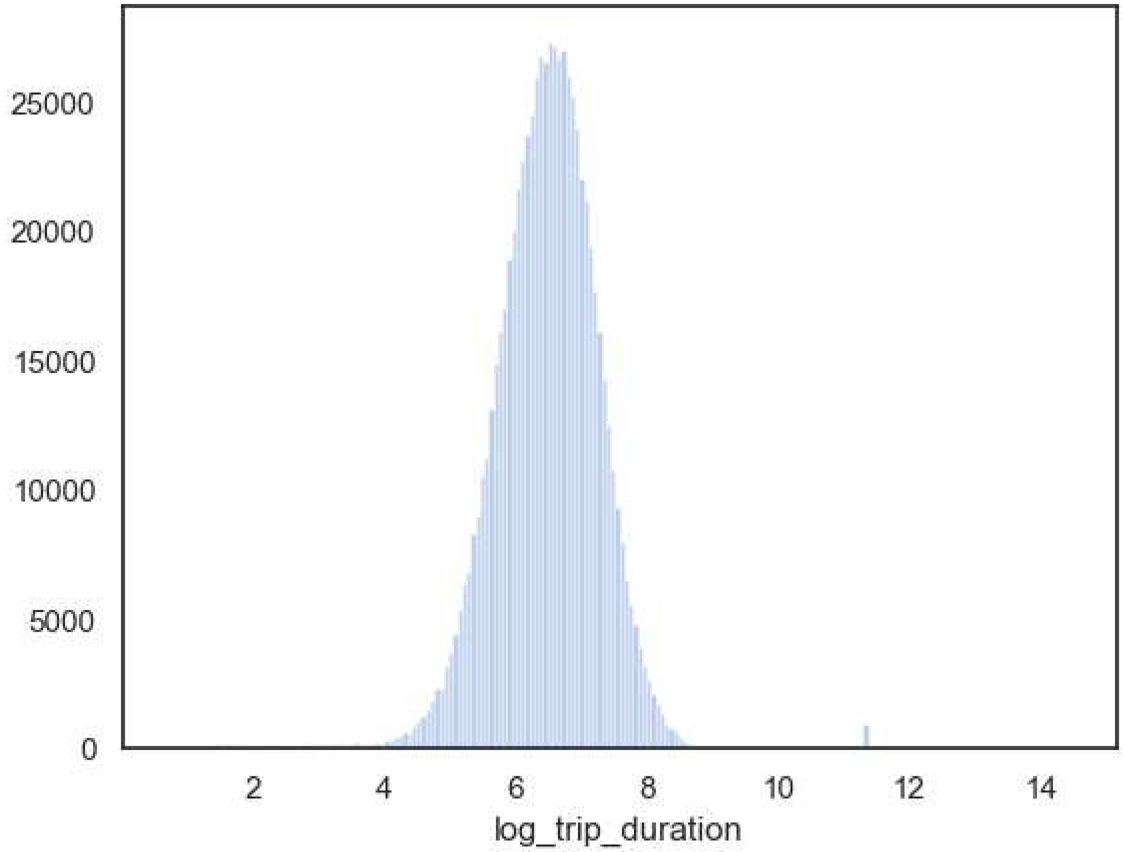
In [148]:
```
#Getting the summary of the trip_duration dataset
df['trip_duration'].describe()/3600 # Trip duration in hours
```

Out[148]:
```
count    202.579722
mean       0.264515
std        1.073531
min        0.000278
25%        0.110278
50%        0.184167
75%        0.298611
max      538.815556
Name: trip_duration, dtype: float64
```

There is a trip with maximum duration of 538 hours. This is a huge outlier and might create problems at the prediction stage. One idea is to log transform this feature.

In [149]:

```
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:

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.

There are several suspiciously short rides with less than 10 seconds duration.

As discussed earlier, there are a few huge outliers near 12.

In [150]:
```
df.head()
```

Out[150]:

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

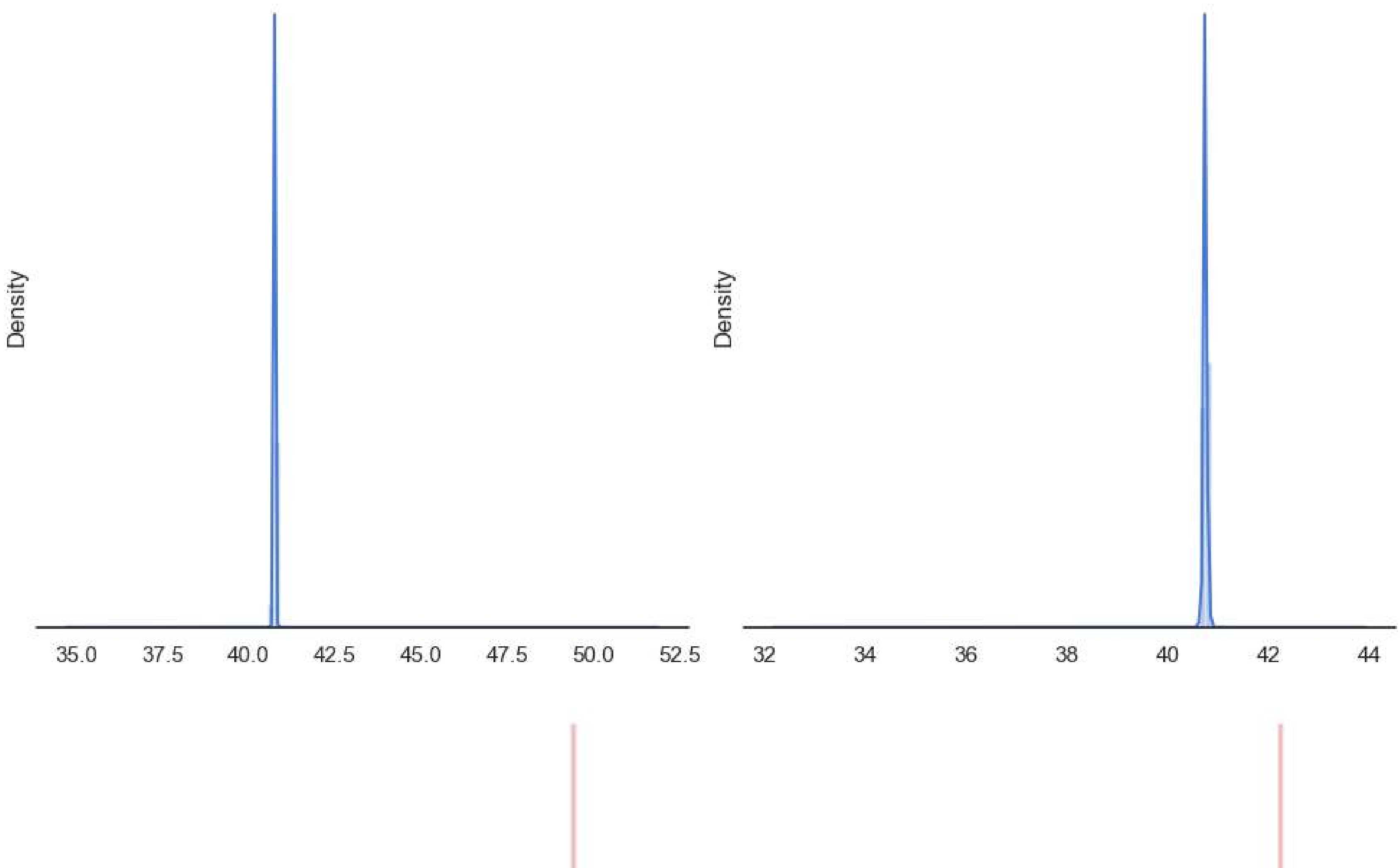
In [151]:
```
df.shape
```

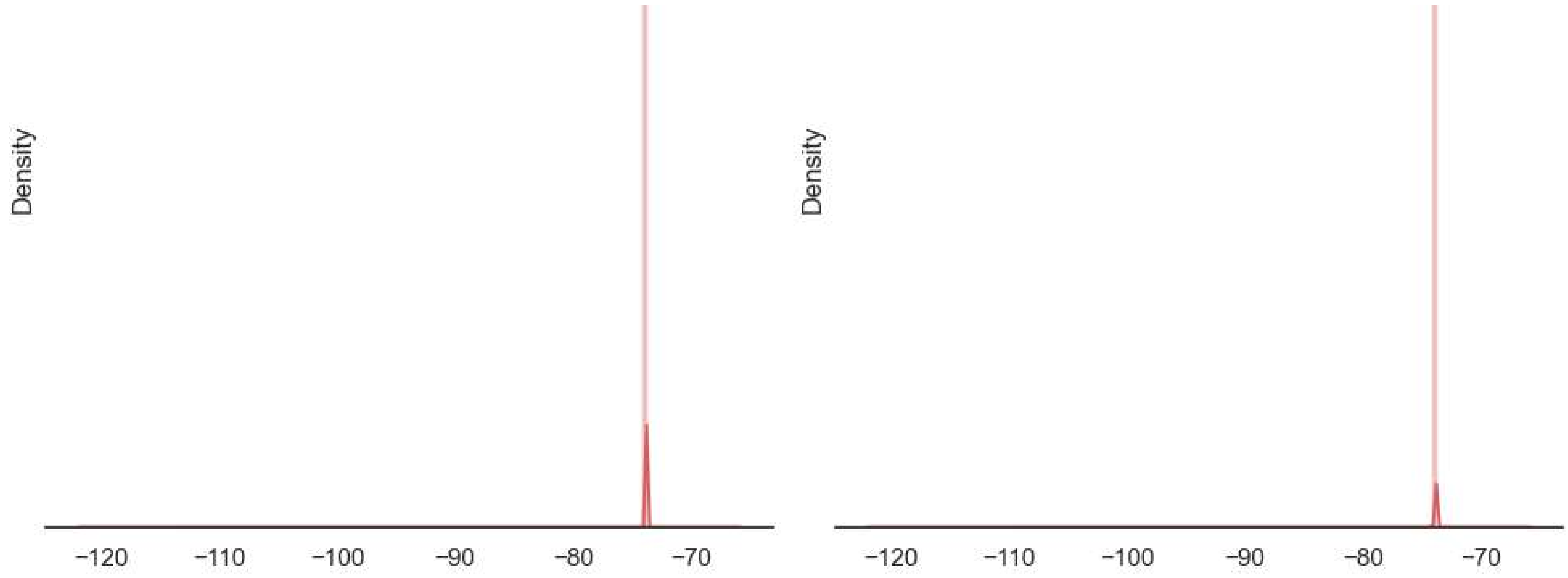
Out[151]:
```
(729287, 19)
```

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 [152]:
```
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

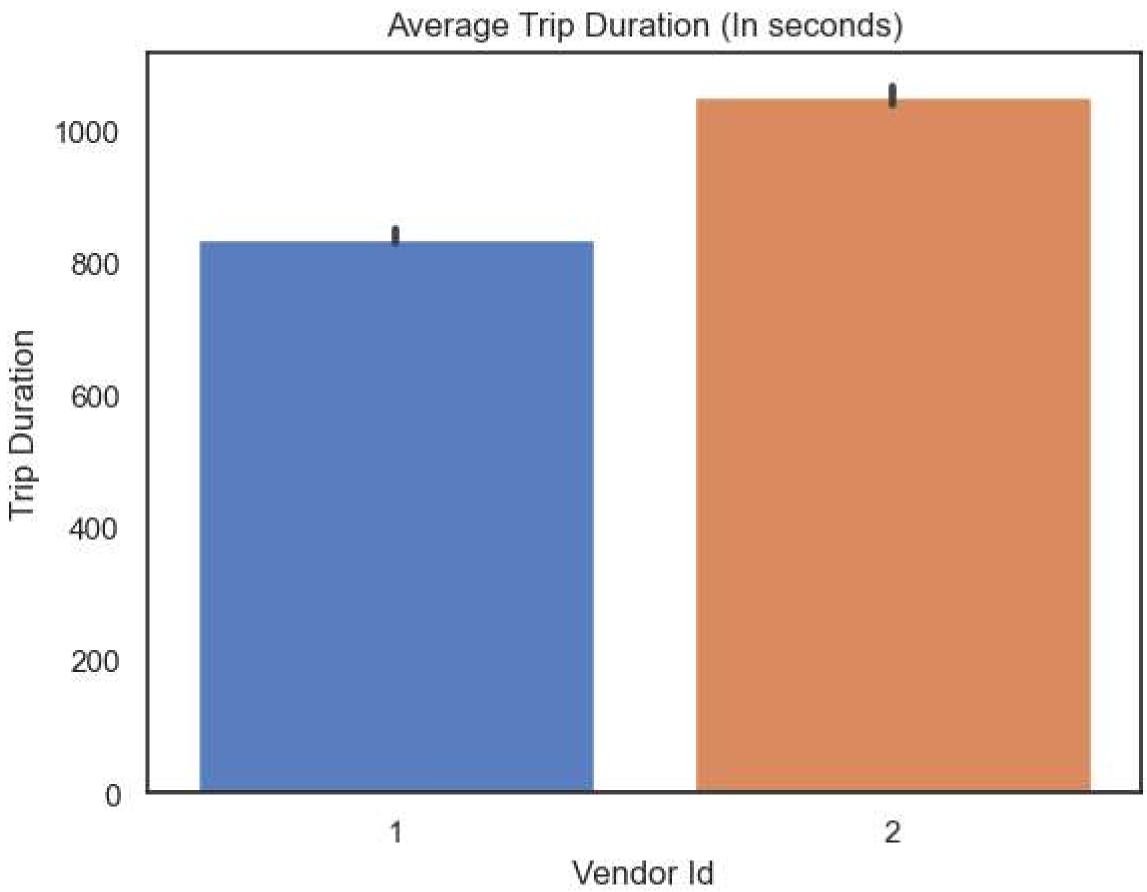
Bivariate Relations with Target

In [153]:
```
df.columns
```

Out[153]:
```
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', 'pickup_day', 'pickup_hour',
       'pickup_weekday', 'dropoff_day', 'dropoff_hour', 'dropoff_weekday',
       'log_trip_duration'],
      dtype='object')
```

Trip Duration vs Vendor Id

In [154]:
```
sns.barplot(x="vendor_id", y="trip_duration",data=df);
plt.title("Average Trip Duration (In seconds)");
plt.xlabel("Vendor Id");
plt.ylabel("Trip Duration");
```



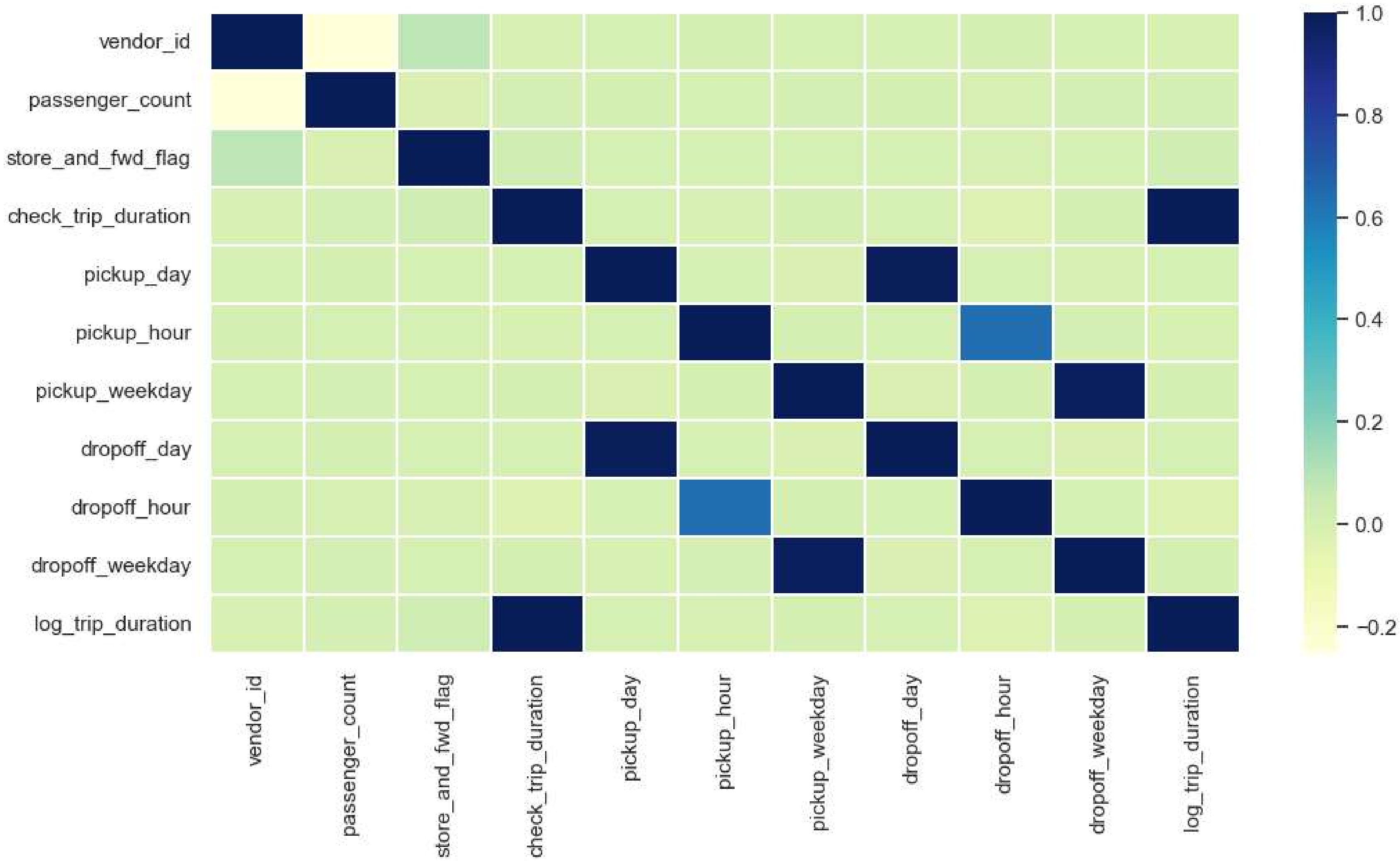
The average trip duration of vendor 2 is greater than vendor 1

Correlation Heatmap

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

In [155]:
```
df1 = df.drop(columns=['id', 'pickup_datetime', 'dropoff_datetime', 'pickup_longitude','pickup_latitude',
                       'dropoff_longitude','dropoff_latitude','trip_duration'])
```

In [156]:
```
# checking the correlation among all features
plt.figure(figsize=(12, 6))
corr = df1.apply(lambda x: pd.factorize(x)[0]).corr()
#corr = df1.corr()
ax = sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns,
                 linewidths=.2, cmap="YlGnBu")
```



1.Predictive Modeling

Root Mean Squared Error(RMSE) is a evaluation model I have choosed for this model to build

RMSE is a very simple metric to be used for evaluation. Since, we will be comparing our models and we will create a benchmark model as a baseline, RMSE will easy to compare these different models. Lower, the value of RMSE, better the model. It will help in getting the elbow curve.

Building a Benchmark Model

In [157]:
```
#seperating independent and dependent variables
X = df1.drop('log_trip_duration', axis=1)
y = df1['log_trip_duration']
```

In [158]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(X)
X = pd.DataFrame(x_scaled, columns=X.columns)
```

In [159]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

In [160]:
```
train = pd.concat([X_train, y_train], axis=1, join="inner")
test = pd.concat([X_test, y_test], axis=1, join="inner"
                )
```

In [164]:
```
test.head()
```

Out[164]:

	vendor_id	passenger_count	store_and_fwd_flag	check_trip_duration	pickup_day	pickup_hour	pickup_weekday	dropoff_day	dr
719491	0.0	0.0	0.0	0.000192	0.433333	0.826087	0.833333	0.433333	
485894	1.0	0.0	0.0	0.000091	0.166667	0.608696	0.666667	0.166667	
255979	1.0	0.0	0.0	0.000780	0.933333	0.565217	1.000000	0.933333	
335113	0.0	0.0	0.0	0.000289	0.533333	0.956522	0.166667	0.533333	
470288	1.0	0.0	0.0	0.000525	0.433333	0.956522	0.500000	0.433333	

In [165]:
```
test.tail()
```

Out[165]:

	vendor_id	passenger_count	store_and_fwd_flag	check_trip_duration	pickup_day	pickup_hour	pickup_weekday	dropoff_day	dr
12030	1.0	0.0	0.0	0.000786	0.200000	0.782609	0.833333	0.200000	
418952	1.0	0.0	0.0	0.000311	0.100000	0.782609	0.333333	0.100000	
662046	1.0	0.0	0.0	0.000639	0.133333	0.739130	0.500000	0.133333	
264794	0.0	0.0	0.0	0.000171	0.800000	0.304348	0.500000	0.800000	
595862	1.0	0.0	0.0	0.000871	0.300000	0.478261	0.166667	0.300000	

In [166]:
```
#storing simple mean in a new column in the test set as "simple_mean"
test['simple_mean'] = train['log_trip_duration'].mean()
```

In [167]:
```
#importing the library
from sklearn.metrics import mean_squared_error as MSE
from math import sqrt

#calculating root mean squared error
error = sqrt(MSE(test['log_trip_duration'] , test['simple_mean']))
error
```

Out[167]: 0.7881204914893064

In [168]:
```
trip_store = pd.pivot_table(train, values='log_trip_duration', index =['store_and_fwd_flag'], aggfunc=np.mean)
trip_store
```

Out[168]:

	log_trip_duration
store_and_fwd_flag	
0.0	6.468011
1.0	6.450863

In [169]:
```
# initializing new column to zero
test['trip_store_mean'] = 0

# For every unique entry in Outlet_Identifier
for i in train['store_and_fwd_flag'].unique():
  # Assign the mean value corresponding to unique entry
  test['trip_store_mean'][test['store_and_fwd_flag'] == i] = train['log_trip_duration'][train['store_and_fwd_fla
```

In [170]:
```
#calculating root mean squared error
error = sqrt(MSE(test['log_trip_duration'] , test['trip_store_mean']))
error
```

Out[170]: 0.7881269979288708

Error value = 0.7881 for Benchmark model

K-Nearest neighbours' Model

In [174]:
```
# Creating instance of Knn
knn = KNN(n_neighbors=5)

# Fitting the model
knn.fit(X_train, y_train)

# Predicting over the Train Set and calculating RMSE
y_pred = knn.predict(X_test)

error = sqrt(MSE(y_test, y_pred))

print("Test RMSE: ", error)
```

```
Test RMSE:  0.41305272705045804
```

Elbow Curve to determine value of K

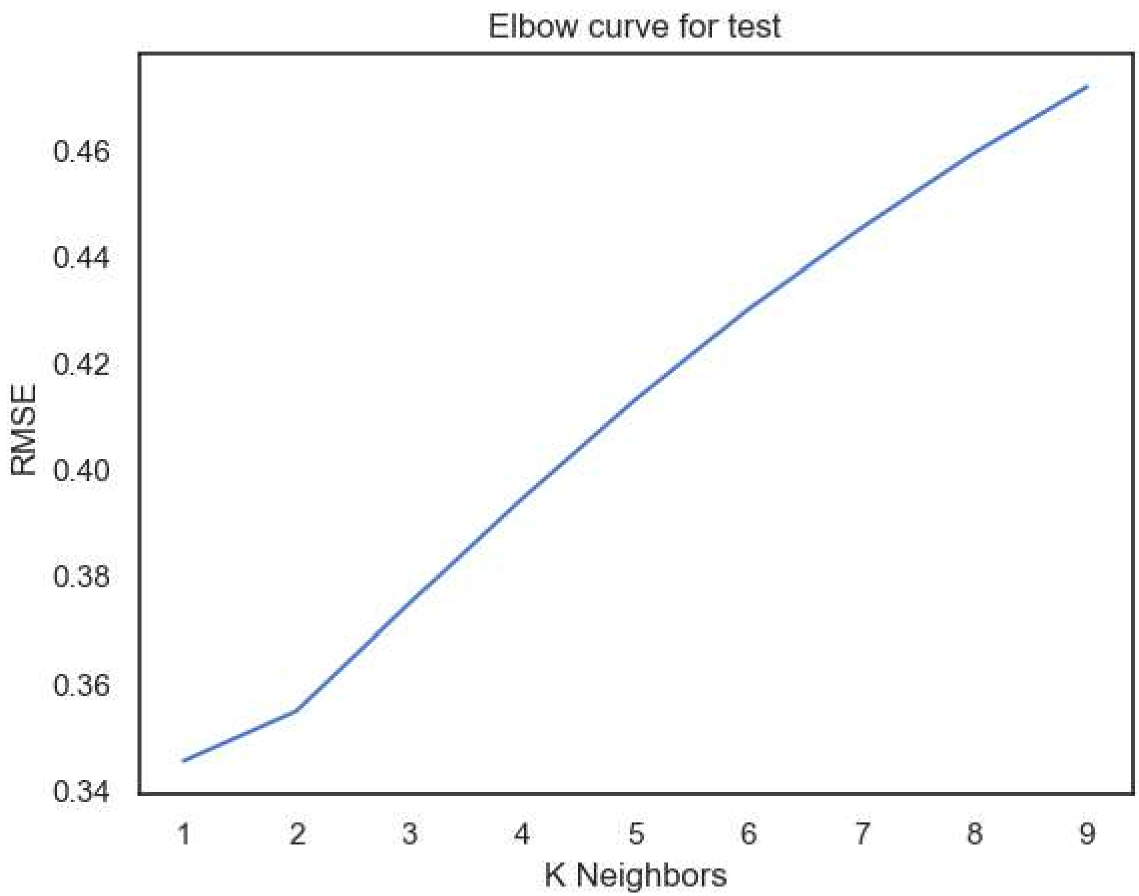
In [175]:
```
def Elbow(k):
  test = []
#training model for evey value of K
  for i in k:
    #Instance of KNN
    reg = KNN(n_neighbors=i)
    reg.fit(X_train, y_train)
    #Appending RMSE value to empty list claculated using the predictions
    tmp_pred = reg.predict(X_test)
    temp_error = sqrt(MSE(tmp_pred, y_test))
    test.append(temp_error)

  return test
```

In [176]:
```
#Defining K range
k = range(1, 10)
```

In [177]:
```
# calling above defined function
test = Elbow(k)
```

In [178]:
```
# plotting the curve
plt.plot(k, test)
plt.xlabel('K Neighbors')
plt.ylabel('RMSE')
plt.title('Elbow curve for test')
```

Out[178]:
```
Text(0.5, 1.0, 'Elbow curve for test')
```



Train Score for K-Nearest neighbours' model

In [189]:
```
# Predicting over the Train Set and calculating RMSE
y_pred = knn.predict(X_train)

knn_train_rmse = sqrt(MSE(y_train, y_pred))

print("Train RMSE: ", knn_train_rmse)
```

```
Train RMSE:  0.2821119956078097
```

Test Score for K-Nearest neighbours' model

In [179]:
```
# Creating instance of KNN again at the value of n_neighbours=6
knn = KNN(n_neighbors=4)

# Fitting the model
knn.fit(X_train, y_train)

# Predicting over the Test Set and calculating RMSE
y_pred = knn.predict(X_test)

error = sqrt(MSE(y_test, y_pred))

print("Test RMSE: ", error)
```

```
Test RMSE:  0.39450047377576736
```

Train Score for K-Nearest neighbours' model

Linear Model

In [181]:
```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

Out[181]:
```
LinearRegression()
```

Train score for Linear Model

In [186]:
```
y_pred = lr.predict(X_train)

lm_train_rmse = sqrt(MSE(y_train, y_pred))

print("RMSE of linear regressor model: ", lm_train_rmse)
```

```
RMSE of linear regressor model:  0.7534906679640115
```

Test Score for Linear Model

In [188]:
```
y_pred = lr.predict(X_test)

lm_test_rmse = sqrt(MSE(y_test, y_pred))
print("RMSE of linear regressor model: ", lm_test_rmse)
```

```
RMSE of linear regressor model:  0.7381668274657416
```

In [ ]: