In [2]:
```
# Importing the libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from geopy.distance import great_circle

from sklearn.model_selection import train_test_split

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

from sklearn.metrics import mean_squared_error
from math import sqrt

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

In [3]:
```
# Loading the given dataset in notebook

df = pd.read_csv('Datasets/nyc_taxi_trip_duration.csv')
```

The given data in the description according to the given data files given as columns :

1) id = a unique identifier for each trip

2) vendor_id = a code indicating the provider associated with the trip record

3) pickup_datetime = date and time when the meter was engaged

4) dropoff_datetime = date and time when the meter was disengaged

5) passenger_count = the number of passengers in the vehicle

6) pickup_longitude = the longitude where the meter was engaged

7) pickup_latitude = the latitude where the meter was engaged

8) dropoff_longitude = the longitude where the meter was disengaged

9) dropoff_latitude = the latitude where the meter was disengaged

10) 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

11) trip_duration = target duration of the trip in seconds

In [4]:
```
# getting the rows and columns data present in the data set
print("Number. of rows: ", df.shape[0])
print("Number. of columns: ", df.shape[1])
```

```
Number. of rows:  729322
Number. of columns:  11
```

In [5]:
```
df.head()
```

Out[5]:

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

In [6]:
```
df.tail()
```

Out[6]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longit
729317	id3905982	2	2016-05-21 13:29:38	2016-05-21 13:34:34	2	-73.965
729318	id0102861	1	2016-02-22 00:43:11	2016-02-22 00:48:26	1	-73.996
729319	id0439699	1	2016-04-15 18:56:48	2016-04-15 19:08:01	1	-73.997
729320	id2078912	1	2016-06-19 09:50:47	2016-06-19 09:58:14	1	-74.006
729321	id1053441	2	2016-01-01 17:24:16	2016-01-01 17:44:40	4	-74.003

MISSING VALUES

In [7]:
```
# studying the missing values in the given data
print("Number of missing values in each columns: \n")
print(df.isnull().sum())
```

```
Number of missing values in each columns: 

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
```

There is no missing values in the given data = 0

In [8]:
```
# number of duplicate records present in the data set
print("The duplicate records are present in the given data set : ", df.duplicated(
```

```
The duplicate records are present in the given data set :  0
```

There are 0 duplicate records are present in the given data set

Unique values and datatypes

In [9]:
```
# getting the unique values from the dataset
df.nunique()
```

Out[9]:
```
id                    729322
vendor_id                  2
pickup_datetime       709359
dropoff_datetime      709308
passenger_count            9
pickup_longitude       19729
pickup_latitude        39776
dropoff_longitude      27892
dropoff_latitude       53579
store_and_fwd_flag         2
trip_duration           6296
dtype: int64
```

This dataset contains 729322 unique id's and this is the exact number rows that the dataset will have.

In [10]:
```
# getting the different datatypes from the given dataset
df.dtypes
```

Out[10]:
```
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 [11]:
```
# getting the count value of the dataset
df['store_and_fwd_flag'].value_counts()
```

Out[11]:
```
N    725282
Y      4040
Name: store_and_fwd_flag, dtype: int64
```

In this variable it will only contain yes and no variables. for YES (Y) and NO (N)

In [12]:
```
# object using label encoder
label_encoder = LabelEncoder()

# The labels are encoded in the column called "discount variable"
```

```
df['store_and_fwd_flag_encoded']= label_encoder.fit_transform(df['store_and_fwd_fla
df['store_and_fwd_flag_encoded']= label_encoder.fit_transform(df['store_and_fwd_fla
```

PICKUP_DATETIME AND DROPPING_DATETIME

In [13]:
```
# FEATURING THE DATATYPE DATE AND TIME:
print(df[['pickup_datetime', 'dropoff_datetime']].dtypes)
```

```
pickup_datetime     object
dropoff_datetime    object
dtype: object
```

In [14]:
```
#  strings converting into datetime feature:

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

In [15]:
```
# studying the datetime datatype after converting string into datatype
print(df[['pickup_datetime', 'dropoff_datetime']].dtypes)
```

```
pickup_datetime     datetime64[ns]
dropoff_datetime    datetime64[ns]
dtype: object
```

In [16]:
```
# Printing the start and end datetiming
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 duration of the trip checked as per datetime featuring. The duration of trip data is collected from the time period of first 6 months of the year 2016

In [17]:
```
# getting extra features from variable datetime:

df['pickup_day'] = df['pickup_datetime'].dt.day
df['pickup_hour'] = df['pickup_datetime'].dt.hour
df['pickup_weekday'] = df['pickup_datetime'].dt.weekday
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 [18]:
```
# after getting extra features from datetime variable checking the data
df.head()
```

Out[18]:

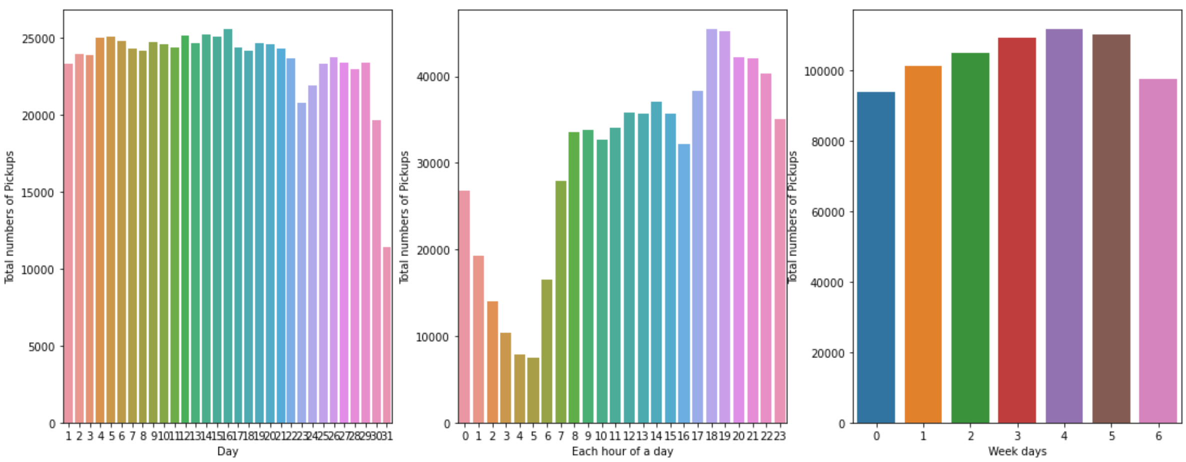
	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314
3	id3744273	2	2016-01-05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

In [19]:

```python
# getting the graph of DAY, HOUR OF THE DAY AND WEEK DAY as per the data

# plotting the size of the graph
plt.figure(figsize=(25, 7))

# Count of the passengers
plt.subplot(141)
sns.countplot(df['pickup_day'])
plt.xlabel('Day')
plt.ylabel('Total numbers of Pickups')

# identity of the vendor
plt.subplot(142)
sns.countplot(df['pickup_hour'])
plt.xlabel('Each hour of a day')
plt.ylabel('Total numbers of Pickups')

# Count of passengers
plt.subplot(143)
sns.countplot(df['pickup_weekday'])
plt.xlabel('Week days')
plt.ylabel('Total numbers of Pickups')
```

Out[19]:

```
Text(0, 0.5, 'Total numbers of Pickups')
```



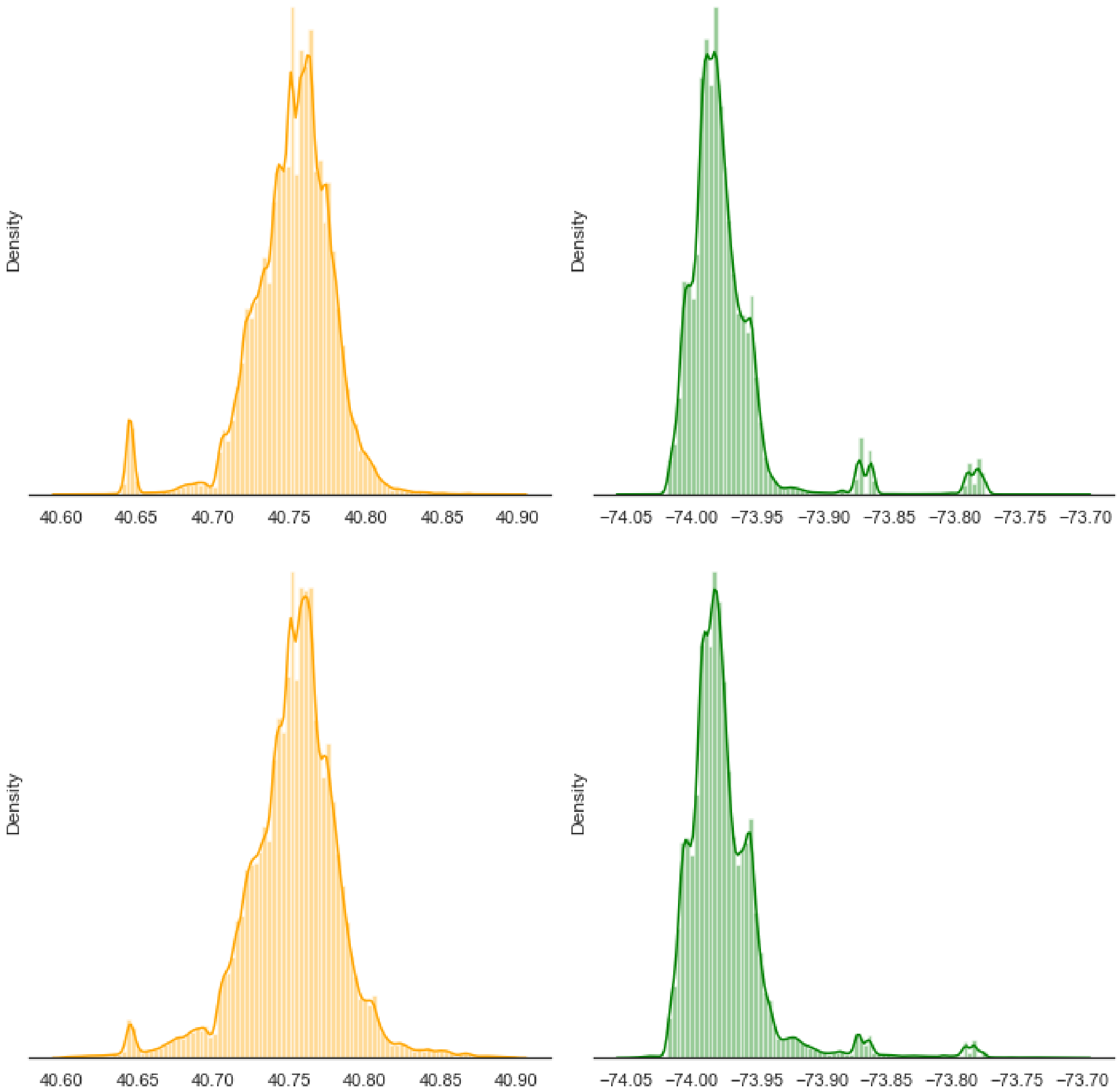
1) First and Second week of the month has more rides than other

2) Rides are very less in morning times and very high in the late evening of a hour in the day.

3) Ride is at the peakstage on Thursday(4) in weekdays.

Latitude and longitude

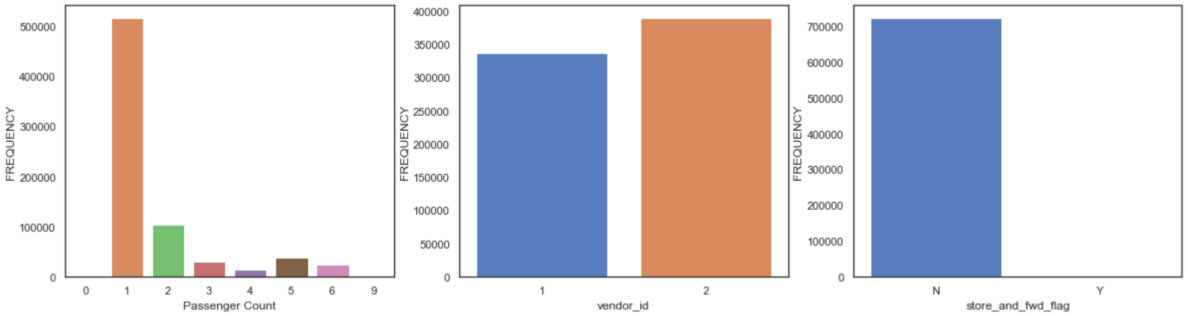
In [20]:
```
df = df.loc[(df.pickup_latitude > 40.6) & (df.pickup_latitude < 40.9)]
df = df.loc[(df.dropoff_latitude>40.6) & (df.dropoff_latitude < 40.9)]
df = df.loc[(df.dropoff_longitude > -74.05) & (df.dropoff_longitude < -73.7)]
df = df.loc[(df.pickup_longitude > -74.05) & (df.pickup_longitude < -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',colo
sns.distplot(df_data_new['pickup_longitude'].values, label = 'pickup_longitude',co
sns.distplot(df_data_new['dropoff_latitude'].values, label = 'dropoff_latitude',co
sns.distplot(df_data_new['dropoff_longitude'].values, label = 'dropoff_longitude',
plt.setp(axes, yticks=[])
plt.tight_layout()

plt.show()
```



In [21]:
```
# plotting the size of the figures
plt.figure(figsize=(20, 5))
```

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

# Identity of the vendor
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[21]: Text(0, 0.5, 'FREQUENCY')



In [22]:
```
# Count of passengers and datatype
df['passenger_count'].value_counts()
```

Out[22]:
```
1    515243
2    104576
5     38776
3     29561
6     24035
4     13972
0        31
9         1
Name: passenger_count, dtype: int64
```

In [23]:
```
df=df[df['passenger_count']!=0]
df=df[df['passenger_count']<=6]
```

In [24]:
```
#The value distribution of passenger_count
df['passenger_count'].value_counts()
```

Out[24]:
```
1    515243
2    104576
5     38776
3     29561
6     24035
4     13972
Name: passenger_count, dtype: int64
```

In [25]:
```
df.head()
```

Out[25]:

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

In [26]:
```
df.tail()
```

Out[26]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longit
729317	id3905982	2	2016-05-21 13:29:38	2016-05-21 13:34:34	2	-73.965
729318	id0102861	1	2016-02-22 00:43:11	2016-02-22 00:48:26	1	-73.996
729319	id0439699	1	2016-04-15 18:56:48	2016-04-15 19:08:01	1	-73.997
729320	id2078912	1	2016-06-19 09:50:47	2016-06-19 09:58:14	1	-74.006
729321	id1053441	2	2016-01-01 17:24:16	2016-01-01 17:44:40	4	-74.003

In [27]:
```
# The unique values of id variables
df['id'].nunique()
```

Out[27]:
```
726163
```

In [28]:
```
# The shape of the given dataset is
df.shape
```

Out[28]:
```
(726163, 18)
```

Trip duration

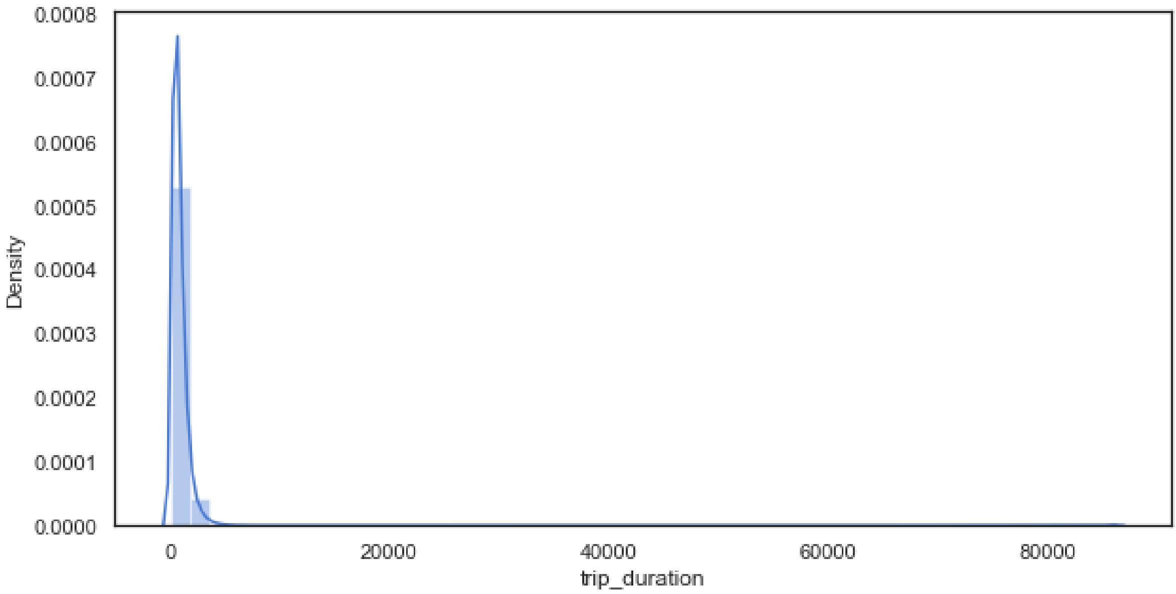
In [29]:
```
# Trip duration in one hour
df['trip_duration'].describe()/3600
```

Out[29]:
```
count    201.711944
mean       0.262934
std        1.073500
min        0.000278
25%        0.110000
50%        0.183611
75%        0.296944
max      538.815556
Name: trip_duration, dtype: float64
```

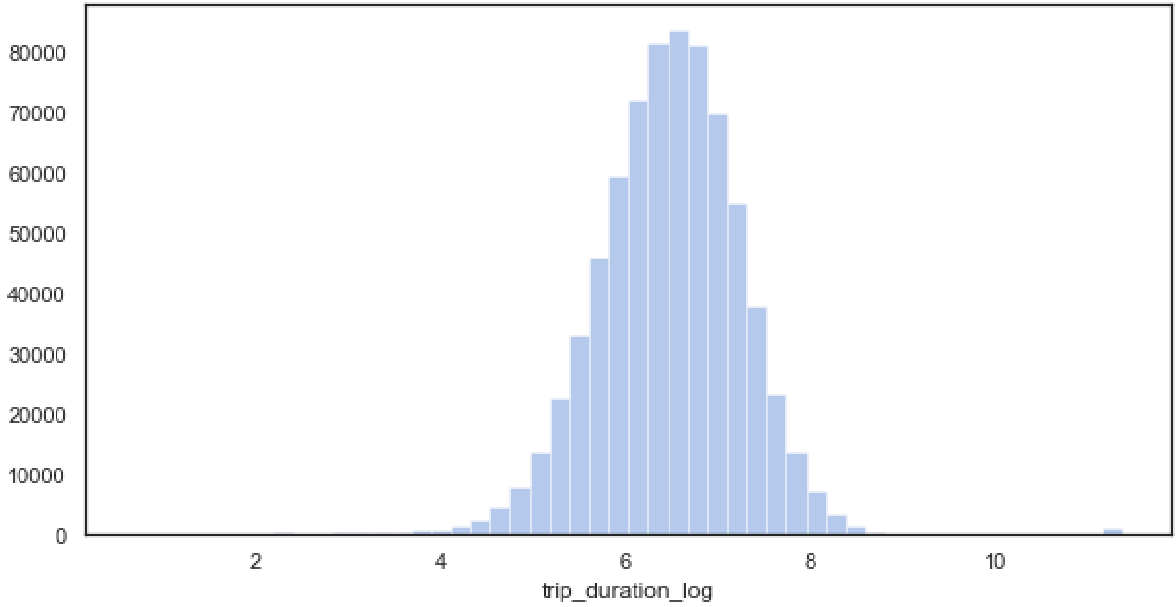
In [30]:
```
# Trip duration in hours
df['trip_duration_in_hour'] = df['trip_duration'].apply(lambda x : x/3600)

# deleting the outliers considering rides are not supposed to exceed in 24 hours
df = df[df['trip_duration_in_hour']<=24]

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

Out[30]:
```
(0.0002777777777777778, 23.9975)
```

In [31]:
```
# plotting the trip dsuration figure
plt.figure(figsize=[10, 5])
sns.distplot(df['trip_duration'])
plt.show()
```



In [32]:
```
# plotting the trip duration log figure
plt.figure(figsize=[10, 5])
df['trip_duration_log'] = np.log(df['trip_duration'].values + 1)
sns.distplot(df['trip_duration_log'], kde = False)
plt.show()
```



In [33]:
```
# dataset checking head of trip_duration_log
df.head()
```

Out[33]:

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

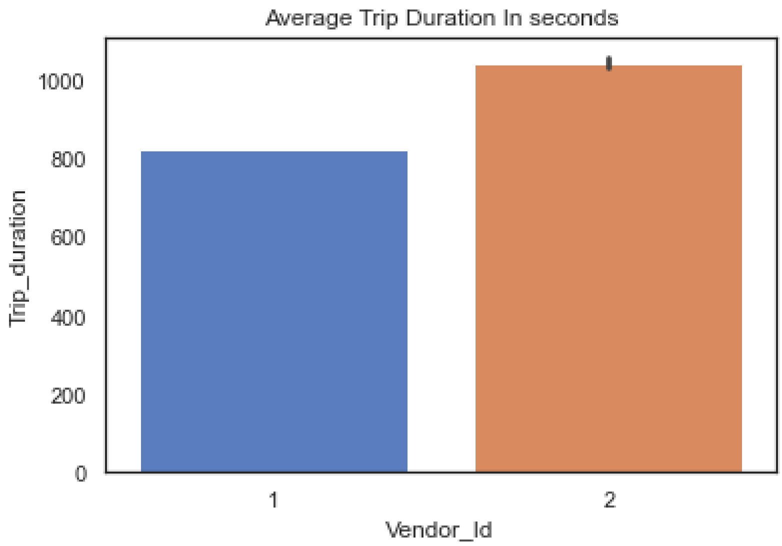
In [34]:
```
# dataset checking tail of trip_duration_log
df.tail()
```

Out[34]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longit
729317	id3905982	2	2016-05-21 13:29:38	2016-05-21 13:34:34	2	-73.96
729318	id0102861	1	2016-02-22 00:43:11	2016-02-22 00:48:26	1	-73.99
729319	id0439699	1	2016-04-15 18:56:48	2016-04-15 19:08:01	1	-73.99
729320	id2078912	1	2016-06-19 09:50:47	2016-06-19 09:58:14	1	-74.00
729321	id1053441	2	2016-01-01 17:24:16	2016-01-01 17:44:40	4	-74.00

Trip Duration vs Vendor Id

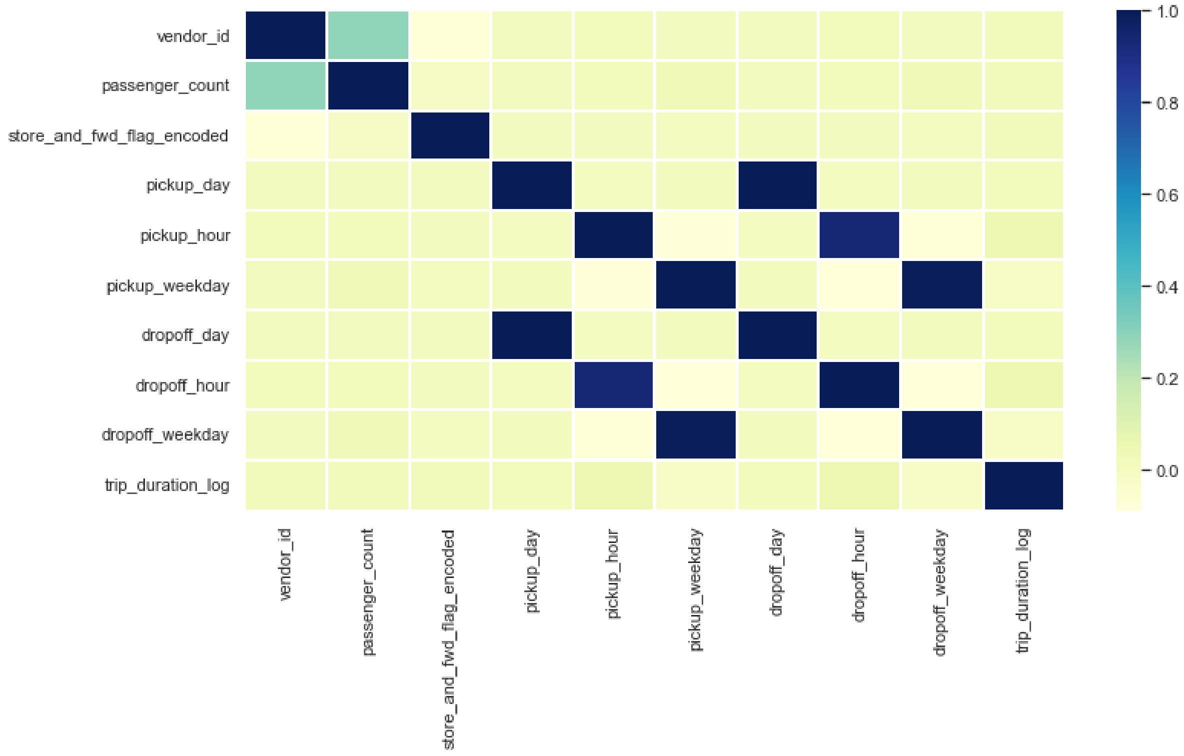
In [35]:
```
# plotting a barplot to know the vendor id vs trip duration

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");
```



Correlation Heatmap

In [36]:
```
df1 = df.drop(columns=['id', 'pickup_datetime', 'dropoff_datetime', 'pickup_longitu
                       'dropoff_longitude',     'dropoff_latitude',     'store_and_
                       'trip_duration_in_hour'])
```

In [37]:
```
# studying the correlation among all the 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")
```



In [38]:
```
df1.head()
```

Out[38]:

	vendor_id	passenger_count	store_and_fwd_flag_encoded	pickup_day	pickup_hour	pickup_wee
0	2	1	0	29	16	
1	1	2	0	11	23	
2	2	2	0	21	17	
3	2	6	0	5	9	
4	1	1	0	17	6	

In [39]:
```
X = df1.drop('trip_duration_log', axis=1)
y = df1['trip_duration_log']
```

Scaling the data

In [40]:
```
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(X)

X = pd.DataFrame(x_scaled, columns=X.columns)
```

In [41]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_st
```

BENCHMARK MODEL

In [42]:
```
# The train and test set for the benchmark model
benchmark_train = pd.concat([X_train, y_train], axis=1, join="inner")
benchmark_test = pd.concat([X_test, y_test], axis=1, join="inner")
```

In [43]:
```
benchmark_train.head()
```

Out[43]:

	vendor_id	passenger_count	store_and_fwd_flag_encoded	pickup_day	pickup_hour	picku
239636	0.0	0.0	0.0	0.200000	0.478261	
121036	1.0	0.0	0.0	0.800000	0.391304	
515707	0.0	0.0	0.0	0.600000	0.956522	
137568	0.0	0.0	0.0	0.133333	0.869565	
145505	0.0	0.0	0.0	0.033333	0.391304	

In [44]:
```
benchmark_test.head()
```

Out[44]:

	vendor_id	passenger_count	store_and_fwd_flag_encoded	pickup_day	pickup_hour	picku
528555	0.0	0.0	0.0	1.000000	0.347826	
191872	1.0	0.2	0.0	0.000000	0.000000	
62203	1.0	0.0	0.0	0.366667	1.000000	
328285	0.0	0.0	0.0	0.666667	0.521739	
425857	0.0	0.0	0.0	0.233333	0.478261	

In [45]:
```
# The value of predicted
benchmark_test['simple_mean'] = benchmark_train['trip_duration_log'].mean()
```

In [46]:
```
# Simple mean model error

error = sqrt(mean_squared_error(benchmark_test['trip_duration_log'], benchmark_tes
print("Score of r-squared simple mean model: ", error)
```

```
Score of r-squared simple mean model:  0.7917401303810214
```

The Error value after the calculation is : 0.7917401303810214 for THE Benchmark Model

K-NN MODEL

In [47]:
```
knnr = KNeighborsRegressor(n_neighbors=5)
knnr.fit(X_train, y_train)
```

Out[47]:
```
KNeighborsRegressor()
```

In [48]:
```
y_pred = knnr.predict(X_test)
error = sqrt(mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error (RMSE) of test k-nn model: ", error)
```

```
Root Mean Squared Error (RMSE) of test k-nn model:  0.7984758569051194
```

Elbow curve to k determine

In [49]:
```
def elbow(k):
  test = []
```

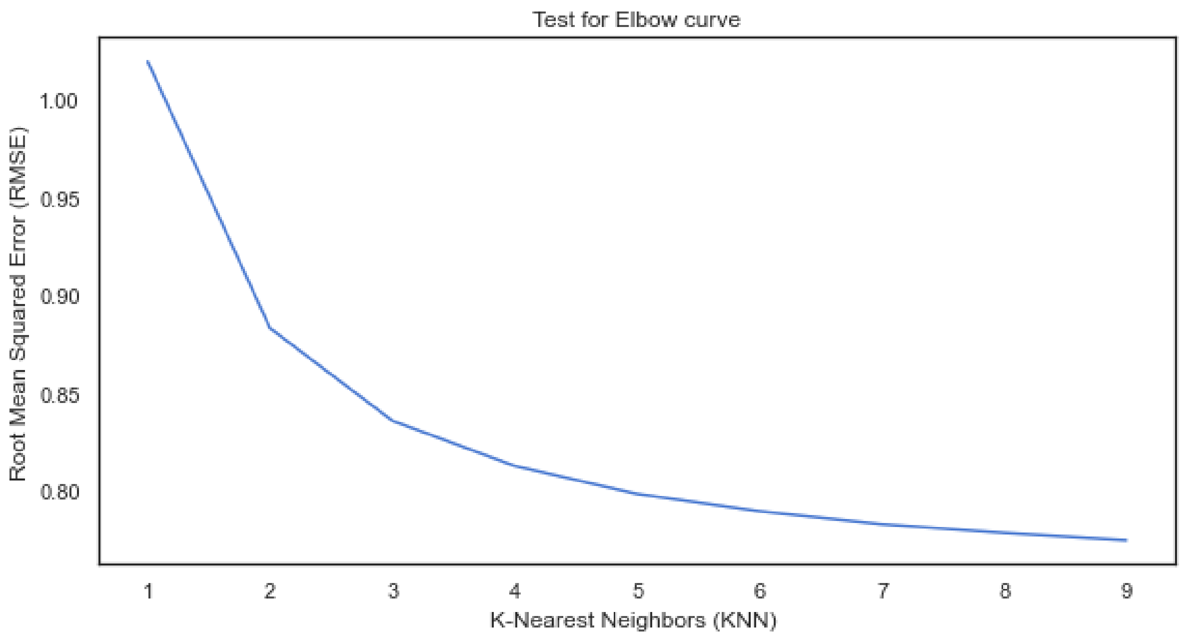
```
    for i in k:
        reg = KNeighborsRegressor(n_neighbors=i)
        reg.fit(X_train, y_train)

        tmp_pred = reg.predict(X_test)
        temp_error = sqrt(mean_squared_error(tmp_pred, y_test))
        test.append(temp_error)

    return test
```

In [50]:
```
k = range(1, 10)
```

In [51]:
```
# calling the elbow function
test = elbow(k)
```

In [52]:
```
# plotting the test elbow curve
plt.figure(figsize=[10, 5])
plt.plot(k, test)
plt.xlabel('K-Nearest Neighbors (KNN) ')
plt.ylabel('Root Mean Squared Error (RMSE)')
plt.title('Test for Elbow curve')
```

Out[52]:
```
Text(0.5, 1.0, 'Test for Elbow curve')
```



In [53]:
```
# studying the error after changing of K-Nearest Neighbors algorithm

knnr = KNeighborsRegressor(n_neighbors=9)
knnr.fit(X_train, y_train)
```

Out[53]:
```
KNeighborsRegressor(n_neighbors=9)
```

In [54]:
```
# Test score
y_pred = knnr.predict(X_test)
knn_test_rmse = sqrt(mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error (RMSE) of knn model: ", knn_test_rmse)
```

```
Root Mean Squared Error (RMSE) of knn model:  0.774831472033997
```

In [55]:
```
# Train score
y_pred = knnr.predict(X_train)
```

```
knn_train_rmse = sqrt(mean_squared_error(y_train, y_pred))
print("Root Mean Squared Error (RMSE) of knn model: ", knn_train_rmse)
```

```
Root Mean Squared Error (RMSE) of knn model:  0.7287635276787346
```

Root Mean Squared Error (RMSE) Value of Train and Test score of KNN model

Test score = 0.774831472033997

Train score = 0.7287635276787346

Linear regression model

In [56]:
```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

Out[56]:
```
LinearRegression()
```

In [57]:
```
# Test score
y_pred = lr.predict(X_test)
lm_test_rmse = sqrt(mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error (RMSE) of linear regressor model: ", lm_test_rmse)
```

```
Root Mean Squared Error (RMSE) of linear regressor model:  0.7919851915355973
```

In [58]:
```
# Train score
y_pred = lr.predict(X_train)
lm_train_rmse = sqrt(mean_squared_error(y_train, y_pred))
print(" Root Mean Squared Error (RMSE) of linear regressor model: ", lm_train_rmse
```

```
 Root Mean Squared Error (RMSE) of linear regressor model:  0.7871162193683029
```

Root Mean Squared Error (RMSE) Value of Train and Test score of Linear regression model

Test Score = 0.7919851915355973

Train Score = 0.7871162193683029

Decision tree model

In [59]:
```
dtr = DecisionTreeRegressor(random_state=42)
dtr.fit(X_train, y_train)
```

Out[59]:
```
DecisionTreeRegressor(random_state=42)
```

In [60]:
```
# Test Score
y_pred = dtr.predict(X_test)
dtr_test_rmse = sqrt(mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error (RMSE) of decision tree regressor model: ", dtr_tes
```

```
Root Mean Squared Error (RMSE) of decision tree regressor model:  0.75517749744449
99
```

In [61]:
```
# Train Score
y_pred = dtr.predict(X_train)
dtr_train_rmse = sqrt(mean_squared_error(y_train, y_pred))
print("Root Mean Squared Error (RMSE) of decision tree regressor model: ", dtr_tra
```

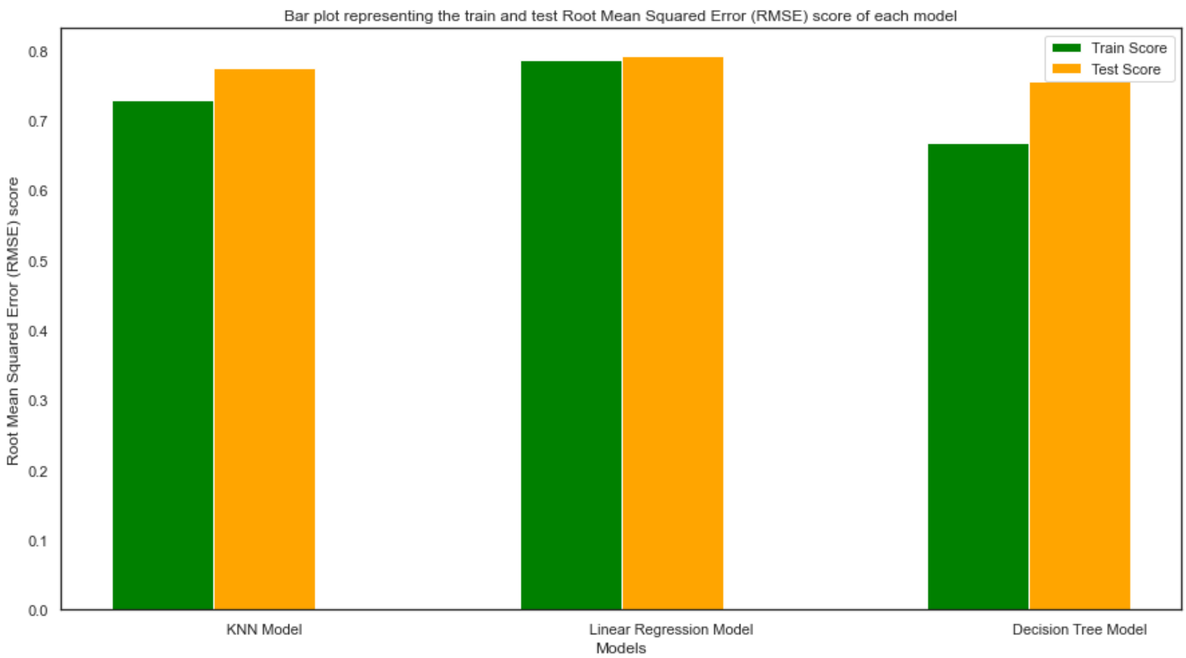
```
Root Mean Squared Error (RMSE) of decision tree regressor model:  0.66731001116152
37
```

Root Mean Squared Error (RMSE) Value of Train and Test score of Decision tree model

Test Score = 0.7551774974444999

Train Score = 0.6673100111615237

In [62]:
```
# Setting the width and height of the (X and Y) axis
plt.figure(figsize=[15, 8])

train_scores = [0.7287, 0.7871, 0.6673]
test_scores = [0.7748, 0.7919, 0.7551]

# To align the side by side bars using X
X = np.arange(len(train_scores))

# To align the side by side bars using X and deploying the colours to indicate
plt.bar(X, train_scores, color = 'green', width = 0.25)
plt.bar(X + 0.25, test_scores, color = 'orange', width = 0.25)

# Implementing the legend of the bars in the graph plot
plt.legend(['Train Score', 'Test Score'])

labels = ['KNN Model', 'Linear Regression Model', 'Decision Tree Model']

# The x axis with the label names
plt.xticks([i + 0.25 for i in range(3)], labels)

# Naming the title of the barplot
plt.title("Bar plot representing the train and test Root Mean Squared Error (RMSE)
# Naming the axis of X and Y
plt.xlabel('Models')
plt.ylabel('Root Mean Squared Error (RMSE) score')

# Displaying the created bar plot
plt.show()
```



In [ ]:

In [ ]:

```

```