In [1]:

```
#Loading Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
test['simple_mean'] = train['trip_duration'].mean()
from sklearn.metrics import mean_squared_error as MSE
```

In [2]:

```
#loding data
data=pd.read_csv('nyc_taxi_trip_duration.csv')
```

In [ ]:

```
#Evaluation Metric used for the models is Mean Squared Error as it is most
```

In [3]:

```
data.head()
```

Out[3]:

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

In [4]:

```
#checking of null values
data.isnull().sum()
```

Out[4]:

```
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 [5]:

```
#dropping unnecessary coloumns
data.drop('id',inplace=True,axis=1)
```

In [6]:

```
data.drop('dropoff_datetime',inplace=True,axis=1)
```

In [7]:

```
1 data.head()
```

Out[7]:

	vendor_id	pickup_datetime	passenger_count	pickup_longitude	pickup_latitud
0	2	2016-02-29 16:40:21	1	-73.953918	40.77887
1	1	2016-03-11 23:35:37	2	-73.988312	40.73174
2	2	2016-02-21 17:59:33	2	-73.997314	40.72145
3	2	2016-01-05 09:44:31	6	-73.961670	40.7597
4	1	2016-02-17 06:42:23	1	-74.017120	40.70846

In [8]:

```
1 #Taking log for easier calculation
2 data['trip_duration']=np.log(data['trip_duration'])
```

In [9]:

```
1 data['passenger_count'].unique()
```

Out[9]:

```
array([1, 2, 6, 3, 4, 5, 0, 7, 9], dtype=int64)
```

In [10]:

```
1 data.drop(data[data['passenger_count'] ==0].index, inplace = True)
```

In [11]:

```
1 data.head()
```

Out[11]:

	vendor_id	pickup_datetime	passenger_count	pickup_longitude	pickup_latitud
0	2	2016-02-29 16:40:21	1	-73.953918	40.77887
1	1	2016-03-11 23:35:37	2	-73.988312	40.73174
2	2	2016-02-21 17:59:33	2	-73.997314	40.72145
3	2	2016-01-05 09:44:31	6	-73.961670	40.75972
4	1	2016-02-17 06:42:23	1	-74.017120	40.70846

In [12]:

```
1 data['passenger_count'].value_counts()
```

Out[12]:

```
1    517415
2    105097
5     38926
3     29692
6     24107
4     14050
7         1
9         1
Name: passenger_count, dtype: int64
```

In [13]:

```
1 data["pickup_datetime"]=pd.to_datetime(data["pickup_datetime"])
```

In [14]:

```
#Extracting Month
data['Month'] = data['pickup_datetime'].dt.month

#Extracting day of month
data['DayofMonth'] = data['pickup_datetime'].dt.day

#Extracting day of week
data['dayofweek'] =data['pickup_datetime'].dt.dayofweek


#Extracting hour of the day
data['Hour'] = data['pickup_datetime'].dt.hour
```

In [15]:

```

# Converting yes/no flag to 1 and 0


data['store_and_fwd_flag']=data['store_and_fwd_flag'].map({'N':0,'Y':1})

```

In [16]:

```
import math
```

In [17]:

```
def haversine(lat1, lon1, lat2, lon2):

    # distance between latitudes
    # and longitudes
    dLat = (lat2 - lat1) * math.pi / 180.0
    dLon = (lon2 - lon1) * math.pi / 180.0

    # convert to radians
    lat1 = (lat1) * math.pi / 180.0
    lat2 = (lat2) * math.pi / 180.0

    # apply formulae
    a = (pow(math.sin(dLat / 2), 2) +
         pow(math.sin(dLon / 2), 2) *
             math.cos(lat1) * math.cos(lat2));
    rad = 6371
    c = 2 * math.asin(math.sqrt(a))
    return rad * c
```

In [18]:

```
#changing longitude and latitiude to distance
data['distance']=data.apply(lambda row:haversine(row['pickup_latitude'],row
data['distance']=data['distance'].astype(float)
```

In [19]:

```
1 data.head()
```

Out[19]:

	vendor_id	pickup_datetime	passenger_count	pickup_longitude	pickup_latitud
0	2	2016-02-29 16:40:21	1	-73.953918	40.77887
1	1	2016-03-11 23:35:37	2	-73.988312	40.73174
2	2	2016-02-21 17:59:33	2	-73.997314	40.72145
3	2	2016-01-05 09:44:31	6	-73.961670	40.75972
4	1	2016-02-17 06:42:23	1	-74.017120	40.70846

In [20]:

```
1 #droppiong longitude and latitude
2 data.drop('pickup_longitude',inplace=True,axis=1)
3 data.drop('pickup_latitude',inplace=True,axis=1)
4 data.drop('dropoff_longitude',inplace=True,axis=1)
5 data.drop('dropoff_latitude',inplace=True,axis=1)
6 data.drop('pickup_datetime',inplace=True,axis=1)
```

In [21]:

```
1 data.head()
```

Out[21]:

	vendor_id	passenger_count	store_and_fwd_flag	trip_duration	Month	DayofM
0	2	1	0	5.991465	2	
1	1	2	0	7.003065	3	
2	2	2	0	7.399398	2	
3	2	6	0	7.039660	1	
4	1	1	0	6.742881	2	

In [22]:

```
#getting rid of outliers
data.drop(data[data['distance'] > 200].index, inplace = True)
```

In [23]:

```
data.drop('store_and_fwd_flag',inplace=True,axis=1)
```

In [24]:

```
data.head()
```

Out[24]:

	vendor_id	passenger_count	trip_duration	Month	DayofMonth	dayofweek	H
0	2	1	5.991465	2	29	0	
1	1	2	7.003065	3	11	4	
2	2	2	7.399398	2	21	6	
3	2	6	7.039660	1	5	1	
4	1	1	6.742881	2	17	2	

In [25]:

```
data.shape
```

Out[25]:

```
(729283, 8)
```

In [26]:

```
from sklearn.utils import shuffle

# Shuffling the Dataset
data = shuffle(data, random_state = 42)

#creating 4 divisions
div = int(data.shape[0]/4)

# 3 parts to train set and 1 part to test set
train = data.loc[:3*div+1,:]
test = data.loc[3*div+1:]
```

In [27]:

```
1 train.head()
```

Out[27]:

	vendor_id	passenger_count	trip_duration	Month	DayofMonth	dayofwe
267686	1	1	6.588926	6	3	
217626	2	2	6.386879	6	25	
540092	1	1	5.407172	4	19	
488315	2	1	6.614726	5	27	
529358	2	1	5.583496	6	28	

In [28]:

```
1 train.shape
```

Out[28]:

```
(570718, 8)
```

In [29]:

```
1 test.head()
```

Out[29]:

	vendor_id	passenger_count	trip_duration	Month	DayofMonth	dayofwe
546961	2	1	5.590987	2	10	
708968	2	6	6.440947	5	24	
688012	1	1	6.717805	3	9	
673321	1	1	5.937536	2	19	
556530	2	2	6.302619	4	8	

In [30]:

```
1 test.shape
```

Out[30]:

```
(158566, 8)
```

In [31]:

```
1
```

In [32]:

```
1 #calculating Error for Simple Mean
2
3 simple_mean_error = MSE(test['trip_duration'] , test['simple_mean'])
4 simple_mean_error
```

Out[32]:

```
0.6414769712789729
```

In [33]:

```
hour= pd.pivot_table(train, values='trip_duration', index = ['Hour'], aggfu
hour
```

Out[33]:

Hour	trip_duration
0	6.419300
1	6.375759
2	6.325873
3	6.316918
4	6.319188
5	6.248041
6	6.185225
7	6.346175
8	6.462101
9	6.480601
10	6.486776
11	6.517891
12	6.517866
13	6.531802
14	6.570180
15	6.566781
16	6.543516
17	6.532548
18	6.504891
19	6.440187
20	6.416309
21	6.432378
22	6.470023
23	6.464092

In [34]:

```
test['hour_mean'] = 0

# For every unique entry in Hour
for i in train['Hour'].unique():
  # Assign the mean value corresponding to unique entry
  test['hour_mean'][test['Hour'] == i] = train['trip_duration'][train['Hou
```

In [35]:

```
#calculating mean squared error
hour_error = MSE(test['trip_duration'] , test['hour_mean'] )
hour_error
```

Out[35]:

```
0.6348134684473961
```

In [36]:

```
weekday= pd.pivot_table(train, values='trip_duration', index = ['dayofweek
weekday
```

Out[36]:

dayofweek	trip_duration
0	6.425561
1	6.489393
2	6.516227
3	6.523728
4	6.500830
5	6.417368
6	6.371678

In [37]:

```
test['weekday_mean'] = 0

# For every unique entry in dayofweek
for i in train['dayofweek'].unique():
  # Assign the mean value corresponding to unique entry
  test['weekday_mean'][test['dayofweek'] == i] = train['trip_duration'][tra
```

In [38]:

```
#calculating mean squared error
weekday_error = MSE(test['trip_duration'] , test['weekday_mean'] )
weekday_error
```

Out[38]:

```
0.6386881642731397
```

In [39]:

```
#seperating independent and dependent variables
x = data.drop(['trip_duration'], axis=1)
y = data['trip_duration']
x.shape, y.shape
```

Out[39]:

```
((729283, 7), (729283,))
```

In [40]:

```
# Importing MinMax Scaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)
```

In [41]:

```
x = pd.DataFrame(x_scaled)
```

In [42]:

```
# Importing Train test split
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(x,y, random_state = 56)
```

In [43]:

```
#importing KNN regressor
from sklearn.neighbors import KNeighborsRegressor as KNN
```

In [44]:

```
reg = KNN(n_neighbors = 5)

# Fitting the model
reg.fit(train_x, train_y)

# Predicting over the Train Set and calculating MSE
test_predict = reg.predict(test_x)
k = MSE(test_predict, test_y)
print('Test MSE    ', k )
```

```
Test MSE     0.3201870115350543
```

In [45]:

```
def Elbow(K):
  #initiating empty list
    test_mse = []

  #training model for evey value of K
    for i in K:
        #Instance of KNN
        reg = KNN(n_neighbors = i)
        reg.fit(train_x, train_y)
        #Appending mse value to empty list claculated using the predictions
        tmp = reg.predict(test_x)
        tmp = MSE(tmp,test_y)
        test_mse.append(tmp)

    return test_mse
```

In [46]:

```
#Defining K range
k = range(1,11,2)
```

In [47]:

```
# calling above defined function
test = Elbow(k)
```

In [48]:

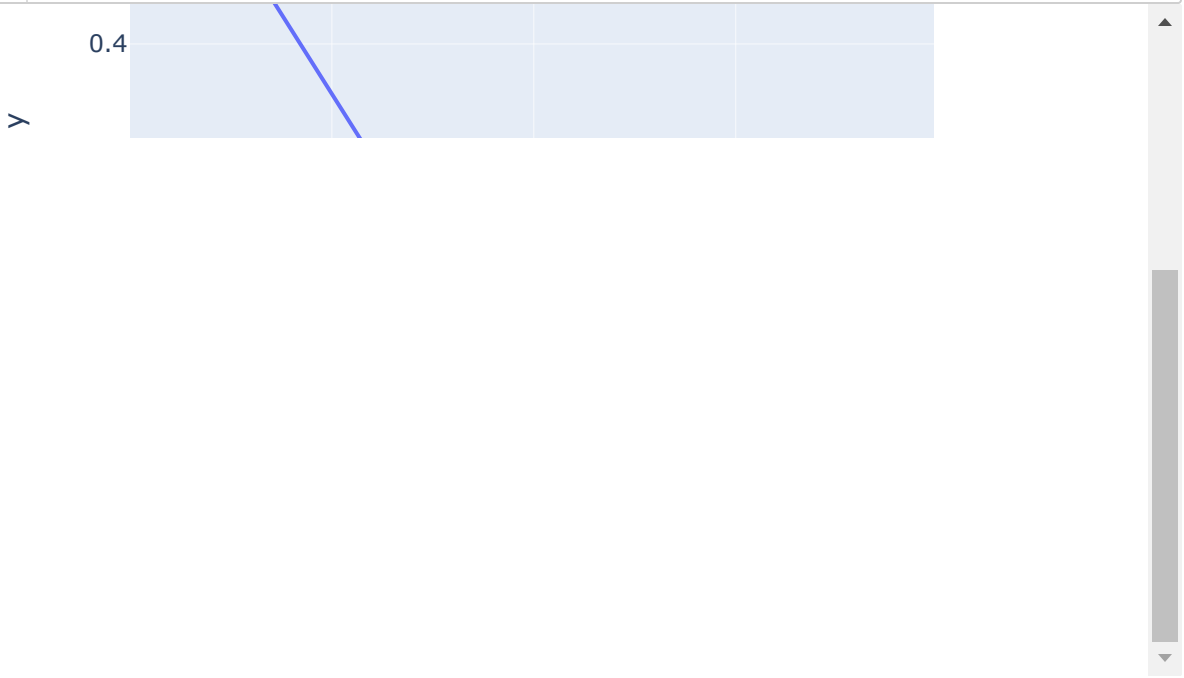
```
import plotly.express as px
```

In [49]:

```
# plotting the Curves
fig=px.line(x=k,y=test)
```

In [50]:

```
fig.show()
```



In [51]:

```
reg = KNN(n_neighbors = 5)
# Fitting the model
reg.fit(train_x, train_y)

# Predicting over the Train Set and calculating MSE
train_predict = reg.predict(train_x)
train_k = MSE(train_predict, train_y)
```

In [52]:

```
print('Train MSE   ', train_k)
```

```
Train MSE    0.20800164109882655
```

In [205]:

```
reg = KNN(n_neighbors = 5)
# Fitting the model
reg.fit(train_x, train_y)

# Predicting over the Test Set and calculating MSE
test_predict = reg.predict(test_x)
test_k = MSE(test_predict, test_y)
print('Test MSE    ', test_k )
```

```
Test MSE     0.26665160784965847
```

In [54]:

```
data.head()
```

Out[54]:

	vendor_id	passenger_count	trip_duration	Month	DayofMonth	dayofwe
267686	1	1	6.588926	6	3	
217626	2	2	6.386879	6	25	
540092	1	1	5.407172	4	19	
488315	2	1	6.614726	5	27	
529358	2	1	5.583496	6	28	

In [181]:

```
#seperating independent and dependent variables
x = data.drop(['trip_duration'], axis=1)
y = data['trip_duration']
x.shape, y.shape
```

Out[181]:

```
((729283, 7), (729283,))
```

In [182]:

```
# Importing the train test split function
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(x,y, random_state = 56)
```

In [183]:

```
#importing Linear Regression
from sklearn.linear_model import LinearRegression as LR
```

In [184]:

```
# Creating instance of Linear Regresssion
lr = LR()

# Fitting the model
lr.fit(train_x, train_y)
```

Out[184]:

```
LinearRegression()
```

In [185]:

```
# Predicting over the Train Set and calculating error
train_predict = lr.predict(train_x)
train_lr_k = MSE(train_predict, train_y)
print('Training Mean Square Error', train_lr_k )
```

```
Training Mean Square Error 0.38964204246525314
```

In [186]:

```
# Predicting over the Test Set and calculating error
test_predict = lr.predict(test_x)
test_lr_k = MSE(test_predict, test_y)
print('Test Mean Absolute Error     ', test_lr_k)
```

```
Test Mean Absolute Error     0.38458672319254134
```

In [187]:

```
# Importing ridge from sklearn's linear_model module
from sklearn.linear_model import Ridge
```

In [188]:

```
rr = Ridge(alpha=0.1)
```

In [189]:

```
rr.fit(train_x, train_y)
```

Out[189]:

```
Ridge(alpha=0.1)
```

In [190]:

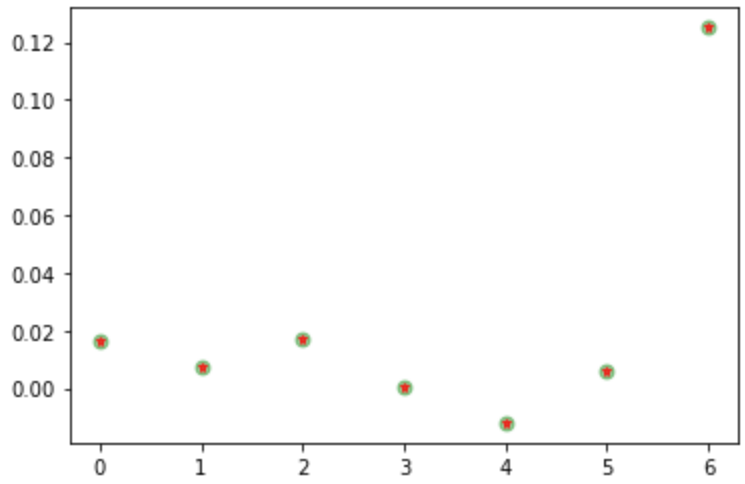
```
train_score=lr.score(train_x, train_y)
test_score=lr.score(test_x, test_y)
Ridge_train_score = rr.score(train_x,train_y)
Ridge_test_score = rr.score(test_x, test_y)

```

In [191]:

```
plt.plot(rr.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color=
plt.plot(lr.coef_,alpha=0.4,linestyle='none',marker='o',markersize=7,color=
```

Out[191]:

```
[<matplotlib.lines.Line2D at 0x160982da2e0>]
```



In [152]:

```
rr.coef_
```

Out[152]:

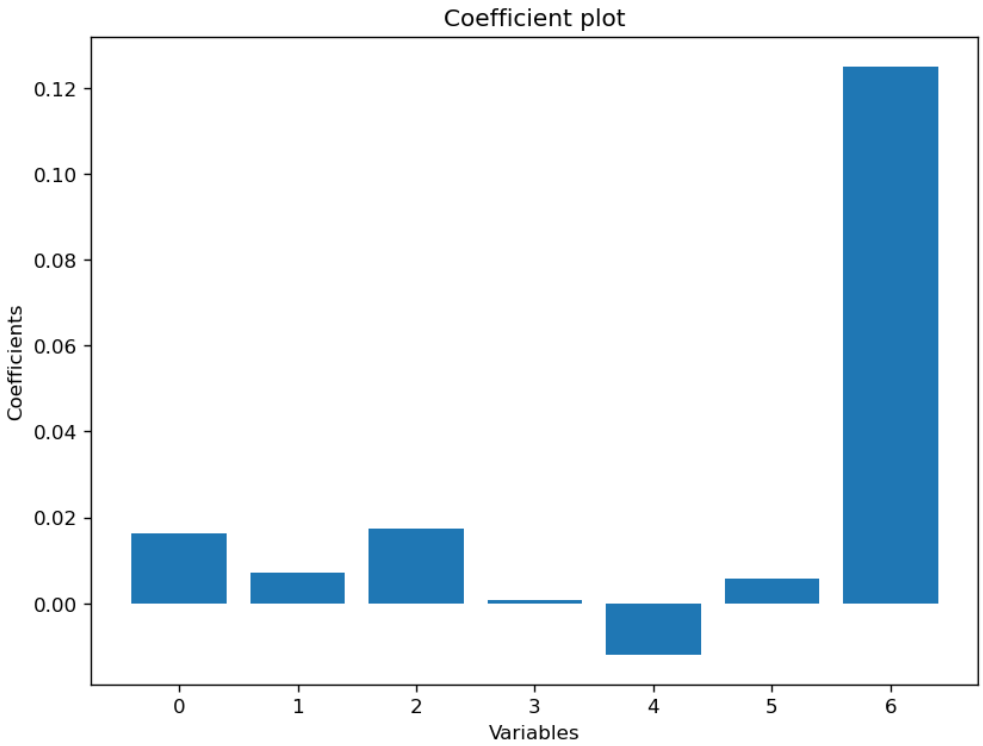
```
array([ 0.016295  ,  0.00716117,  0.01730813,  0.0006261 , -0.01
200293,
        0.00583826,  0.1250478 ])
```

In [153]:

```
plt.figure(figsize=(8, 6), dpi=120, facecolor='w', edgecolor='b')
x = range(len(train_x.columns))
y = lr.coef_
plt.bar( x, y )
plt.xlabel( "Variables")
plt.ylabel('Coefficients')
plt.title('Coefficient plot')
```

Out[153]:

```
Text(0.5, 1.0, 'Coefficient plot')
```



In [154]:

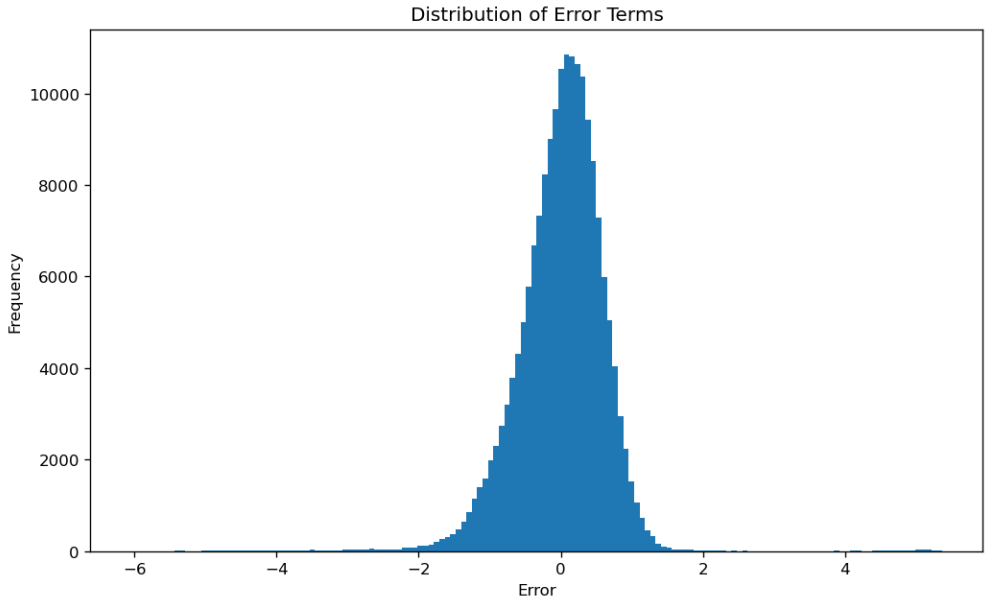
```
# Arranging and calculating the Residuals
residuals = pd.DataFrame({
    'fitted values' : test_y,
    'predicted values' : test_predict,
})

residuals['residuals'] = residuals['fitted values'] - residuals['predicted
residuals.head()
```

Out[154]:

	fitted values	predicted values	residuals
347029	6.1	6.2	-0.026
397789	8.2	7.2	0.93
122614	7.5	6.6	0.89
603659	5.6	6.3	-0.72
259950	5.1	6	-0.95

In [173]:

```
# Histogram for distribution
plt.figure(figsize=(10, 6), dpi=120, facecolor='w', edgecolor='b')
plt.hist(residuals.residuals, bins = 150)
plt.xlabel('Error')
plt.ylabel('Frequency')
plt.title('Distribution of Error Terms')
plt.show()
```



In [156]:

```
# importing the QQ-plot from the from the statsmodels
from statsmodels.graphics.gofplots import qqplot

## Plotting the QQ plot
fig, ax = plt.subplots(figsize=(5,5) , dpi = 120)
qqplot(residuals.residuals, line = 's' , ax = ax)
plt.ylabel('Residual Quantiles')
plt.xlabel('Ideal Scaled Quantiles')
plt.title('Checking distribution of Residual Errors')
plt.show()
```

In [157]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# Calculating VIF for every column (only works for the not Catagorical)
VIF = pd.Series([variance_inflation_factor(data.values, i) for i in range(
VIF
```

Out[157]:

```
vendor_id          11
passenger_count   2.8
trip_duration      25
Month             5.3
DayofMonth        4.1
dayofweek         3.3
Hour              5.4
distance          2.2
dtype: float64
```

In [158]:

```
lr = LR(normalize = True)

# Fitting the model
lr.fit(train_x, train_y)
```

Out[158]:

```
LinearRegression(normalize=True)
```

In [193]:

```
train_predict = lr.predict(train_x)
train_k_normal = MSE(train_predict, train_y)
print('Training Mean Absolute Error', k )
```

```
Training Mean Absolute Error 0.38516877731830523
```

In [194]:

```
test_predict = lr.predict(test_x)
test_k_normal = MSE(test_predict, test_y)
print('Test Mean Absolute Error    ', k )
```

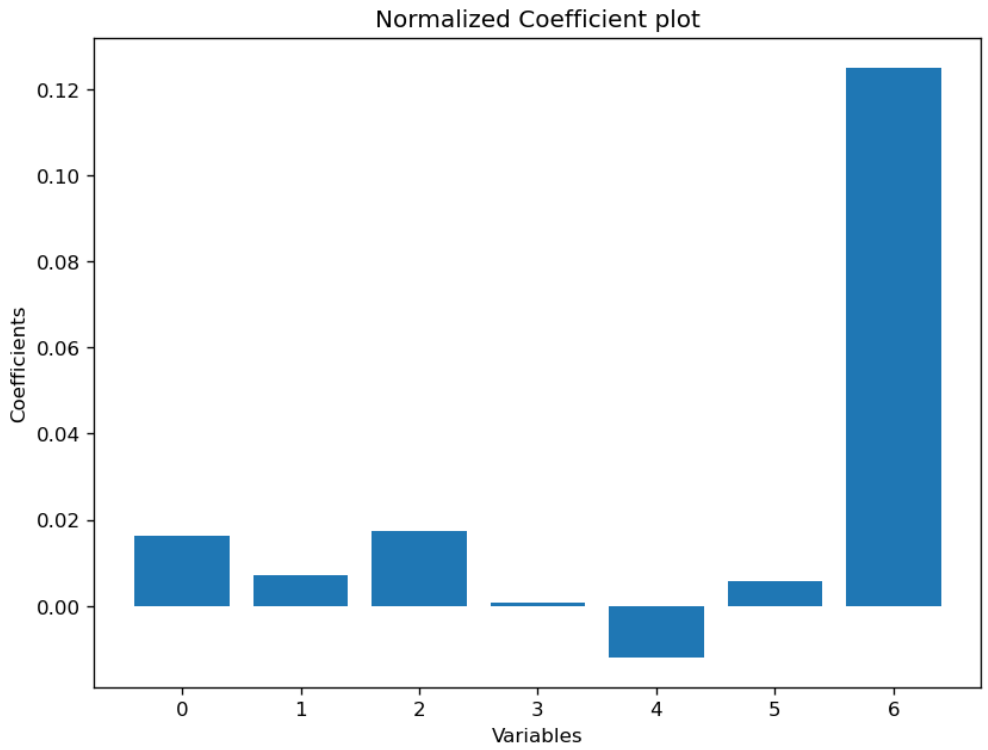
```
Test Mean Absolute Error     0.38516877731830523
```

In [161]:

```
plt.figure(figsize=(8, 6), dpi=120, facecolor='w', edgecolor='b')
x = range(len(train_x.columns))
y = lr.coef_
plt.bar( x, y )
plt.xlabel( "Variables")
plt.ylabel('Coefficients')
plt.title('Normalized Coefficient plot')
```

Out[161]:

```
Text(0.5, 1.0, 'Normalized Coefficient plot')
```



In [162]:

```
x = data.drop(['trip_duration'], axis=1)
y = data['trip_duration']
x.shape, y.shape
```

Out[162]:

```
((729283, 7), (729283,))
```

In [163]:

```
Coefficients = pd.DataFrame({
    'Variable'    : x.columns,
    'coefficient' : lr.coef_
})
Coefficients
```

Out[163]:

	Variable	coefficient
0	vendor_id	0.016
1	passenger_count	0.0072
2	Month	0.017
3	DayofMonth	0.00063
4	dayofweek	-0.012
5	Hour	0.0058
6	distance	0.13

In [164]:

```
sig_var = Coefficients[Coefficients.coefficient > 0.001]
```

In [165]:

```
subset = data[sig_var['Variable'].values]
subset.head()
```

Out[165]:

	vendor_id	passenger_count	Month	Hour	distance
267686	1	1	6	9	0.88
217626	2	2	6	11	3.5
540092	1	1	4	7	0.96
488315	2	1	5	17	1.9
529358	2	1	6	16	1.9

In [166]:

```
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(subset, y , random_state =
```

In [167]:

```
from sklearn.linear_model import LinearRegression as LR

```

In [168]:

```
lr = LR(normalize = True)

# Fitting the model
lr.fit(train_x, train_y)
```

Out[168]:

```
LinearRegression(normalize=True)
```

In [196]:

```
train_predict = lr.predict(train_x)
coef_adj_train_k = MSE(train_predict, train_y)
print('Training Mean Absolute Error', k )
```

```
Training Mean Absolute Error 0.38516877731830523
```

In [197]:

```
test_predict = lr.predict(test_x)
coef_adj_test_k = MSE(test_predict, test_y)
print('Test Mean Absolute Error    ', k )
```

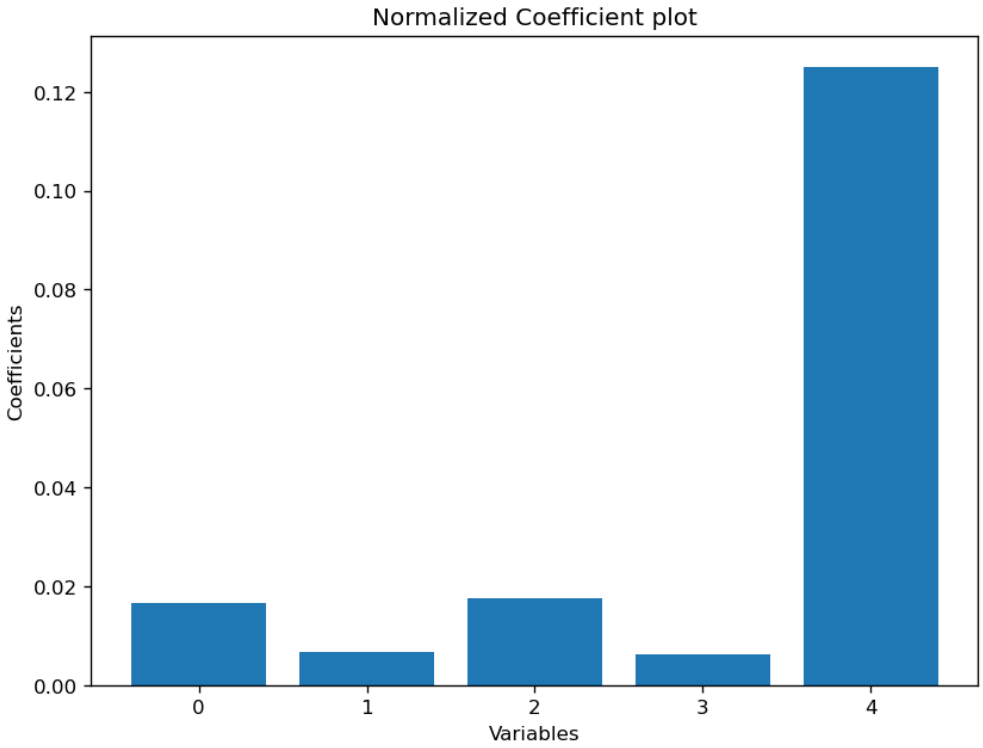
```
Test Mean Absolute Error     0.38516877731830523
```

In [171]:

```
plt.figure(figsize=(8, 6), dpi=120, facecolor='w', edgecolor='b')
x = range(len(train_x.columns))
y = lr.coef_
plt.bar( x, y )
plt.xlabel( "Variables")
plt.ylabel('Coefficients')
plt.title('Normalized Coefficient plot')
```

Out[171]:

```
Text(0.5, 1.0, 'Normalized Coefficient plot')
```



In [202]:

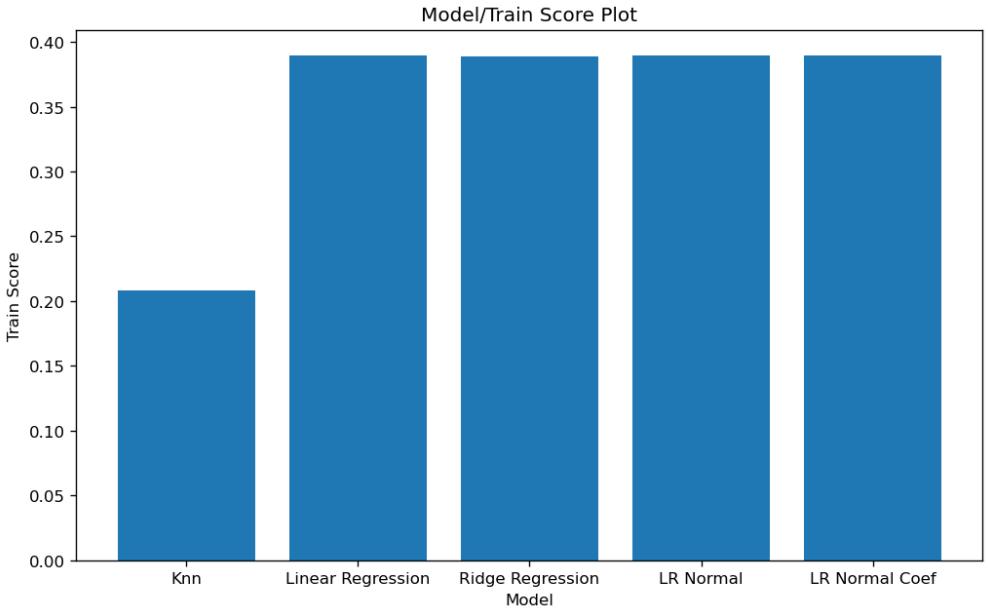
```
model=['Knn','Linear Regression','Ridge Regression','LR Normal','LR Normal
train_score=[train_k,train_lr_k,Ridge_train_score,train_k_normal,coef_adj_
```

In [203]:

```
plt.figure(figsize=(10, 6), dpi=120, facecolor='w', edgecolor='b')
plt.bar( model,train_score )
plt.xlabel( "Model")
plt.ylabel('Train Score')
plt.title('Model/Train Score Plot')
```

Out[203]:

```
Text(0.5, 1.0, 'Model/Train Score Plot')
```

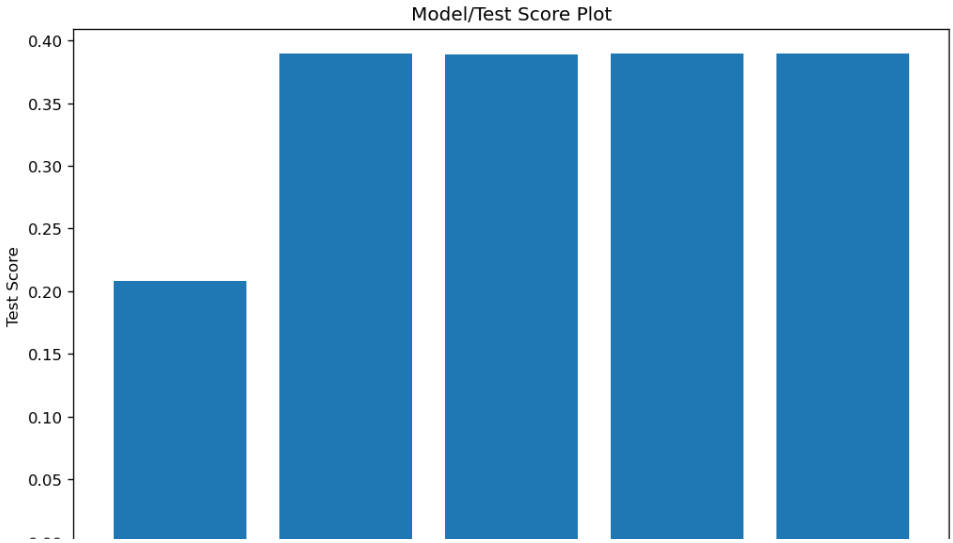


In [206]:

```
model=['Knn','Linear Regression','Ridge Regression','LR Normal','LR Normal
test_score=[test_k,test_lr_k,Ridge_test_score,test_k_normal,coef_adj_test_l
plt.figure(figsize=(10, 6), dpi=120, facecolor='w', edgecolor='b')
plt.bar( model,train_score )
plt.xlabel( "Model")
plt.ylabel('Test Score')
plt.title('Model/Test Score Plot')
```

Out[206]:

```
Text(0.5, 1.0, 'Model/Test Score Plot')
```



In [ ]:

```
1
```