Build a K-Nearest neighbours' model for the given dataset and find the best value of K.

Importing Libraries

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

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

Load the data

In [2]:
```
data = pd.read_csv('nyc_taxi_trip_duration.csv')
data.shape
```

Out[2]: (729322, 11)

In [3]:
```
data.head()
```

Out[3]:

	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 [4]:
```
def UVA_outlier(data, var):

    # calculating descriptives of variable
    quant25 = data[var].quantile(0.25)
    quant75 = data[var].quantile(0.75)
    IQR = quant75 - quant25
    med = data[var].median()
    whis_low = quant25-(1.5*IQR)
    whis_high = quant75+(1.5*IQR)

    ls = data.index[(data[var] < whis_low) | (data[var] > whis_high)]

    return ls
```

In [5]:
```
def remove(df,ls):
    ls = sorted(set(ls))
    df = df.drop(ls)
    return df
```

In [6]:
```
# import pdb
index_list1 = []

# for j in data.drop(['id','vendor_id','pickup_datetime','dropoff_datetime','store_and_fwd_flag'], axis=1).columns:
for j in ['trip_duration']:#,'pickup_longitude','dropoff_longitude','pickup_latitude','dropoff_latitude']:
# for j in data.columns:
#     pdb.set_trace()
    for i in [j]:
        index_list1.extend(UVA_outlier(data,i))
        data_cleaned = remove(data,index_list1)
        index_list1.clear()
```

In [7]:
```
data_cleaned.shape
```

Out[7]: (692359, 11)

In [8]:
```
data_cleaned.head()
```

Out[8]:

	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 [9]:
```
data = data_cleaned
```

In [10]:
```
data.shape
```

Out[10]: (692359, 11)

In [11]:
```
# creating an instance(date) of DatetimeIndex class using "pickup_datetime"
date_pick = pd.DatetimeIndex(data['pickup_datetime'])
# creating an instance(date) of DatetimeIndex class using "dropoff_datetime"
date_drop = pd.DatetimeIndex(data['dropoff_datetime'])

# extracting new columns from "pick datetime"

# last day of year when pickup was done
data['doy_pick'] = date_pick.dayofyear

# week of year when pickup was done
data['woy_pick'] = date_pick.weekofyear

# month of year when pickup was done
data['moy_pick'] = date_pick.month

# day of week when pickup was done
data['dow_pick'] = date_pick.dayofweek

# hour of day when pickup was done
data['hod_pick'] = date_pick.hour


# extracting new columns from "dropoff datetime"

# last day of year dropoff was done
data['doy_drop'] = date_drop.dayofyear

# week of year when dropoff was done
data['woy_drop'] = date_drop.weekofyear

# month of year when dropoff was done
data['moy_drop'] = date_drop.month

# day of week when dropoff was done
data['dow_drop'] = date_drop.dayofweek

# hour of day when dropoff was done
data['hod_drop'] = date_drop.hour
```

In [12]:
```
data = pd.get_dummies(data.drop('id',axis=1), columns = ['store_and_fwd_flag'])
```

In [13]:
```
data.tail()
```

Out[13]:

	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration	doy_pick	...	moy_pick	dow_pick	hod_pick	doy_
729317	2	2016-05-21 13:29:38	2016-05-21 13:34:34	2	-73.965919	40.789780	-73.952637	40.789181	296	142	...	5	5	13	
729318	1	2016-02-22 00:43:11	2016-02-22 00:48:26	1	-73.996666	40.737434	-74.001320	40.731911	315	53	...	2	0	0	
729319	1	2016-04-15 18:56:48	2016-04-15 19:08:01	1	-73.997849	40.761696	-74.001488	40.741207	673	106	...	4	4	18	
729320	1	2016-06-19 09:50:47	2016-06-19 09:58:14	1	-74.006706	40.708244	-74.013550	40.713814	447	171	...	6	6	9	
729321	2	2016-01-01 17:24:16	2016-01-01 17:44:40	4	-74.003342	40.743839	-73.945847	40.712841	1224	1	...	1	4	17	

5 rows × 21 columns

Segregating variables: Independent and Dependent Variables

In [14]:
```
#seperating independent and dependent variables
# we have already taken required information from pickup_datetime column and placed in separate columns - so we dont need them
# we dont need dropoff_datetime since we have pickup and duration- removed to improve code processing speed
x = data.drop(['trip_duration','pickup_datetime','dropoff_datetime'], axis=1)
y = data['trip_duration']
x.shape, y.shape
```

Out[14]: ((692359, 18), (692359,))

In [15]:
```
x.head()
```

Out[15]:

	vendor_id	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	doy_pick	woy_pick	moy_pick	dow_pick	hod_pick	doy_drop	woy_drop	moy_drop	dow_drop	hod_dr
0	2	1	-73.953918	40.778873	-73.963875	40.771164	60	9	2	0	16	60	9	2	0	
1	1	2	-73.988312	40.731743	-73.994751	40.694931	71	10	3	4	23	71	10	3	4	
2	2	2	-73.997314	40.721458	-73.948029	40.774918	52	7	2	6	17	52	7	2	6	
3	2	6	-73.961670	40.759720	-73.956779	40.780628	5	1	1	1	9	5	1	1	1	
4	1	1	-74.017120	40.708469	-73.988182	40.740631	48	7	2	2	6	48	7	2	2	

In [16]:
```
y.head()
```

Out[16]:
```
0     400
1    1100
2    1635
3    1141
4     848
Name: trip_duration, dtype: int64
```

Scaling the data (Using MinMax Scaler)

In [17]:
```
# Importing MinMax Scaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)
```

In [18]:
```
x = pd.DataFrame(x_scaled)
```

In [19]:
```
x.head()
```

Out[19]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	1.0	0.111111	0.856226	0.353352	0.856048	0.732273	0.325967	0.153846	0.2	0.000000	0.695652	0.324176	0.153846	0.166667	0.000000	0.695652	1.0	0.0
1	0.0	0.222222	0.855612	0.350606	0.855497	0.725775	0.386740	0.173077	0.4	0.666667	1.000000	0.384615	0.173077	0.333333	0.666667	1.000000	1.0	0.0
2	1.0	0.222222	0.855451	0.350007	0.856331	0.732593	0.281768	0.115385	0.2	1.000000	0.739130	0.280220	0.115385	0.166667	1.000000	0.782609	1.0	0.0
3	1.0	0.666667	0.856087	0.352236	0.856174	0.733080	0.022099	0.000000	0.0	0.166667	0.391304	0.021978	0.000000	0.000000	0.166667	0.434783	1.0	0.0
4	0.0	0.111111	0.855098	0.349251	0.855614	0.729671	0.259669	0.115385	0.2	0.333333	0.260870	0.258242	0.115385	0.166667	0.333333	0.260870	1.0	0.0

In [20]:
```
# 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)
```

Implementing KNN Regressor

In [21]:
```
#importing KNN regressor and metric mse

from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error as MAE
```

In [22]:
```
# Creating instance of KNN
reg = KNN(n_neighbors = 5)

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

Out[22]:
```
KNeighborsRegressor()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [23]:
```
# Predicting over the Train Set and calculating MSE
test_predict = reg.predict(test_x)
k = MAE(test_predict, test_y)
print('Test MAE    ', k )
```

```
Test MAE     326.99994222658734
```

Elbow for Classifier

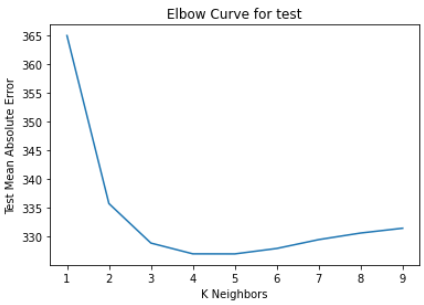
In [24]:
```
def Elbow(K):
  #initiating empty list
    test_mae = []

  #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 = MAE(tmp,test_y)
        test_mae.append(tmp)

    return test_mae
```

In [25]:
```
#Defining K range
k = range(1,10)
```

In [26]:
```
# calling above defined function
test = Elbow(k)
```

In [27]:
```
# plotting the Curves
plt.plot(k, test)
plt.xlabel('K Neighbors')
plt.ylabel('Test Mean Absolute Error')
plt.title('Elbow Curve for test')
```

Out[27]:
```
Text(0.5, 1.0, 'Elbow Curve for test')
```



In [28]:
```
# Creating instance of KNN
reg = KNN(n_neighbors = 4)

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

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

```
Test MAE     327.01053064879545
```

so the best value for K could be 4

In [29]:
```
# Over fitting and Under fitting
```

In [30]:
```
# Creating instance of KNN
reg = KNN(n_neighbors = 4)

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

# Predicting over the Train Set and calculating MAE
train_predict = reg.predict(train_x)
k = MAE(train_predict, train_y)
print('Train MAE    ', k )

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

```
Train MAE     245.92629831551662
Test MAE     327.01053064879545
```

Checking Consistency , using Cross Validation

In [31]:
```
from sklearn.model_selection import cross_val_score
score = cross_val_score( KNN(n_neighbors = 4), X = train_x, y = train_y, cv = 10)
score
```

Out[31]:
```
array([0.06967275, 0.06434958, 0.06005126, 0.06393296, 0.07181012,
       0.06466215, 0.0587845 , 0.0571514 , 0.06409931, 0.06229975])
```

In [32]:
```
# Consistency using Mean and standard deviation in percentage
score.mean()*100, score.std()*100
```

Out[32]:
```
(6.368137739609385, 0.43049505484479067)
```

Automating the process of cross validation for different K-Neighbors

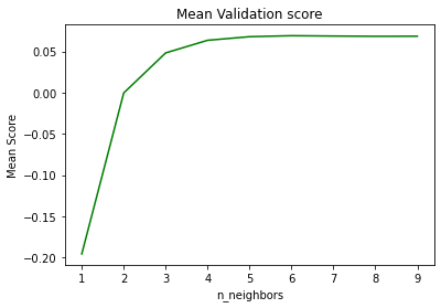
In [33]:
```
def Val_score(n_neighbors):
    '''
  takes range of n_neighbors as input
  returns Mean and Standard Deviation for each value of n_neighbors
  '''
    avg = []
    std = []

    for i in n_neighbors:

    # 10 fold cross validation for every value of n_neighbor
      score = cross_val_score( KNN(n_neighbors = i) , X = train_x, y = train_y, cv = 10)

    # adding mean to avg list
      avg.append(score.mean())

    # adding standard deviation to std list
      std.append(score.std())

    return avg, std
```

In [47]:
```
n_neighbors = range(1,10)
mean, std = Val_score(n_neighbors)
```

Ploting Mean Validation Score for each K value

In [48]:
```
plt.plot(n_neighbors, mean, color = 'green', label = 'mean' )
plt.xlabel('n_neighbors')
plt.ylabel('Mean Score')
plt.title('Mean Validation score')
```

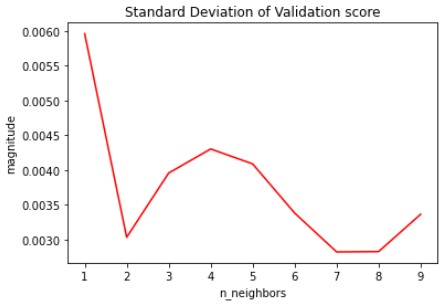
Out[48]:
```
Text(0.5, 1.0, 'Mean Validation score')
```



Ploting Standard Deaviation Validation Score for each K value

In [49]:
```
plt.plot(n_neighbors, std, color = 'red', label = 'Standard deviation' )
plt.xlabel('n_neighbors')
plt.ylabel('magnitude')
plt.title('Standard Deviation of Validation score')
```

Out[49]: Text(0.5, 1.0, 'Standard Deviation of Validation score')



Trying the optimal model over test set

In [50]:
```
clf = KNN(n_neighbors = 7 )
clf.fit(train_x, train_y)

score1 = clf.score(train_x, train_y)

score = clf.score(test_x, test_y)
score, score1
```

Out[50]: (0.08539003216879149, 0.32535481363367225)

In [53]:
```
clf = KNN(n_neighbors = 8 )
clf.fit(train_x, train_y)

score1 = clf.score(train_x, train_y)

score = clf.score(test_x, test_y)
score, score1
```

Out[53]: (0.08332357856047168, 0.29545000059241133)

In [55]:
```
clf = KNN(n_neighbors = 2 )
clf.fit(train_x, train_y)

score1 = clf.score(train_x, train_y)

score = clf.score(test_x, test_y)
score, score1
```

Out[55]: (0.01930944589950323, 0.7063027173029341)

In [56]:
```
# Creating instance of KNN
reg = KNN(n_neighbors = 2)

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

# Predicting over the Train Set and calculating MAE
train_predict = reg.predict(train_x)
k = MAE(train_predict, train_y)
print('Train MAE    ', k )

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

```
Train MAE     182.4545957875398
Test MAE     335.749881564504
```

In [57]:
```
# Creating instance of KNN
reg = KNN(n_neighbors = 7)

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

# Predicting over the Train Set and calculating MAE
train_predict = reg.predict(train_x)
k = MAE(train_predict, train_y)
print('Train MAE    ', k )

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

```
Train MAE     281.32469243164087
Test MAE     329.4823997424956
```

In [60]:
```
# So 7 seems to be our better k value.
```

In [ ]:
```
```