In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from math import radians,cos,sin,sqrt
import warnings
warnings.filterwarnings('ignore')
```

Importing Dataset

In [2]:
```
df=pd.read_csv(r'nyc_taxi_trip_duration.csv')
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 [3]:
```
#data=data.iloc[0 : 10000]
```

In [4]:
```
df.shape
```

Out[4]:
```
(729322, 11)
```

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

Out[5]:

	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	id0323939	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 [6]:
```
df.isnull().sum()
```

Out[6]:
```
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 [7]:
```
df['trip_duration'] = df['trip_duration'].apply(lambda x: x/3600)
df.drop(columns=['trip_duration'], inplace=True)
```

In [8]:
```
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 [9]:
```
df['pickup_hour'] = df['pickup_datetime'].dt.hour
df['pickup_weekday'] = df['pickup_datetime'].dt.weekday
df['pickup_day'] = df['pickup_datetime'].dt.day
df['pickup_month'] = df['pickup_datetime'].dt.month
df['pickup_year'] = df['pickup_datetime'].dt.year
df['pickup_date'] = df['pickup_datetime'].dt.date

df['dropoff_hour'] = df['dropoff_datetime'].dt.hour
df['dropoff_weekday'] = df['dropoff_datetime'].dt.weekday
df['dropoff_day'] = df['dropoff_datetime'].dt.day
df['dropoff_month'] = df['dropoff_datetime'].dt.month
df['dropoff_year'] = df['dropoff_datetime'].dt.year
df['dropoff_date'] = df['dropoff_datetime'].dt.date
```

In [10]:
```
def time_of_day(x):
    # to calculate what time of it is now
    if x in range(6,12):
        return 'Morning'
    elif x in range(12,16):
        return 'Afternoon'
    elif x in range(16,22):
        return 'Evening'
    else:
        return 'Late night'

df['pickup_time_of_day'] = df['pickup_hour'].apply(time_of_day)
df['dropoff_time_of_day'] = df['dropoff_hour'].apply(time_of_day)
```

In [11]:
```
df.drop(columns=['pickup_hour','pickup_weekday','pickup_day','pickup_month','pickup_year','dropoff_hour','dropoff_weekday','dropoff_day','dropoff_month','dropoff_year','pickup_date','dropoff_date'], inplace=True)
```

In [12]:
```
#simple predictive model
df["trip_duration_hour_mean"]=df["trip_duration_hour"].mean()
df["trip_duration_hour_mean"].head()
```

Out[12]:
```
0    0.264508
1    0.264508
2    0.264508
3    0.264508
4    0.264508
Name: trip_duration_hour_mean, dtype: float64
```

1.Choose the most suitable evaluation metric and state why you chose it.

In [13]:
```
# We have  chosen root mean squared error because:
# 1.RMSE is better in terms of reflecting performance when dealing with large error values
# 2.RMSE is more useful when lower residual values are preferred.
# 3.RMSE penalize large errors.
```

2.Build a benchmark model for the given dataset.

shuffling and creating Train and Test Dataset

In [14]:
```
#importing the shuffle library
from sklearn.utils import shuffle

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

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

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

In [15]:
```
div
```

Out[15]:
```
182330
```

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

Out[16]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration_hour	pickup
469114	id2380741	2	2016-05-21 10:40:14	2016-05-21 10:51:11	1	-73.981796	40.762035	-73.972267	40.781265	N	0.182500	2016
694852	id3946961	2	2016-01-08 18:49:27	2016-01-08 18:52:42	5	-73.980965	40.747677	-73.982704	40.741161	N	0.054167	2016
696324	id0833913	1	2016-05-22 00:54:10	2016-05-22 01:08:10	1	-73.951065	40.782722	-73.867691	40.833664	N	0.233333	2016
356496	id1330649	1	2016-06-11 10:32:12	2016-06-11 10:38:50	1	-73.987625	40.762791	-73.973518	40.762909	N	0.110556	2016
645318	id1610858	1	2016-04-03 10:45:51	2016-04-03 10:57:13	3	-73.964333	40.792503	-73.988609	40.758369	N	0.189444	2016

In [17]:
```
test.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_hour	pickup
546991	id2240736	1	2016-05-25 07:59:16	2016-05-25 08:05:02	1	-73.991364	40.732590	-74.000526	40.742283	N	0.096111	2016
43126	id1423404	1	2016-01-18 12:17:13	2016-01-18 12:21:13	2	-73.966225	40.768059	-73.967606	40.763073	N	0.066667	2016
641450	id1317268	2	2016-03-02 18:39:01	2016-03-02 18:50:12	1	-73.994926	40.766018	-74.004219	40.742523	N	0.186389	2016
611380	id3335546	1	2016-04-06 19:17:20	2016-04-06 19:18:03	1	-73.974388	40.793781	-73.976006	40.792339	N	0.011944	2016
62690	id2174190	2	2016-06-21 18:35:31	2016-06-21 18:40:56	3	-73.963440	40.798557	-73.979736	40.777878	N	0.090278	2016

In [18]:
```
# storing simple mean in a new column in the test set as "simple_mean"
df["trip_duration_hour_mean"]=df["trip_duration_hour"].mean()
df["trip_duration_hour_mean"]
```

Out[18]:
```
469114    0.264508
694852    0.264508
696324    0.264508
356496    0.264508
645318    0.264508
            ...   
259178    0.264508
365838    0.264508
131932    0.264508
671155    0.264508
121958    0.264508
Name: trip_duration_hour_mean, Length: 729322, dtype: float64
```

In [19]:
```
from sklearn.metrics import mean_squared_error as mse
from math import sqrt

trip_mean_error = sqrt(mse(test['trip_duration_hour'] , test['trip_duration_hour_mean']))
trip_mean_error
```

Out[19]:
```
0.8904067655425832
```

In [20]:
```
pickup = pd.pivot_table(train, values='trip_duration_hour', index = ['pickup_time_of_day'], aggfunc=np.mean)
pickup
```

Out[20]:

pickup_time_of_day	trip_duration_hour
Afternoon	0.291531
Evening	0.264078
Late night	0.255589
Morning	0.250610

In [21]:
```
# initializing new column to zero
test['pickup'] = 0


for i in train['pickup_time_of_day'].unique():

  test['pickup'][test['pickup_time_of_day'] == str(i)] = train['trip_duration_hour'][train['pickup_time_of_day'] == str(i)].mean()
```

In [22]:
```
#calculating RMSE
pickup_error = sqrt(mse(test['trip_duration_hour'] , test['pickup'] ))
pickup_error
```

Out[22]:
```
0.8903816016040094
```

In [23]:
```
#trip duration mean with respect to the mean of dropoff time of the day
dropoff =pd.pivot_table(train, values='trip_duration_hour', index = ['dropoff_time_of_day'], aggfunc=np.mean)
dropoff
```

Out[23]:

dropoff_time_of_day	trip_duration_hour
Afternoon	0.284258
Evening	0.269666
Late night	0.256343
Morning	0.247905

In [24]:
```
# initializing new column to zero
test['dropoff'] = 0

# For every unique entry in pickup latitude
for i in train['dropoff_time_of_day'].unique():
  # Assign the mean value corresponding to unique entry
  test['dropoff'][test['dropoff_time_of_day'] == str(i)] = train['trip_duration_hour'][train['dropoff_time_of_day'] == str(i)].mean()
```

In [25]:
```
#calculating mean absolute error
dropoff_error = sqrt(mse(test['trip_duration_hour'] , test['dropoff'] ))
dropoff_error
```

Out[25]:
```
0.8903865449170373
```

In [26]:
```
#trip duration mean with respect to the mean of passenger count
pass_count = pd.pivot_table(train, values='trip_duration_hour', index = ["passenger_count"], aggfunc=np.mean)
pass_count
```

Out[26]:

passenger_count	trip_duration_hour
0	0.092981
1	0.255343
2	0.277822
3	0.287332
4	0.285759
5	0.299641
6	0.300193

In [27]:
```
# initializing new column to zero
test['pass_count'] = 0

# For every unique entry in passenger count
for i in train['passenger_count'].unique():
  # Assign the mean value corresponding to unique entry
  test['pass_count'][test['passenger_count'] == str(i)] = train['trip_duration_hour'][train['passenger_count'] == str(i)].mean()
```

In [28]:
```
pass_count_error = sqrt(mse(test['trip_duration_hour'] , test['pass_count'] ))
pass_count_error
```

Out[28]:
```
0.9290781075032716
```

In [29]:
```
store_and_fwd = pd.pivot_table(train, values='trip_duration_hour', index = ["store_and_fwd_flag"], aggfunc=np.mean)
store_and_fwd
```

Out[29]:

store_and_fwd_flag	trip_duration_hour
N	0.264109
Y	0.304058

In [30]:
```
# initializing new column to zero
test['store_and_fwd'] = 0

# For every unique entry in pickup latitude
for i in train['store_and_fwd_flag'].unique():
  # Assign the mean value corresponding to unique entry
  test['store_and_fwd'][test['store_and_fwd_flag'] == str(i)] = train['trip_duration_hour'][train['store_and_fwd_flag'] == str(i)].mean()
```

In [31]:
```
str_and_fwd_error = sqrt(mse(test['store_and_fwd'] , test['trip_duration_hour'] ))
str_and_fwd_error
```

Out[31]:
```
0.8904020727484228
```

In [32]:
```
combo = pd.pivot_table(train, values = 'trip_duration_hour', index = ['passenger_count','pickup_time_of_day','dropoff_time_of_day'], aggfunc = np.mean)
combo
```

Out[32]:

			trip_duration_hour
passenger_count	pickup_time_of_day	dropoff_time_of_day	
0	**Afternoon**	**Afternoon**	0.305417
	Evening	**Evening**	0.054352
		Late night	0.106944
	Late night	**Late night**	0.023856
	Morning	**Afternoon**	0.432222
	...	...	...
6	**Late night**	**Late night**	0.282054
	Morning	**Morning**	0.313738
		Afternoon	0.376369
		Late night	16.731944
		Morning	0.267228

75 rows × 1 columns

In [38]:
```python
#importing the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [39]:
```python
#reading the csv file
df = pd.read_csv("nyc_taxi_trip_duration.csv")
df.head()
```

Out[39]:

	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 [40]:
```python
# removing the outlier
```

In [41]:
```python
df["passenger_count"].value_counts()
```

Out[41]:
```
1    517415
2    105097
5     38926
3     29692
6     24107
4     14050
0        33
7         1
9         1
Name: passenger_count, dtype: int64
```

In [42]:
```python
df = df.loc[~(df['passenger_count'] == 0)]
df = df.loc[~(df['passenger_count'] == 7)]
df = df.loc[~(df['passenger_count'] == 9)]
```

In [ ]:

In [43]:
```python
df['trip_duration_hour'] = df['trip_duration'].apply(lambda x: x/3600)
df.drop(columns=['trip_duration'], inplace=True)
```

In [44]:
```python
df.dtypes
```

Out[44]:
```
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_hour    float64
dtype: object
```

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

In [87]:
```python
custom_df = df.sample(10000)
```

In [88]:
```python
sample_df=custom_df
```

In [89]:
```python
#seperate features and target
features = sample_df.drop(["id","vendor_id","trip_duration_hour","pickup_datetime","dropoff_datetime","store_and_fwd_flag"],axis = 1)
target = sample_df["trip_duration_hour"]
```

In [90]:
```python
#seperate features and target
features = sample_df.drop(["id","vendor_id","trip_duration_hour","pickup_datetime","dropoff_datetime","store_and_fwd_flag"],axis = 1)
target = sample_df["trip_duration_hour"]
```

In [91]:
```python
sample_df.dtypes
```

Out[91]:
```
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_hour    float64
dtype: object
```

In [92]:
```python
#converting the store and fwd flag to int type
df["passenger_count"].value_counts()
```

Out[92]:
```
1    7097
2    1376
5     584
3     402
6     355
4     186
Name: passenger_count, dtype: int64
```

In [93]:
```python
df1 = pd.concat([sample_df, pd.get_dummies(sample_df[['passenger_count']].astype('str'))], axis=1)
try:
    df1.drop(['id','vendor_id','trip_duration_hour','pickup_datetime','dropoff_datetime','store_and_fwd_flag','passenger_count'], axis=1, inplace=True)
except KeyError:
    pass

df1.head()
```

Out[93]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count_1	passenger_count_2	passenger_count_3	passenger_count_4	passenger_count_5	passenger_count_6
146191	-74.013412	40.709381	-73.975502	40.681133	0	1	0	0	0	0
265532	-74.000076	40.761559	-73.994354	40.760761	1	0	0	0	0	0
513052	-73.974457	40.757908	-73.974205	40.753754	1	0	0	0	0	0
607286	-73.985367	40.737347	-74.011597	40.702568	0	0	0	0	1	0
658563	-73.943840	40.788502	-73.950119	40.778938	1	0	0	0	0	0

In [96]:
```python
#seperate features and target
x = df1
y = df["trip_duration_hour"]
x.shape,y.shape
```

Out[96]:
```
((10000, 10), (10000,))
```

In [97]:
```python
#scaling the data(using MinMax Scaler)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)
```

In [98]:
```python
x = pd.DataFrame(x_scaled,columns=x.columns)
x.head()
```

Out[98]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count_1	passenger_count_2	passenger_count_3	passenger_count_4	passenger_count_5	passenger_count_6
0	0.248894	0.254970	0.329205	0.288621	0.0	1.0	0.0	0.0	0.0	0.0
1	0.285032	0.422099	0.300734	0.505790	1.0	0.0	0.0	0.0	0.0	0.0
2	0.354455	0.410406	0.331163	0.486678	1.0	0.0	0.0	0.0	0.0	0.0
3	0.324891	0.344546	0.274694	0.347080	0.0	0.0	0.0	0.0	1.0	0.0
4	0.437420	0.508400	0.367538	0.555364	1.0	0.0	0.0	0.0	0.0	0.0

In [99]:
```python
#importing the train_test_split from sklearn
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 [100]:
```python
#importing knn regressor and mse metrics
from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn.metrics import mean_squared_error as mse
from math import sqrt
```

In [101]:
```python
#creating instance of KNN
reg = KNN(n_neighbors = 10)
#fitting the model
reg.fit(train_x,train_y)
#predicting over the train set and calculating F1
test_predict = reg.predict(test_x)
k= sqrt(mse(test_predict,test_y))
print("test rmse ", k)
```

```
test rmse  0.8482900308922127
```

Finding the value of k using elbow method

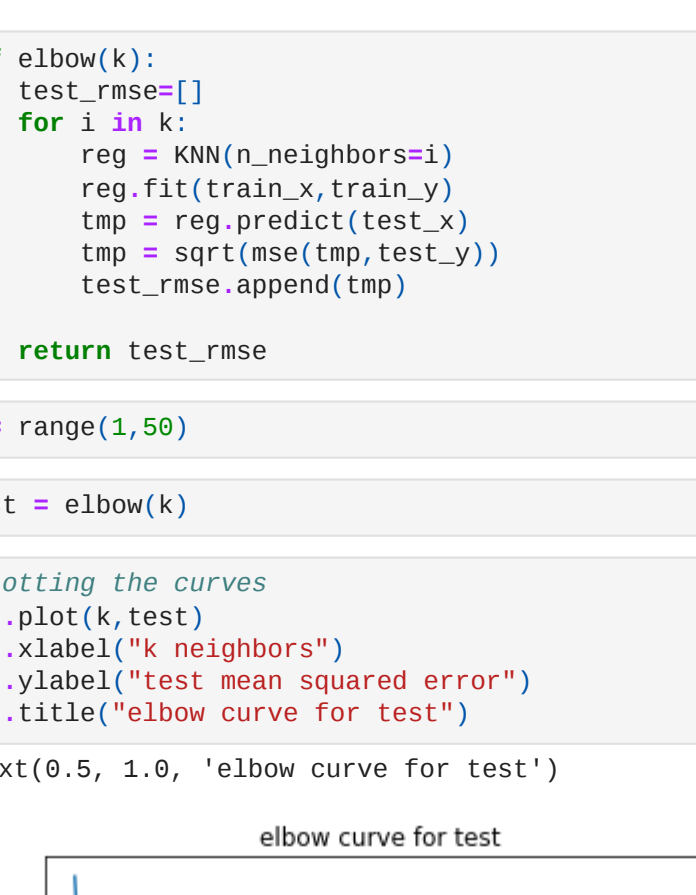
In [102]:
```python
def elbow(k):
    test_rmse=[]
    for i in k:
        reg = KNN(n_neighbors=i)
        reg.fit(train_x,train_y)
        tmp = reg.predict(test_x)
        tmp = sqrt(mse(tmp,test_y))
        test_rmse.append(tmp)

    return test_rmse
```

In [103]:
```python
k = range(1,50)
```

In [104]:
```python
test = elbow(k)
```

In [105]:
```python
#plotting the curves
plt.plot(k,test)
plt.xlabel("k neighbors")
plt.ylabel("test mean squared error")
plt.title("elbow curve for test")
```

Out[105]:
```
Text(0.5, 1.0, 'elbow curve for test')
```



In [106]:
```python
#creating instance of KNN
reg = KNN(n_neighbors = 5)
#fitting the model
reg.fit(train_x,train_y)
#predicting over the train set and calculating F1
test_predict = reg.predict(test_x)
k= sqrt(mse(test_predict,test_y))
print("test rmse ", k)
```

```
test rmse  0.879542551542604
```

The best value of k is 10

In [107]:
```python
knn_train_score = reg.score(train_x,train_y)
knn_train_score*100
```

Out[107]:
```
22.066087471354766
```

In [108]:
```python
knn_test_score = reg.score(test_x,test_y)
knn_test_score*100
```

Out[108]:
```
-13.45060950590189
```

In [109]:
```python
#storing the value of train score which can be further used in other files for plotting the bar graph
%store knn_test_score
```

```
Stored 'knn_test_score' (float64)
```

In [110]:
```python
#storing the value of test score which can be further used in other files for plotting the bar graph
%store knn_train_score
```

```
Stored 'knn_train_score' (float64)
```

In [111]:
```python
custom_df=sample_df
```

In [112]:
```python
sample_df.dtypes
```

Out[112]:
```
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_hour    float64
dtype: object
```

In [113]:
```python
#seperate features and target
features = sample_df.drop(["id","vendor_id","trip_duration_hour","pickup_datetime","dropoff_datetime"],axis = 1)
target = sample_df["trip_duration_hour"]
```

In [114]:
```python
df1 = pd.concat([sample_df, pd.get_dummies(sample_df[['passenger_count']].astype('str'))], axis=1)
try:
    df1.drop(['id','vendor_id','trip_duration_hour','pickup_datetime','dropoff_datetime','store_and_fwd_flag','passenger_count'], axis=1, inplace=True)
except KeyError:
    pass

df1.head()
```

Out[114]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count_1	passenger_count_2	passenger_count_3	passenger_count_4	passenger_count_5	passenger_count_6
146191	-74.013412	40.709381	-73.975502	40.681133	0	1	0	0	0	0
265532	-74.000076	40.761559	-73.994354	40.760761	1	0	0	0	0	0
513052	-73.974457	40.757908	-73.974205	40.753754	1	0	0	0	0	0
607286	-73.985367	40.737347	-74.011597	40.702568	0	0	0	0	1	0
658563	-73.943840	40.788502	-73.950119	40.778938	1	0	0	0	0	0

In [115]:
```python
df1.shape
```

Out[115]:
```
(10000, 10)
```

In [116]:
```python
#seperate features and target
x = df1
y = sample_df["trip_duration_hour"]
x.shape,y.shape
```

Out[116]:
```
((10000, 10), (10000,))
```

In [117]:
```python
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(x,y,random_state=56)
```

Build a Linear model for the given dataset with regularisation. Attempt to interpret the variable coefficients of the Linear Model.

In [154]:
```python
from sklearn.linear_model import LinearRegression as LR
from sklearn.metrics import mean_squared_error as mse
```

In [198]:
```python
#creating an instance of LR
lr = LR()
#fitting the model
lr.fit(train_x,train_y)
```

Out[198]:
```
LinearRegression()
```

In [156]:
```python
#predicting the train values and finding the RMSE
train_predict = lr.predict(train_x)
k = sqrt(mse(train_predict,train_y))
print("training rmse ",k)
```

```
training rmse  0.6346149233354497
```

In [157]:
```python
#predicting the test values and finding the RMSE
test_predict = lr.predict(test_x)
k = sqrt(mse(test_predict,test_y))
print("testing rmse ",k)
```

```
testing rmse  1.1463409343038833
```

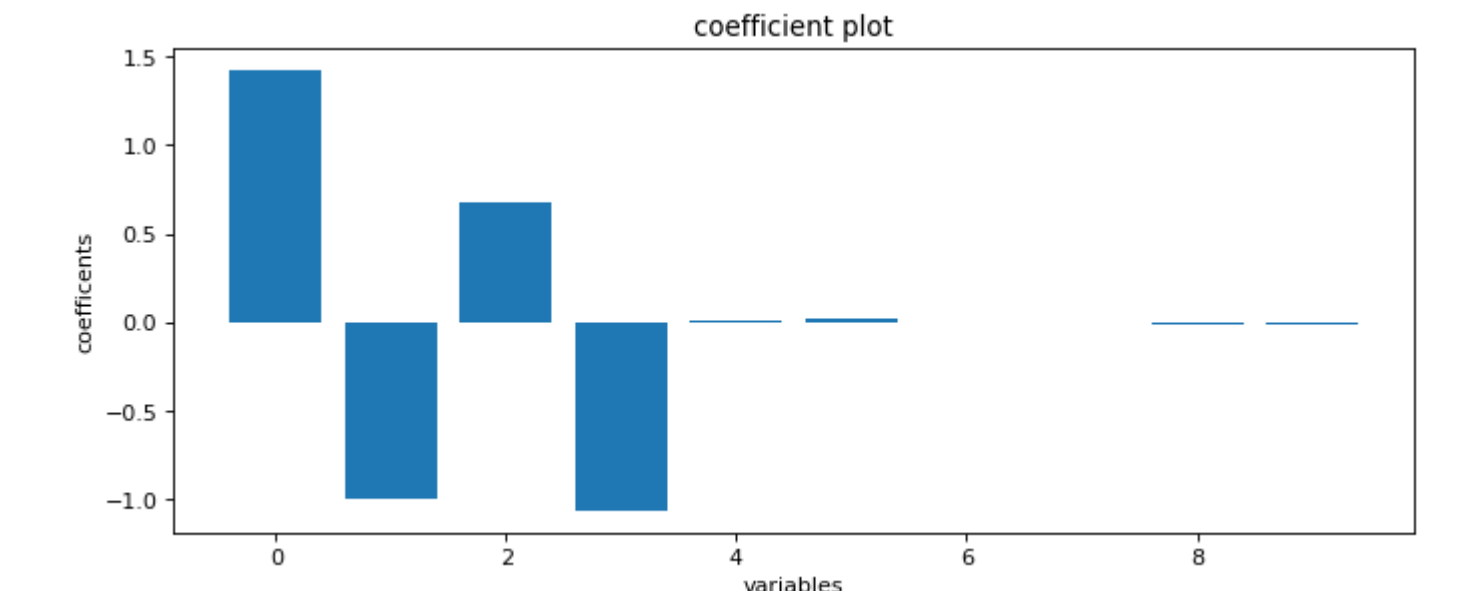
In [158]:
```python
#computing the coefficients of above model
lr.coef_
```

Out[158]:
```
array([ 1.42080279, -0.99258854,  0.68009856, -1.06158804,  0.01102592,
        0.02619976, -0.00204601, -0.00311208, -0.01788712, -0.01418048])
```

plotting the coefficients

In [197]:
```python
plt.figure(figsize=(10,4),dpi=80,facecolor="w",edgecolor="b")
x = range(len(train_x.columns))
y = lr.coef_
plt.bar(x,y)
plt.xlabel("variables")
plt.ylabel("coefficents")
plt.title("coefficient plot")
```

Out[197]:
```
Text(0.5, 1.0, 'coefficient plot')
```



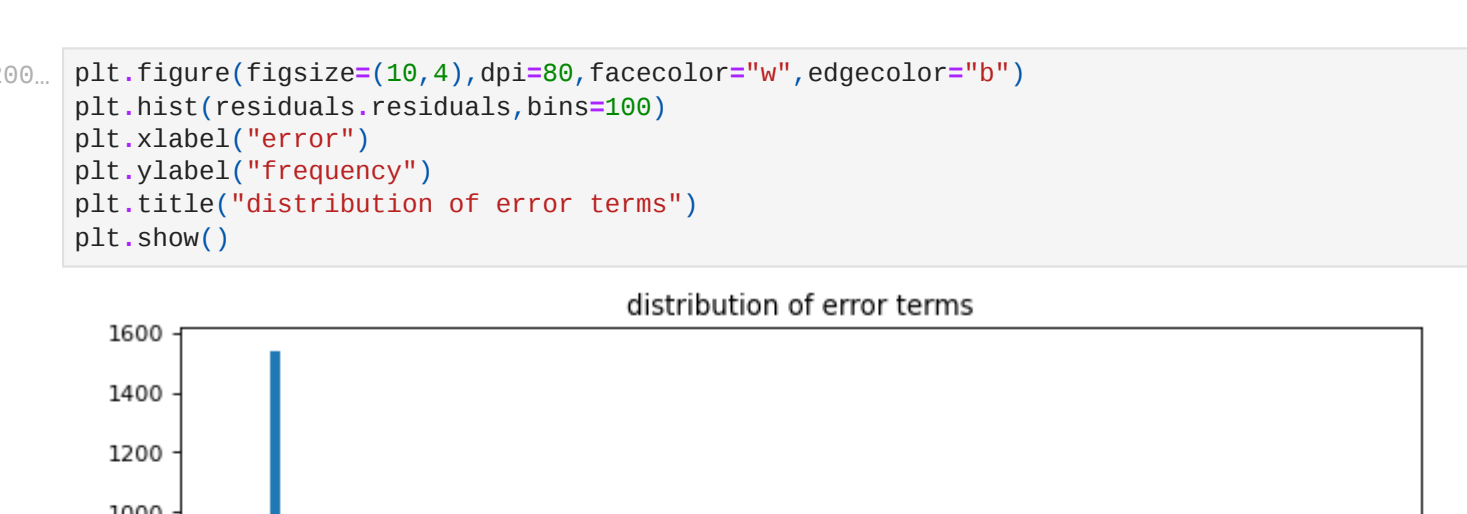
Here we can see that the model depends upon some Independent variables too much, But these coefficients are not suitable for interpretation because these are not scaled.

In [160]:
```python
#calculating the residuals
residuals = pd.DataFrame({
    "fitted_values":test_y,
    "predicted_values":test_predict
})
residuals["residuals"]=residuals["fitted_values"]-residuals["predicted_values"]
residuals.head()
```

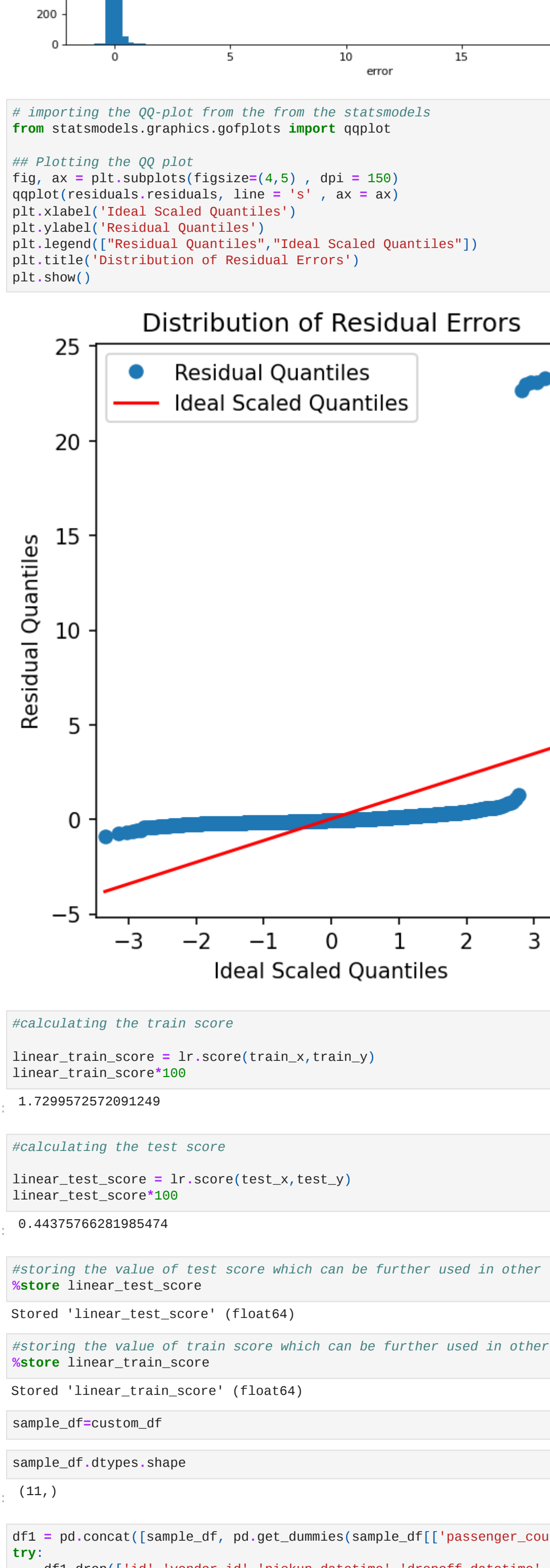
Out[160]:

	fitted_values	predicted_values	residuals
534484	0.201111	0.237518	-0.036407
569166	0.047778	0.177900	-0.130123
700651	0.196667	0.224883	-0.028217
492236	0.258889	0.206411	0.052478
147163	0.408611	0.260428	0.148183

In [200]:
```python
plt.figure(figsize=(10,4),dpi=80,facecolor="w",edgecolor="b")
plt.hist(residuals.residuals,bins=100)
plt.xlabel("error")
plt.ylabel("frequency")
plt.title("distribution of error terms")
plt.show()
```



In [201]:
```python
# importing the QQ-plot from the from the statsmodels
from statsmodels.graphics.gofplots import qqplot

## Plotting the QQ plot
fig, ax = plt.subplots(figsize=(4,5) , dpi = 150)
qqplot(residuals.residuals, line = 's' , ax = ax)
plt.xlabel('Ideal Scaled Quantiles')
plt.ylabel('Residual Quantiles')
plt.legend(["Residual Quantiles","Ideal Scaled Quantiles"])
plt.title('Distribution of Residual Errors')
plt.show()
```



In [202]:
```python
#calculating the train score

linear_train_score = lr.score(train_x,train_y)
linear_train_score*100
```

Out[202]:
```
1.7299572572091249
```

In [164]:
```python
#calculating the test score

linear_test_score = lr.score(test_x,test_y)
linear_test_score*100
```

Out[164]:
```
0.44375766281985474
```

In [165]:
```python
#storing the value of test score which can be further used in other files for plotting the bar graph
%store linear_test_score
```

```
Stored 'linear_test_score' (float64)
```

In [166]:
```python
#storing the value of train score which can be further used in other files for plotting the bar graph
%store linear_train_score
```

```
Stored 'linear_train_score' (float64)
```

In [167]:
```python
sample_df=custom_df
```

In [168]:
```python
sample_df.dtypes.shape
```

Out[168]:
```
(11,)
```

In [169]:
```python
df1 = pd.concat([sample_df, pd.get_dummies(sample_df[['passenger_count']].astype('str'))], axis=1)
try:
    df1.drop(['id','vendor_id','pickup_datetime','dropoff_datetime','store_and_fwd_flag','passenger_count'], axis=1, inplace=True)
except KeyError:
    pass

df1.head()
```

Out[169]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration_hour	passenger_count_1	passenger_count_2	passenger_count_3	passenger_count_4	passenger_count_5
146191	-74.013412	40.709381	-73.975502	40.681133	0.261944	0	1	0	0	0
265532	-74.000076	40.761559	-73.994354	40.760761	0.055278	1	0	0	0	0
513052	-73.974457	40.757908	-73.974205	40.753754	0.070833	1	0	0	0	0
607286	-73.985367	40.737347	-74.011597	40.702568	0.179722	0	0	0	0	1
658563	-73.943840	40.788502	-73.950119	40.778938	0.085556	1	0	0	0	0

In [170]:
```python
x = df1.drop(["trip_duration_hour"],axis = 1)
y = df1["trip_duration_hour"]
```

In [171]:
```python
y.shape
```

Out[171]:
```
(10000,)
```

In [172]:
```python
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(x,y,random_state=56)
```

Build a Decision tree model for the given dataset. Attempt to interpret the variable importance

In [173]:
```python
from sklearn.tree import DecisionTreeRegressor as DecisionTreeRegressor
from sklearn.metrics import mean_squared_error as mse
regressor = DecisionTreeRegressor()
regressor.fit(x, y)
```

Out[173]:
```
DecisionTreeRegressor()
```

In [174]:
```python
from math import sqrt
train_predict = regressor.predict(train_x)
k = sqrt(mse(train_predict,train_y))
print("training rmse ",k)
```

```
training rmse  0.0
```

In [175]:
```python
from math import sqrt
test_predict = regressor.predict(test_x)
k1 = sqrt(mse(test_predict,test_y))
print("testing rmse ",k1)
```

```
testing rmse  0.0
```

To interpret the variable importance.

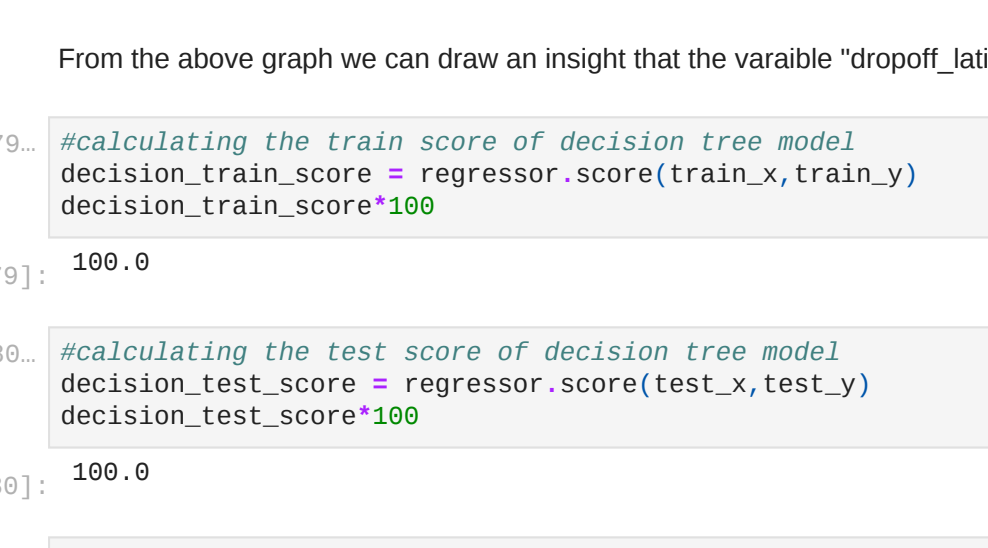
In [176]:
```python
x.columns
```

Out[176]:
```
Index(['pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count_1', 'passenger_count_2',
       'passenger_count_3', 'passenger_count_4', 'passenger_count_5',
       'passenger_count_6'],
      dtype='object')
```

In [177]:
```python
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\dell\anaconda3\lib\site-packages (1.7.1)Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: numpy in c:\users\dell\anaconda3\lib\site-packages (from xgboost) (1.21.5)
Requirement already satisfied: scipy in c:\users\dell\anaconda3\lib\site-packages (from xgboost) (1.7.3)
```

In [178]:
```python
#Using XGBoost in order to interpret the variable importance
from xgboost import XGBRegressor
from xgboost import plot_importance
import matplotlib.pyplot as plt
model = XGBRegressor()
model.fit(x,y)
plot_importance(model)
plt.show()
```



From the above graph we can draw an insight that the varaible "dropoff_latitude" has more impact on the trip duration variable followed by "pickup_longitude" then "pickup_latitude"

In [179]:
```python
#calculating the train score of decision tree model
decision_train_score = regressor.score(train_x,train_y)
decision_train_score*100
```

Out[179]:
```
100.0
```

In [180]:
```python
#calculating the test score of decision tree model
decision_test_score = regressor.score(test_x,test_y)
decision_test_score*100
```

Out[180]:
```
100.0
```

In [181]:
```python
#storing the test score to use it in other file
%store decision_test_score
```

```
Stored 'decision_test_score' (float64)
```

In [182]:
```python
#storing the train score to use it in other file
%store decision_train_score
```

```
Stored 'decision_train_score' (float64)
```

Plot the following Bar plots:

1. train score of all the above models.
2. test (not validation!) score of all the above models.
3. Attempt to explain the observations from the plots (optional)

In [183]:
```python
decision_train_score,linear_train_score,knn_train_score
```

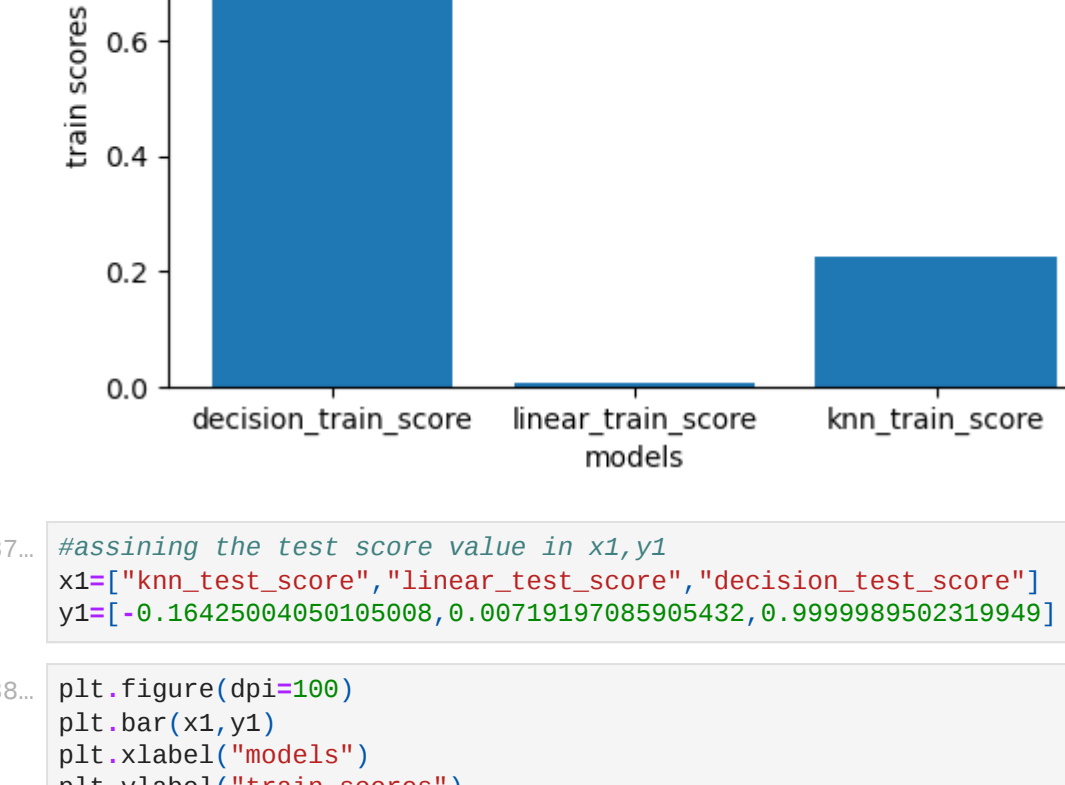
Out[183]:
```
(1.0, 0.01729957257209125, 0.22066087471354767)
```

In [184]:
```python
#assining the train score values in x,y
x=["decision_train_score","linear_train_score","knn_train_score"]
y=[0.9999996442282492, 0.005868613774166032, 0.22451293627095015]
```

In [185]:
```python
knn_test_score, linear_test_score,decision_test_score
```

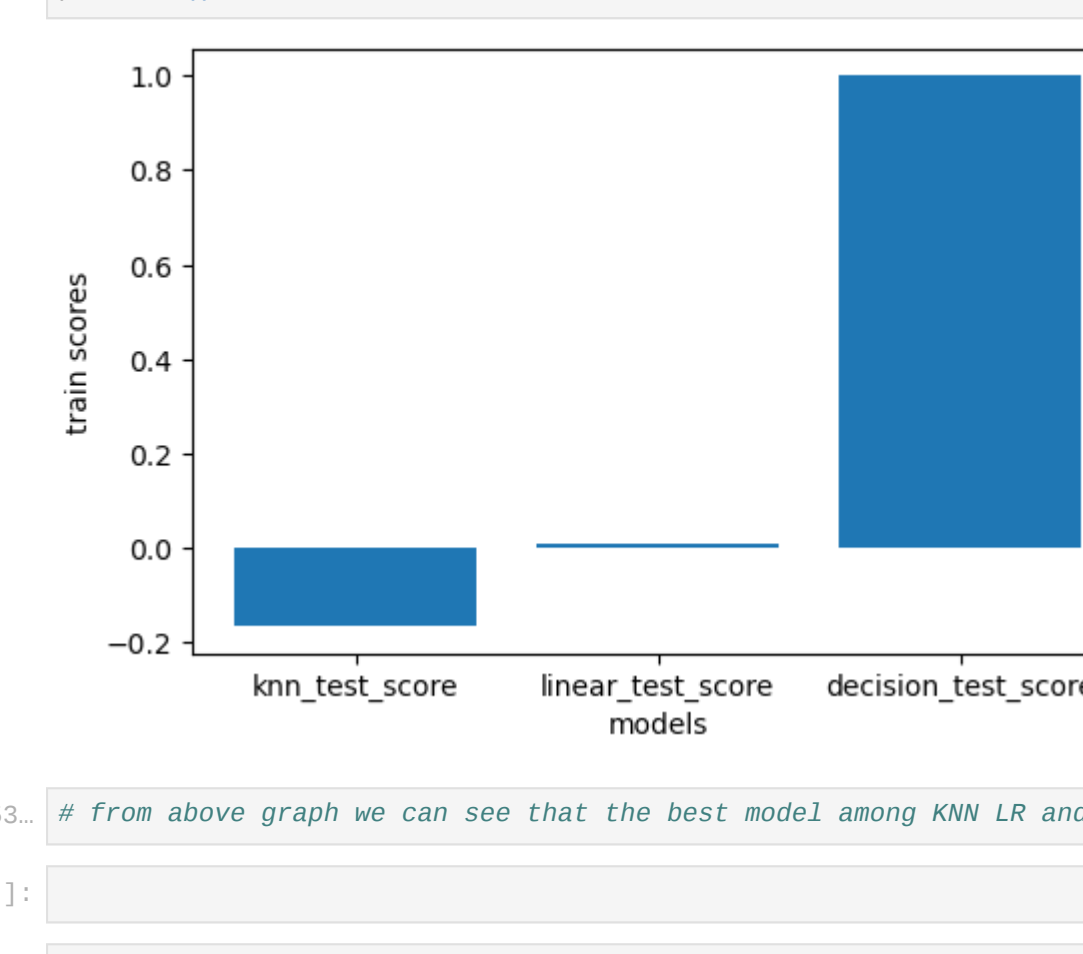
Out[185]:
```
(-0.1345060950590189, 0.004437576628198547, 1.0)
```

In [186]:
```python
plt.figure(dpi=100)
plt.bar(x,y)
plt.xlabel("models")
plt.ylabel("train scores")
plt.show()
```



In [187]:
```python
#assining the test score value in x1,y1
x1=["knn_test_score","linear_test_score","decision_test_score"]
y1=[-0.16425004050105008,0.00719197085905432,0.9999989502319949]
```

In [188]:
```python
plt.figure(dpi=100)
plt.bar(x1,y1)
plt.xlabel("models")
plt.ylabel("train scores")
plt.show()
```



In [153]:
```python
# from above graph we can see that the best model among KNN LR and DT(decision tree) the best one is DT in train score
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: