EDA PROJECT

1.DEFINING THE PROBLEM

```
PREDICTING THE DURATION OF A TOTAL TAXI RIDE IN NEW YORK CITY.
WE NEED TO PLAN THEIR FLEET IN MUCH BETTER MANNER. FOR THAT WE NEED TO PREDICT THAT WHERE THE RIDE END AND WHEN THE RIDE END
THIS WILL HELP TO PREDICT THE FLEET ACCORDINGLY.
```

2 . HYPOTHESIS GENERATION

1. Taxi Features

2. Driver and Passenger Details

3. Day or Time based Factors

4. Demographics

5. Vendor Details

DATA COLLECTION

```
nyc_taxi_trip_duration.csv
```

Data Exploration

Our Dataset contains the following set of variables-

- **id** - a unique identifier for each trip
- **vendor_id** - a code indicating the provider associated with the trip record
- **pickup_datetime** - date and time when the meter was engaged
- **dropoff_datetime** - date and time when the meter was disengaged
- **passenger_count** - the number of passengers in the vehicle (driver entered value)
- **pickup_longitude** - the longitude where the meter was engaged
- **pickup_latitude** - the latitude where the meter was engaged
- **dropoff_longitude** - the longitude where the meter was disengaged
- **dropoff_latitude** - the latitude where the meter was disengaged
- **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)
- **trip_duration** - (target) duration of the trip in seconds

IMPORT LIBRARIES

In [1]:

```
import pandas as pd
import numpy as np
import datetime as dt
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

LOAD A DATASET

In [2]:

```
df = pd.read_csv('nyc_taxi_trip_duration.csv')
df.head()
```

Out[2]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lat
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	-73.963875	40.7
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	-73.994751	40.6
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	-73.948029	40.7
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	-73.956779	40.7
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	-73.988182	40.7

datatypes of dataset

In [3]:

```
df.dtypes
```

Out[3]:

```
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
```

shape of the dataset

In [4]:

```
df.shape
```

Out[4]:

```
(729322, 11)
```

THERE IS 729322 ROWS AND 11 COLUMNS IN A DATASET

check missing values in dataset

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
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
```

DOES NOT HAVE ANY MISSING VALUES

In [6]:

```
df.describe()
```

Out[6]:

	vendor_id	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration
count	729322.000000	729322.000000	729322.000000	729322.000000	729322.000000	729322.000000	7.293220e+05
mean	1.535403	1.662055	-73.973513	40.750919	-73.973422	40.751775	9.522291e+02
std	0.498745	1.312446	0.069754	0.033594	0.069588	0.036037	3.864626e+03
min	1.000000	0.000000	-121.933342	34.712234	-121.933304	32.181141	1.000000e+00
25%	1.000000	1.000000	-73.991859	40.737335	-73.991318	40.735931	3.970000e+02
50%	2.000000	1.000000	-73.981758	40.754070	-73.979759	40.754509	6.630000e+02
75%	2.000000	2.000000	-73.967361	40.768314	-73.963036	40.769741	1.075000e+03
max	2.000000	9.000000	-65.897385	51.881084	-65.897385	43.921028	1.939736e+06

In [7]:

```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime)
df.dropoff_datetime = pd.to_datetime(df.dropoff_datetime)
df.dtypes
```

Out[7]:

```
id                            object
vendor_id                      int64
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            object
trip_duration                  int64
dtype: object
```

In [8]:

```
df.vendor_id=df["vendor_id"].astype("category")
df.store_and_fwd_flag=df["store_and_fwd_flag"].astype("category")
df.dtypes
```

Out[8]:

```
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 above step we have change a datatype of vendorid and store_and_fwd_flag dataset into categorical

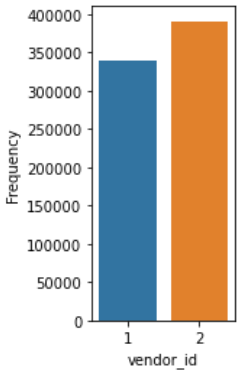
univarite analysis

In [9]:

```
plt.subplot(132)
sns.countplot(df['vendor_id'])
plt.xlabel('vendor_id')
plt.ylabel('Frequency')
```

Out[9]:

```
Text(0, 0.5, 'Frequency')
```



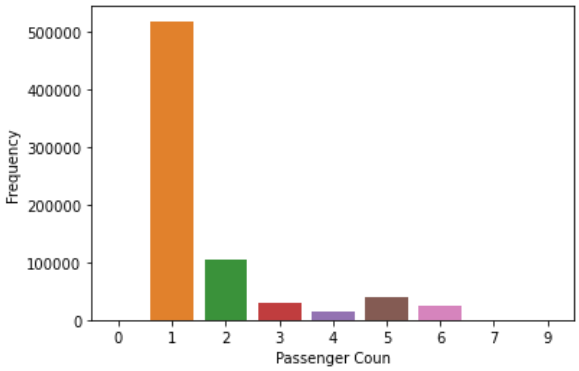

```
as you can see there are 2 vendor and second vendor is slightly greater than first vendor
```

In [10]:

```
sns.countplot(df['passenger_count'])
plt.xlabel('Passenger Coun')
plt.ylabel('Frequency')
```

Out[10]:

```
Text(0, 0.5, 'Frequency')
```



In [11]:

```
df["passenger_count"].value_counts()
```

Out[11]:

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

```
so in this dataset mostly rides has one passenger and then there is another which has 2 passenger
and rest all you see in the count and graph
```

In [12]:

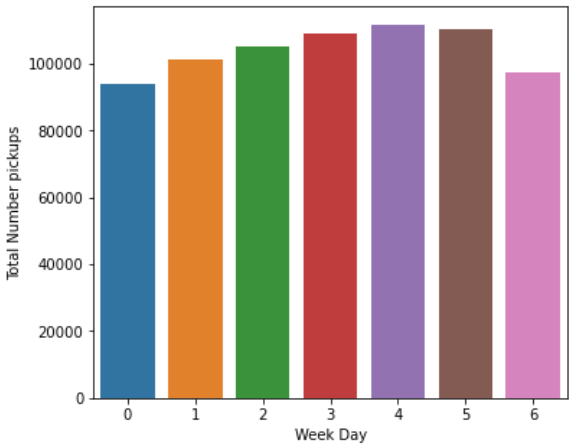
```
#getting pickupdate from weekday and hour of day
df['day_of_week'] = df['pickup_datetime'].dt.weekday
df['hour_of_day'] = df['pickup_datetime'].dt.hour
```

In [13]:

```
plt.figure(figsize=(20, 5))
plt.subplot(132)
sns.countplot(df['day_of_week'])
plt.xlabel('Week Day')
plt.ylabel('Total Number pickups')
```

Out[13]:

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

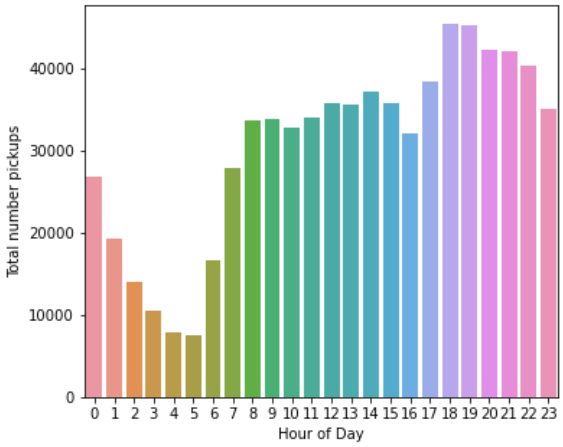


In [14]:

```
plt.figure(figsize=(20, 5))
plt.subplot(132)
sns.countplot(df['hour_of_day'])
plt.xlabel('Hour of Day')
plt.ylabel('Total number pickups')
```

Out[14]:

```
Text(0, 0.5, 'Total number pickups')
```




```
# as we can see in both the graph and compare we can observe pickups for week day is higher as compare weekends
# where 0 is sunday and 6 is saturday so we can see 4 is peak day which is thursday
# 7-8PM,8-9PM are the busiest.
# so we can assume most trip are evening and least trip in late night or early morning
```

Benchmark model

In [15]:

```
from sklearn.utils import shuffle
```

In [16]:

```
df["mean_of_trip_duration"] = df["trip_duration"].mean()
df["mean_of_trip_duration"].head()
```

Out[16]:

```
0    952.229133
1    952.229133
2    952.229133
3    952.229133
4    952.229133
Name: mean_of_trip_duration, dtype: float64
```

In [17]:

```
# shuffle the dataset
df = shuffle(df, random_state = 35)
```

In [18]:

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

In [19]:

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

In [20]:

```
train.shape, test.shape
```

Out[20]:

```
((452490, 14), (276833, 14))
```

In [21]:

```
train.head()
```

Out[21]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	drop
465965	id1379098	1	2016-04-15 23:48:24	2016-04-15 23:59:03	1	-73.982430	40.777637	-73.972588	
186947	id2525840	1	2016-04-02 09:11:50	2016-04-02 09:15:34	1	-73.925232	40.768703	-73.934181	
573122	id1746879	2	2016-04-25 17:43:48	2016-04-25 17:49:22	1	-73.990196	40.735191	-74.001663	
374920	id1335310	1	2016-04-18 19:09:23	2016-04-18 19:23:49	1	-74.012817	40.702354	-73.971588	
554920	id2502413	2	2016-04-14 00:10:51	2016-04-14 00:31:41	2	-73.992798	40.724201	-73.984596	

In [22]:

```
test.head()
```

Out[22]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	drop
546991	id2240736	1	2016-05-25 07:59:16	2016-05-25 08:05:02	1	-73.991364	40.732590	-74.000526	
350010	id1111913	1	2016-01-06 16:47:56	2016-01-06 17:06:07	2	-73.962173	40.779232	-73.990540	
374045	id3012940	1	2016-05-31 09:51:04	2016-05-31 09:58:57	2	-73.988579	40.744457	-73.994545	
217227	id1664156	2	2016-05-11 07:30:21	2016-05-11 07:35:41	2	-73.948341	40.778431	-73.955833	
157584	id3138666	1	2016-06-20 00:32:16	2016-06-20 00:40:10	2	-73.998466	40.735508	-74.011055	

simple model

In [23]:

```
test['Simple_mode'] = train['trip_duration'].mean()
```

In [24]:

```
# importing RMSE from skearn.metrics and evaluating our model
from sklearn.metrics import mean_squared_error as mse
```

In [25]:

```
error = np.sqrt(mse(test['trip_duration'] , test['Simple_mode']))
print(error)
```

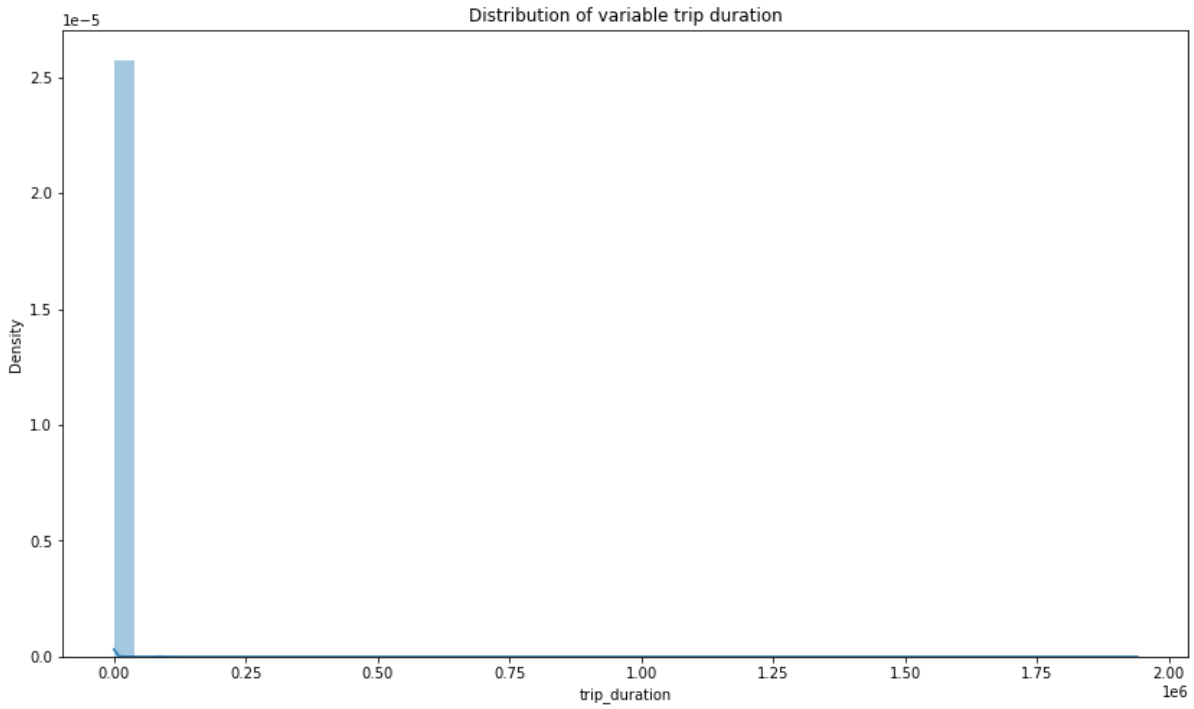
```
4820.989407004606
```

In [26]:

```
plt.figure(figsize = (14,8))
sns.distplot(df['trip_duration'])
plt.xlabel('trip_duration')
plt.title('Distribution of variable trip duration')
```

Out[26]:

```
Text(0.5, 1.0, 'Distribution of variable trip duration')
```



```
now we can see outliers
```

adding vendor id parameter to model and predict

In [27]:

```
# we are firstly constructing a pivot table with index vendor id column
ven_id = pd.pivot_table(train , values = 'trip_duration' , index = ['vendor_id'] , aggfunc = np.mean)
ven_id
```

Out[27]:

vendor_id	trip_duration
1	831.593502
2	1053.555311

In [28]:

```
test['ven_id_mean'] = 0
for i in train['vendor_id'].unique():
    test['ven_id_mean'][test['vendor_id'] == i] = train['trip_duration'][train['vendor_id'] == i].mean()
```

In [29]:

```
vendor_id_error = np.sqrt(mse(test['trip_duration'] , test['ven_id_mean']))
print(vendor_id_error)
```

```
4819.927000821105
```

Adding passanger count parameter

In [30]:

```
pass_count = pd.pivot_table(train , values = 'trip_duration' , index = ['passenger_count'] , aggfunc = np.mean)
pass_count
```

Out[30]:

passenger_count	trip_duration
0	436.409091
1	917.190507
2	1002.711406
3	1036.484664
4	1021.201436
5	1084.195345
6	1075.871661
9	560.000000

In [31]:

```
test['pass_count_mean'] = 0
for i in train['passenger_count'].unique():
    test['pass_count_mean'][test['passenger_count'] == i] = train['trip_duration'][train['passenger_count'] == i].mean()
```

In [32]:

```
pass_count_mean_error = np.sqrt(mse(test['trip_duration'] , test['pass_count_mean']))
print(pass_count_mean_error)
```

```
4820.756827883175
```

now combine both and build a model

In [33]:

```
combined_table = pd.pivot_table(train , values = 'trip_duration' , index = ['vendor_id' , 'passenger_count'] , aggfunc = np.mea
combined_table
```

Out[33]:

		trip_duration
vendor_id	**passenger_count**	
1	**0**	785.000000
	1	808.095683
	2	928.322936
	3	936.954617
	4	971.078125
	5	872.255319
	6	962.702703
2	**0**	18.100000
	1	1041.160513
	2	1060.669649
	3	1097.143545
	4	1057.881444
	5	1085.023615
	6	1076.152685
	9	560.000000

In [34]:

```
# Calculating the super mean by using both the parameters as a constraint.

test['Super_mean'] = 0
s1 = 'vendor_id'
s2 = 'passenger_count'

for i in test[s1].unique():
    for j in test[s2].unique():
        test['Super_mean'][(test[s1] == i) & (test[s2] == j)] = train['trip_duration'][(train[s1] == i) & (train[s2] == j)].mea
```

In [35]:

```
filler = test['Super_mean'].mean()
```

In [36]:

```
test['Super_mean'] = test['Super_mean'].fillna(filler)
```

In [37]:

```
error_related = np.sqrt(mse(test['trip_duration'] , test['Super_mean']))
print(error_related)
```

```
4819.83412817597
```

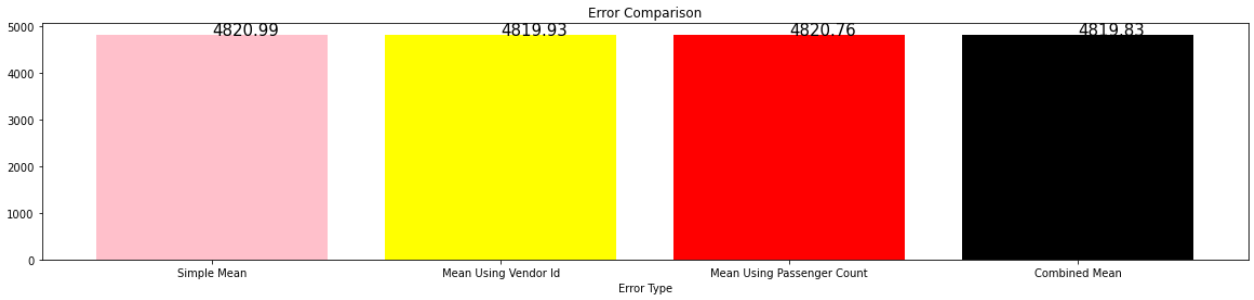
after combined we get 4819

In [38]:

```
# Plotting all the errors on the bar plot.

plt.figure(figsize = (20,4))
X = ['Simple Mean' , 'Mean Using Vendor Id' , 'Mean Using Passenger Count' , 'Combined Mean']
Y = [round(error,2) , round(vendor_id_error,2) , round(pass_count_mean_error,2) , round(error_related,2)]#

x_pos = np.arange(len(X))
plt.bar(x_pos, Y, color=[ 'pink', 'yellow', 'red', 'black'])
for index , data in enumerate(Y):
    plt.text(x = index , y = data+1 , s = f'{data}' , fontdict = dict(fontsize = 15))

plt.xticks(x_pos , X)
plt.xlabel("Error Type")
plt.title("Error Comparison")
```

Out[38]:

```
Text(0.5, 1.0, 'Error Comparison')
```



K-Nearest Neighbours' model

In [39]:

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

In [40]:

```
sample_data = df.sample(120000)
```

In [41]:

```
sample_data["passenger_count"].value_counts()
```

Out[41]:

```
1    84730
2    17491
5     6444
3     5023
6     3962
4     2343
0        6
7        1
Name: passenger_count, dtype: int64
```

In [42]:

```
#seperate features and target
features = sample_data.drop(["id","vendor_id","trip_duration","pickup_datetime","dropoff_datetime","store_and_fwd_flag"],axis =
target = sample_data["trip_duration"]
added_data= pd.concat([sample_data, pd.get_dummies(sample_data[['passenger_count']].astype('str'))], axis=1)
added_data=added_data.drop(['id','vendor_id','pickup_datetime','dropoff_datetime','store_and_fwd_flag','passenger_count','trip_

added_data.head()
```

Out[42]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count_0	passenger_count_1	passenger_count_2	pas
436739	-73.991905	40.738007	-73.998337	40.724869	0	1	0	
157879	-73.948204	40.784756	-73.998138	40.731983	0	1	0	
96128	-73.969994	40.764622	-73.963097	40.774986	0	1	0	
467005	-73.972733	40.762875	-74.014153	40.717640	0	0	1	
174799	-73.961845	40.773849	-73.954910	40.767288	0	0	1	

In [43]:

```
x=added_data
y=sample_data['trip_duration']
x.shape,y.shape
```

Out[43]:

```
((120000, 12), (120000,))
```

In [44]:

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

In [45]:

```
x = pd.DataFrame(x_scaled,columns=x.columns)
x.head()
```

Out[45]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count_0	passenger_count_1	passenger_count_2	passeng
0	0.161597	0.064143	0.215802	0.086458	0.0	1.0	0.0	
1	0.178959	0.077928	0.215876	0.088497	0.0	1.0	0.0	
2	0.170302	0.071991	0.228858	0.100825	0.0	1.0	0.0	
3	0.169214	0.071475	0.209943	0.084386	0.0	0.0	1.0	
4	0.173539	0.074712	0.231891	0.098618	0.0	0.0	1.0	

In [46]:

```
#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=40)
```

KNN REGRESSOR

In [47]:

```
# 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 [48]:

```
#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 RMSE
test_predict = reg.predict(test_x)
k= sqrt(mse(test_predict,test_y))
print("RMSE = ", k)
```

```
RMSE =  3539.9165199655486
```

ELBOW

In [49]:

```
# Using elbow method
def Elbow(k):
    # Initiating empty list
    test_rmse = []

    # Training model for every 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 the empty list calculated using the predictions
        tmp = reg.predict(test_x)
        tmp = sqrt(mse(tmp, test_y))
        test_rmse.append(tmp)

    return test_rmse
```

In [50]:

```
 #Defining K range
k = range(1, 50)
```

In [51]:

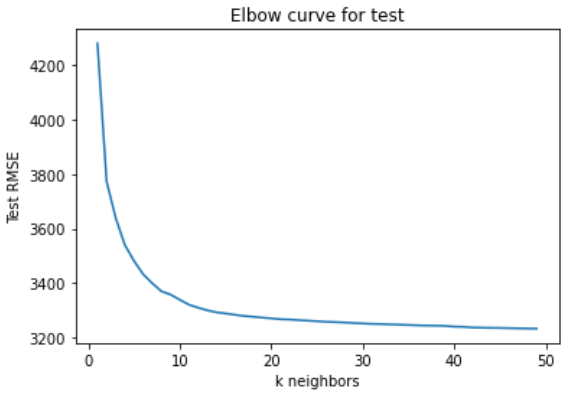
```
test=Elbow(k)
```

In [52]:

```
# plotting the Curves
plt.plot(k,test)
plt.xlabel("k neighbors")
plt.ylabel("Test RMSE")
plt.title("Elbow curve for test")
```

Out[52]:

```
Text(0.5, 1.0, 'Elbow curve for test')
```



In [53]:

```
#creating instance of KNN
reg = KNN(n_neighbors = 6)

#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  3432.813786478796
```

In [59]:

```
knn_train_rate = reg.score(train_x,train_y)
knn_train_rate*100
```

Out[59]:

```
19.000445247735843
```

In [60]:

```
knn_test_rate = reg.score(test_x,test_y)
knn_test_rate*100
```

Out[60]:

```
-11.308299988179172
```

In [56]:

```
%store knn_test_rate
%store knn_train_rate
```

```
Stored 'knn_test_rate' (float64)
Stored 'knn_train_rate' (float64)
```

linear model

In [62]:

```
features = sample_data.drop(["id","vendor_id","trip_duration","pickup_datetime","dropoff_datetime","store_and_fwd_flag"],axis =
target = sample_data["trip_duration"]
```

In [63]:

```
added_data= pd.concat([sample_data, pd.get_dummies(sample_data[['passenger_count']].astype('str'))], axis=1)
added_data=added_data.drop(['id','vendor_id','pickup_datetime','dropoff_datetime','store_and_fwd_flag','passenger_count','trip_

added_data.head()
```

Out[63]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count_0	passenger_count_1	passenger_count_2	pas
436739	-73.991905	40.738007	-73.998337	40.724869	0	1	0	
157879	-73.948204	40.784756	-73.998138	40.731983	0	1	0	
96128	-73.969994	40.764622	-73.963097	40.774986	0	1	0	
467005	-73.972733	40.762875	-74.014153	40.717640	0	0	1	
174799	-73.961845	40.773849	-73.954910	40.767288	0	0	1	

In [64]:

```
x=added_data
y=sample_data['trip_duration']
x.shape,y.shape
```

Out[64]:

```
((120000, 12), (120000,))
```

In [65]:

```
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 [66]:

```
from sklearn.linear_model import LinearRegression as LR
from sklearn.metrics import mean_squared_error as mse
```

In [67]:

```
lrr=LR()
lrr.fit(train_x,train_y)
```

Out[67]:

```
LinearRegression()
```

In [68]:

```
train_predict = lrr.predict(train_x)
k = sqrt(mse(train_predict,train_y))
print("train",k)
```

```
train 3171.3198101545495
```

In [69]:

```
test_predict = lrr.predict(test_x)
k = sqrt(mse(test_predict,test_y))
print("test ",k)
```

```
test  3085.158810039596
```

In [71]:

```
lrr.coef_
```

Out[71]:

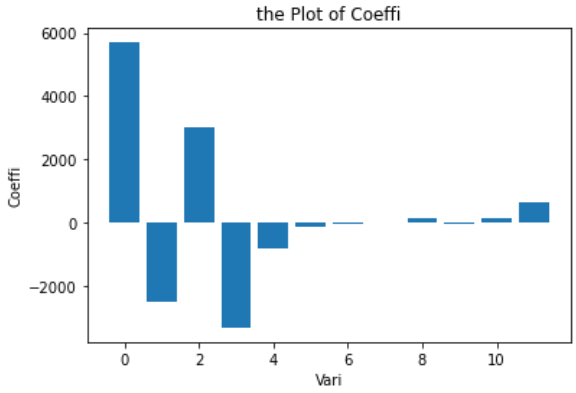
```
array([ 5694.10276561, -2484.23301957,  2997.36723117, -3310.41766584,
        -811.69149025,  -119.96272579,   -45.07535652,    27.3616685 ,
         164.85342283,   -22.15964087,   166.13820625,   640.53591585])
```

In [72]:

```
x = range(len(train_x.columns))
y = lrr.coef_
plt.bar(x,y)
plt.xlabel("Vari")
plt.ylabel("Coeffi")
plt.title("the Plot of Coeffi")
```

Out[72]:

```
Text(0.5, 1.0, 'the Plot of Coeffi')
```



In [73]:

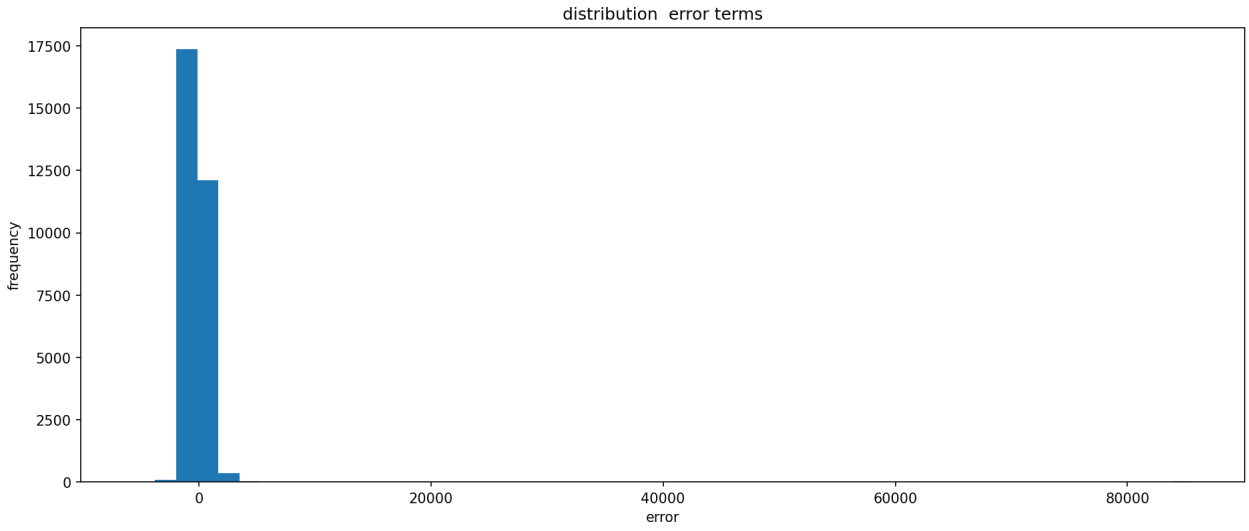
```
residuals = pd.DataFrame({
    "fitted_values":test_y,
    "predicted_values":test_predict
})
residuals["residuals"]=residuals["fitted_values"]-residuals["predicted_values"]
residuals.head()
```

Out[73]:

	fitted_values	predicted_values	residuals
503283	237	795.322839	-558.322839
504474	501	729.120602	-228.120602
157570	597	780.973099	-183.973099
622519	999	885.192803	113.807197
50568	193	846.234180	-653.234180

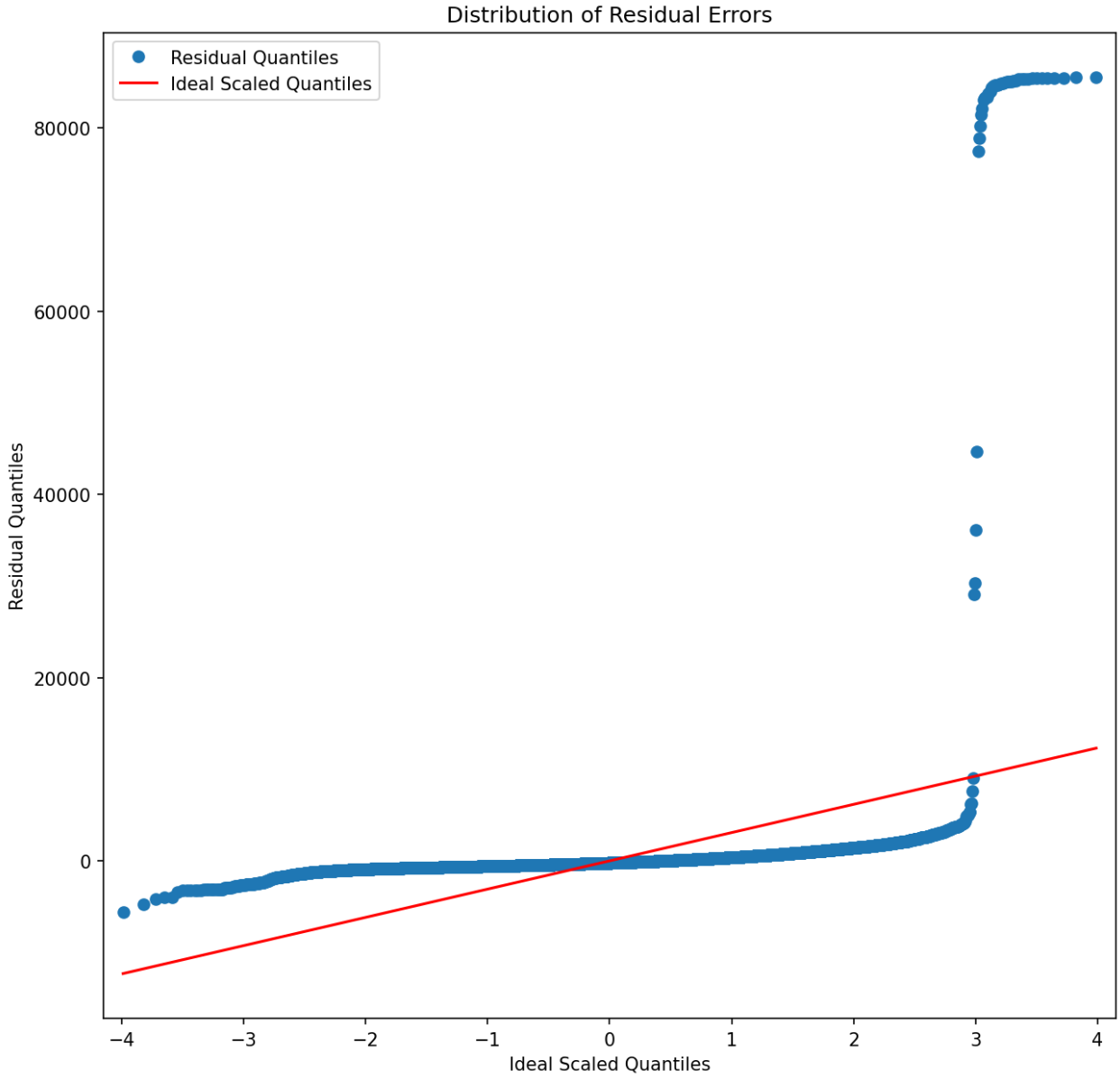
In [76]:

```
plt.figure(figsize=(15,6),dpi=150)
plt.hist(residuals.residuals,bins=50)
plt.xlabel("error")
plt.ylabel("frequency")
plt.title("distribution  error terms")
plt.show()
```



In [77]:

```
from statsmodels.graphics.gofplots import qqplot

# Plotting the qq plot
fig, ax = plt.subplots(figsize=(10,10) , 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 [78]:

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

Out[78]:

```
0.9638108643812426
```

In [79]:

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

Out[79]:

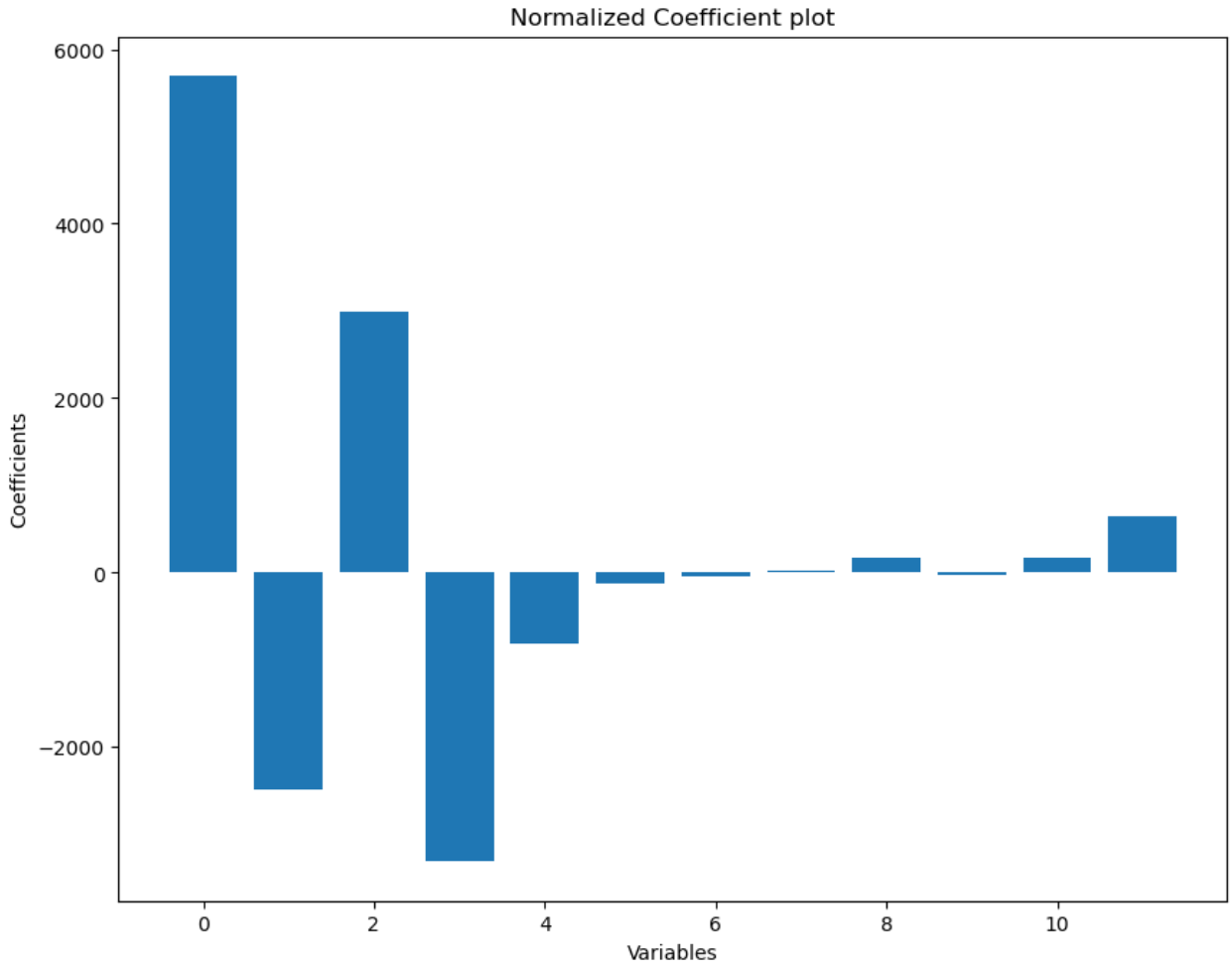
```
0.8230563689823889
```

In [80]:

```
plt.figure(figsize=(10, 8), dpi=100)
x = range(len(train_x.columns))
lrr.coef_
plt.bar( x, y )
plt.xlabel( "Variables")
plt.ylabel('Coefficients')
plt.title('Normalized Coefficient plot')
```

Out[80]:

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



In [82]:

```
%store linear_train_score
%store linear_test_score
```

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

Decision tree model

In [83]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsRegressor
```

In [86]:

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

In [87]:

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

In [88]:

```
df_y = np.log(data['trip_duration'])

df.loc[:, 'pickup_weekday']  = df['pickup_datetime'].dt.weekday
df.loc[:, 'pickup_weekofyear'] = df['pickup_datetime'].dt.weekofyear
df.loc[:, 'pickup_hour'] = df['pickup_datetime'].dt.hour
df.loc[:, 'pickup_minute'] = df['pickup_datetime'].dt.minute
df.loc[:, 'pickup_minute'] = df['pickup_datetime'].dt.minute
df.loc[:, 'pickup_dt'] = (df['pickup_datetime'] - df['pickup_datetime'].min()).dt.total_seconds()
df.loc[:, 'pickup_week_hour'] = df['pickup_weekday'] * 24 + df['pickup_hour']
```

DISTANCE EUCLIDEAN

In [89]:

```
y_dist = df['pickup_longitude'] - df['dropoff_longitude']
x_dist = df['pickup_latitude'] - df['dropoff_latitude']
df['dist_sq'] = (y_dist ** 2) + (x_dist ** 2)
df['dist_sq'] = df['dist_sq'] ** 0.5
```

In [91]:

```
df['pickup_latitude_round3'] = np.round(df['pickup_latitude'], 3)
df['pickup_longitude_round3'] = np.round(df['pickup_longitude'], 3)

df['dropoff_latitude_round3'] = np.round(df['dropoff_latitude'], 3)
df['dropoff_longitude_round3'] = np.round(df['dropoff_longitude'], 3)
```

In [92]:

```
def harvesine_array(lat1, lng1, lat2, lng2):
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    AVG_EARTH_RADIUS = 6371
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5)**2 + np.cos(lat1) *  np.cos(lat2) * np.sin(lng * 0.5)**2
    h = 2 *  AVG_EARTH_RADIUS + np.arcsin(np.sqrt(d))
    return h

def direction_array(lat1, lng1, lat2, lng2):
    AVG_EARTH_RADIUS = 6371
    lng_delta_rad = np.radians(lng2 - lng1)
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2) * np.cos(lng_delta_rad)
    return np.degrees(np.arctan(y,x))

df['harvesine_distance'] = harvesine_array(df['pickup_latitude'].values, df['pickup_longitude'].values, df['dropoff_latitude'].
df['direction'] = direction_array(df['pickup_latitude'].values, df['pickup_longitude'].values, df['dropoff_latitude'].values, d
```

In [93]:

```
df['vendor_id'] = df['vendor_id'] - 1
```

In [94]:

```
np.sum(pd.isnull(df))
```

Out[94]:

```
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
pickup_weekday              0
pickup_weekofyear           0
pickup_hour                 0
pickup_minute               0
pickup_dt                   0
pickup_week_hour            0
dist_sq                     0
harvesine_distance          0
direction                   0
pickup_latitude_round3      0
pickup_longitude_round3     0
dropoff_latitude_round3     0
dropoff_longitude_round3    0
dtype: int64
```

In [95]:

```
df.fillna(0, inplace=True)
```

In [96]:

```
df = df.drop(['id', 'pickup_datetime', 'dropoff_datetime', 'trip_duration', 'store_and_fwd_flag'], axis=1)
```

In [97]:

```
from sklearn.metrics import mean_squared_error
from math import sqrt
```

In [103]:

```
xtrain, xtest, ytrain, ytest = train_test_split(df, df_y, test_size=1/3, random_state=40)
```

In [104]:

```
mean_pred = np.repeat(ytrain.mean(), len(ytest))
sqrt(mean_squared_error(ytest, mean_pred))
```

Out[104]:

```
0.8026060258650528
```

In [118]:

```
def cv_score(ml_model, rstate=11, cols = df.columns):
    i = 1
    cv_scores = []
    df1 = df.copy()
    df1 = df[cols]

    kf = KFold(n_splits=5, random_state=rstate, shuffle=True)

    for train_index, test_index in kf.split(df1, df_y):
        print('\n {} of Kfold {}'.format(i, kf.n_splits))
        xtr, xvl = df1.loc[train_index], df1.loc[test_index]
        ytr, yvl = df_y[train_index], df_y[test_index]

        model = ml_model
        model.fit(xtr, ytr)
        train_val = model.predict(xtr)
        pred_val = model.predict(xvl)

        rmse_score_train = sqrt(mean_squared_error(ytr, train_val))
        rmse_score = sqrt(mean_squared_error(yvl, pred_val))
        suffix = ""
        msg = ""

        msg += "Valid RMSE: {:5f}".format(rmse_score)
        print(msg)

        cv_scores.append(rmse_score)
        i += 1
    return cv_scores
```

In [119]:

```
linreg_score = cv_score(LinearRegression())
```

```
 1 of Kfold 5
Valid RMSE: 0.642333

 2 of Kfold 5
Valid RMSE: 0.637824

 3 of Kfold 5
Valid RMSE: 1.014804

 4 of Kfold 5
Valid RMSE: 0.651147

 5 of Kfold 5
Valid RMSE: 0.663225
```

In [122]:

```
dtree_score = cv_score(DecisionTreeRegressor(min_samples_leaf=25, min_samples_split=40))
```

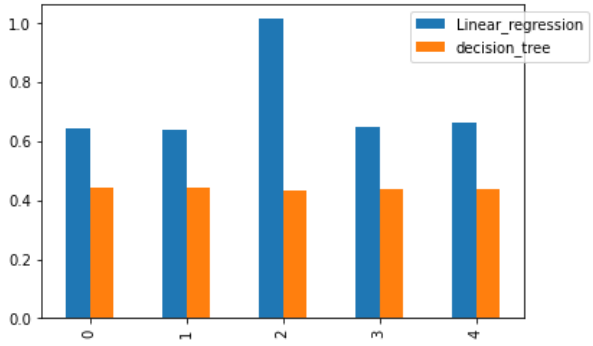
```
 1 of Kfold 5
Valid RMSE: 0.441791

 2 of Kfold 5
Valid RMSE: 0.443386

 3 of Kfold 5
Valid RMSE: 0.434108

 4 of Kfold 5
Valid RMSE: 0.438741

 5 of Kfold 5
Valid RMSE: 0.439476
```

In [123]:

```
results_df = pd.DataFrame({'Linear_regression': linreg_score, 'decision_tree': dtree_score})
```

In [126]:

```
results_df.plot(y = ['Linear_regression', 'decision_tree'], kind='bar', legend=False)
plt.legend(bbox_to_anchor = (1.15, 1), loc=1, borderaxespad = 0.5)
plt.show()
```



In [127]:

```
from sklearn import tree
```

In [128]:

```
dtree = DecisionTreeRegressor(min_samples_leaf=25, min_samples_split=25)
dtree.fit(xtrain, ytrain)
```

Out[128]:

```
DecisionTreeRegressor(min_samples_leaf=25, min_samples_split=25)
```

In [129]:

```
knn_train_rate,linear_train_score
```

Out[129]:

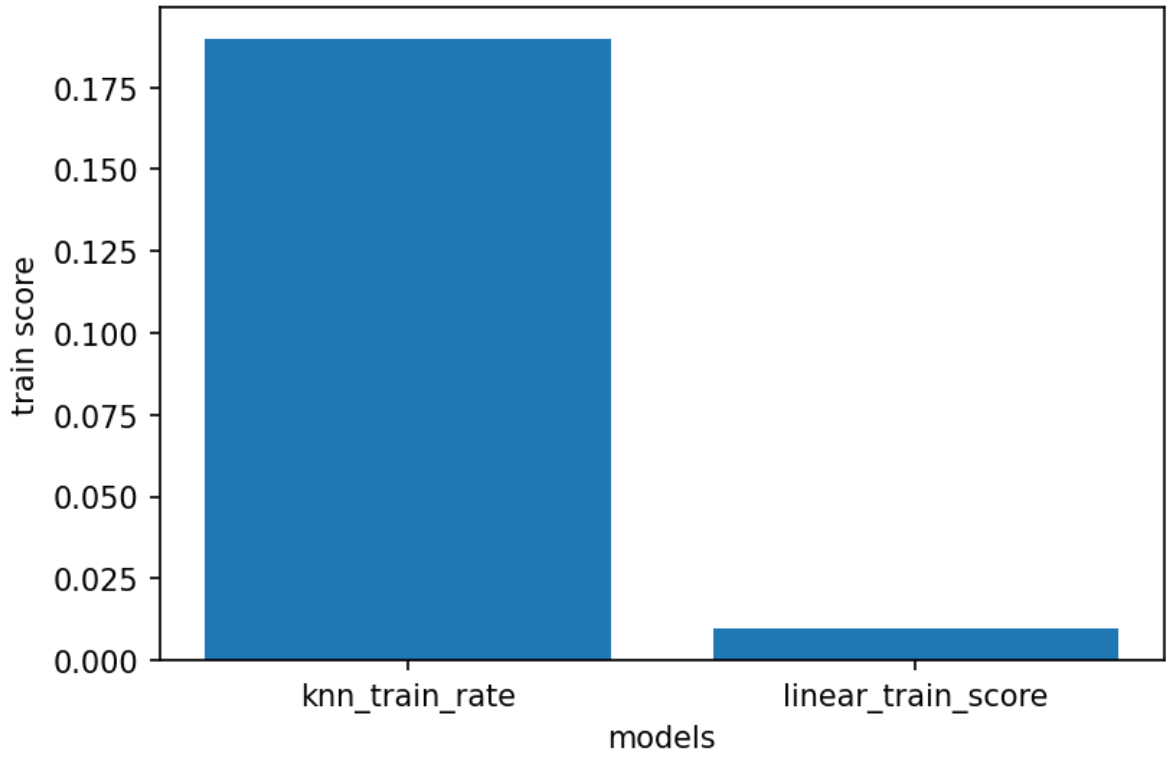
```
(0.19000445247735842, 0.009638108643812426)
```

In [131]:

```
x=["knn_train_rate","linear_train_score"]
y=[round(knn_train_rate,10), round(linear_train_score,5)]
```

In [132]:

```
plt.figure(dpi=150)
plt.bar(x,y)
plt.xlabel("models")
plt.ylabel("train score")
plt.show()
```



In [133]:

```
# Test values in x,y
knn_test_rate,linear_test_score
```

Out[133]:

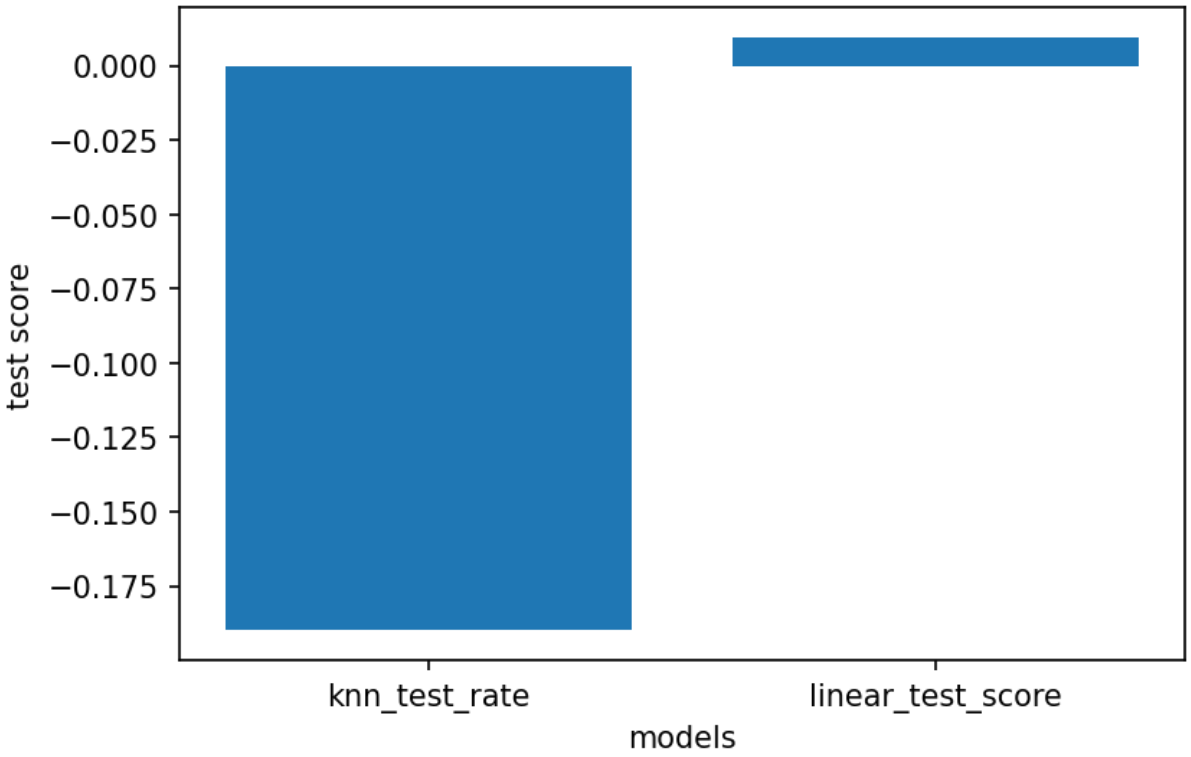
```
(-0.11308299988179171, 0.008230563689823889)
```

In [134]:

```
x=["knn_test_rate","linear_test_score"]
y=[-round(knn_train_rate,5), round(linear_train_score,5)]
```

In [135]:

```
plt.figure(dpi=150)
plt.bar(x,y)
plt.xlabel("models")
plt.ylabel("test score")
plt.show()
```



In [ ]: