In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from haversine import haversine
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
```

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

In [3]:
```python
#checking the dataset
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]:
```python
data.shape
```

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

In [5]:
```python
#checking for null values in the dataset
data.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
```

- There are no null values in this dataset

In [6]:
```python
#converting datetime variable
data['pickup_datetime'] = pd.to_datetime(data['pickup_datetime'])
data['dropoff_datetime'] = pd.to_datetime(data['dropoff_datetime'])
```

In [7]:
```python
#Generating new variables from datetime variable
data['year'] = data.pickup_datetime.dt.year
data['month'] = data.pickup_datetime.dt.month
data['weekno.'] = data.pickup_datetime.dt.weekday
data['hour'] = data.pickup_datetime.dt.hour
```

In [8]:
```python
#using haversine for calculation of distance
def dist(df):
    pick = (df['pickup_latitude'], df['pickup_longitude'])
    drop = (df['dropoff_latitude'], df['dropoff_longitude'])
    return haversine(pick,drop)
```

In [9]:
```python
#Calculating speed and distance for each trip
data['distance'] = data.apply(lambda x: dist(x), axis = 1)
data['speed'] = (data.distance/(data.trip_duration/3600))
```

In [10]:
```python
#dummy encoding the categorical variable 'store_and_fwd_flag'
data = pd.concat([data, pd.get_dummies(data['store_and_fwd_flag'], prefix= 'flag')], axis= 1).drop(['store_and_fwd_flag'],axis= 1)
```

In [11]:
```python
#dropping unnecessary variables
data = data.drop(['pickup_datetime', 'dropoff_datetime', 'pickup_longitude', 'pickup_latitude','dropoff_longitude', 'dropoff_latitude','flag_Y'], axis= 1)
```

In [12]:
```python
data.head()
```

Out[12]:

	id	vendor_id	passenger_count	trip_duration	year	month	weekno.	hour	distance	speed	flag_N
0	id1080784	2	1	400	2016	2	0	16	1.199074	10.791668	1
1	id0889885	1	2	1100	2016	3	4	23	4.129117	13.513473	1
2	id0857912	2	2	1635	2016	2	6	17	7.250763	15.964983	1
3	id3744273	2	6	1141	2016	1	1	9	2.361101	7.449573	1
4	id0232939	1	1	848	2016	2	2	6	4.328540	18.375877	1

In [13]:
```python
data.shape
```

Out[13]:
```
(729322, 11)
```

In [14]:
```python
data.dtypes
```

Out[14]:
```
id                  object
vendor_id            int64
passenger_count      int64
trip_duration        int64
year                 int64
month                int64
weekno.              int64
hour                 int64
distance           float64
speed              float64
flag_N               uint8
dtype: object
```

Removing outliers and tuning the data

In [15]:
```python
data.passenger_count.value_counts()
```

Out[15]:
```
1    517415
2    105097
5     38926
3     29692
6     24107
4     14050
0        33
7         1
9         1
Name: passenger_count, dtype: int64
```

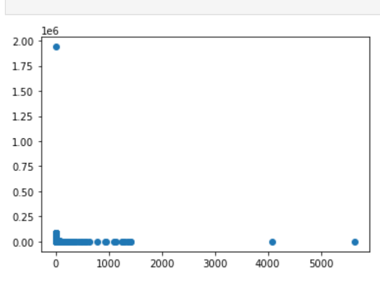
In [16]:
```python
data.passenger_count.describe()
```

Out[16]:
```
count    729322.000000
mean          1.662055
std           1.312446
min           0.000000
25%           1.000000
50%           1.000000
75%           2.000000
max           9.000000
Name: passenger_count, dtype: float64
```

- There are 33 trips with 0 passenger counts. Since the mean is approx 1 we will convert 0 passenger counts to 1.
- We will remove passenger count more than 6 as we consider it an outlier.

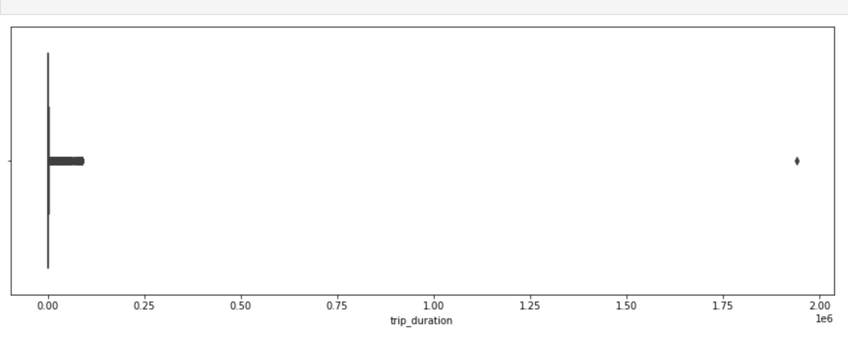
In [17]:
```python
data['passenger_count'] = data.passenger_count.apply(lambda x : 1 if x==0 else x)
data = data[data.passenger_count <7]
```

In [18]:
```python
#checking speed and trip duration
plt.scatter(data['speed'],data['trip_duration'])
plt.show()
```



- the outliers we see in this plot will cause inconsistency in our predictions.

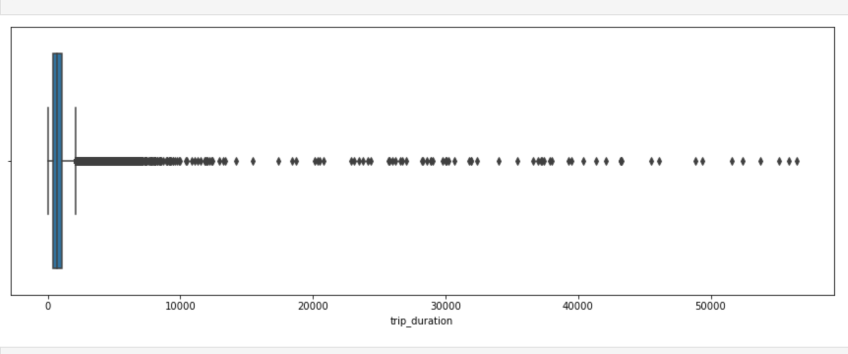
In [19]:
```python
#plotting trip duration to check for outliers
plt.figure(figsize=(15,5))
sns.boxplot(data.trip_duration)
plt.show()
```



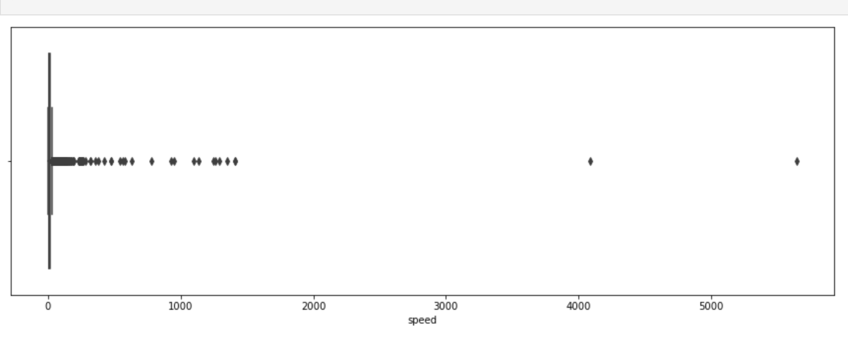
- We will focus on the data with less than 24 hours of trip duration to avoid inconsistency due to outliers.

In [20]:
```python
data = data[data.trip_duration <= 57600]
```

In [21]:
```python
plt.figure(figsize=(15,5))
sns.boxplot(data.trip_duration)
plt.show()
```

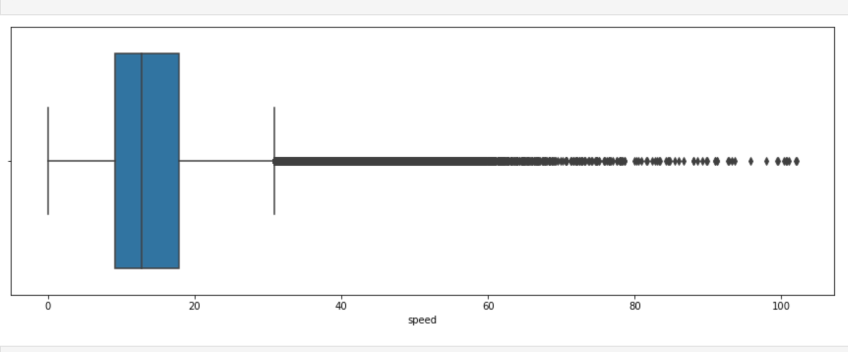


In [22]:
```python
#checking for outliers in speed
plt.figure(figsize=(15,5))
sns.boxplot(data.speed)
plt.show()
```



- the speed limit in NYC is 105km/h so we remove the data with speed more than 105km/h.

In [23]:
```python
data= data[data.speed<= 105]
plt.figure(figsize=(15,5))
sns.boxplot(data.speed)
plt.show()
```

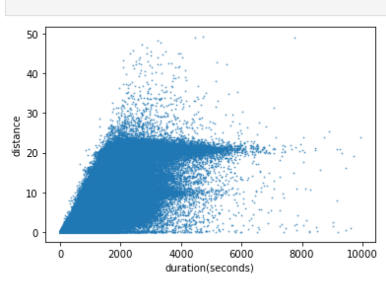


In [24]:
```python
#typecasting the categorical variable and dropping the 'id' variable.
data = data.drop(['id'], axis = 1)
data['vendor_id'] = data['vendor_id'].astype('category')
```

In [25]:
```python
data.shape
```

Out[25]:
```
(728287, 10)
```

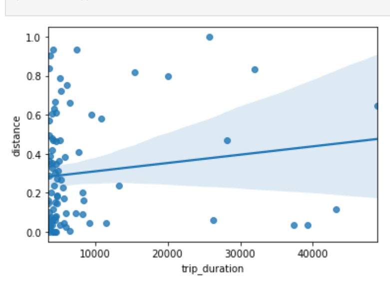
In [26]:
```python
# looking at data of less than 3 hours and less than 50 km of distance travelled.
bi_dist = data.loc[(data.trip_duration<10000) & (data.distance< 50)]
plt.scatter(bi_dist.trip_duration, bi_dist.distance, s=1, alpha=0.5)
plt.xlabel('duration(seconds)')
plt.ylabel('distance')
plt.show()
```



- there are a lot of trips with more than 30 and 60 minutes which cover 0km distance. It rarely happens that passenger keeps sitting in the taxi for more than and hour and travels nowhere.

In [27]:
```python
#focusing on distances with only 1km and more than 60 minutes of trip duration.
min= data.loc[(data['distance'] <= 1) & (data['trip_duration'] >= 3600),['distance','trip_duration']].reset_index(drop=True)
```

In [28]:
```python
sns.regplot(min.trip_duration, min.distance)
plt.show()
```



- Although having some linear relationship, still it is not a linear distribution and shows no correlation between the two.
- these values must be removed for better consistency in our dataset.

In [29]:
```python
data= data[~((data['trip_duration'] <=1) & (data['distance']<=3600))]
```

In [30]:
```python
data.shape
```

Out[30]:
```
(728276, 10)
```

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

In [31]:
```python
#separating the independent and dependent variables.
x= data.drop(['trip_duration'], axis = 1)
y = data['trip_duration']
```

In [32]:
```python
from sklearn.preprocessing import StandardScaler
#scaling the dataset
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
#putting the array into dataframe
x= pd.DataFrame(x_scaled)
```

In [33]:
```python
from sklearn.model_selection import train_test_split as tts
```

- we will use the same split data for all the models in this notebook

In [34]:
```python
# splitting the data
train_x, test_x ,train_y, test_y = tts(x,y,test_size = 0.25, random_state= 12)
train_x.shape, test_x.shape, train_y.shape , test_y.shape
```

Out[34]:
```
((546207, 9), (182069, 9), (546207,), (182069,))
```

In [35]:
```python
from sklearn.neighbors import KNeighborsRegressor as KNN
```

In [36]:
```python
# Creating instance of KNN
kreg = KNN()
```

In [37]:
```python
#importing mean squared error and r square
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score
```

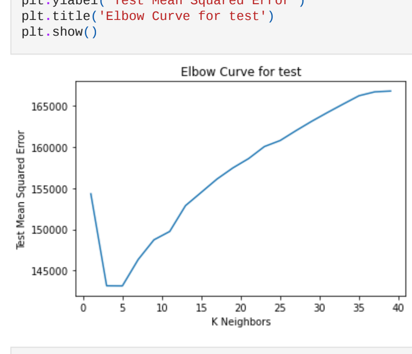
In [38]:
```python
def Elbow(K):
  #initiating empty list
    test_error = []

  #training model for every value of K
    for i in K:
        #Instance of KNN
        kreg = KNN(n_neighbors = i,n_jobs = -1)
        kreg.fit(train_x, train_y)
        #Appending mse value to empty list calculated using the predictions
        tmp = kreg.predict(test_x)
        tmp = mse(tmp,test_y)
        test_error.append(tmp)

    return test_error
```

In [39]:
```python
#setting k range and assigning it to a variable
k = range(1,40,2)
test = Elbow(k)
```

In [40]:
```python
#plotting the elbow curve
plt.plot(k, test)
plt.xlabel('K Neighbors')
plt.ylabel('Test Mean Squared Error')
plt.title('Elbow Curve for test')
plt.show()
```



In [41]:
```python
# Creating instance of KNN
kreg = KNN(n_neighbors = 5)

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

# Predicting over the Train Set and calculating MSE
pred_knn = kreg.predict(test_x)
k = r2_score(test_y, pred_knn)
print('R squared error:', k)
```

```
R squared error: 0.7543655349328509
```

2. Build a Linear model for the given dataset with regularisation.

In [42]:
```python
from numpy import arange
from sklearn.linear_model import Lasso
from sklearn.metrics import r2_score
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import RandomizedSearchCV
```

In [43]:
```python
from sklearn.metrics import mean_absolute_error
```

In [44]:
```python
# define model
lassoreg = Lasso()
# define model evaluation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['alpha'] = arange(0,100,0.01)
# define search
search = RandomizedSearchCV(lassoreg, grid, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)
# perform the search
results = search.fit(x, y)

print('mae: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
```

```
mae: -192.161
Config: {'alpha': 14.96}
```

In [46]:
```python
#training the model
lassoreg = Lasso(alpha = 14.96)
lassoreg.fit(train_x,train_y)
```

Out[46]:
```
Lasso(alpha=14.96)
```

In [47]:
```python
#calculating the r squared error
pred_lasso = lassoreg.predict(test_x)
r_squared = r2_score(test_y,pred_lasso)
print('Test R squared error is:', r_squared)
```

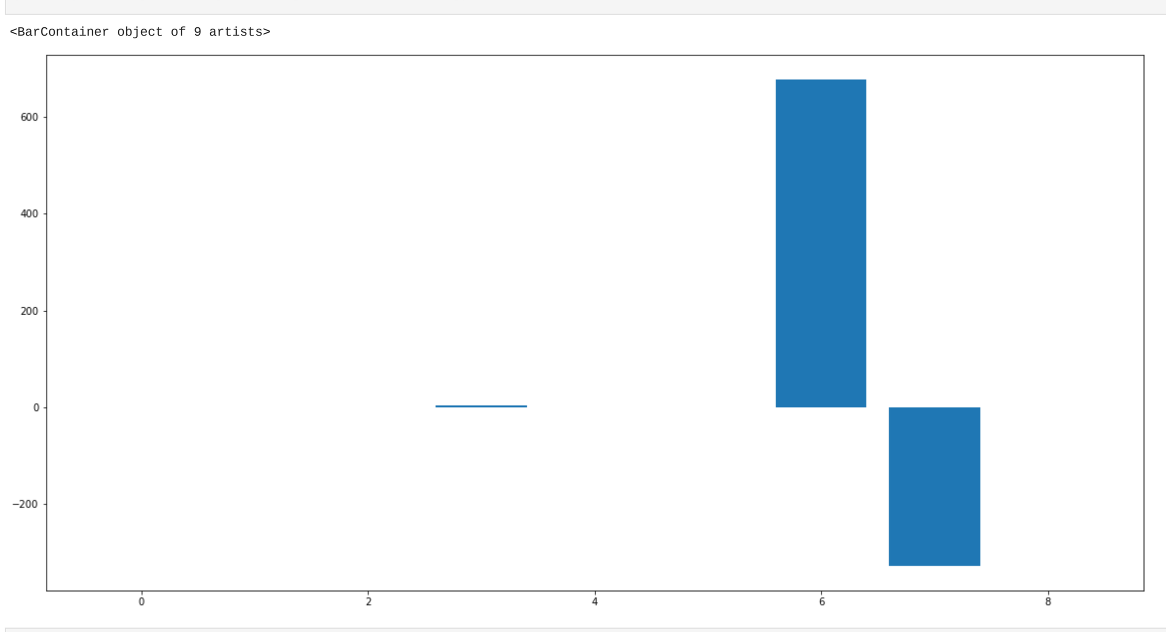
```
Test R squared error is: 0.6080526076906043
```

In [48]:
```python
# putting together the coefficients and their corresponding variable names
coeff_df = pd.DataFrame()
coeff_df['Columns'] = train_x.columns
coeff_df['coefficient_values'] = pd.Series(lassoreg.coef_)
print(coeff_df.head(15))
```

```
   Columns  coefficient_values
0        0            0.000000
1        1            0.000000
2        2            0.000000
3        3            3.299920
4        4           -0.000000
5        5           -0.000000
6        6          678.394727
7        7         -328.995558
8        8           -0.000000
```

In [49]:
```python
#plotting the coefficient values and columns.
plt.rcParams['figure.figsize']= 20,10
plt.bar(coeff_df['Columns'],coeff_df['coefficient_values'])
```

Out[49]:
```
<BarContainer object of 9 artists>
```



In [50]:
```python
data.shape
```

Out[50]:
```
(728276, 10)
```

3. Build a Random Forest model for the given dataset.

In [51]:
```python
#importing random forest regressor
from sklearn.ensemble import RandomForestRegressor
```

In [52]:
```python
Reg = RandomForestRegressor()
```

In [53]:
```python
#importing random search for better hyperparameters
from sklearn.model_selection import RandomizedSearchCV
```

In [54]:
```python
#Giving inputs for hyperparameter search
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 200, num = 10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10,100, num = 11)]
max_depth.append(None)
min_samples_split = [2,4,8]
```

In [55]:
```python
#putting the lists in a dictionary
random_search = {'n_estimators': n_estimators,
                 'max_features':max_features,
                 'max_depth':max_depth,
                 'min_samples_split': min_samples_split}
print(random_search)
```

```
{'n_estimators': [10, 31, 52, 73, 94, 115, 136, 157, 178, 200], 'max_features': ['auto', 'sqrt'], 'max_depth': [10, 19, 28, 37, 46, 55, 64, 73, 82, 91, 100, None], 'min_samples_split': [2, 4, 8]}
```

In [56]:
```python
random_grid = RandomizedSearchCV(estimator = Reg, param_distributions = random_search, scoring = r2_score, cv = 3, random_state = 21, verbose = 2, n_jobs = -1)
```

In [57]:
```python
random_grid.fit(train_x, train_y)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

Out[57]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_jobs=-1,
                   param_distributions={'max_depth': [10, 19, 28, 37, 46, 55,
                                                      64, 73, 82, 91, 100,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_split': [2, 4, 8],
                                        'n_estimators': [10, 31, 52, 73, 94,
                                                         115, 136, 157, 178,
                                                         200]},
                   random_state=21,
                   scoring=<function r2_score at 0x0000023505A72820>,
                   verbose=2)
```

In [58]:
```python
random_grid.best_params_
```

Out[58]:
```
{'n_estimators': 136,
 'min_samples_split': 4,
 'max_features': 'sqrt',
 'max_depth': 100}
```

In [59]:
```python
reg = RandomForestRegressor(n_estimators = 136, min_samples_split = 4, max_features = 'sqrt', max_depth = 100, random_state = 21)
```

In [60]:
```python
reg.fit(train_x,train_y)
```

Out[60]:
```
RandomForestRegressor(max_depth=100, max_features='sqrt', min_samples_split=4,
                      n_estimators=136, random_state=21)
```

In [61]:
```python
#calculating error
pred_rf = reg.predict(test_x)
r_squared = r2_score(test_y,pred_rf)
print(r_squared)
```

```
0.899216386878106
```

4. Build a Gradient Boosting model for the given dataset.

In [62]:
```python
# importing gradient boosting regressor
from sklearn.ensemble import GradientBoostingRegressor as gbr
```

In [63]:
```python
# fitting the model
grad = gbr(random_state = 21)
grad.fit(train_x,train_y)
```

Out[63]:
```
GradientBoostingRegressor(random_state=21)
```

In [64]:
```python
#calculating error
preds = grad.predict(test_x)
Error_rsq = r2_score(test_y,preds)
print(Error_rsq)
```

```
0.9451530602618291
```

In [65]:
```python
from sklearn.model_selection import RandomizedSearchCV
```

In [66]:
```python
#setting inputs for hyperparameter tuning
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 200, num = 10)]
max_depth = [int(x) for x in np.linspace(10,120, num = 11)]
max_depth.append(None)
learning_rate = [0.15,0.1,0.3,0.5,0.7,0.9,0.05]
```

In [67]:
```python
random_search1 = {'n_estimators': n_estimators,
                 'learning_rate' :[0.15,0.1,0.3,0.5,0.7,0.9,0.05],
                 'max_depth':max_depth
                 }
print(random_search1)
```

```
{'n_estimators': [10, 31, 52, 73, 94, 115, 136, 157, 178, 200], 'learning_rate': [0.15, 0.1, 0.3, 0.5, 0.7, 0.9, 0.05], 'max_depth': [10, 21, 32, 43, 54, 65, 76, 87, 98, 109, 120, None]}
```

In [68]:
```python
random_grid1 = RandomizedSearchCV(estimator = grad, param_distributions = random_search1, scoring = r2_score, cv= 3, random_state = 21, verbose = 2, n_jobs = -1)
```

In [69]:
```python
random_grid1.fit(train_x,train_y)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

Out[69]:
```
RandomizedSearchCV(cv=3, estimator=GradientBoostingRegressor(random_state=21),
                   n_jobs=-1,
                   param_distributions={'learning_rate': [0.15, 0.1, 0.3, 0.5,
                                                          0.7, 0.9, 0.05],
                                        'max_depth': [10, 21, 32, 43, 54, 65,
                                                      76, 87, 98, 109, 120,
                                                      None],
                                        'n_estimators': [10, 31, 52, 73, 94,
                                                         115, 136, 157, 178,
                                                         200]},
                   random_state=21,
                   scoring=<function r2_score at 0x0000023505A72820>,
                   verbose=2)
```

In [70]:
```python
random_grid1.best_params_
```

Out[70]:
```
{'n_estimators': 94, 'max_depth': None, 'learning_rate': 0.7}
```

In [71]:
```python
GBR = gbr(n_estimators = 94, max_depth = None, learning_rate = 0.7)
GBR.fit(train_x,train_y)
```

Out[71]:
```
GradientBoostingRegressor(learning_rate=0.7, max_depth=None, n_estimators=94)
```

In [72]:
```python
#calculating error
pred_gb = GBR.predict(test_x)
error = r2_score(test_y,pred_gb)
```

In [73]:
```python
# r squared error
print(error)
```

```
0.9591025901710818
```

In [74]:
```python
# getting train score
pr = GBR.predict(train_x)
print(r2_score(train_y,pr))
```

```
0.9993861830025543
```

5. Combine all the models above using the averaging technique to generate the final predictions.

In [86]:
```python
# averaging all the above models.
w_avg = ((pred_lasso) + (pred_knn) + (pred_rf) + (pred_gb))/(4)
r_sqr = r2_score(test_y,w_avg)
print('R squared error of weighted average of all models is: ',r_sqr)
```

```
R squared error of weighted average of all models is:  0.8703041881535512
```

In [ ]: