ford-car-price-prediction

July 19, 2023

Project: **Ford car price prediction**

Algorithm Used: **Multiple regression,Random forest regression,Gradient boost regressor and XGboost regressor**

Steps:

- Loading the dataset using pandas library
- preprocessing the dataset.
- Exploratory Data Analysis
- Model building using Multiple regression Algorithm, Rnadom forest regressor, Gradient boost regressor and XGboost regressor
- Model testing using MSE,MAE,R2-SCORE,RMSE

[ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error,mean_squared_error, r2_score
df=pd.read_csv('/content/ford.csv')
df
```

[ ]:
```
         model  year  price transmission  mileage fuelType  tax   mpg  \
0         Kuga  2019  18990       Manual     8389   Petrol  150  35.3
1       Fiesta  2019  21999       Manual     4000   Petrol  145  40.3
2          Ka+  2020  11999       Manual     2000   Petrol  145  43.5
```

```
3             KA  2018   9899       Manual     6000   Petrol  145  43.5
4            Ka+  2018   9999       Manual    15000   Petrol  145  43.5
…          …   …    …          …      …       …   …    …
17961    Mustang  2017  27890    Semi-Auto    26452   Petrol  580  23.5
17962    Mustang  2020  42999       Manual       10   Petrol  145  23.7
17963    Mustang  2020  48000       Manual       50   Petrol  145  23.9
17964    Mustang  2020  40495    Semi-Auto     3200   Petrol  145  24.8
17965     Mondeo  2017  15499    Automatic    10162   Petrol  235  38.2

       engineSize
0             0.0
1             0.0
2             0.0
3             0.0
4             0.0
…           …
17961         5.0
17962         5.0
17963         5.0
17964         5.0
17965         5.0

[17966 rows x 9 columns]
```

Data Preprocessing

[ ]:
```
df.dtypes
```

[ ]:
```
model            object
year              int64
price             int64
transmission     object
mileage           int64
fuelType         object
tax               int64
mpg             float64
engineSize      float64
dtype: object
```

[ ]:
```
df.isna().sum()
```

[ ]:
```
model           0
year            0
price           0
transmission    0
mileage         0
fuelType        0
```

```
tax              0
mpg              0
engineSize       0
dtype: int64
```

[ ]:
```
df.duplicated().sum()
```

[ ]:
```
154
```

[ ]:
```
#dropping duplicate values
df1 = df.drop_duplicates().reset_index(drop=True)
```

[ ]:
```
df1.duplicated().sum()
```

[ ]:
```
0
```

[ ]:
```
df1.shape
```

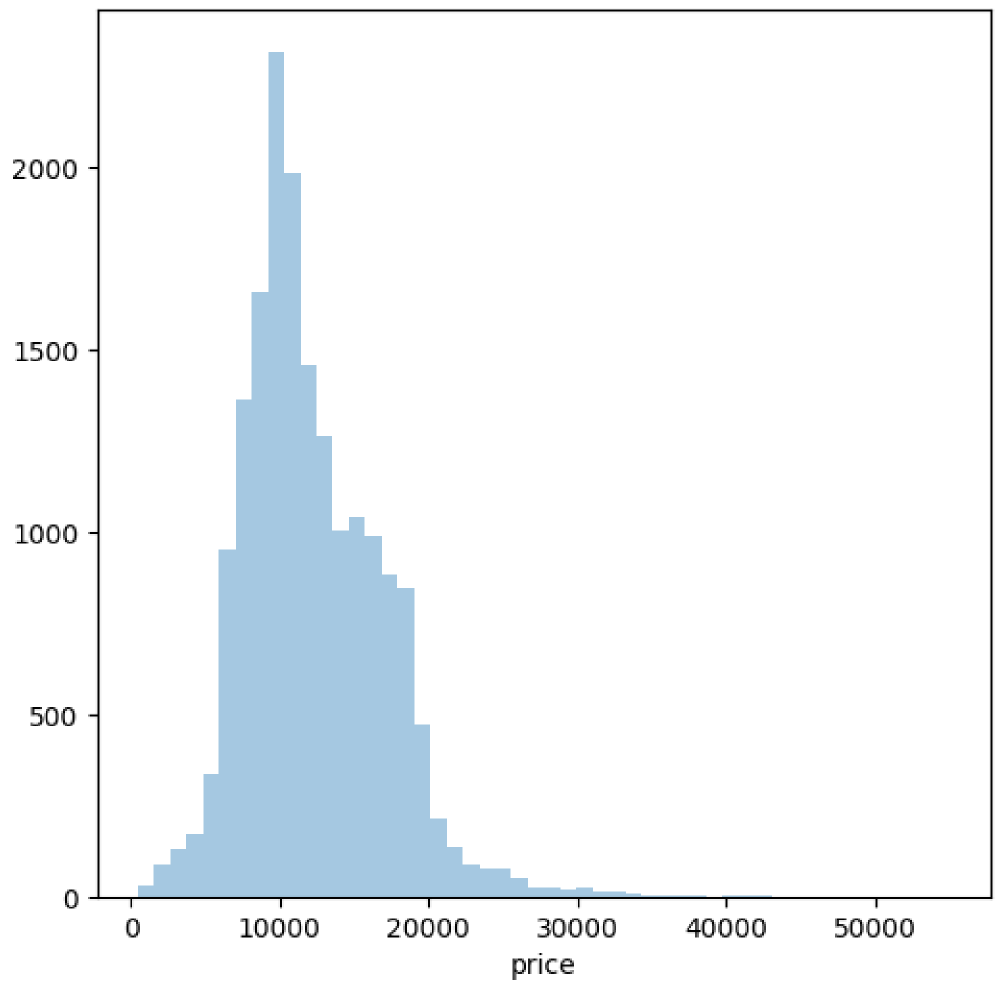
[ ]:
```
(17812, 9)
```

[ ]:
```
#removing negative values
df1=df1[df1['price'] >= 0]
```

Explortory Data Analysis

[ ]:
```
fig, ax = plt.subplots(figsize=(6,6))
sns.distplot(df1['price'],kde=False)
```

[ ]:
```
<Axes: xlabel='price'>
```



[ ]:
```
a=df1.groupby('model')['price'].sum().reset_index()
index_to_drop=23
a=a.drop(index_to_drop)
a1=a.sort_values(by='price',ascending=False)[['model','price']].head(10)
a1
```

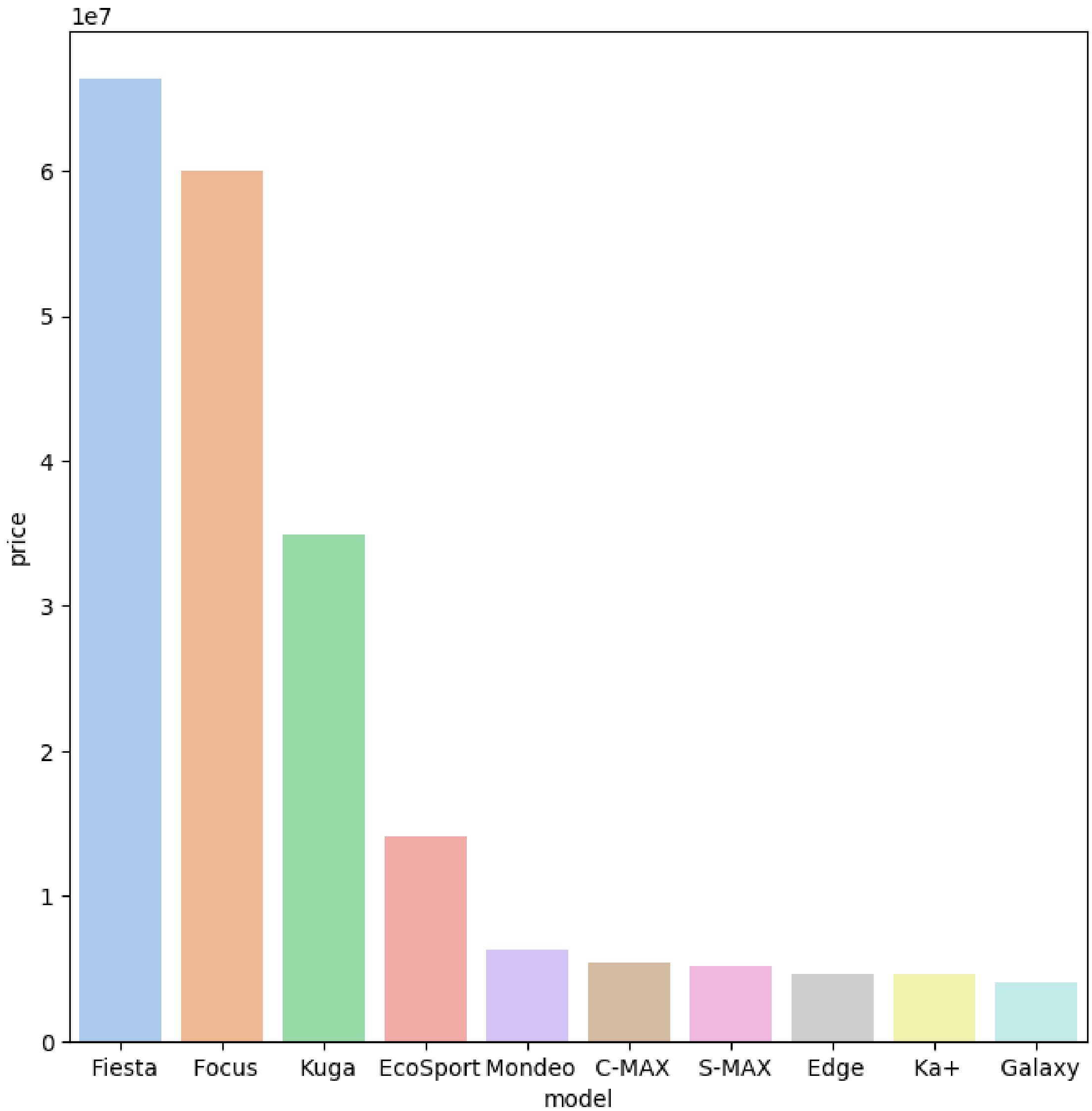
[ ]:

	model	price
5	Fiesta	66332283
6	Focus	60030224
13	Kuga	34928705
2	EcoSport	14052864
14	Mondeo	6276581
1	C-MAX	5373119
18	S-MAX	5190688
3	Edge	4655584
12	Ka+	4549602

```
8      Galaxy    4051648
```

[ ]:
```
fig,ax=plt.subplots(figsize=(8,8))
sns.barplot(x='model',y='price',data=a1,ax=ax,palette='pastel')
```

[ ]: <Axes: xlabel='model', ylabel='price'>



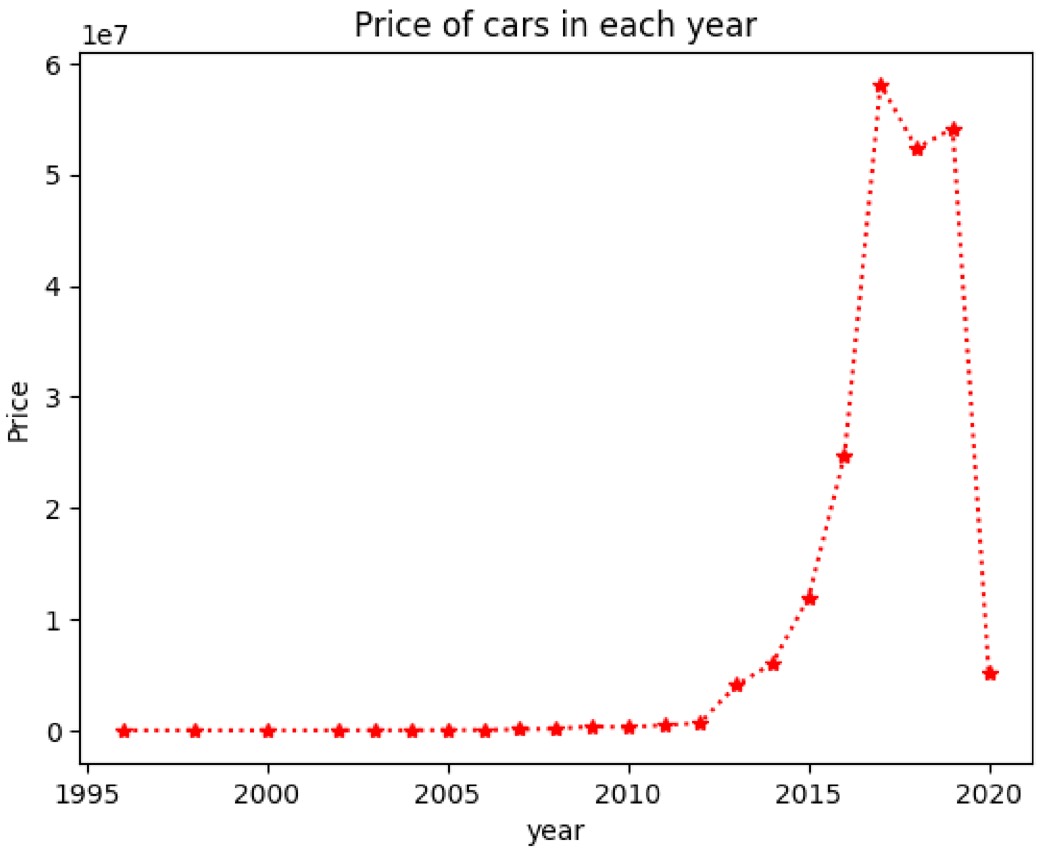
[43]:
```
b=df1.groupby('year')['price'].sum().reset_index()
b1=b.sort_values(by='year')
b1=b1.drop(22)
b1
```

[43]:

	year	price
0	1996	3000
1	1998	2699
2	2000	1995
3	2002	5785
4	2003	6189
5	2004	5744
6	2005	25488
7	2006	28634
8	2007	83314
9	2008	146342
10	2009	338495
11	2010	271903
12	2011	472113
13	2012	653307
14	2013	4061935
15	2014	6021380
16	2015	11908049
17	2016	24677121
18	2017	58082890
19	2018	52355498
20	2019	54139409
21	2020	5247552

[45]:

```
x=b1['year']
y=b1['price']
plt.plot(x,y,"*:r")
plt.xlabel("year")
plt.ylabel("Price")
plt.title("Price of cars in each year")
plt.show()
```

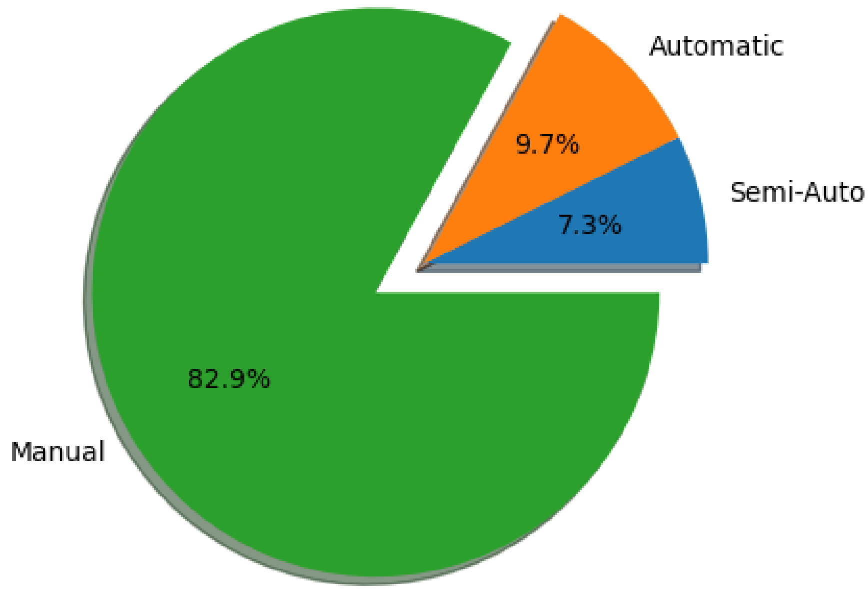


[ ]:
```
c=df1.groupby('transmission')['price'].sum().reset_index()
c1=c.sort_values(by='price')
c1
```

[ ]:

	transmission	price
2	Semi-Auto	16015323
0	Automatic	21262532
1	Manual	181267482

[ ]:
```
y=c1['price']
labels=['Semi-Auto','Automatic','Manual']
myexplode=[0,0,0.2]
plt.pie(y,labels=labels,explode = myexplode,shadow=True,autopct='%1.1f%%')
plt.show()
```



[ ]:
```
d=df1.groupby('fuelType')['tax'].sum().reset_index()
d1=d.sort_values(by='tax')
d1
```

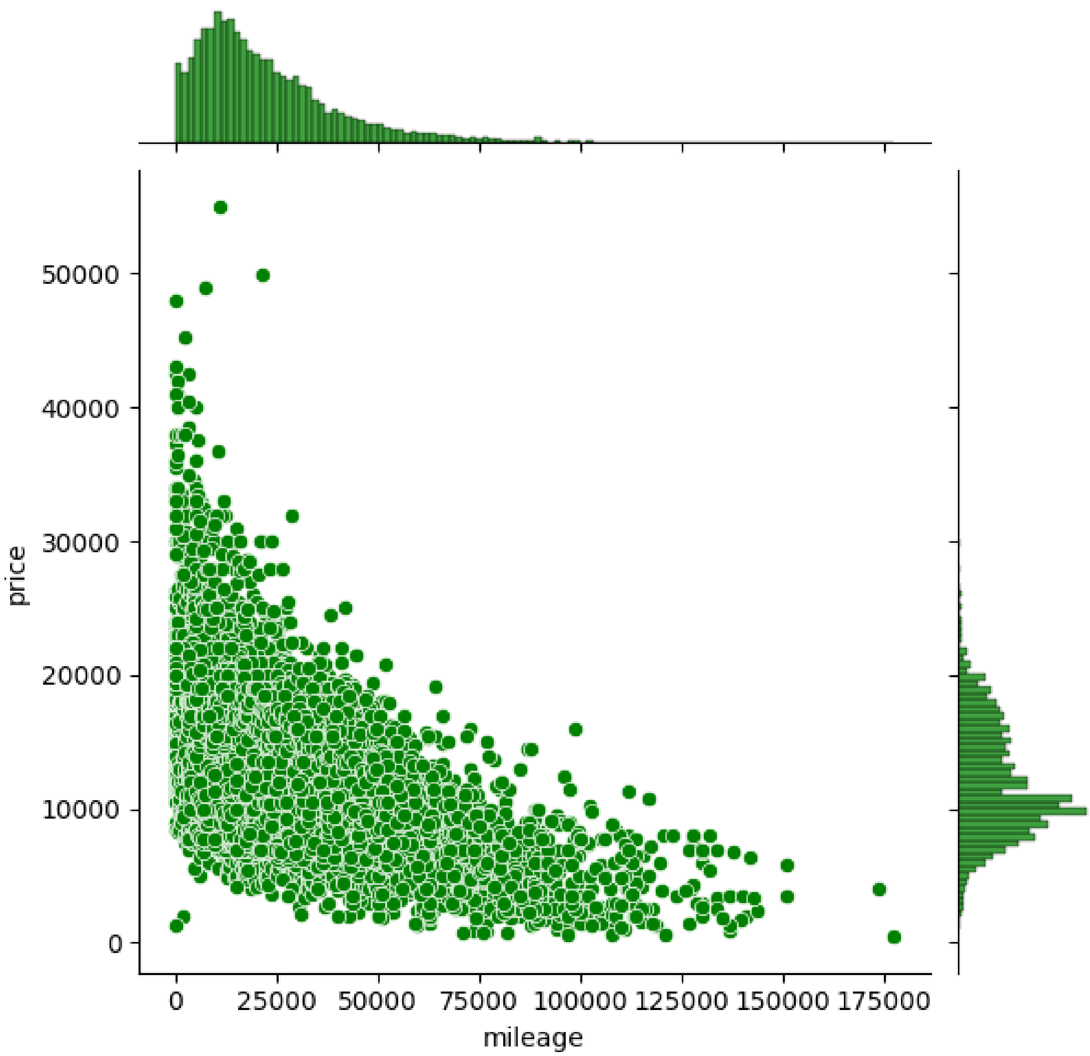
[ ]:

	fuelType	tax
1	Electric	0
3	Other	0
2	Hybrid	2215
0	Diesel	580305
4	Petrol	1435847

[47]:
```
size=[200, 400, 600, 800, 1000]
color=[1, 2, 3, 4, 5]
px.scatter(d1, x='fuelType', y='tax',size=size, color=color)
```

[ ]:
```
sns.jointplot(data=df1, x=df1['mileage'], y=df1['price'],color='g')
```

[ ]: <seaborn.axisgrid.JointGrid at 0x7d735ab0b550>




```
[ ]: sns.heatmap(df1.corr(),annot=True,linewidths=1)
```

```
[ ]: <Axes: >
```

Observations

- More number of cars were purchased in the price range **10000** Dollars to **20000** Dollars
- **Fiesta**, **Focus** and **kugo** are the cars in that comes under highest price range in the Ford cars list.
- The prices were high between the years **2015** to **2018**.
- **Manual** cars are highly priced than automatic and semi auto
- Cars that run on **petrol** gives highest tax followed by diesel and **Electric** does cars doesnot have to pay tax
- As the mileage increases price decreases.

Model Building

[ ]:
```
X=df1.drop(['price'],axis=1)
y=df1['price']
```

[ ]:
```
le=LabelEncoder()
lst=['model','transmission','fuelType']
for i in lst:
  X[i]=le.fit_transform(X[i])
```

[ ]:
```
X
```

[ ]:
```
       model  year  transmission  mileage  fuelType  tax   mpg  engineSize
0         13  2019             1     8389         4  150  35.3         0.0
1          5  2019             1     4000         4  145  40.3         0.0
2         12  2020             1     2000         4  145  43.5         0.0
3         11  2018             1     6000         4  145  43.5         0.0
4         12  2018             1    15000         4  145  43.5         0.0
…        …     …             …        …       …    …     …           …
17807     15  2017             2    26452         4  580  23.5         5.0
17808     15  2020             1       10         4  145  23.7         5.0
17809     15  2020             1       50         4  145  23.9         5.0
17810     15  2020             2     3200         4  145  24.8         5.0
17811     14  2017             0    10162         4  235  38.2         5.0

[17812 rows x 8 columns]
```

[ ]:
```
#scaling using standard scaler
ms=MinMaxScaler()
X_ms=ms.fit_transform(X)
```

[ ]:
```
#Performing train_test_split
X_train,X_test,y_train,y_test=train_test_split(X_ms,y,test_size=0.
↪3,random_state=0)
```

[ ]:
```
#model building using Multiple regression
mlr=LinearRegression()
mlr.fit(X_train,y_train)
y_pred=mlr.predict(X_test)
y_pred
```

[ ]:
```
array([12548.90861712,  9106.24387537,  8051.36725607, …,
       11849.31347771,  8568.45511522, 12324.32982938])
```

Multiple Regression

[ ]:
```
#model validation
import numpy as np
print("mean absolute error:",mean_absolute_error(y_test,y_pred))
print("mean squared error:",mean_squared_error(y_test,y_pred))
print("root mean squared error:",np.sqrt(mean_squared_error(y_test,y_pred)))
print("r2-score:",r2_score(y_test,y_pred))
```

```
mean absolute error: 1758.779360016115
mean squared error: 6758138.433333946
root mean squared error: 2599.64198176094
r2-score: 0.7083660598390757
```

Random Forest Regressor

[ ]:
```
rs=RandomForestRegressor()
rs.fit(X_train,y_train)
y_pred1=rs.predict(X_test)
y_pred1
```

[ ]:
```
array([11854.45,  8205.83,  8469.2 , …, 12626.35,  7742.25, 12058.46])
```

[ ]:
```
print("mean absolute error:",mean_absolute_error(y_test,y_pred1))
print("mean squared error:",mean_squared_error(y_test,y_pred1))
print("root mean squared error:",np.sqrt(mean_squared_error(y_test,y_pred1)))
print("R2 score:",r2_score(y_test,y_pred1))
```

```
mean absolute error: 889.8758965112929
mean squared error: 1778514.9257368413
root mean squared error: 1333.6097351687417
R2 score: 0.9232517474236209
```

Gradient Boosting Regressor

[ ]:
```
gb=GradientBoostingRegressor()
gb.fit(X_train,y_train)
y_pred3=gb.predict(X_test)
y_pred3
```

[ ]:
```
array([12024.84664223,  8764.37171307,  8592.75237629, …,
       12368.23635446,  7234.98658503, 13614.79528895])
```

[ ]:
```
print("mean absolute error:",mean_absolute_error(y_test,y_pred3))
print("mean squared error:",mean_squared_error(y_test,y_pred3))
print("root mean squared error:",np.sqrt(mean_squared_error(y_test,y_pred3)))
print("R2 score:",r2_score(y_test,y_pred3))
```

```
mean absolute error: 977.5392003798046
mean squared error: 2071177.3886393986
root mean squared error: 1439.1585696647185
R2 score: 0.91062248449339
```

XGB Regressor

[ ]:
```
xgb=XGBRegressor()
xgb.fit(X_train,y_train)
y_pred4=xgb.predict(X_test)
y_pred4
```

[ ]:
```
array([11974.356 ,  8401.951 ,  8561.761 , …, 12414.846 ,  7270.1777,
       13043.783 ], dtype=float32)
```

[ ]:
```python
print("mean absolute error:",mean_absolute_error(y_test,y_pred4))
print("mean squared error:",mean_squared_error(y_test,y_pred4))
print("root mean squared error:",np.sqrt(mean_squared_error(y_test,y_pred4)))
print("R2 score:",r2_score(y_test,y_pred4))
```

```
mean absolute error: 833.9477964732462
mean squared error: 1580760.1082320393
root mean squared error: 1257.2828274624765
R2 score: 0.9317854608394733
```

[48]:
```python
#Comparing Actual and predicted Values
Result=pd.DataFrame({'Actual Values':y_test,'linear_Model':
 ↪y_pred,'Random_Forest':y_pred1,'Gradient_boost':y_pred3,'Xg_Boost':y_pred4})
Result
```

[48]:
```
       Actual Values  linear_Model  Random_Forest  Gradient_boost  \
2122           11450  12548.908617       11854.45    12024.846642
12465           8497   9106.243875        8205.83     8764.371713
5989            9450   8051.367256        8469.20     8592.752376
1243           18000  16140.979185       17013.66    17811.490350
16105          16500  16971.815460       16850.58    17151.406430
…                …           …             …             …
12498           8495   7431.343928        8172.31     7541.589404
8676            6498   7428.964502        7074.39     6337.538206
1530           11290  11849.313478       12626.35    12368.236354
12995           7499   8568.455115        7742.25     7234.986585
4848           11997  12324.329829       12058.46    13614.795289

           Xg_Boost
2122   11974.356445
12465   8401.951172
5989    8561.760742
1243   16567.275391
16105  17185.714844
…               …
12498   7758.489746
8676    6383.739258
1530   12414.845703
12995   7270.177734
4848   13043.783203

[5344 rows x 5 columns]
```

Here We can see that XG boost model is more efficient than Linear model.So we predict the price using XG boost regressor model.

[50]:

```
y_new=xgb.predict(ms.transform([[4,2015,1,10,4,150,40,2]]))
print(y_new)
```

```
[18073.955]
```