Oasis Infobyte Internship

Intern Name-Akshay Anandkar

Task 3-CAR PRICE PREDICTION WITH MACHINE LEARNING

Problem Statement- The price of a car depends on a lot of factors like the goodwill of the brand of the car,features of the car, horsepower and the mileage it gives and many more. Car price prediction is one of the major research areas in machine learning. So if you want to learn how to train a car price prediction model then this project is for you.

In [2]:
```
#import all liabraries for project
import pandas as pd
import numpy as np
```

In [3]:
```
#importing dataset
cars=pd.read_csv(r"D:\Data-Science-Internship\car data.csv")
cars.head(10)
```

Out[3]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0
5	vitara brezza	2018	9.25	9.83	2071	Diesel	Dealer	Manual	0
6	ciaz	2015	6.75	8.12	18796	Petrol	Dealer	Manual	0
7	s cross	2015	6.50	8.61	33429	Diesel	Dealer	Manual	0
8	ciaz	2016	8.75	8.89	20273	Diesel	Dealer	Manual	0
9	ciaz	2015	7.45	8.92	42367	Diesel	Dealer	Manual	0

In [4]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Car_Name       301 non-null    object
 1   Year           301 non-null    int64
 2   Selling_Price  301 non-null    float64
 3   Present_Price  301 non-null    float64
 4   Driven_kms     301 non-null    int64
 5   Fuel_Type      301 non-null    object
 6   Selling_type   301 non-null    object
 7   Transmission   301 non-null    object
 8   Owner          301 non-null    int64
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
```

In [5]:
```
cars.isnull().sum()
```

Out[5]:
```
Car_Name         0
Year             0
Selling_Price    0
Present_Price    0
Driven_kms       0
Fuel_Type        0
Selling_type     0
Transmission     0
Owner            0
dtype: int64
```

In [6]:
```
cars.duplicated().sum()
```

Out[6]:
```
2
```

In [7]:
```
#now finding out duplicates
duplicate_rows=cars[cars.duplicated()]
print(duplicate_rows)
```

```
   Car_Name  Year  Selling_Price  Present_Price  Driven_kms Fuel_Type  \
17    ertiga  2016           7.75          10.79       43000    Diesel
93  fortuner  2015          23.00          30.61       40000    Diesel

   Selling_type Transmission  Owner
17       Dealer       Manual      0
93       Dealer    Automatic      0
```

In [8]:
```
no_duplicate_cars=cars.drop_duplicates(keep='first')
no_duplicate_cars
```

Out[8]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0
...	...	...	...	...	...	...	...	...	...
296	city	2016	9.50	11.60	33988	Diesel	Dealer	Manual	0
297	brio	2015	4.00	5.90	60000	Petrol	Dealer	Manual	0
298	city	2009	3.35	11.00	87934	Petrol	Dealer	Manual	0
299	city	2017	11.50	12.50	9000	Diesel	Dealer	Manual	0
300	brio	2016	5.30	5.90	5464	Petrol	Dealer	Manual	0

299 rows × 9 columns

In [9]:
```
#Checking for unique data
cars['Car_Name'].unique()
```

Out[9]:
```
array(['ritz', 'sx4', 'ciaz', 'wagon r', 'swift', 'vitara brezza',
       's cross', 'alto 800', 'ertiga', 'dzire', 'alto k10', 'ignis',
       '800', 'baleno', 'omni', 'fortuner', 'innova', 'corolla altis',
       'etios cross', 'etios g', 'etios liva', 'corolla', 'etios gd',
       'camry', 'land cruiser', 'Royal Enfield Thunder 500',
       'UM Renegade Mojave', 'KTM RC200', 'Bajaj Dominar 400',
       'Royal Enfield Classic 350', 'KTM RC390', 'Hyosung GT250R',
       'Royal Enfield Thunder 350', 'KTM 390 Duke ',
       'Mahindra Mojo XT300', 'Bajaj Pulsar RS200',
       'Royal Enfield Bullet 350', 'Royal Enfield Classic 500',
       'Bajaj Avenger 220', 'Bajaj Avenger 150', 'Honda CB Hornet 160R',
       'Yamaha FZ S V 2.0', 'Yamaha FZ 16', 'TVS Apache RTR 160',
       'Bajaj Pulsar 150', 'Honda CBR 150', 'Hero Extreme',
       'Bajaj Avenger 220 dtsi', 'Bajaj Avenger 150 street',
       'Yamaha FZ  v 2.0', 'Bajaj Pulsar  NS 200', 'Bajaj Pulsar 220 F',
       'TVS Apache RTR 180', 'Hero Passion X pro', 'Bajaj Pulsar NS 200',
       'Yamaha Fazer ', 'Honda Activa 4G', 'TVS Sport ',
       'Honda Dream Yuga ', 'Bajaj Avenger Street 220',
       'Hero Splender iSmart', 'Activa 3g', 'Hero Passion Pro',
       'Honda CB Trigger', 'Yamaha FZ S ', 'Bajaj Pulsar 135 LS',
       'Activa 4g', 'Honda CB Unicorn', 'Hero Honda CBZ extreme',
       'Honda Karizma', 'Honda Activa 125', 'TVS Jupyter',
       'Hero Honda Passion Pro', 'Hero Splender Plus', 'Honda CB Shine',
       'Bajaj Discover 100', 'Suzuki Access 125', 'TVS Wego',
       'Honda CB twister', 'Hero Glamour', 'Hero Super Splendor',
       'Bajaj Discover 125', 'Hero Hunk', 'Hero  Ignitor Disc',
       'Hero  CBZ Xtreme', 'Bajaj  ct 100', 'i20', 'grand i10', 'i10',
       'eon', 'xcent', 'elantra', 'creta', 'verna', 'city', 'brio',
       'amaze', 'jazz'], dtype=object)
```

In [10]:
```
#Start building model for car prediction
#using label encoder
from sklearn.preprocessing import LabelEncoder,StandardScaler
lb=LabelEncoder()
cars['Car_Name']=lb.fit_transform(cars['Car_Name'])
cars['Fuel_Type']=lb.fit_transform(cars['Fuel_Type'])
cars['Selling_type']=lb.fit_transform(cars['Selling_type'])
cars['Transmission']=lb.fit_transform(cars['Transmission'])
```

In [11]:
```
#selecting dependent and independent variables
X=cars.drop('Selling_Price',axis=1)
```

In [12]:
```
y=cars['Selling_Price']
```

In [13]:
```
#now train_test_split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.25,random_state=23)
```

In [14]:
```
#feature scaling
sc=StandardScaler()
X=sc.fit_transform(X)
```

In [15]:
```
#applying model
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X_train,y_train)
```

Out[15]:
```
▾ LinearRegression
LinearRegression()
```

In [16]:
```
y_pred=lr.predict(X_test)
print(y_pred)
```

```
[ 1.54539789  6.23108154  1.24813403  0.0924863   2.40025249 -2.82425458
  1.88783774 11.24223525 -1.03503083  1.59450622  0.10545208 10.08208147
  6.3696335   5.77080975  5.94495352  0.71157071  4.5559797   5.7041711
  4.40939077  1.86120518  2.76746519 20.99288908  0.81975797  4.40270507
  6.83677229  3.66282609 -1.69578598  1.71044003  0.41156628 19.49045397
  8.36510108  4.13202677  6.38490443  7.25655985  4.50911866  1.08340243
  3.91515674  3.12888298  0.76049952  4.46892957  1.97238744  6.87068418
  1.94238368  2.14597549 -0.58720387  1.23194897  2.39916962  3.65924474
  5.04584404  3.32352299  1.51177086  2.10011665 -1.48986907  1.84921632
  0.55544784  7.47714566  7.47772005  8.78736839  8.42618685  5.76864413
  2.07712201  1.61417312  3.66602897 19.86236501  4.78577625 -0.79616853
  0.45071284  7.75769009  2.25493333  7.09184555  3.18353488 17.67927442
  0.51214656  4.86651879  2.41530935  1.48993559]
```

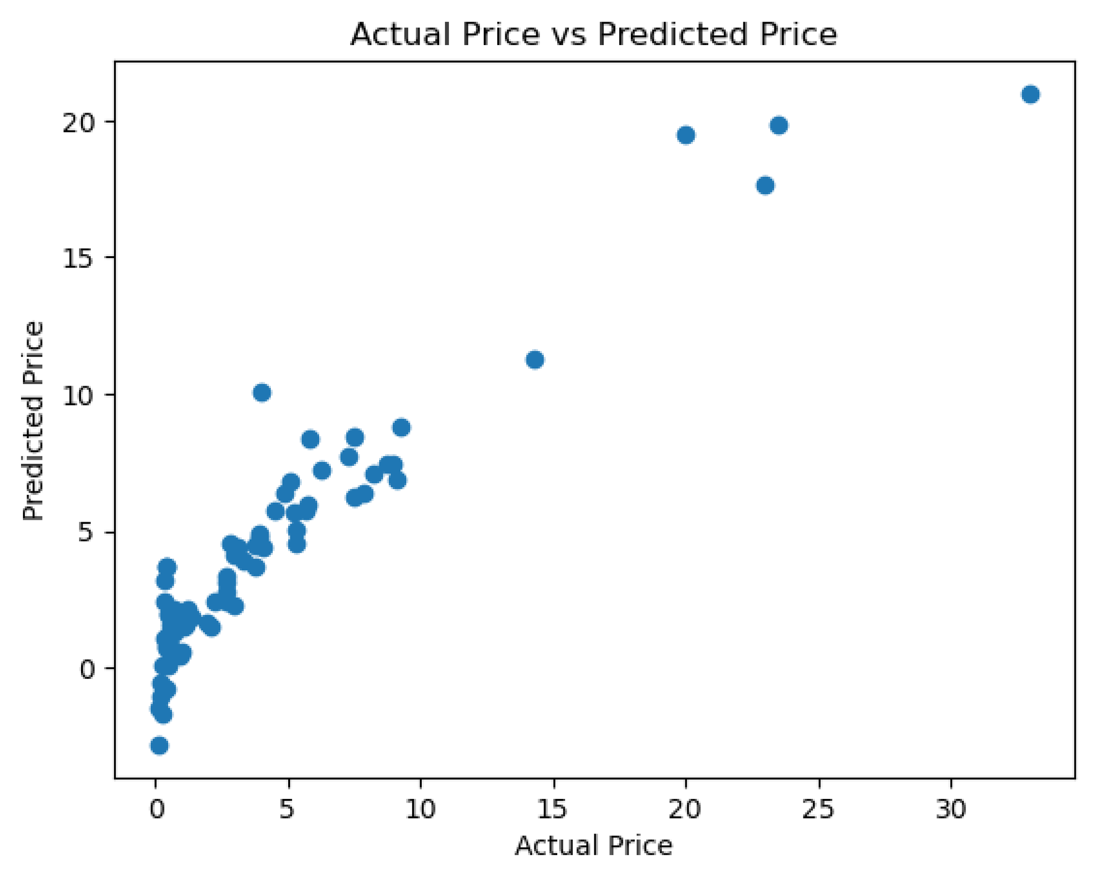
In [17]:
```
print(f'test data accuracy:{lr.score(X_test,y_test)*100:.2f}')
print(f'train data accuracy:{lr.score(X_train,y_train)*100:.2f}')
```

```
test data accuracy:86.68
train data accuracy:87.89
```

In [19]:
```
#calculating r2_score
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
mse=mean_squared_error(y_test,y_pred)
rmse=np.sqrt(mse)
print("root mean squared error{:4f}",format(rmse))
print("r2_score for linear model{:4f}",format(r2_score(y_test,y_pred)))
print("mean squared error{:4f}",format(mean_absolute_error(y_test,y_pred)))
```

```
root mean squared error{:4f} 2.1122763483255875
r2_score for linear model{:4f} 0.8668372274997985
mean squared error{:4f} 1.3027307035656903
```

In [20]:
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.scatter(y_test,y_pred)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual Price vs Predicted Price")
plt.show()
```



In [ ]: