Task-3:

Car Price Prediction With Machine Learning:

Predicting car prices using machine learning involves training a model on historical data with features like brand, model, mileage, age, etc., and then using this model to estimate the price of a car based on its attributes. This predictive model can assist buyers and sellers in making informed decisions about car pricing.

In [1]:
```
#Import required libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn import metrics
```

Data Collection and Processing

In [2]:
```
data=pd.read_csv("car data.csv")
```

In [3]:
```
data.head()
```

Out[3]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Trans
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	

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

Out[4]:
```
(301, 9)
```

In [5]:
```
data. 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 [6]:
```
#Checking gthe numbeer of missing values
data.isnull().sum()
```

Out[6]:
```
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 [7]:
```
data.describe()
```

Out[7]:

	Year	Selling_Price	Present_Price	Driven_kms	Owner
count	301.000000	301.000000	301.000000	301.000000	301.000000
mean	2013.627907	4.661296	7.628472	36947.205980	0.043189
std	2.891554	5.082812	8.642584	38886.883882	0.247915
min	2003.000000	0.100000	0.320000	500.000000	0.000000
25%	2012.000000	0.900000	1.200000	15000.000000	0.000000
50%	2014.000000	3.600000	6.400000	32000.000000	0.000000
75%	2016.000000	6.000000	9.900000	48767.000000	0.000000
max	2018.000000	35.000000	92.600000	500000.000000	3.000000

In [8]:
```
#Checking the distribution of categorical  data

print(data.Fuel_Type.value_counts())
print(data.Selling_type.value_counts())
print(data.Transmission.value_counts())
```

```
Fuel_Type
Petrol    239
Diesel     60
CNG         2
Name: count, dtype: int64
Selling_type
Dealer        195
Individual    106
Name: count, dtype: int64
Transmission
Manual       261
Automatic     40
Name: count, dtype: int64
```

Encoding the Categorical Data

In [9]:
```
#encoding "Fuel_Type" Column
data.replace({'Fuel_Type':{'Petrol':0,'Diesel':1,'CNG':2}},inplace=True

#encoding the "Selling_type" Column
data.replace({'Selling_type':{'Dealer':0,'Individual':1}},inplace=True)

#ncoding the "Transmission" Column
data.replace({'Transmission':{'Manual':0,'Automatic':1}},inplace=True)
```

In [10]:
```
data.head()
```

Out[10]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Trans
0	ritz	2014	3.35	5.59	27000	0	0	
1	sx4	2013	4.75	9.54	43000	1	0	
2	ciaz	2017	7.25	9.85	6900	0	0	
3	wagon r	2011	2.85	4.15	5200	0	0	
4	swift	2014	4.60	6.87	42450	1	0	

In [11]:
```
data.tail()
```

Out[11]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Tra
296	city	2016	9.50	11.6	33988	1	0	
297	brio	2015	4.00	5.9	60000	0	0	
298	city	2009	3.35	11.0	87934	0	0	
299	city	2017	11.50	12.5	9000	1	0	
300	brio	2016	5.30	5.9	5464	0	0	

Splitting the data and Target

In [13]:

```
#Splitting the data and Target

x=data.drop(["Car_Name","Selling_Price"],axis=1)#here axis-1 because ic
                                               #Otherwise axis=0 when
```

In [14]:

```
y=data["Selling_Price"]
```

In [15]:

```
print(x)
```

```
     Year  Present_Price  Driven_kms  Fuel_Type  Selling_type  Transmissio
n  \
0    2014           5.59       27000          0             0
0
1    2013           9.54       43000          1             0
0
2    2017           9.85        6900          0             0
0
3    2011           4.15        5200          0             0
0
4    2014           6.87       42450          1             0
0
..    ...            ...         ...        ...           ...
...
296  2016          11.60       33988          1             0
0
297  2015           5.90       60000          0             0
0
298  2009          11.00       87934          0             0
0
299  2017          12.50        9000          1             0
0
300  2016           5.90        5464          0             0
0

     Owner
0        0
1        0
2        0
3        0
4        0
..     ...
296      0
297      0
298      0
299      0
300      0

[301 rows x 7 columns]
```

In [16]:
```
print(y)
```

```
0       3.35
1       4.75
2       7.25
3       2.85
4       4.60
       ...
296     9.50
297     4.00
298     3.35
299    11.50
300     5.30
Name: Selling_Price, Length: 301, dtype: float64
```

Splitting Training and Test data

In [17]:
```
X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size=0.2, rando
```

Model training

1.Linear Regression

In [18]:
```
#Loading the linear regression model

lin_reg_model=LinearRegression()
```

In [19]:
```
lin_reg_model.fit(X_train,Y_train)
```

Out[19]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model evaluation

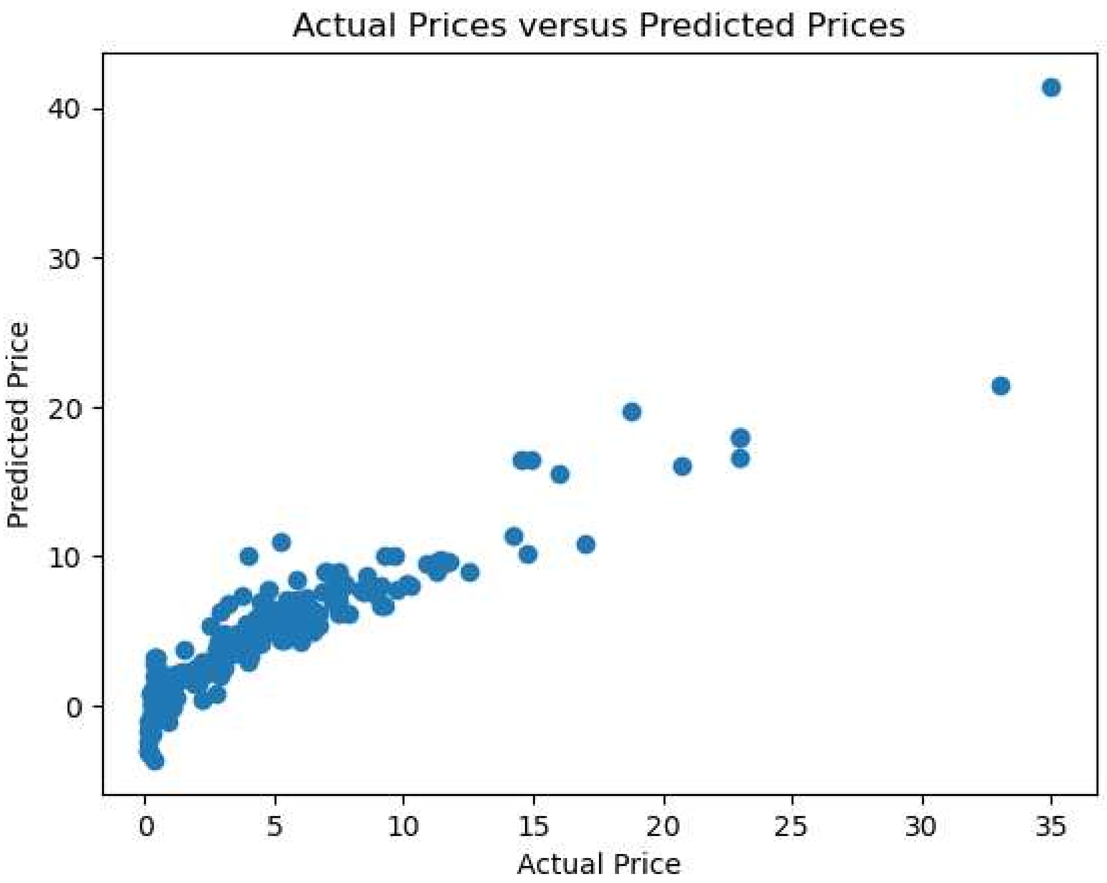
In [20]:
```
#prediction on traning data

training_data_prediction=lin_reg_model.predict(X_train)
```

In [21]:
```
# R-squared error
error_score=metrics.r2_score(Y_train,training_data_prediction)
print("R squared Eroor:",error_score)
```

```
R squared Eroor: 0.8680830940612677
```

Visualize the actual prices and Predicted prices:

In [22]:
```
#Visualize the prices and predicted prices
plt.scatter(Y_train,training_data_prediction)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual Prices versus Predicted Prices")
plt.show()
```

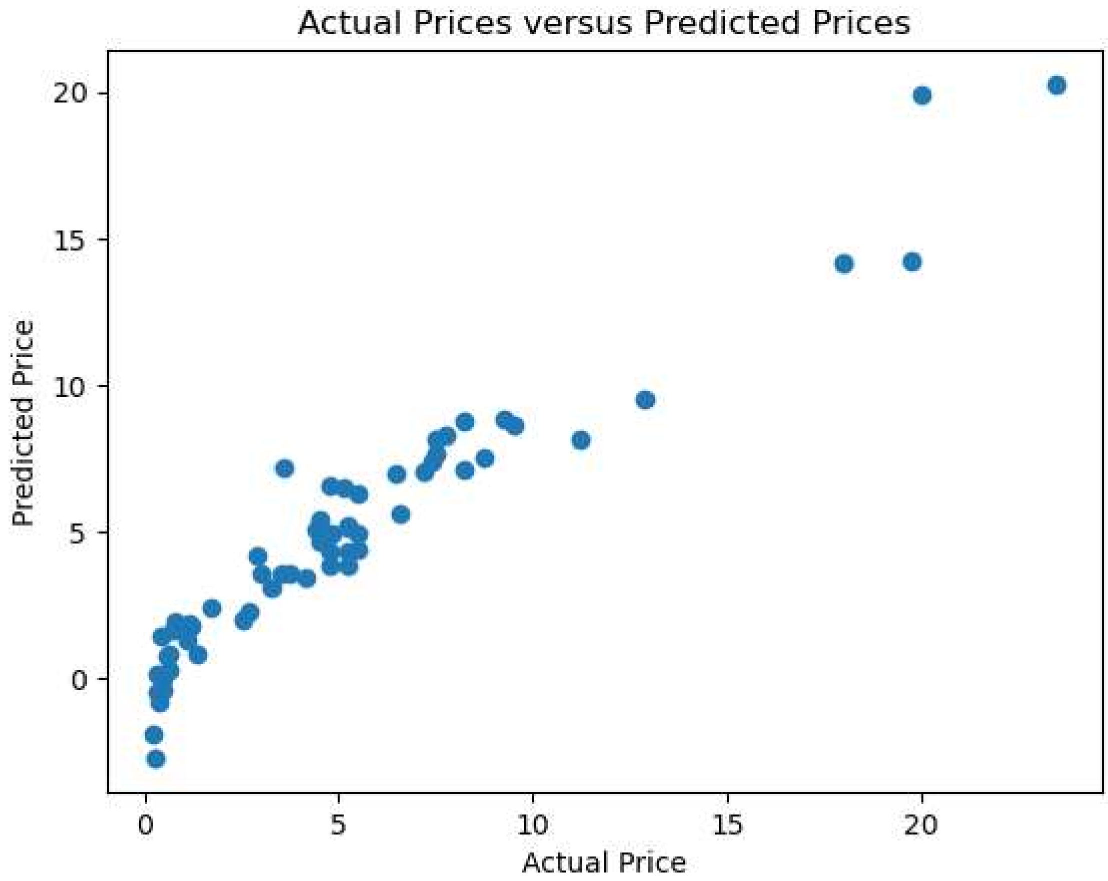


In [23]:
```
#prediction on test data

test_data_prediction=lin_reg_model.predict(X_test)
```

In [24]:
```
# R-squared error
error_score=metrics.r2_score(Y_test,test_data_prediction)
print("R squared Eroor:",error_score)
```

```
R squared Eroor: 0.9133788577646775
```

In [25]:
```
#Visualize the prices and predicted prices
plt.scatter(Y_test,test_data_prediction)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual Prices versus Predicted Prices")
plt.show()
```



2.Lasso Regression model:

In [26]:
```
#Loading the Lasso Regression model

lass_reg_model=Lasso()
```

In [27]:
```
lass_reg_model.fit(X_train,Y_train)
```

Out[27]: Lasso()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [28]:
```
#prediction on traning data

training_data_prediction=lass_reg_model.predict(X_train)
```

In [29]:

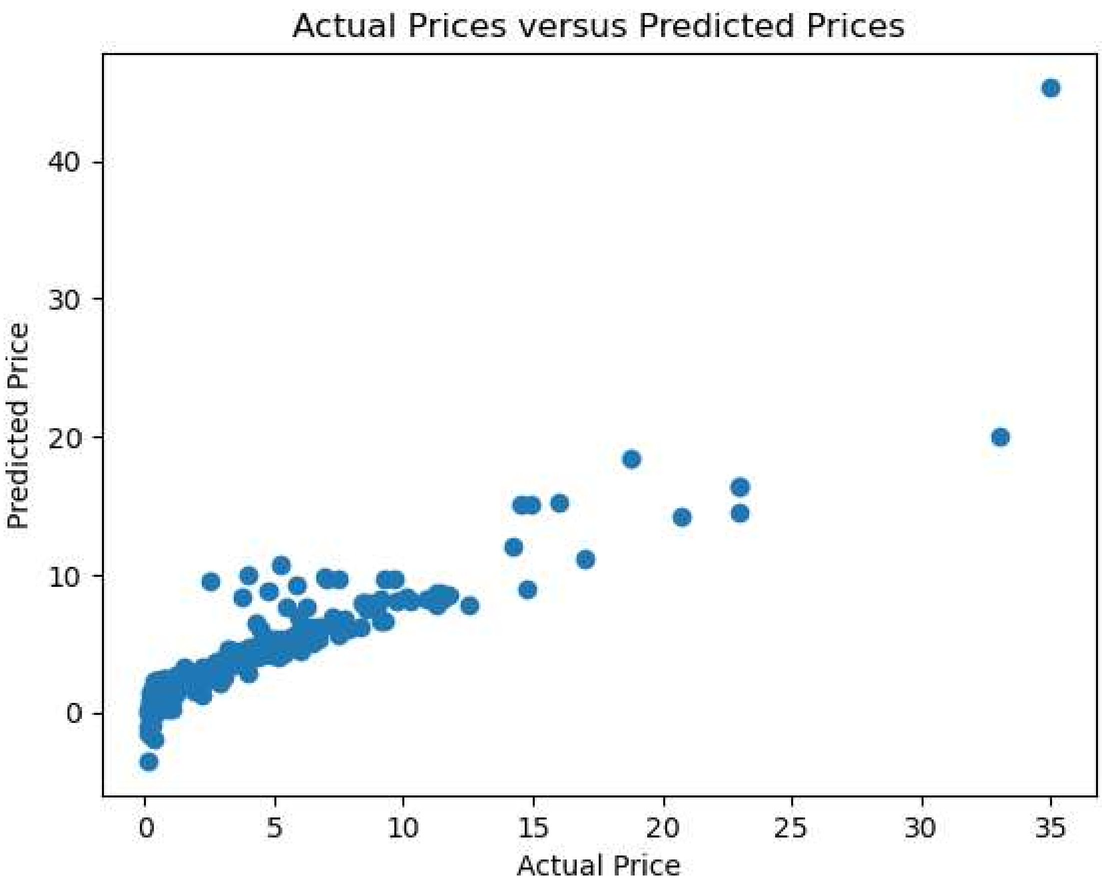
```
# R-squared error
error_score=metrics.r2_score(Y_train,training_data_prediction)
print("R squared Eroor:",error_score)
```

```
R squared Eroor: 0.8315232865153553
```

Visualize the actual prices and Predicted prices:

In [30]:

```
#Visualize the prices and predicted prices
plt.scatter(Y_train,training_data_prediction)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual Prices versus Predicted Prices")
plt.show()
```

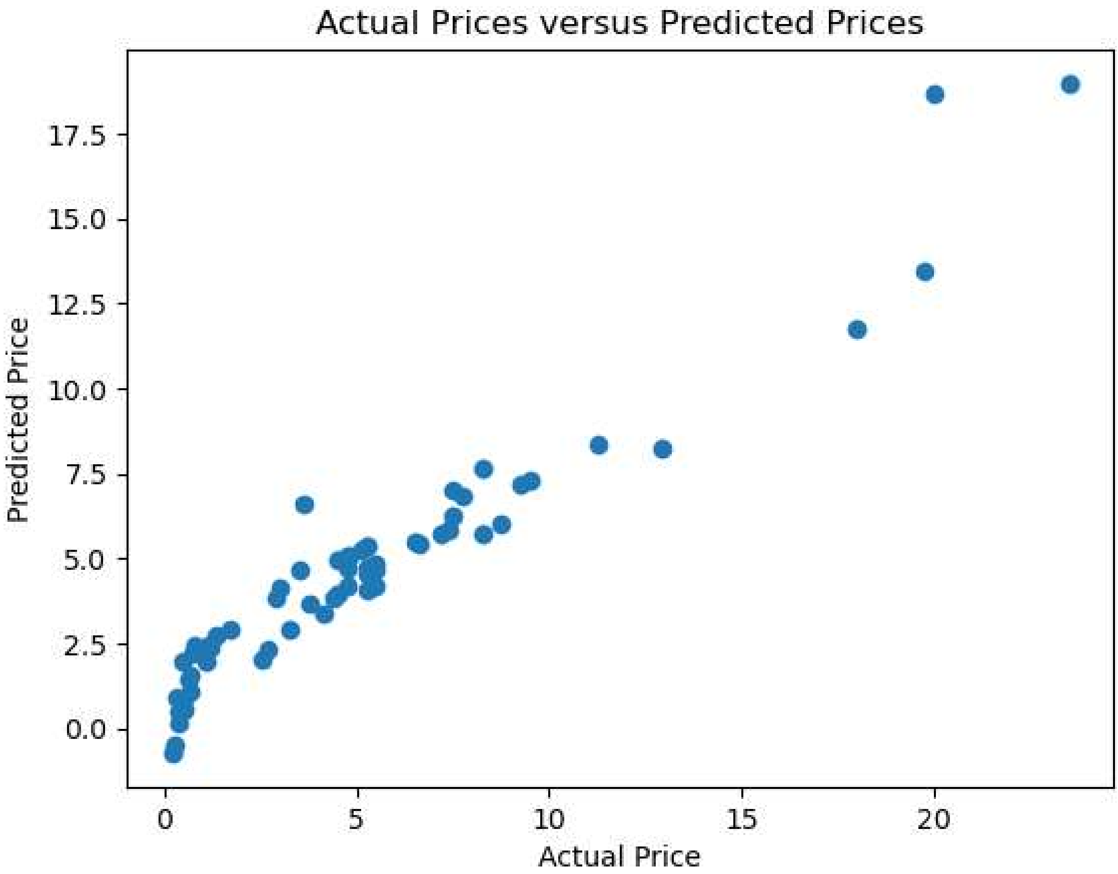


In [31]:

```
#prediction on test data

test_data_prediction=lass_reg_model.predict(X_test)
```

In [32]:

```
#Visualize the prices and predicted prices
plt.scatter(Y_test,test_data_prediction)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual Prices versus Predicted Prices")
plt.show()
```



In [ ]:

```

```