Importing the Dependencies

In [1]:
```
import numpy as np  # used for array
import pandas as pd # used for data processing (data loading, data manipulation, etc)
import seaborn as sns # used for data visualization
import matplotlib.pyplot as plt # used for plotting graphs
from sklearn.model_selection import train_test_split  # used for splitting the data into training data and testi
from sklearn.linear_model import LinearRegression # import LinearRegression model
from sklearn.linear_model import Lasso
from sklearn import metrics # r2_error
from sklearn.metrics import mean_absolute_error # mean absolute error
```

Data Collection and Processing

In [2]:
```
# loading the data from csv file to pandas dataframe
car_dataset = pd.read_csv('car data.csv')
```

In [3]:
```
# .head() displays the first 5 rows, and first n columns

# python indexes the column and row number starting from 0.car_dataset.head()
car_dataset.head
```

Out[3]:
```
<bound method NDFrame.head of     Car_Name  Year  Selling_Price  Present_Price  Kms_Driven Fuel_Type  \
0       ritz  2014           3.35           5.59       27000    Petrol
1        sx4  2013           4.75           9.54       43000    Diesel
2       ciaz  2017           7.25           9.85        6900    Petrol
3    wagon r  2011           2.85           4.15        5200    Petrol
4      swift  2014           4.60           6.87       42450    Diesel
..       ...   ...            ...            ...         ...       ...
296     city  2016           9.50          11.60       33988    Diesel
297     brio  2015           4.00           5.90       60000    Petrol
298     city  2009           3.35          11.00       87934    Petrol
299     city  2017          11.50          12.50        9000    Diesel
300     brio  2016           5.30           5.90        5464    Petrol

    Seller_Type Transmission  Owner
0        Dealer       Manual      0
1        Dealer       Manual      0
2        Dealer       Manual      0
3        Dealer       Manual      0
4        Dealer       Manual      0
..          ...          ...    ...
296      Dealer       Manual      0
297      Dealer       Manual      0
298      Dealer       Manual      0
299      Dealer       Manual      0
300      Dealer       Manual      0

[301 rows x 9 columns]>
```

In [4]:
```
# checking the number of rows and columns

car_dataset.shape
```

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

In [5]:
```
# getting some information about the dataset

car_dataset.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   Kms_Driven     301 non-null    int64
 5   Fuel_Type      301 non-null    object
 6   Seller_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 for missing values per column

car_dataset.isnull().sum()
```

Out[6]:
```
Car_Name         0
Year             0
Selling_Price    0
Present_Price    0
Kms_Driven       0
Fuel_Type        0
Seller_Type      0
Transmission     0
Owner            0
dtype: int64
```

In [7]:
```
# checking the distribution of categorical data
print(car_dataset.Fuel_Type.value_counts())
print(car_dataset.Seller_Type.value_counts())
print(car_dataset.Transmission.value_counts())
```

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

Encoding the Categorical Data

In [8]:
```
# encoding "Fuel_Type" Column
car_dataset.replace({'Fuel_Type':{'Petrol':0,'Diesel':1,'CNG':2}},inplace=True)

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

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

In [9]:
```
car_dataset.head()

# checking the replaced values
```

Out[9]:

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

Splitting the data and Target

In [10]:
```
X = car_dataset.drop(['Car_Name','Selling_Price'],axis=1) # data values
Y = car_dataset['Selling_Price'] # target column
```

In [11]:
```
print(X)
```

```
     Year  Present_Price  Kms_Driven  Fuel_Type  Seller_Type  Transmission  \
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 [12]:
```
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 [13]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.1, random_state=2)
# split the into 4 variables, X_train, X_test, Y_train, Y_test.

# test_size is 10%, therefore 90% of the data and labels will be in X_train, Y_train, and remaining 10% will be i

# stratify = Y references that we split the data according to Y
```

Model Training

1. Linear Regression

In [14]:
```
# loading the linear regression model
lin_reg_model = LinearRegression()
```

In [15]:
```
lin_reg_model.fit(X_train,Y_train)

# training the model with the split data - X_train, Y_train
```

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

Model Evaluation

In [16]:
```
# prediction on Training data
training_data_prediction = lin_reg_model.predict(X_train)
```

In [17]:
```
# R squared Error
```

```
error_score = metrics.r2_score(Y_train, training_data_prediction)
print("R squared Error : ", error_score)
```

```
R squared Error :  0.8799451660493711
```

In [18]:

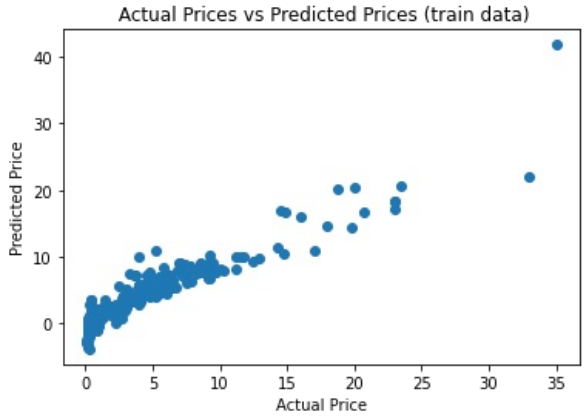
```
# Mean absolute error
m_a_e = mean_absolute_error(Y_train, training_data_prediction)
print('Mean absolute error: ', m_a_e)
```

```
Mean absolute error:  1.216617409391433
```

Visualize the actual prices and Predicted prices

In [19]:

```
plt.scatter(Y_train, training_data_prediction)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title(" Actual Prices vs Predicted Prices (train data)")
plt.show()
```



In [20]:

```
# prediction on test data
test_data_prediction = lin_reg_model.predict(X_test)
```

In [21]:

```
# R squared Error
error_score = metrics.r2_score(Y_test, test_data_prediction)
print("R squared Error : ", error_score)
```

```
R squared Error :  0.8365766715026457
```

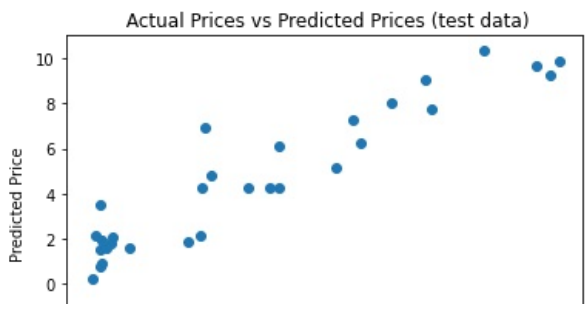
In [22]:

```
# Mean absolute error
m_a_e = mean_absolute_error(Y_test, test_data_prediction)
print('Mean absolute error: ', m_a_e)
```

```
Mean absolute error:  1.1516382156616716
```

In [23]:

```
plt.scatter(Y_test, test_data_prediction)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title(" Actual Prices vs Predicted Prices (test data)")
plt.show()
```



1. Lasso Regression

In [24]:
```
# loading the lasso regression model
lass_reg_model = Lasso()
```

In [25]:
```
lass_reg_model.fit(X_train,Y_train)
```

Out[25]:
```
Lasso()
```

Model Evaluation

In [26]:
```
# prediction on Training data
training_data_prediction = lass_reg_model.predict(X_train)
```

In [27]:
```
# R squared Error
error_score = metrics.r2_score(Y_train, training_data_prediction)
print("R squared Error : ", error_score)
```

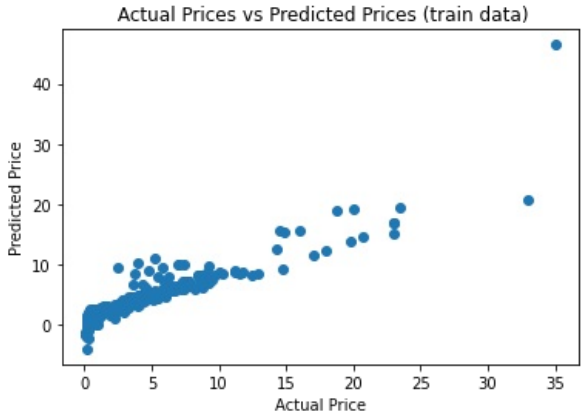
```
R squared Error :  0.8427856123435794
```

In [28]:
```
# mean absolute error
m_a_e_lasso = mean_absolute_error(Y_train, training_data_prediction)
print('Mean absolute error :',m_a_e_lasso)
```

```
Mean absolute error : 1.2863097696916528
```

Visualize the actual prices and Predicted prices

In [29]:
```
plt.scatter(Y_train, training_data_prediction)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title(" Actual Prices vs Predicted Prices (train data)")
plt.show()
```



In [30]:
```
# prediction on Training data
test_data_prediction = lass_reg_model.predict(X_test)
```

In [31]:
```
# R squared Error
error_score = metrics.r2_score(Y_test, test_data_prediction)
print("R squared Error : ", error_score)
```

```
R squared Error :  0.8709167941173195
```

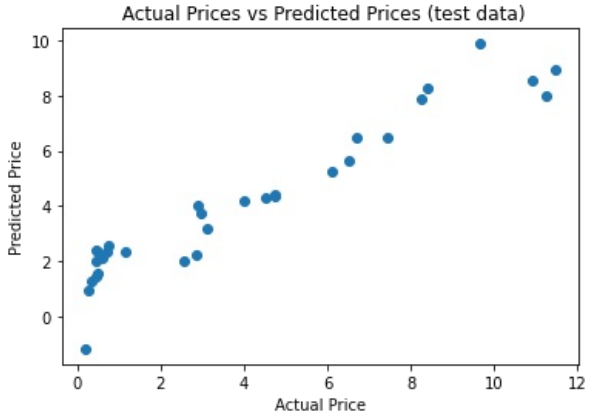
In [32]:

```
# mean absolute error
m_a_e_lasso = mean_absolute_error(Y_test, test_data_prediction)
print('Mean absolute error :',m_a_e_lasso)
```

```
Mean absolute error : 1.0507413774170433
```

In [33]:

```
plt.scatter(Y_test, test_data_prediction)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title(" Actual Prices vs Predicted Prices (test data)")
plt.show()
```



In [ ]: