Import libraries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Read data

In [4]:
```
df = pd.read_csv("CarPrice_Assignment.csv")
print(df)
```

```
     car_ID  symboling                   CarName fueltype aspiration  \
0         1          1         alfa-romeo giulia      gas        std
1         2          1        alfa-romeo stelvio      gas        std
2         3          1  alfa-romeo Quadrifoglio      gas        std
3         4          2               audi 100 ls      gas        std
4         5          2                audi 100ls      gas        std
..      ...        ...                       ...      ...        ...
200     201         22           volvo 145e (sw)      gas        std
201     202         22               volvo 144ea      gas      turbo
202     203         22               volvo 244dl      gas        std
203     204         22                 volvo 246   diesel      turbo
204     205         22               volvo 264gl      gas      turbo

    doornumber      carbody drivewheel enginelocation  wheelbase  ...  \
0          two  convertible        rwd          front       88.6  ...
1          two  convertible        rwd          front       88.6  ...
2          two    hatchback        rwd          front       94.5  ...
3         four        sedan        fwd          front       99.8  ...
4         four        sedan        4wd          front       99.4  ...
..         ...          ...        ...            ...        ...  ...
200       four        sedan        rwd          front      109.1  ...
201       four        sedan        rwd          front      109.1  ...
202       four        sedan        rwd          front      109.1  ...
203       four        sedan        rwd          front      109.1  ...
204       four        sedan        rwd          front      109.1  ...

     enginesize  fuelsystem  boreratio  stroke compressionratio horsepower  \
0           130        mpfi       3.47    2.68              9.0        111
1           130        mpfi       3.47    2.68              9.0        111
2           152        mpfi       2.68    3.47              9.0        154
3           109        mpfi       3.19    3.40             10.0        102
4           136        mpfi       3.19    3.40              8.0        115
..          ...         ...        ...     ...              ...        ...
200         141        mpfi       3.78    3.15              9.5        114
201         141        mpfi       3.78    3.15              8.7        160
202         173        mpfi       3.58    2.87              8.8        134
203         145         idi       3.01    3.40             23.0        106
204         141        mpfi       3.78    3.15              9.5        114

     peakrpm citympg  highwaympg    price
0       5000      21          27  13495.0
1       5000      21          27  16500.0
2       5000      19          26  16500.0
3       5500      24          30  13950.0
4       5500      18          22  17450.0
..       ...     ...         ...      ...
200     5400      23          28  16845.0
201     5300      19          25  19045.0
202     5500      18          23  21485.0
203     4800      26          27  22470.0
204     5400      19          25  22625.0

[205 rows x 26 columns]
```

In [6]:
```
df.shape
```

Out[6]:
```
(205, 26)
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

In [8]:
```
df.head()
```

Out[8]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsy
0	1	1	alfa-romeo giulia	gas	std	two	convertible	rwd	front	88.6	...	130	
1	2	1	alfa-romeo stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	
2	3	1	alfa-romeo Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	

5 rows × 26 columns

In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   car_ID            205 non-null    int64  
 1   symboling         205 non-null    int64  
 2   CarName           205 non-null    object 
 3   fueltype          205 non-null    object 
 4   aspiration        205 non-null    object 
 5   doornumber        205 non-null    object 
 6   carbody           205 non-null    object 
 7   drivewheel        205 non-null    object 
 8   enginelocation    205 non-null    object 
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64  
 14  enginetype        205 non-null    object 
 15  cylindernumber    205 non-null    object 
 16  enginesize        205 non-null    int64  
 17  fuelsystem        205 non-null    object 
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64  
 22  peakrpm           205 non-null    int64  
 23  citympg           205 non-null    int64  
 24  highwaympg        205 non-null    int64  
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

In [10]:

```
df.dtypes
```

Out[10]:

	0
car_ID	int64
symboling	int64
CarName	object
fueltype	object
aspiration	object
doornumber	object
carbody	object
drivewheel	object
enginelocation	object
wheelbase	float64
carlength	float64
carwidth	float64
carheight	float64
curbweight	int64
enginetype	object
cylindernumber	object
enginesize	int64
fuelsystem	object
boreratio	float64
stroke	float64
compressionratio	float64
horsepower	int64
peakrpm	int64
citympg	int64
highwaympg	int64
price	float64

dtype: object

In [11]:
```
df.describe()
```

Out[11]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressio
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.0
mean	103.000000	13.195122	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329756	3.255415	10.1
std	59.322565	6.274831	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313597	3.9
min	1.000000	1.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.0
25%	52.000000	9.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.6
50%	103.000000	13.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.0
75%	154.000000	20.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410000	9.4
max	205.000000	22.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.0

Data transformation

Check missing values

In [12]:
```
df.isnull().sum()
```

Out[12]:

	0
car_ID	0
symboling	0
CarName	0
fueltype	0
aspiration	0
doornumber	0
carbody	0
drivewheel	0
enginelocation	0
wheelbase	0
carlength	0
carwidth	0
carheight	0
curbweight	0
enginetype	0
cylindernumber	0
enginesize	0
fuelsystem	0
boreratio	0
stroke	0
compressionratio	0
horsepower	0
peakrpm	0
citympg	0
highwaympg	0
price	0

dtype: int64

In [13]:

```
df_cleaned = df.dropna()


df_cleaned_columns = df.dropna(axis=1)
```

Data cleaning

Calculate z score

In [14]:

```
from scipy import stats
z_scores = stats.zscore(df.select_dtypes(include=['float64', 'int64']))
df_outliers = df[(z_scores > 3).any(axis=1) | (z_scores < -3).any(axis=1)]
```

Remove Outliers

In [15]:

```
df_no_outliers = df[(z_scores < 3).all(axis=1) & (z_scores > -3).all(axis=1)]
df_no_outliers
```

Out[15]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuel
0	1	1	alfa-romeo giulia	gas	std	two	convertible	rwd	front	88.6	...	130	
1	2	1	alfa-romeo stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	
2	3	1	alfa-romeo Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	
...	...	...	...	...	...	...	...	...	...	...	...	...	
199	200	22	volvo diesel	gas	turbo	four	wagon	rwd	front	104.3	...	130	
200	201	22	volvo 145e (sw)	gas	std	four	sedan	rwd	front	109.1	...	141	
201	202	22	volvo 144ea	gas	turbo	four	sedan	rwd	front	109.1	...	141	
202	203	22	volvo 244dl	gas	std	four	sedan	rwd	front	109.1	...	173	
204	205	22	volvo 264gl	gas	turbo	four	sedan	rwd	front	109.1	...	141	

181 rows × 26 columns

Data Visualization

In [16]:

```
#determining the average price of each such car brands and storing them in a numpy array
no_of_brands=df['symboling'].max()
avg_prices=np.zeros(no_of_brands)
for i in range(1,no_of_brands+1):
    df_temp=df[df['symboling']==i]
    avg_prices[i-1]=df_temp['price'].mean()
avg_prices=sorted(avg_prices, reverse=True)
avg_prices
```

Out[16]:

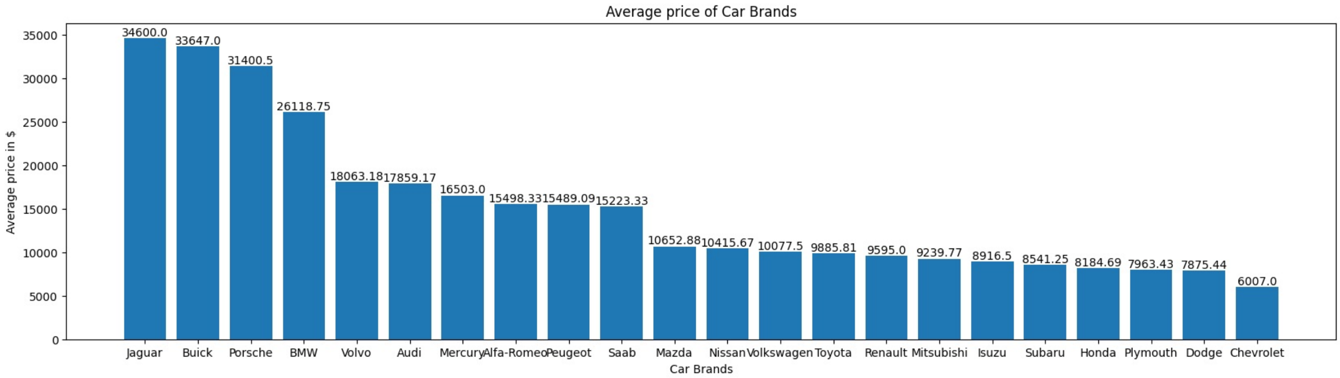
```
[34600.0,
 33647.0,
 31400.5,
 26118.75,
 18063.18181818182,
 17859.166714285715,
 16503.0,
 15498.333333333334,
 15489.09090909091,
 15223.333333333334,
 10652.882352941177,
 10415.666666666666,
 10077.5,
 9885.8125,
 9595.0,
 9239.76923076923,
 8916.5,
 8541.25,
 8184.692307692308,
 7963.428571428572,
 7875.444444444444,
 6007.0]
```

In [17]:

```
import matplotlib.pyplot as plt
labels=['Jaguar','Buick','Porsche','BMW','Volvo','Audi','Mercury','Alfa-Romeo','Peugeot','Saab','Mazda','Nissan
plt.figure(figsize=(20,5))
bars=plt.bar(labels,avg_prices)
bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.5, round(yval, 2),
             ha='center', va='bottom')
plt.xlabel('Car Brands')
plt.ylabel('Average price in $')
plt.title('Average price of Car Brands')
```

Out[17]:

```
Text(0.5, 1.0, 'Average price of Car Brands')
```



Grouped bar chart

In [18]:
```
avg_carheight=np.zeros(no_of_brands)
for i in range(1,no_of_brands+1):
    df_temp=df[df['symboling']==i]
    avg_carheight[i-1]=df_temp['carheight'].mean()
avg_carheight
```

Out[18]:
```
array([50.        , 54.42857143, 54.825     , 52.4       , 51.64444444,
       53.23846154, 52.225     , 51.13333333, 53.35882353, 55.725     ,
       54.8       , 50.69230769, 53.63333333, 57.18181818, 51.97142857,
       51.1       , 52.85      , 56.1       , 53.75      , 53.721875  ,
       55.18333333, 56.23636364])
```

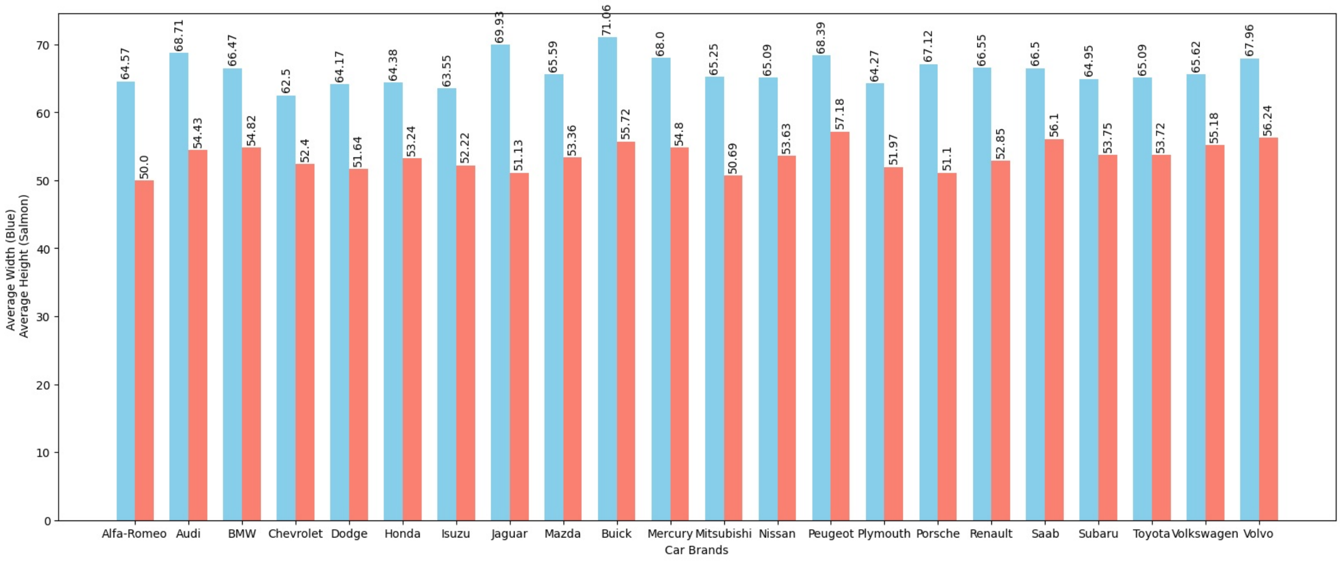
In [19]:
```
avg_carwidth=np.zeros(no_of_brands)
for i in range(1,no_of_brands+1):
    df_temp=df[df['symboling']==i]
    avg_carwidth[i-1]=df_temp['carwidth'].mean()
avg_carwidth
```

Out[19]:
```
array([64.56666667, 68.71428571, 66.475     , 62.5       , 64.16666667,
       64.38461538, 63.55      , 69.93333333, 65.58823529, 71.0625    ,
       68.        , 65.25384615, 65.08888889, 68.39090909, 64.27142857,
       67.12      , 66.55      , 66.5       , 64.95      , 65.090625  ,
       65.61666667, 67.96363636])
```

In [20]:
```
labels=['Alfa-Romeo','Audi','BMW','Chevrolet','Dodge','Honda','Isuzu','Jaguar','Mazda','Buick','Mercury','Mitsu
```

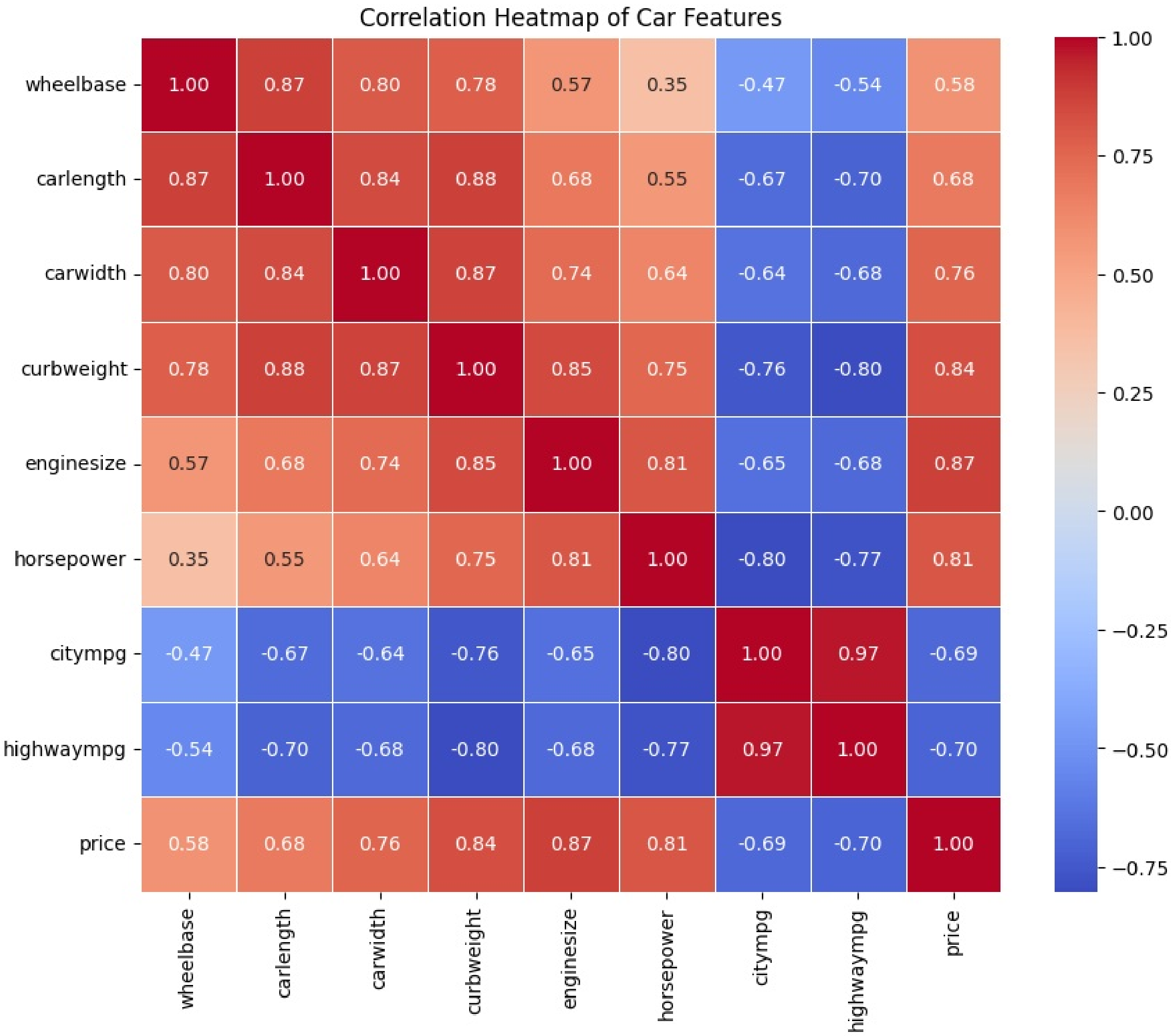
In [21]:
```
# Positions for the bars
x = np.arange(no_of_brands)  # The label locations
width = 0.35  # Width of the bars

# Create the figure and axis
plt.figure(figsize=(20,8))

# Plot the first set of bars
plt.bar(x - width/2, avg_carwidth, width, color='skyblue')

# Plot the second set of bars, shifting them to the right
plt.bar(x + width/2, avg_carheight, width, color='salmon')

plt.xticks(x, labels)  # Replace x-axis ticks with category names

bars_1=plt.bar(x - width/2, avg_carwidth, width, color='skyblue')
for bar in bars_1:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.5, round(yval, 2),
             ha='center', va='bottom', rotation=90)
bars_2=plt.bar(x + width/2, avg_carheight, width, color='salmon')
for bar in bars_2:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.5, round(yval, 2),
             ha='center', va='bottom', rotation=90)


plt.xlabel('Car Brands')
plt.ylabel('Average Width (Blue) \n Average Height (Salmon)')
```

Out[21]:
```
Text(0, 0.5, 'Average Width (Blue) \n Average Height (Salmon)')
```



Correlation Heatmap

In [ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Select relevant numerical columns for the correlation heatmap
correlation_columns = ['wheelbase', 'carlength', 'carwidth', 'curbweight', 'enginesize', 'horsepower', 'citympg
correlation_matrix = df[correlation_columns].corr()

# Create the heatmap
plt.figure(figsize=(10, 8))
heatmap = sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", linewidths=0.5, fmt=".2f")
plt.title("Correlation Heatmap of Car Features")
plt.show()
```



Multiple Regression

```
In [ ]: import statsmodels.api as sm

# Define the independent variables and the dependent variable
X = df[['wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight',
        'enginesize', 'boreratio', 'stroke', 'compressionratio', 'horsepower',
        'peakrpm', 'citympg', 'highwaympg']]
y = df['price']

# Add a constant to the independent variables (for the intercept)
X = sm.add_constant(X)

# Fit the regression model
model = sm.OLS(y, X).fit()

# Get the summary of the regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.851
Model:                            OLS   Adj. R-squared:                  0.841
Method:                 Least Squares   F-statistic:                     83.78
Date:                Tue, 29 Oct 2024   Prob (F-statistic):           1.68e-71
Time:                        12:32:56   Log-Likelihood:                -1937.5
No. Observations:                 205   AIC:                             3903.
Df Residuals:                     191   BIC:                             3949.
Df Model:                          13
Covariance Type:            nonrobust
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
const             -4.75e+04   1.53e+04     -3.111      0.002   -7.76e+04   -1.74e+04
wheelbase          122.6169    100.465      1.220      0.224     -75.546     320.780
carlength          -94.6752     55.557     -1.704      0.090    -204.259      14.909
carwidth           505.5716    246.013      2.055      0.041      20.319     990.824
carheight          163.1801    135.721      1.202      0.231    -104.524     430.884
curbweight           1.8846      1.737      1.085      0.279      -1.542       5.312
enginesize         117.3461     13.837      8.481      0.000      90.054     144.638
boreratio        -1002.5654   1195.798     -0.838      0.403   -3361.231    1356.100
stroke           -3034.6060    778.604     -3.897      0.000   -4570.373   -1498.839
compressionratio   298.1369     82.914      3.596      0.000     134.592     461.682
horsepower          30.8086     16.216      1.900      0.059      -1.177      62.795
peakrpm              2.3751      0.671      3.540      0.001       1.052       3.698
citympg           -320.3545    177.769     -1.802      0.073    -670.998      30.289
highwaympg         202.8221    159.760      1.270      0.206    -112.299     517.943
==============================================================================
Omnibus:                       24.541   Durbin-Watson:                   0.930
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               81.326
Skew:                           0.383   Prob(JB):                     2.19e-18
Kurtosis:                       5.989   Cond. No.                     3.94e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.94e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [ ]:

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

# Define the independent variables and the dependent variable
X = df[['wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight',
        'enginesize', 'boreratio', 'stroke', 'compressionratio', 'horsepower',
        'peakrpm', 'citympg', 'highwaympg']]
y = df['price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a StandardScaler instance and scale the features
scaler = StandardScaler()

# Fit the scaler on the training data and transform both training and testing sets
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)
print(f"R-squared: {r2}")
```

```
R-squared: 0.8146957315667049
```

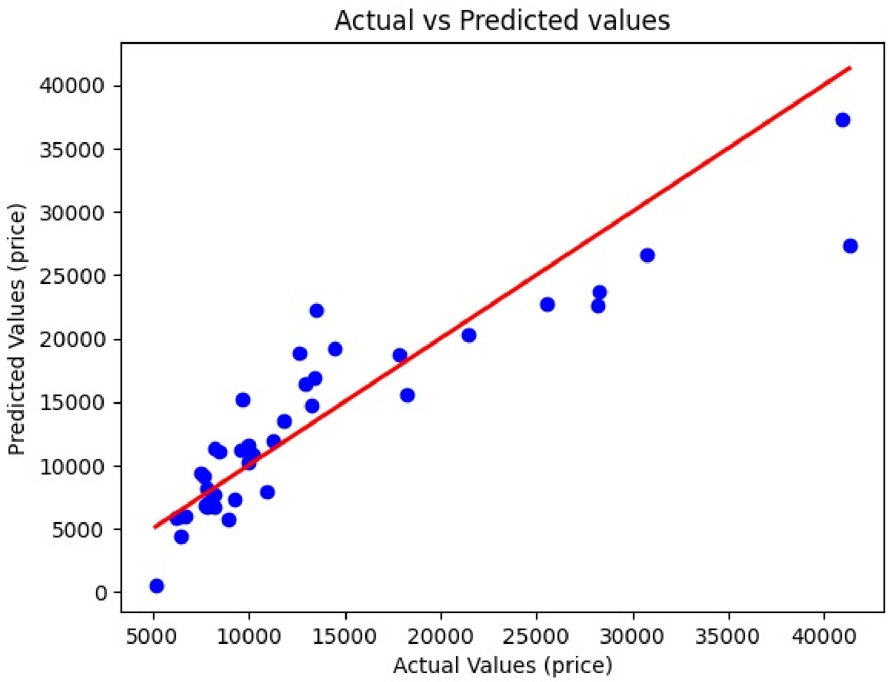
Find the R^2 value beween actual & predicted price

In [ ]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Load data (assuming df is already loaded)
X = df[['enginesize', 'curbweight', 'horsepower']]  # Independent variables
y = df['price']  # Dependent variable
```

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Calculate R-squared
r2 = r2_score(y_test, y_pred)

# Calculate adjusted R-squared
n = len(y_test)  # Number of observations
k = X_train.shape[1]  # Number of predictors

adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - k - 1)
print(f"R-squared: {r2}")
print(f"Adjusted R-squared: {adjusted_r2}")

# Plotting the Actual vs Predicted values
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', lw=2)  # 45-degree line
plt.title('Actual vs Predicted values')
plt.xlabel('Actual Values (price)')
plt.ylabel('Predicted Values (price)')
plt.show()
```

```
R-squared: 0.8208130980062112
Adjusted R-squared: 0.806284430276985
```



Curve fitting (Polynomial equation)

In [ ]:

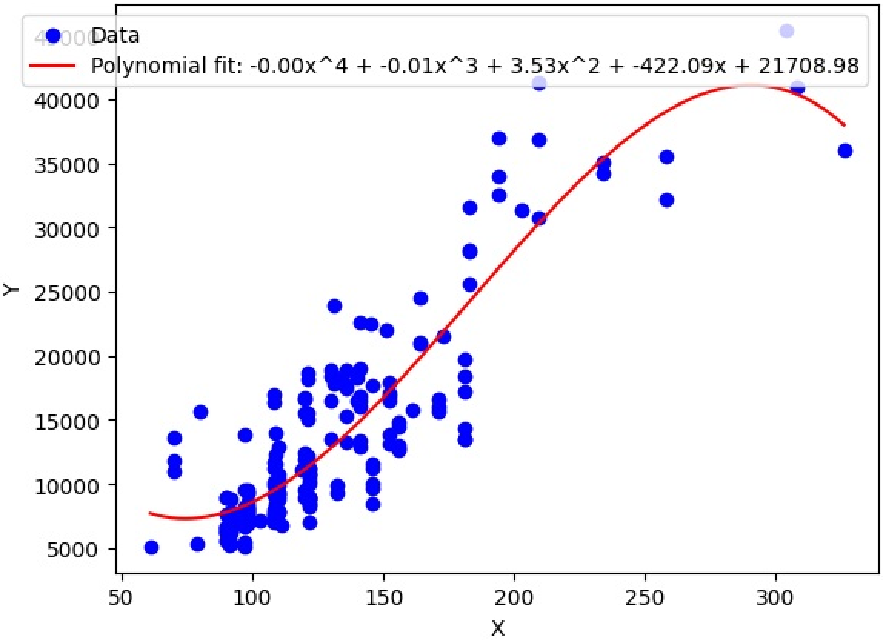
```
```

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Data
x_data = df['enginesize'].values
y_data = df['price'].values

# Define the 4th-degree polynomial model
def polynomial_model(x, a, b, c, d, e):
    return a * x**4 + b * x**3 + c * x**2 + d * x + e

# Fit the 4th-degree polynomial model to the data
polynomial_params, _ = curve_fit(polynomial_model, x_data, y_data)

# Extract the coefficients (a, b, c, d, e)
a, b, c, d, e = polynomial_params
print(f"Polynomial coefficients: a = {a}, b = {b}, c = {c}, d = {d}, e = {e}")

# Generate the polynomial fit line
```

```python
x_fit = np.linspace(min(x_data), max(x_data), 100)
y_polynomial_fit = polynomial_model(x_fit, a, b, c, d, e)

# Plotting the data and the polynomial fit
plt.scatter(x_data, y_data, color='blue', label='Data')
plt.plot(x_fit, y_polynomial_fit, color='red', label=f'Polynomial fit: {a:.2f}x^4 + {b:.2f}x^3 + {c:.2f}x^2 + {
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

```
Polynomial coefficients: a = -7.932093656702786e-07, b = -0.006128503949338147, c = 3.530275633369346, d = -422
.08871731704414, e = 21708.98411034999
```



In [ ]:

```python
import numpy as np
import pandas as pd
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt



x_data = df['enginesize'].values
y_data = df['price'].values

# 3. Define potential models
def linear_model(x, a, b):
    return a * x + b

def quadratic_model(x, a, b, c):
    return a * x**2 + b * x + c

def exponential_model(x, a, b):
    return a * np.exp(b * x)

def logarithmic_model(x, a, b):
    return a * np.log(x) + b

def polynomial_model(x, a, b, c,d,e):
    return a * x**4 + b * x**3 + c * x**2 + d*x+e

# 4. Fit the models to the data
linear_params, _ = curve_fit(linear_model, x_data, y_data)
quadratic_params, _ = curve_fit(quadratic_model, x_data, y_data)
exponential_params, _ = curve_fit(exponential_model, x_data, y_data)
logarithmic_params, _ = curve_fit(logarithmic_model, x_data, y_data)
polynomial_params, _ = curve_fit(polynomial_model, x_data, y_data)

print("Linear fit parameters:", linear_params)
print("Quadratic fit parameters:", quadratic_params)
print("Exponential fit parameters:", exponential_params)
print("Logarithmic fit parameters:", logarithmic_params)
print("Polynomial fit parameters:", polynomial_params)

# 5. Define R-squared function to evaluate fit
def r_squared(y_true, y_pred):
    residuals = y_true - y_pred
    ss_res = np.sum(residuals**2)
    ss_tot = np.sum((y_true - np.mean(y_true))**2)
    r2 = 1 - (ss_res / ss_tot)
    return r2

# 6. Compute predicted y-values for each model
```

```
y_linear_fit = linear_model(x_data, *linear_params)
y_quadratic_fit = quadratic_model(x_data, *quadratic_params)
y_exponential_fit = exponential_model(x_data, *exponential_params)
y_logarithmic_fit = logarithmic_model(x_data, *logarithmic_params)
y_polynomial_fit = polynomial_model(x_data, *polynomial_params)
# 7. Evaluate R-squared for each model
print("R-squared for linear fit:", r_squared(y_data, y_linear_fit))
print("R-squared for quadratic fit:", r_squared(y_data, y_quadratic_fit))
print("R-squared for exponential fit:", r_squared(y_data, y_exponential_fit))
print("R-squared for logarithmic fit:", r_squared(y_data, y_logarithmic_fit))
print("R-squared for polynomial fit:", r_squared(y_data, y_polynomial_fit))
```

```
Linear fit parameters: [  167.69841668 -8005.44557058]
Quadratic fit parameters: [-2.55397620e-02  1.76231991e+02 -8.63301592e+03]
Exponential fit parameters: [1.56878817e-15 1.00000000e+00]
Logarithmic fit parameters: [  23875.16545936 -101328.48660362]
Polynomial fit parameters: [-7.93209366e-07 -6.12850395e-03  3.53027563e+00 -4.22088717e+02
  2.17089841e+04]
R-squared for linear fit: 0.7641291357806177
R-squared for quadratic fit: 0.7642334305374976
R-squared for exponential fit: -2.7323223540835977e+243
R-squared for logarithmic fit: 0.7151419495741502
R-squared for polynomial fit: 0.7938699575446959
```

Piechart

In [ ]:
```
avg_enginesize=np.zeros(22)
for i in range(1,23):
    df_temp=df[df['symboling']==i]
    avg_enginesize[i-1]=df_temp['enginesize'].mean()
#avg_enginesize = sorted(avg_enginesize, reverse=True)
avg_enginesize
```

Out[ ]:
```
array([137.33333333, 130.71428571, 166.875     ,  80.33333333,
       102.66666667,  99.30769231, 102.5       , 280.66666667,
       103.        , 226.5       , 140.        , 118.30769231,
       127.88888889, 135.81818182, 106.28571429, 187.2       ,
       132.        , 121.        , 107.08333333, 118.8125    ,
       107.25      , 142.27272727])
```

In [ ]:
```
avg_enginesize = sorted(avg_enginesize, reverse=True)    #8,10,16,3,22,11,1,14,17,2,13,18,20,12,21,19,15,9,5,7,
avg_enginesize
```

Out[ ]:
```
[280.6666666666667,
 226.5,
 187.2,
 166.875,
 142.27272727272728,
 140.0,
 137.33333333333334,
 135.8181818181818,
 132.0,
 130.71428571428572,
 127.88888888888889,
 121.0,
 118.8125,
 118.3076923076923,
 107.25,
 107.08333333333333,
 106.28571428571429,
 103.0,
 102.66666666666667,
 102.5,
 99.3076923076923,
 80.33333333333333]
```

In [ ]:
```
labels=['Jaguar','Buick','Porsche','BMW','Volvo','Mercury','Alfa-Romeo','Peugeot','Renault','Audi','Nissan','Sa
```

In [ ]:
```
top_10_brands=labels[:10]
top_10_brands
```

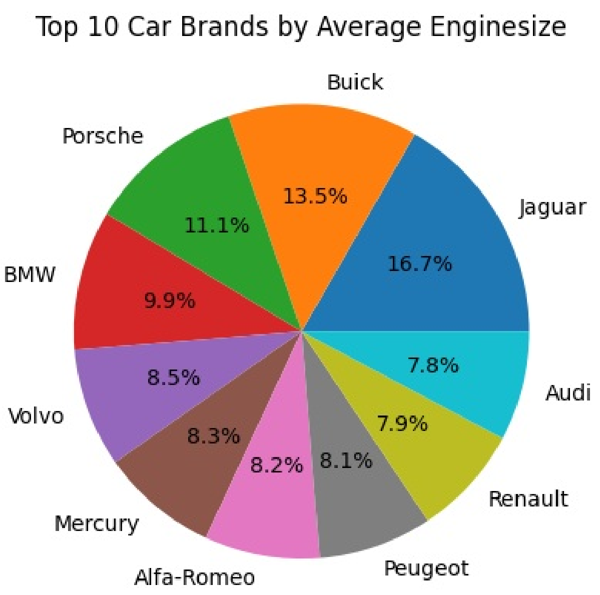
Out[ ]:
```
['Jaguar',
 'Buick',
 'Porsche',
 'BMW',
 'Volvo',
 'Mercury',
 'Alfa-Romeo',
 'Peugeot',
 'Renault',
 'Audi']
```

In [ ]:
```
top_10_enginesize_values=avg_enginesize[:10]
top_10_enginesize_values
```

Out[ ]:
```
[280.6666666666667,
 226.5,
 187.2,
 166.875,
 142.27272727272728,
 140.0,
 137.33333333333334,
 135.8181818181818,
 132.0,
 130.71428571428572]
```

In [ ]:
```
plt.pie(top_10_enginesize_values, labels=top_10_brands, autopct='%1.1f%%')
plt.title('Top 10 Car Brands by Average Enginesize')
```

Out[ ]:
```
Text(0.5, 1.0, 'Top 10 Car Brands by Average Enginesize')
```



Machine Learning Model

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

In [ ]:
```
door = {'two':2,'four':4}
```

In [ ]:
```
df['doornumber'] = df['doornumber'].map(door)
df['doornumber']
```

Out[ ]:

	doornumber
0	2
1	2
2	2
3	4
4	4
...	...
200	4
201	4
202	4
203	4
204	4

205 rows × 1 columns

dtype: int64

In [ ]:
```
body = {'convertible':1,'hatchback':2,'sedan':3, 'wagon':4, 'hardtop':5, 'convertible':6}
```

In [ ]:
```
df['carbody'] = df['carbody'].map(body)
```

In [ ]:

```
df['carbody'] = df['carbody'].map(body)
df['carbody']
```

Out[ ]:

	carbody
0	6
1	6
2	2
3	3
4	3
...	...
200	3
201	3
202	3
203	3
204	3

205 rows × 1 columns

dtype: int64

In [ ]:

```
df['enginetype'].unique()
```

Out[ ]:

```
array(['dohc', 'ohcv', 'ohc', 'l', 'rotor', 'ohcf', 'dohcv'], dtype=object)
```

In [ ]:

```
engine = {'dohc':1, 'ohcv':2, 'ohc':3, 'l':4, 'rotor':5, 'ohcf':6, 'dohcv':7}
```

In [ ]:

```
df['enginetype'] = df['enginetype'].map(engine)
df['enginetype']
```

Out[ ]:

	enginetype
0	1
1	1
2	2
3	3
4	3
...	...
200	3
201	3
202	2
203	3
204	3

205 rows × 1 columns

dtype: int64

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   car_ID            205 non-null    int64  
 1   symboling         205 non-null    int64  
 2   CarName           205 non-null    object 
 3   fueltype          205 non-null    object 
 4   aspiration        205 non-null    object 
 5   doornumber        205 non-null    int64  
 6   carbody           205 non-null    int64  
 7   drivewheel        205 non-null    object 
 8   enginelocation    205 non-null    object 
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64  
 14  enginetype        205 non-null    int64  
 15  cylindernumber    205 non-null    object 
 16  enginesize        205 non-null    int64  
 17  fuelsystem        205 non-null    object 
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64  
 22  peakrpm           205 non-null    int64  
 23  citympg           205 non-null    int64  
 24  highwaympg        205 non-null    int64  
 25  price             205 non-null    float64
dtypes: float64(8), int64(11), object(7)
memory usage: 41.8+ KB
```

In [ ]:
```python
df.drop(columns=['CarName','fueltype','aspiration','drivewheel','enginelocation','cylindernumber','fuelsystem',
```

In [ ]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 19 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   car_ID            205 non-null    int64  
 1   symboling         205 non-null    int64  
 2   doornumber        205 non-null    int64  
 3   carbody           205 non-null    int64  
 4   wheelbase         205 non-null    float64
 5   carlength         205 non-null    float64
 6   carwidth          205 non-null    float64
 7   carheight         205 non-null    float64
 8   curbweight        205 non-null    int64  
 9   enginetype        205 non-null    int64  
 10  enginesize        205 non-null    int64  
 11  boreratio         205 non-null    float64
 12  stroke            205 non-null    float64
 13  compressionratio  205 non-null    float64
 14  horsepower        205 non-null    int64  
 15  peakrpm           205 non-null    int64  
 16  citympg           205 non-null    int64  
 17  highwaympg        205 non-null    int64  
 18  price             205 non-null    float64
dtypes: float64(8), int64(11)
memory usage: 30.6 KB
```

In [ ]:
```python
X=df.drop(['price'], axis=1)
```

In [ ]:
```python
y=df['price']
```

In [ ]:
```python
from sklearn.feature_selection import mutual_info_regression
```

In [ ]:
```python
imp=mutual_info_regression(X,y)
```

In [ ]:
```python
pd.DataFrame(imp)
```

Out[ ]:

	0
0	0.436250
1	0.214741
2	0.000000
3	0.051091
4	0.580270
5	0.553567
6	0.663565
7	0.345273
8	0.856740
9	0.189631
10	0.814015
11	0.455993
12	0.370516
13	0.170294
14	0.855570
15	0.189717
16	0.738283
17	0.851571

In [ ]:
```
imp_df=pd.DataFrame(imp,index=X.columns)
```

In [ ]:
```
imp_df.columns=['importance']
```

In [ ]:
```
imp_df.sort_values(by ='importance', ascending=False)
```

Out[ ]:

	importance
curbweight	0.856740
horsepower	0.855570
highwaympg	0.851571
enginesize	0.814015
citympg	0.738283
carwidth	0.663565
wheelbase	0.580270
carlength	0.553567
boreratio	0.455993
car_ID	0.436250
stroke	0.370516
carheight	0.345273
symboling	0.214741
peakrpm	0.189717
enginetype	0.189631
compressionratio	0.170294
carbody	0.051091
doornumber	0.000000

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.25,random_state=42)
```

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [ ]:
```
ml_model = RandomForestRegressor()
```

In [ ]:
```
ml_model.fit(X_train,y_train)
```

Out[ ]:

```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [ ]:
```
y_pred = ml_model.predict(X_test)
y_pred
```

Out[ ]:
```
array([35672.535, 18776.8  ,  9170.58 , 12921.88 , 27975.05 ,  6702.79 ,
        8000.61 ,  8072.07 ,  9399.59 ,  8333.35 , 14315.99 ,  7973.57 ,
       14228.7  , 10802.17 , 40903.255,  6502.535,  6037.485, 14471.61 ,
        8538.56 ,  8899.07 , 10042.66 , 15163.18 ,  7125.52 ,  6072.97 ,
        7315.425, 35772.805,  9359.2  , 16584.89 ,  7253.12 , 16433.76 ,
       28037.89 ,  6730.09 ,  8437.64 , 19560.92 ,  8060.86 , 28017.69 ,
        9813.58 , 12457.61 ,  7217.355, 14654.3  ,  8419.5  , 10849.31 ,
       14270.46 ,  8048.28 ,  7227.14 ,  8509.89 ,  6907.14 ,  7503.975,
       15298.49 , 15585.43 ,  6840.64 , 16585.63 ])
```

In [ ]:
```
from sklearn import metrics
```

In [ ]:
```
metrics.r2_score(y_test,y_pred)
```

Out[ ]:
```
0.9423238534600755
```

In [ ]:
```
r2 = metrics.r2_score(y_test,y_pred)

n = len(y_test)
p = X_test.shape[1]

# Calculate Adjusted R^2
adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)

print(f"Adjusted R^2: {adjusted_r2}")
```

```
Adjusted R^2: 0.9108641371655712
```

In [ ]:
```
def predict(ml_model):
  model=ml_model.fit(X_train,y_train)
  print('Training score:{}',format(model.score(X_train,y_train)))
  y_prediction = model.predict(X_test)
  print('Predictions are:{}'.format(y_prediction))
  print('\n')
  r2_score= metrics.r2_score(y_test,y_pred)
  print('r2 score:{}'.format(r2_score))
  print('MAE:{}'.format(metrics.mean_absolute_error(y_prediction,y_test)))
  print('MSE:{}'.format(metrics.mean_squared_error(y_prediction,y_test)))
  print('RMSE:{}'.format(np.sqrt(metrics.mean_squared_error(y_prediction,y_test))))
```

In [ ]:
```
predict(RandomForestRegressor())
```

```
Training score:{} 0.9880582178978995
Predictions are:[35509.115 18891.47   9031.34  13163.29  27208.64   6522.11   7912.15
  8080.22   9433.33   8238.44  14349.99   7953.13  14159.82  10764.76
 40841.465  6337.94   5669.945 14418.77   8334.36   8952.35   9910.17
 14691.59   7182.31   5873.7    7268.52  35465.455  9294.3   16729.89
  7263.7   16356.39  27351.58   6597.76   8448.26  19996.12   8093.74
 27738.755 10082.71  12710.87   7428.78  14699.76   8484.78  11016.25
 14085.65   8074.845  7215.82   8697.23   6703.76   7617.64  15641.33
 15385.91   6640.45  16578.73 ]


r2 score:0.9423238534600755
MAE:1296.1694807692306
MSE:3775289.184699211
RMSE:1943.010340862655
```

XGBOOST

In [ ]:
```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd

# Features and target
X = df.drop("price", axis=1)
y = df["price"]

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Define and train the model using scikit-learn API
model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, learning_rate=0.1)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)


# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
print(f"R-squared Score: {r2}")
```

```
R-squared Score: 0.9134656914414635
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js