ML Tasks

ML Tasks:1

Measures of Descriptive statistics-Central Tendency,spread

You are given house_price.csv which contains property prices in the city of Bangalore. You need to examine price per square feet do the following:

Detect the outliers and remove it using:

1. Mean Function
2. Percentile method
3. IQR(Inter quartile range method)
4. Normal distribution
5. Zscore method

Also, plot the box plot(for all the numerical columns), histplot(to check the normality of the column(price per sqft column))
Check the correlation between all the numerical columns and plot heatmap.
Scatter plot between the variables to check the correlation between them.
For the percentile method, you can consider less than 5% and greater than 95% .

[5]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[6]:
```python
#load the data set

data1 = pd.read_csv("house_price.csv")
data1
```

[6]:

	location	size	total_sqft	bath	price	bhk	price_per_sqft
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2	3699
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4	4615
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3	4305
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3	6245
4	Kothanur	2 BHK	1200.0	2.0	51.00	2	4250
...	...	...	...	...	...	...	...
13195	Whitefield	5 Bedroom	3453.0	4.0	231.00	5	6689
13196	other	4 BHK	3600.0	5.0	400.00	4	11111
13197	Raja Rajeshwari Nagar	2 BHK	1141.0	2.0	60.00	2	5258
13198	Padmanabhanagar	4 BHK	4689.0	4.0	488.00	4	10407
13199	Doddathoguru	1 BHK	550.0	1.0	17.00	1	3090

13200 rows × 7 columns

[7]:
```python
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13200 entries, 0 to 13199
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   location        13200 non-null  object 
 1   size            13200 non-null  object 
 2   total_sqft      13200 non-null  float64
 3   bath            13200 non-null  float64
 4   price           13200 non-null  float64
 5   bhk             13200 non-null  int64  
 6   price_per_sqft  13200 non-null  int64  
dtypes: float64(3), int64(2), object(2)
memory usage: 722.0+ KB
```

[8]:
```python
data1.describe()
```

[8]:

	total_sqft	bath	price	bhk	price_per_sqft
count	13200.000000	13200.000000	13200.000000	13200.000000	1.320000e+04
mean	1555.302783	2.691136	112.276178	2.800833	7.920337e+03
std	1237.323445	1.338915	149.175995	1.292843	1.067272e+05
min	1.000000	1.000000	8.000000	1.000000	2.670000e+02
25%	1100.000000	2.000000	50.000000	2.000000	4.267000e+03
50%	1275.000000	2.000000	71.850000	3.000000	5.438000e+03
75%	1672.000000	3.000000	120.000000	3.000000	7.317000e+03
max	52272.000000	40.000000	3600.000000	43.000000	1.200000e+07

```
[26]: #UPPER LIMIT



upper_limit = np.percentile(data1['price_per_sqft'], 0.95)
percentile_upper= data1 ["price_per_sqft"].quantile(0.95)

percentile_upper
```

```
[26]: 15312.099999999984
```

```
[29]: percentile_outliers = data1 [(data1['price_per_sqft'] < lower_limit) | (data1['price_per_sqft'] > upper_limit)]
```

```
[30]:
cleaned_percentile=data1[~data1["price_per_sqft"].isin(percentile_outliers ["price_per_sqft"])]

print("Outliers Detected Using Percentile:",len(percentile_outliers))
```

```
Outliers Detected Using Percentile: 13081
```

```
[31]: #IQR [Inter Quartile Range]

Q1 = data1['price_per_sqft'].quantile(0.25)
Q1
```

```
[31]: 4267.0
```

```
[33]: Q3 =data1['price_per_sqft'].quantile(0.75)
Q3
```

```
[33]: 7317.0
```

```
[34]: IQR= Q3-Q1
IQR
```

```
[34]: 3050.0
```

```
[37]: lower_bound=Q1-1.5*IQR
lower_bound
```

```
[37]: -308.0
```

```
[9]: #mean function

mean_price = data1['price_per_sqft'].mean()
mean_price
```

```
[9]: 7920.336742424242
```

```
[14]: #STANDARD DEVIATION

std_dev = data1["price_per_sqft"].std()
std_dev
```

```
[14]: 106727.16032810867
```

```
[ ]: threshold =2

outliers_mean=data1[(data1["price_per_sqft"]< mean_price - threshold *std_dev)|
(data1["price_per_sqft"] > mean_price +threshold * std_dev)]
```

percentile method

```
[25]: #LOWER LIMIT

lower_limit = np.percentile(data1["price_per_sqft"], 0.05)

percentile_lower= data1["price_per_sqft"].quantile(0.05)

percentile_lower
```

```
[25]: 3107.8500000000004
```

[37]:
```
lower_bound=Q1-1.5*IQR
lower_bound
```

[37]: -308.0

[38]:
```
upper_bound = Q3 +1.5*IQR
upper_bound
```

[38]: 11892.0

[40]:
```
IQR_outliers = data1[(data1["price_per_sqft"]<lower_bound) | (data1["price_per_sqft"]>upper_bound)]

cleaned_IQR = data1 [~data1 ["price_per_sqft"].isin(percentile_outliers["price_per_sqft"])]

print("Outliers Detected Using IQR:" ,len(IQR_outliers))
```

Outliers Detected Using IQR: 1265

[25]:
```
from scipy.stats import zscore
```

[26]:
```
data1["z_score"] = zscore(data1["price_per_sqft"])
data1.head()
```

[26]:

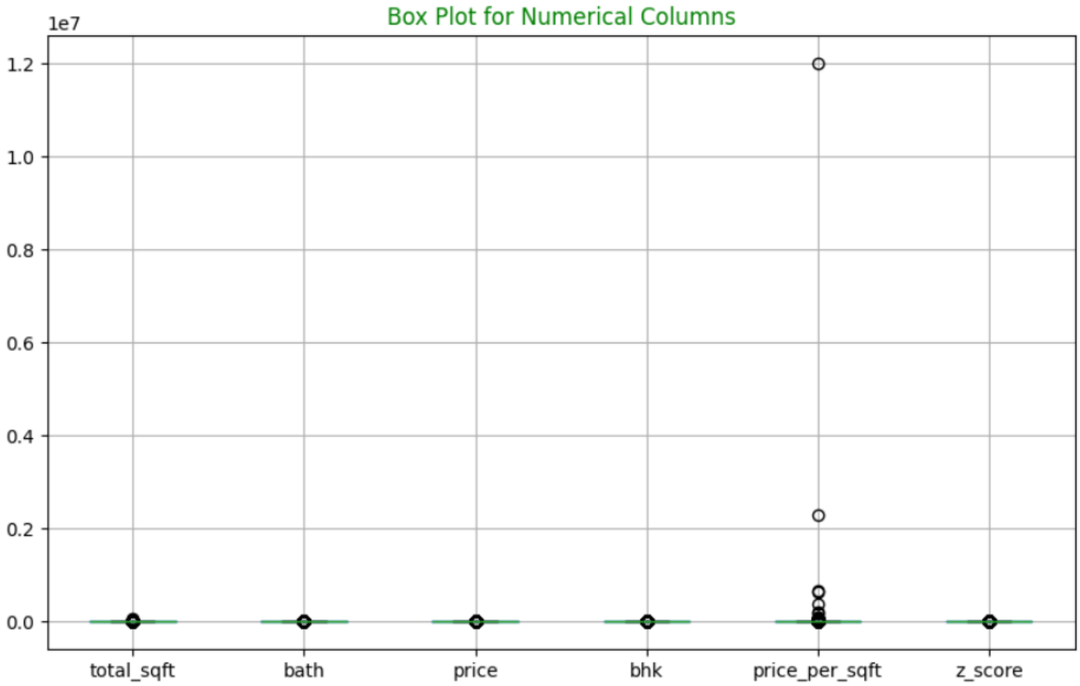
	location	size	total_sqft	bath	price	bhk	price_per_sqft	z_score
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2	3699	-0.039554
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4	4615	-0.030971
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3	4305	-0.033876
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3	6245	-0.015698
4	Kothanur	2 BHK	1200.0	2.0	51.00	2	4250	-0.034391

[27]:
```
threshold = 2
zscore_outliers=data1[(data1["price_per_sqft"]>threshold) | (data1["price_per_sqft"]<-threshold)]
```

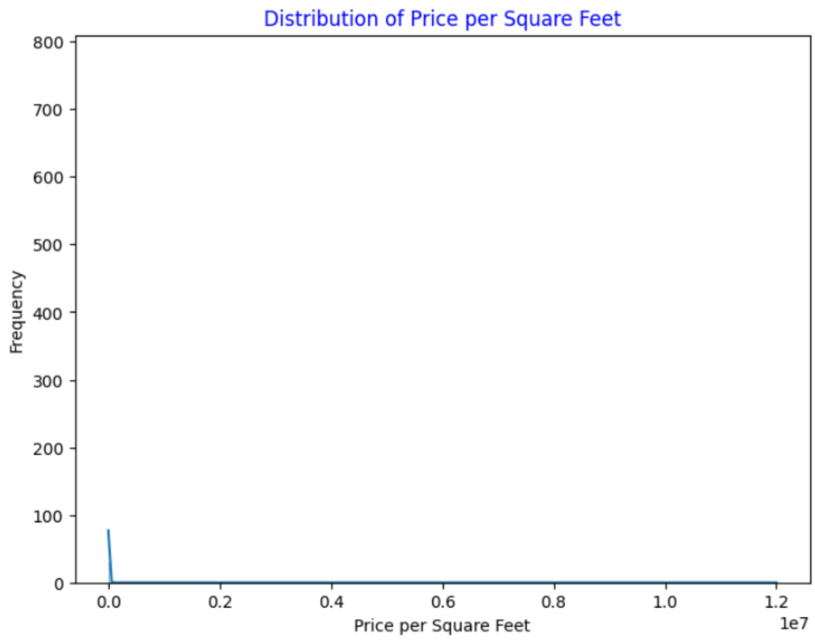
[28]:
```
cleaned_zscore = data1 [~data1["price_per_sqft"].isin(zscore_outliers ["price_per_sqft"])]
print("Outliers Detected Using zscore:" ,len(zscore_outliers))
```

Outliers Detected Using zscore: 13200

[29]:
```
 # Plotting box plot for all numerical columns


data1.boxplot(figsize=(10,6))
plt.title('Box Plot for Numerical Columns', color='g')
plt.show()
```

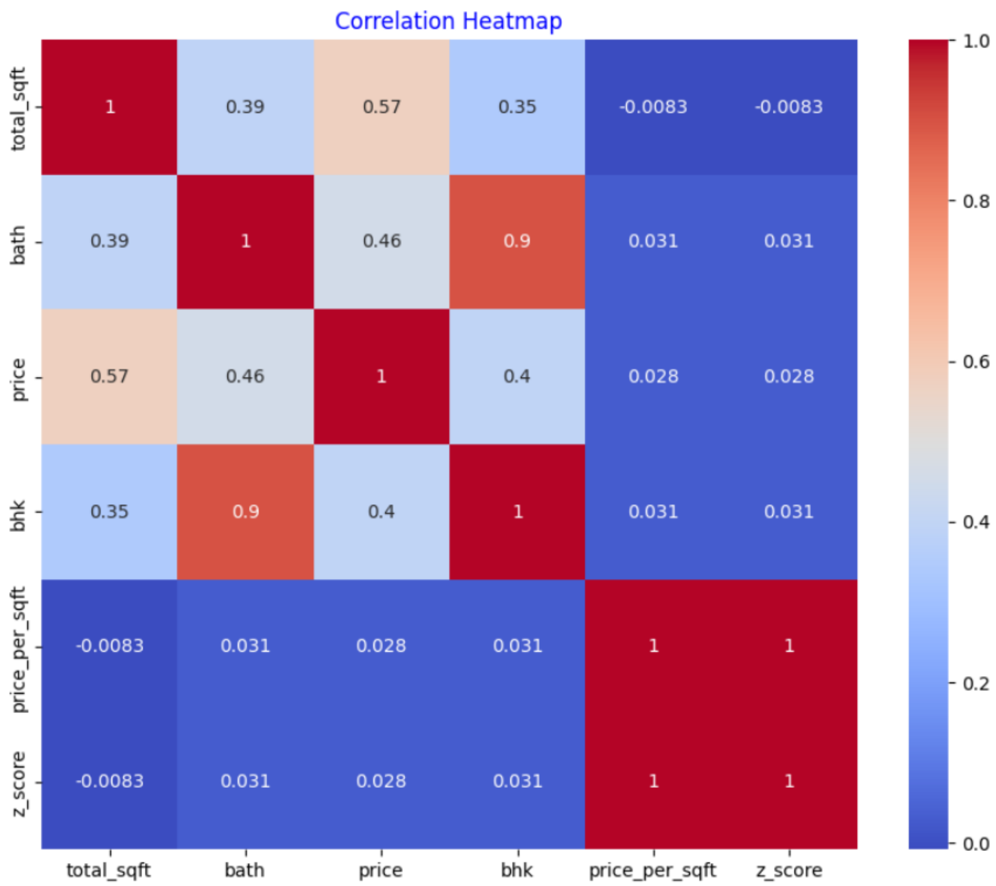


[30]:
```python
# Plotting histplot for price_per_sqft column

plt.figure(figsize=(8, 6))
sns.histplot(data1['price_per_sqft'], kde=True)
plt.title('Distribution of Price per Square Feet', color='b')
plt.xlabel('Price per Square Feet')
plt.ylabel('Frequency')
plt.show()
```



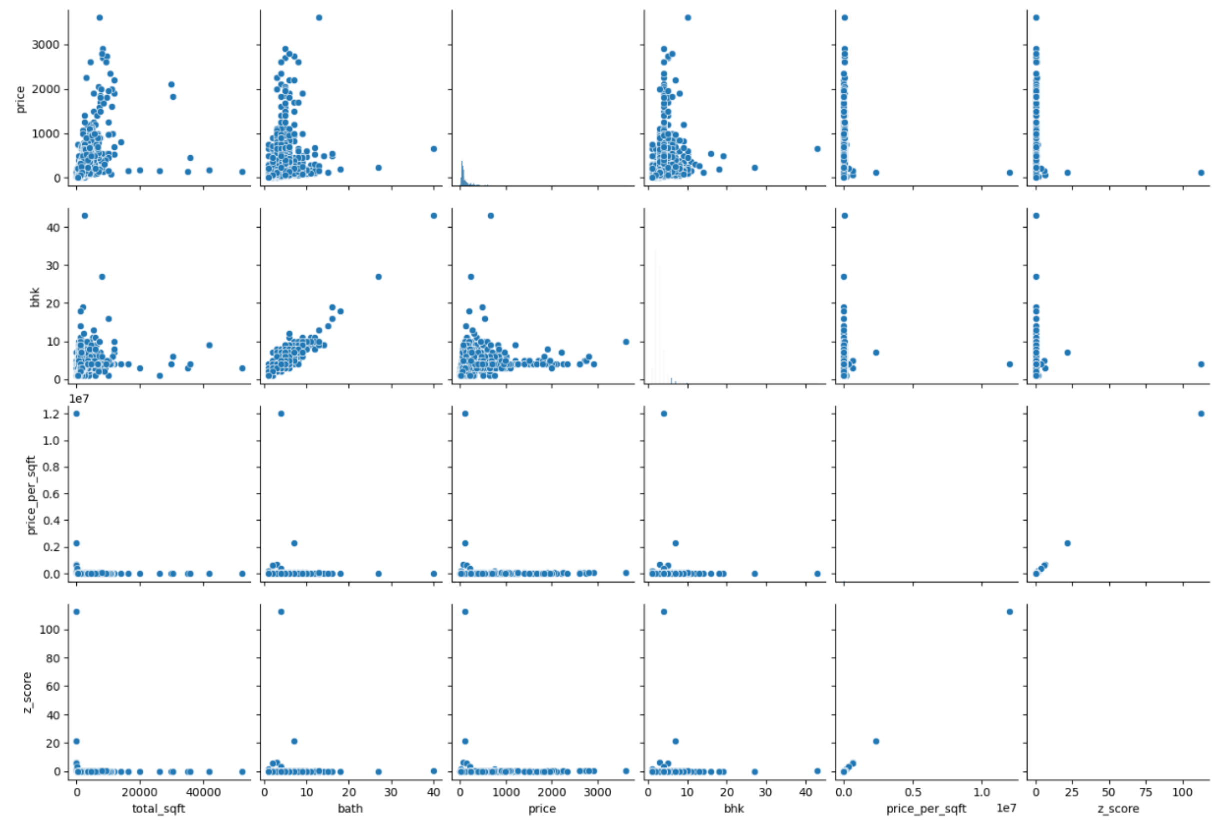
[31]:

```
#correlation between all the numerical columns

columns_numeric=data1.select_dtypes (include=['number'])
matrix_correlation=columns_numeric.corr()


# Checking correlation between numerical columns and plotting heatmap

plt.figure(figsize=(10, 8))
sns.heatmap(matrix_correlation, annot= True, cmap='coolwarm')
plt.title('Correlation Heatmap', color='b')
plt.show()
```



[32]:

```
#Scatter plot between numerical columns to check correlation

sns.pairplot(data1)
plt.suptitle('Pairplot for Numerical Columns', color='g')
plt.show()
```



ML - Task2

Hypothesis testing

Q1. Suppose a child psychologist claims that the average time working mothers spend talking to their children is at least 11 minutes per day. You conduct a random sample of 1000 working mothers and find they spend an average of 11.5 minutes per day talking with their children. Assume prior research suggests the population standard deviation is 2.3 minutes. Conduct a test with a level of significance of alpha = 0.05.

```
[33]: from scipy import stats
      import numpy as np
```

```
[37]: #Given data

      sample_mean= 11.5 # sample mean
      population_mean =11 # population mean under the null hypothesis
      population_std= 2.3 # population standard deviation
      sample_size = 1000 # sample size
      alpha= 0.05 # significance Level
```

```
[39]:
      #Calculate the z-score

      z_score = (sample_mean - population_mean) / (population_std /(sample_size**0.5))
      print(" z-score:", z_score)

       z-score: 6.874516652539955
```

```
[40]: #Calculate the critical value

      critical_value =stats.norm.ppf(1- alpha)
      print("Critical Value:", critical_value)

      Critical Value: 1.6448536269514722
```

```
[41]: #Determine if the null hypothesis is rejected or not

      if z_score > critical_value:
          print("Reject the null hypothesis.")
      else:
          print("Fail to reject the null hypothesis.")

      Reject the null hypothesis.
```

Q2. A coffee shop claims that their average wait time for customers is less than 5 minutes. To test this claim, a sample of 40 customers is taken, and their wait times are recorded. The sample mean wait time is found to be 4.6 minutes with a standard deviation of 0.8 minutes. Perform a hypothesis test at a significance level of 0.05 and determine whether there is enough evidence to support the coffee shop's claim.

[42]:
```python
#Given data

sample_mean = 4.6 # sample mean
population_mean = 5 # population mean under the null hypothesis
population_std= 0.8 # population standard deviation
sample_size = 40 # sample size
alpha =0.05 # significance level
```

[43]:
```python
#Calculate the z-score

z_score = (sample_mean - population_mean) / (population_std/(sample_size**0.5))
print(" z-score:", z_score)
```

```
 z-score: -3.162277660168382
```

[44]:
```python
#Calculate the critical value

critical_value= stats.norm.ppf(1- alpha)
print("Critical Value:", critical_value)
```

```
Critical Value: 1.6448536269514722
```

[45]:
```python
#Determine if the null hypothesis is rejected or not

if z_score > critical_value:
    print("Reject the null hypothesis.")
else:
    print("Fail to reject the null hypothesis.")
```

```
Fail to reject the null hypothesis.
```

ML - Task3

- Data Preprocessing

Objective:

The main objective of this project is to design and implement a robust data preprocessing system that addresses common challenges such as missing values, outliers, inconsistent formatting, and noise. By performing effective data preprocessing, the project aims to enhance the quality, reliability, and usefulness of the data for machine learning.

Key Components to be fulfilled:

Data Exploration: Explore the data, list down the unique values in each feature and find its length. Perform the statistical analysis and renaming of the columns.

Data Cleaning:
Find the missing and inappropriate values, treat them appropriately. Remove all duplicate rows. Find the outliers.

- Replace the value 0 in age as NaN
- Treat the null values in all columns using any measures(removing/ replace the values with mean/median/mode)
-

Data Analysis:

- Filter the data with age >40 and salary<5000
- Plot the chart with age and salary
- Count the number of people from each place and represent it visually

Data Encoding:
Convert categorical variables into numerical representations using techniques such as one-hot encoding, label encoding, making them suitable for analysis by machine learning algorithms.

Feature Scaling:
After the process of encoding, perform the scaling of the features using standardscaler and minmaxscaler.

[51]:
```
#Load the data set

data2=pd.read_csv("employee.csv")
data2
```

[51]:

	Company	Age	Salary	Place	Country	Gender
0	TCS	20.0	NaN	Chennai	India	0
1	Infosys	30.0	NaN	Mumbai	India	0
2	TCS	35.0	2300.0	Calcutta	India	0
3	Infosys	40.0	3000.0	Delhi	India	0
4	TCS	23.0	4000.0	Mumbai	India	0
...	...	...	...	...	...	...
143	TCS	33.0	9024.0	Calcutta	India	1
144	Infosys	22.0	8787.0	Calcutta	India	1
145	Infosys	44.0	4034.0	Delhi	India	1
146	TCS	33.0	5034.0	Mumbai	India	1
147	Infosys	22.0	8202.0	Cochin	India	0

148 rows × 6 columns

[53]:
```python
#data exploration

print("data exploration")
print(data2.head()) #DISPLAY FIRST FEW ROWS

for column in data2.columns:
    print(f"unique values in {column}:{data2[column].unique()}(length:{len(data2[column].unique())})")

print(data2.describe())
```

```
data exploration
   Company   Age  Salary     Place Country  Gender
0      TCS  20.0     NaN   Chennai   India       0
1  Infosys  30.0     NaN    Mumbai   India       0
2      TCS  35.0  2300.0  Calcutta   India       0
3  Infosys  40.0  3000.0     Delhi   India       0
4      TCS  23.0  4000.0    Mumbai   India       0
unique values in Company:['TCS' 'Infosys' 'CTS' nan 'Tata Consultancy Services' 'Congnizant'
 'Infosys Pvt Lmt'](length:7)
unique values in Age:[20. 30. 35. 40. 23. nan 34. 45. 18. 22. 32. 37. 50. 21. 46. 36. 26. 41.
 24. 25. 43. 19. 38. 51. 31. 44. 33. 17.  0. 54.](length:30)
unique values in Salary:[  nan 2300. 3000. 4000. 5000. 6000. 7000. 8000. 9000. 1089. 1234. 3030.
 3045. 3184. 4824. 5835. 7084. 8943. 8345. 9284. 9876. 2034. 7654. 2934.
 4034. 5034. 8202. 9024. 4345. 6544. 6543. 3234. 4324. 5435. 5555. 8787.
 3454. 5654. 5009. 5098. 3033.](length:41)
unique values in Place:['Chennai' 'Mumbai' 'Calcutta' 'Delhi' 'Podicherry' 'Cochin' nan 'Noida'
 'Hyderabad' 'Bhopal' 'Nagpur' 'Pune'](length:12)
unique values in Country:['India'](length:1)
unique values in Gender:[0 1](length:2)
              Age       Salary      Gender
count  130.000000   124.000000  148.000000
mean    30.484615  5312.467742    0.222973
std     11.096640  2573.764683    0.417654
min      0.000000  1089.000000    0.000000
25%     22.000000  3030.000000    0.000000
50%     32.500000  5000.000000    0.000000
75%     37.750000  8000.000000    0.000000
max     54.000000  9876.000000    1.000000
```

[ ]:
```python
#Data Cleaning

data2['Age'].replace(0, np.nan, inplace=True)
data2.dropna(inplace=True)
```

[55]:
```python
from scipy.stats import zscore

# Calculate z-score for 'age' and 'salary'
data2['age_zscore'] = zscore(data2['Age'])
data2['salary_zscore'] = zscore (data2 ['Salary'])
zscore_threshold = 3


# Identify outliers based on z-score

outliers_age = data2[abs(data2 ['age_zscore']) > zscore_threshold]
outliers_salary = data2[abs (data2['salary_zscore']) > zscore_threshold]
print("Outliers based on z-score for age:")
print(outliers_age)
print("\nOutliers based on z-score for salary:")
print(outliers_salary)
```

```
Outliers based on z-score for age:
Empty DataFrame
Columns: [Company, Age, Salary, Place, Country, Gender, age_zscore, salary_zscore]
Index: []

Outliers based on z-score for salary:
Empty DataFrame
Columns: [Company, Age, Salary, Place, Country, Gender, age_zscore, salary_zscore]
Index: []
```

[56]:
```python
#Plot age vs salary

filtered_data=data2 [(data2["Age"]>40)&(data2 ["Salary"]<5000)]
plt.scatter(filtered_data["Age"], filtered_data["Salary"])
plt.xlabel('Age')
plt.ylabel('Salary')
plt.title('Age vs Salary')
plt.show()
```

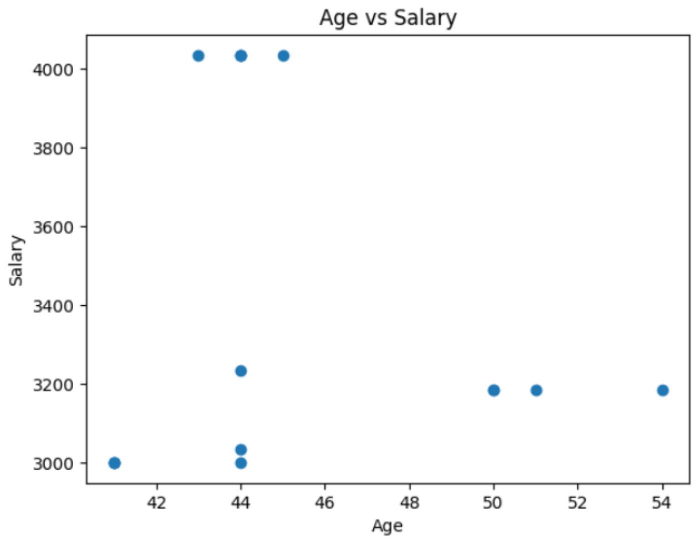
```
[4]: # Data Cleaning

df['Age'].replace(0, np.nan, inplace=True)

df.dropna(inplace=True)
```

```
[5]: from scipy.stats import zscore

# Calculate z-score for 'age' and 'salary'

df['age_zscore'] = zscore(df['Age'])
df['salary_zscore'] = zscore(df['Salary'])

zscore_threshold = 3

# Identify outliers based on z-score

outliers_age = df[abs(df['age_zscore']) > zscore_threshold]
outliers_salary = df[abs(df['salary_zscore']) > zscore_threshold]

print("Outliers based on z-score for age:")
print(outliers_age)
print("\nOutliers based on z-score for salary:")
print(outliers_salary)
```

```
Outliers based on z-score for age:
Empty DataFrame
Columns: [Company, Age, Salary, Place, Country, Gender, age_zscore, salary_zscore]
Index: []

Outliers based on z-score for salary:
Empty DataFrame
Columns: [Company, Age, Salary, Place, Country, Gender, age_zscore, salary_zscore]
Index: []
```

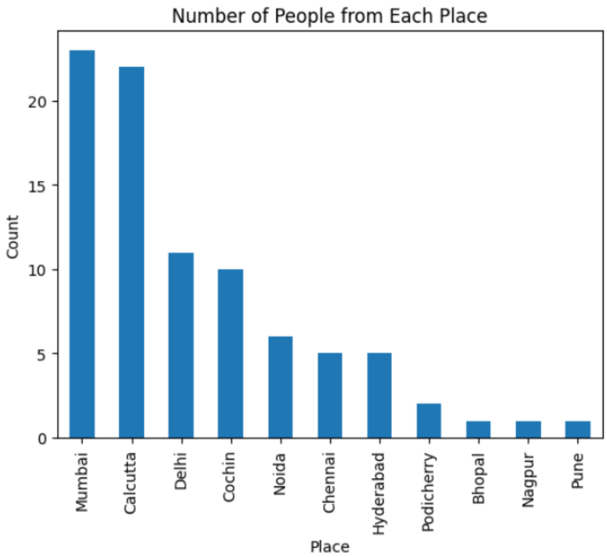
[56]:

```
#Plot age vs salary

filtered_data=data2 [(data2["Age"]>40)&(data2 ["Salary"]<5000)]
plt.scatter(filtered_data["Age"], filtered_data["Salary"])
plt.xlabel('Age')
plt.ylabel('Salary')
plt.title('Age vs Salary')
plt.show()
```



[58]:

```
#Count people from each place and visualize

place_counts= data2['Place'].value_counts()
place_counts.plot(kind='bar')
plt.xlabel('Place')
plt.ylabel('Count')
plt.title('Number of People from Each Place')
plt.show()
```



[3]:
```python
import pandas as pd
```

[ ]:
```python
from sklearn.preprocessing import minmaxscaler,standardscaler
```

[ ]:
```python
#encoded data

encoded_data =pd.get_dummies(data2, columns=['place'])

print(encoded_data.head())
```

```
   Company   Age  Salary Country  Gender  age_zscore  salary_zscore  \
2      TCS  35.0  2300.0   India       0    0.283003      -1.012973
3  Infosys  40.0  3000.0   India       0    0.818249      -0.749656
4      TCS  23.0  4000.0   India       0   -1.001585      -0.373490
7  Infosys  23.0  7000.0   India       1   -1.001585       0.755009
8      TCS  34.0  8000.0   India       1    0.175954       1.131175

   Place_Bhopal  Place_Calcutta  Place_Chennai  Place_Cochin  Place_Delhi  \
2         False            True          False         False        False
3         False           False          False         False         True
4         False           False          False         False        False
7         False           False          False         False        False
8         False            True          False         False        False

   Place_Hyderabad  Place_Mumbai  Place_Nagpur  Place_Noida  Place_Podicherry  \
2            False         False         False        False             False
3            False         False         False        False             False
4            False          True         False        False             False
7            False          True         False        False             False
8            False         False         False        False             False

   Place_Pune
2       False
3       False
4       False
7       False
8       False
```

[6]:
```python
#Create instances of StandardScaler and MinMaxScaler

scaler_standard= StandardScaler()
scaler_minmax= MinMaxScaler()


# Fit and transform the selected columns using StandardScaler

scaled_data_standard =scaler_standard.fit_transform(data2[['Age', 'Salary']])
data2[['age_standard', 'salary_standard']] = scaled_data_standard


# Fit and transform the selected columns using MinMaxScaler

scaled_data_minmax = scaler_minmax.fit_transform(data2[['Age', 'Salary']])
data2[['age_minmax', 'salary_minmax']] = scaled_data_minmax



#Print the scaled data

print("Scaled Data using StandardScaler:")
print(data2[['age_standard', 'salary_standard']].head())
print("\nScaled Data using MinMaxScaler:")
print(data2[['age_minmax', 'salary_minmax']].head())
```

```
Scaled Data using StandardScaler:
   age_standard  salary_standard
2      0.283003        -1.012973
3      0.818249        -0.749656
4     -1.001585        -0.373490
7     -1.001585         0.755009
8      0.175954         1.131175

Scaled Data using MinMaxScaler:
   age_minmax  salary_minmax
2    0.486486       0.137817
3    0.621622       0.217480
4    0.162162       0.331285
7    0.162162       0.672698
8    0.459459       0.786503
```

ML - Task4

Regression

Problem Description

A Chinese automobile company aspires to enter the US market by setting up their manufacturing unit there and producing cars locally to give competition to their US and European counterparts. They have contracted an **automobile consulting company** to understand the factors on which the pricing of cars depends. Specifically, they want to understand the factors affecting the pricing of cars in the American market, since those may be very different from the Chinese market. Essentially, the company wants to know:

- Which variables are significant in predicting the price of a car

- How well those variables describe the price of a car

Based on various market surveys, the consulting firm has gathered a large dataset of different types of cars across the American market.

Business Goal

You are required to model the price of cars with the available independent variables. It will be used by the management to understand how exactly the prices vary with the independent variables. They can accordingly manipulate the design of the cars, the business strategy etc. to meet certain price levels. Further, the model will be a good way for the management to
understand the pricing dynamics of a new market.

Apply any 5 algorithms to the regression problem provided.
For example:
Linear Regression
Decision Tree Regressor
Random Forest Regressor
Gradient Boosting Regressor
Support Vector Regressor

Rootmap:

1. Understand problem statement

2. Import necessary libraries and data

3. Check the data

Info()
Describe((
Isnull()
Duplicated()
Df. Columns
Length of unique values in each column.

4. Data preprocessing

Drop car id

Find unique values in categorical or count plot
extract company name from car name and address this new col to df also remove car name column.
There are spelling mistakes in company name. Treat this.
Label encoding all the categorical columns
Outliers detection and removal(if present)

5. Feature selection

Find correlation matrix
Remove multicolinearity (remove features with High correlation .85 to 1)

6. Data splitting

Test, train

7. Model selection and implementation

8. Model evaluation

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

[2]:
```python
df = pd.read_csv('CarPrice_Assignment.csv')
```

[3]:
```python
print(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
None
```

[4]:
```python
print(df.describe())
```

```
           car_ID   symboling   wheelbase   carlength    carwidth   carheight  \
count  205.000000  205.000000  205.000000  205.000000  205.000000  205.000000   
mean   103.000000    0.834146   98.756585  174.049268   65.907805   53.724878   
std     59.322565    1.245307    6.021776   12.337289    2.145204    2.443522   
min      1.000000   -2.000000   86.600000  141.100000   60.300000   47.800000   
25%     52.000000    0.000000   94.500000  166.300000   64.100000   52.000000   
50%    103.000000    1.000000   97.000000  173.200000   65.500000   54.100000   
75%    154.000000    2.000000  102.400000  183.100000   66.900000   55.500000   
max    205.000000    3.000000  120.900000  208.100000   72.300000   59.800000   

        curbweight  enginesize   boreratio      stroke  compressionratio  \
count   205.000000  205.000000  205.000000  205.000000        205.000000   
mean   2555.565854  126.907317    3.329756    3.255415         10.142537   
std     520.680204   41.642693    0.270844    0.313597          3.972040   
min    1488.000000   61.000000    2.540000    2.070000          7.000000   
25%    2145.000000   97.000000    3.150000    3.110000          8.600000   
50%    2414.000000  120.000000    3.310000    3.290000          9.000000   
75%    2935.000000  141.000000    3.580000    3.410000          9.400000   
max    4066.000000  326.000000    3.940000    4.170000         23.000000   

       horsepower      peakrpm     citympg  highwaympg         price  
count  205.000000   205.000000  205.000000  205.000000    205.000000  
mean   104.117073  5125.121951   25.219512   30.751220  13276.710571  
std     39.544167   476.985643    6.542142    6.886443   7988.852332  
min     48.000000  4150.000000   13.000000   16.000000   5118.000000  
25%     70.000000  4800.000000   19.000000   25.000000   7788.000000  
50%     95.000000  5200.000000   24.000000   30.000000  10295.000000  
75%    116.000000  5500.000000   30.000000   34.000000  16503.000000  
max    288.000000  6600.000000   49.000000   54.000000  45400.000000  
```

[5]:
```
print(df.isnull().sum())
```

```
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
```

[6]:
```
print(df.duplicated().sum())
```

```
0
```

[7]:
```python
print(df.columns)
for column in df.columns:
    print(f"Unique values in {column}: {len(df[column].unique())}")
```

```
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')
Unique values in car_ID: 205
Unique values in symboling: 6
Unique values in CarName: 147
Unique values in fueltype: 2
Unique values in aspiration: 2
Unique values in doornumber: 2
Unique values in carbody: 5
Unique values in drivewheel: 3
Unique values in enginelocation: 2
Unique values in wheelbase: 53
Unique values in carlength: 75
Unique values in carwidth: 44
Unique values in carheight: 49
Unique values in curbweight: 171
Unique values in enginetype: 7
Unique values in cylindernumber: 7
Unique values in enginesize: 44
Unique values in fuelsystem: 8
Unique values in boreratio: 38
Unique values in stroke: 37
Unique values in compressionratio: 32
Unique values in horsepower: 59
Unique values in peakrpm: 23
Unique values in citympg: 29
```

[8]:
```python
# Drop car id
df.drop('car_ID', axis=1, inplace=True,errors='ignore')

df.head()
```

[8]:

	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	...	enginesize	fuelsystem	boreratio	strol
0	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	168.8	...	130	mpfi	3.47	2.(
1	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	168.8	...	130	mpfi	3.47	2.(
2	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	171.2	...	152	mpfi	2.68	3.4
3	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	176.6	...	109	mpfi	3.19	3.4
4	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	176.6	...	136	mpfi	3.19	3.4

5 rows × 25 columns

[9]:
```python
# identify categorical columns
categorical_cols=df.select_dtypes(include=["object","category"]).columns.tolist()
categorical_cols

for col in categorical_cols:
   unique_values = df[col].unique()
   print(f'Unique values in {col}: {unique_values}')
```

```
Unique values in CarName: ['alfa-romero giulia' 'alfa-romero stelvio' 'alfa-romero Quadrifoglio'
 'audi 100 ls' 'audi 100ls' 'audi fox' 'audi 5000' 'audi 4000'
 'audi 5000s (diesel)' 'bmw 320i' 'bmw x1' 'bmw x3' 'bmw z4' 'bmw x4'
 'bmw x5' 'chevrolet impala' 'chevrolet monte carlo' 'chevrolet vega 2300'
 'dodge rampage' 'dodge challenger se' 'dodge d200' 'dodge monaco (sw)'
 'dodge colt hardtop' 'dodge colt (sw)' 'dodge coronet custom'
 'dodge dart custom' 'dodge coronet custom (sw)' 'honda civic'
 'honda civic cvcc' 'honda accord cvcc' 'honda accord lx'
 'honda civic 1500 gl' 'honda accord' 'honda civic 1300' 'honda prelude'
 'honda civic (auto)' 'isuzu MU-X' 'isuzu D-Max ' 'isuzu D-Max V-Cross'
 'jaguar xj' 'jaguar xf' 'jaguar xk' 'maxda rx3' 'maxda glc deluxe'
 'mazda rx2 coupe' 'mazda rx-4' 'mazda glc deluxe' 'mazda 626' 'mazda glc'
 'mazda rx-7 gs' 'mazda glc 4' 'mazda glc custom l' 'mazda glc custom'
 'buick electra 225 custom' 'buick century luxus (sw)' 'buick century'
 'buick skyhawk' 'buick opel isuzu deluxe' 'buick skylark'
 'buick century special' 'buick regal sport coupe (turbo)'
 'mercury cougar' 'mitsubishi mirage' 'mitsubishi lancer'
 'mitsubishi outlander' 'mitsubishi g4' 'mitsubishi mirage g4'
 'mitsubishi montero' 'mitsubishi pajero' 'Nissan versa' 'nissan gt-r'
 'nissan rogue' 'nissan latio' 'nissan titan' 'nissan leaf' 'nissan juke'
 'nissan note' 'nissan clipper' 'nissan nv200' 'nissan dayz' 'nissan fuga'
 'nissan otti' 'nissan teana' 'nissan kicks' 'peugeot 504' 'peugeot 304'
 'peugeot 504 (sw)' 'peugeot 604sl' 'peugeot 505s turbo diesel'
 'plymouth fury iii' 'plymouth cricket' 'plymouth satellite custom (sw)'
 'plymouth fury gran sedan' 'plymouth valiant' 'plymouth duster'
 'porsche macan' 'porcshce panamera' 'porsche cayenne' 'porsche boxter'
 'renault 12tl' 'renault 5 gtl' 'saab 99e' 'saab 99le' 'saab 99gle'
 'subaru' 'subaru dl' 'subaru brz' 'subaru baja' 'subaru r1' 'subaru r2'
 'subaru trezia' 'subaru tribeca' 'toyota corona mark ii' 'toyota corona'
 'toyota corolla 1200' 'toyota corona hardtop' 'toyota corolla 1600 (sw)'
 'toyota carina' 'toyota mark ii' 'toyota corolla'
 'toyota corolla liftback' 'toyota celica gt liftback'
 'toyota corolla tercel' 'toyota corona liftback' 'toyota starlet'
 'toyota tercel' 'toyota cressida' 'toyota celica gt' 'toyouta tercel'
 'vokswagen rabbit' 'volkswagen 1131 deluxe sedan' 'volkswagen model 111'
 'volkswagen type 3' 'volkswagen 411 (sw)' 'volkswagen super beetle'
 'volkswagen dasher' 'vw dasher' 'vw rabbit' 'volkswagen rabbit'
 'volkswagen rabbit custom' 'volvo 145e (sw)' 'volvo 144ea' 'volvo 244dl'
 'volvo 245' 'volvo 264gl' 'volvo diesel' 'volvo 246']
```

```
Unique values in fueltype: ['gas' 'diesel']
Unique values in aspiration: ['std' 'turbo']
Unique values in doornumber: ['two' 'four']
Unique values in carbody: ['convertible' 'hatchback' 'sedan' 'wagon' 'hardtop']
Unique values in drivewheel: ['rwd' 'fwd' '4wd']
Unique values in enginelocation: ['front' 'rear']
Unique values in enginetype: ['dohc' 'ohcv' 'ohc' 'l' 'rotor' 'ohcf' 'dohcv']
Unique values in cylindernumber: ['four' 'six' 'five' 'three' 'twelve' 'two' 'eight']
Unique values in fuelsystem: ['mpfi' '2bbl' 'mfi' '1bbl' 'spfi' '4bbl' 'idi' 'spdi']
```

[10]:

```
print(len(df['CarName'].unique()))
df['CarName'].unique()
```

```
147
```

[10]:
```
array(['alfa-romero giulia', 'alfa-romero stelvio',
       'alfa-romero Quadrifoglio', 'audi 100 ls', 'audi 100ls',
       'audi fox', 'audi 5000', 'audi 4000', 'audi 5000s (diesel)',
       'bmw 320i', 'bmw x1', 'bmw x3', 'bmw z4', 'bmw x4', 'bmw x5',
       'chevrolet impala', 'chevrolet monte carlo', 'chevrolet vega 2300',
       'dodge rampage', 'dodge challenger se', 'dodge d200',
       'dodge monaco (sw)', 'dodge colt hardtop', 'dodge colt (sw)',
       'dodge coronet custom', 'dodge dart custom',
       'dodge coronet custom (sw)', 'honda civic', 'honda civic cvcc',
       'honda accord cvcc', 'honda accord lx', 'honda civic 1500 gl',
       'honda accord', 'honda civic 1300', 'honda prelude',
       'honda civic (auto)', 'isuzu MU-X', 'isuzu D-Max ',
       'isuzu D-Max V-Cross', 'jaguar xj', 'jaguar xf', 'jaguar xk',
       'maxda rx3', 'maxda glc deluxe', 'mazda rx2 coupe', 'mazda rx-4',
       'mazda glc deluxe', 'mazda 626', 'mazda glc', 'mazda rx-7 gs',
       'mazda glc 4', 'mazda glc custom l', 'mazda glc custom',
       'buick electra 225 custom', 'buick century luxus (sw)',
       'buick century', 'buick skyhawk', 'buick opel isuzu deluxe',
       'buick skylark', 'buick century special',
       'buick regal sport coupe (turbo)', 'mercury cougar',
       'mitsubishi mirage', 'mitsubishi lancer', 'mitsubishi outlander',
       'mitsubishi g4', 'mitsubishi mirage g4', 'mitsubishi montero',
       'mitsubishi pajero', 'Nissan versa', 'nissan gt-r', 'nissan rogue',
       'nissan latio', 'nissan titan', 'nissan leaf', 'nissan juke',
       'nissan note', 'nissan clipper', 'nissan nv200', 'nissan dayz',
       'nissan fuga', 'nissan otti', 'nissan teana', 'nissan kicks',
       'peugeot 504', 'peugeot 304', 'peugeot 504 (sw)', 'peugeot 604sl',
       'peugeot 505s turbo diesel', 'plymouth fury iii',
       'plymouth cricket', 'plymouth satellite custom (sw)',
       'plymouth fury gran sedan', 'plymouth valiant', 'plymouth duster',
       'porsche macan', 'porcshce panamera', 'porsche cayenne',
       'porsche boxter', 'renault 12tl', 'renault 5 gtl', 'saab 99e',
```

[11]:
```
#separate column for car company

df['Company'] = df.CarName.str.split(expand=True)[0]
```

[12]:
```
df['Company'].unique()
```

[12]:
```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'maxda', 'mazda', 'buick', 'mercury',
       'mitsubishi', 'Nissan', 'nissan', 'peugeot', 'plymouth', 'porsche',
       'porcshce', 'renault', 'saab', 'subaru', 'toyota', 'toyouta',
       'vokswagen', 'volkswagen', 'vw', 'volvo'], dtype=object)
```

[13]:
```
df['Company']=df.CarName.str.split(expand=True)[0]
df.head()
```

[13]:

	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	...	fuelsystem	boreratio	stroke	compres
0	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	168.8	...	mpfi	3.47	2.68	
1	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	168.8	...	mpfi	3.47	2.68	
2	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	171.2	...	mpfi	2.68	3.47	
3	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	176.6	...	mpfi	3.19	3.40	
4	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	176.6	...	mpfi	3.19	3.40	

5 rows × 26 columns

[14]:
```
# Replace misspelled company names with correct ones
df['Company'].replace(['maxda', 'nissan', 'porcshce', 'toyouta', 'vokswagen', 'vw'],
                      ['Mazda', 'Nissan', 'Porsche', 'Toyota', 'Volkswagen', 'Volkswagen'],
                      inplace=True)

# Display unique values in the 'Company' column to verify corrections
print(df['Company'].unique())
```

```
['alfa-romero' 'audi' 'bmw' 'chevrolet' 'dodge' 'honda' 'isuzu' 'jaguar'
 'Mazda' 'mazda' 'buick' 'mercury' 'mitsubishi' 'Nissan' 'peugeot'
 'plymouth' 'porsche' 'Porsche' 'renault' 'saab' 'subaru' 'toyota'
 'Toyota' 'Volkswagen' 'volkswagen' 'volvo']
```

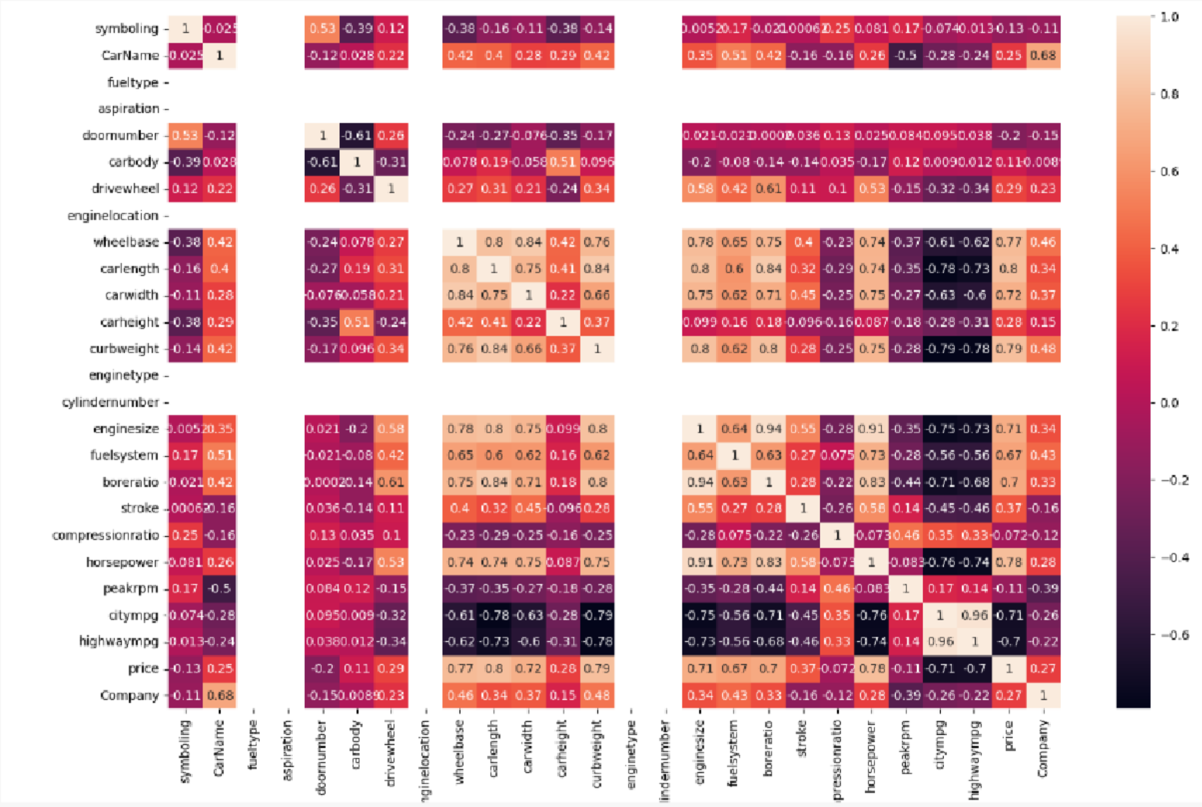
[15]:
```
# Label encoding all the categorical columns
label_encoder = LabelEncoder()
categorical_columns = df.select_dtypes(include=["object","category"]).columns
for col in categorical_columns:
    df[col] = label_encoder.fit_transform(df[col])

# Display the encoded DataFrame
print(df.head())
```

```
   symboling  CarName  fueltype  aspiration  doornumber  carbody  drivewheel  \
0          3        2         1           0           1        0           2
1          3        3         1           0           1        0           2
2          1        1         1           0           1        2           2
3          2        4         1           0           0        3           1
4          2        5         1           0           0        3           0

   enginelocation  wheelbase  carlength  ...  fuelsystem  boreratio  stroke  \
0               0       88.6      168.8  ...           5       3.47    2.68
1               0       88.6      168.8  ...           5       3.47    2.68
2               0       94.5      171.2  ...           5       2.68    3.47
3               0       99.8      176.6  ...           5       3.19    3.40
4               0       99.4      176.6  ...           5       3.19    3.40

   compressionratio  horsepower  peakrpm  citympg  highwaympg    price  \
0               9.0         111     5000       21          27  13495.0
1               9.0         111     5000       21          27  16500.0
2               9.0         154     5000       19          26  16500.0
3              10.0         102     5500       24          30  13950.0
4               8.0         115     5500       18          22  17450.0

   Company
0        5
1        5
2        5
3        6
4        6

[5 rows x 26 columns]
```

```
[16]: numeric_columns=df.select_dtypes(include=["int","float"]).columns
for col in numeric_columns:
    q1=df[col].quantile(0.25)
    q3=df[col].quantile(0.75)
    IQR=q3-q1

    lower_bound=q1-1.5*IQR
    upper_bound=q3+1.5*IQR

    df=df[(df[col]>=lower_bound)&(df[col]<=upper_bound)]

# Display the DataFrame after outlier removal

print("DataFrame after outlier removal:")

print(df.head())
```

```
DataFrame after outlier removal:
    symboling  CarName  fueltype  aspiration  doornumber  carbody  drivewheel  \
3           2        4         1           0           0        3           1
19          1       25         1           0           1        2           1
20          0       26         1           0           0        3           1
21          1       35         1           0           1        2           1
22          1       27         1           0           1        2           1

    enginelocation  wheelbase  carlength  ...  fuelsystem  boreratio  stroke  \
3                0       99.8      176.6  ...           5       3.19    3.40
19               0       94.5      155.9  ...           1       3.03    3.11
20               0       94.5      158.8  ...           1       3.03    3.11
21               0       93.7      157.3  ...           1       2.97    3.23
22               0       93.7      157.3  ...           1       2.97    3.23

    compressionratio  horsepower  peakrpm  citympg  highwaympg    price  \
3              10.00         102     5500       24          30  13950.0
19              9.60          70     5400       38          43   6295.0
20              9.60          70     5400       38          43   6575.0
21              9.41          68     5500       37          41   5572.0
22              9.40          68     5500       31          38   6377.0

    Company
3         6
19        9
20        9
21       10
22       10

[5 rows x 26 columns]
```

```
[17]: # Find correlation matrix

plt.figure(figsize=(16,10))
ax=sns.heatmap(df.corr(),annot=True)
plt.show()
```



[18]:

```
# Remove multicollinearity

correlation_matrix = df.corr().abs()


# Create a mask to identify highly correlated features
mask = (correlation_matrix > 0.85) & (correlation_matrix < 1.0)

# Drop the columns with high correlation
drop_columns = set()
for col in correlation_matrix.columns:
    correlated_features = correlation_matrix.index[mask[col]]
    drop_columns.update(correlated_features)

df_filtered = df.drop(columns=drop_columns)

# Display the DataFrame after removing highly correlated features
print("DataFrame after removing highly correlated features:")
print(df_filtered.head())
```

```
DataFrame after removing highly correlated features:
    symboling  CarName  fueltype  aspiration  doornumber  carbody  drivewheel  \
3           2        4         1           0           0        3           1
19          1       25         1           0           1        2           1
20          0       26         1           0           0        3           1
21          1       35         1           0           1        2           1
22          1       27         1           0           1        2           1

    enginelocation  wheelbase  carlength  ...  carheight  curbweight  \
3                0       99.8      176.6  ...       54.3        2337
19               0       94.5      155.9  ...       52.0        1874
20               0       94.5      158.8  ...       52.0        1909
21               0       93.7      157.3  ...       50.8        1876
22               0       93.7      157.3  ...       50.8        1876

    enginetype  cylindernumber  fuelsystem  stroke  compressionratio  peakrpm  \
3            3               2           5    3.40             10.00     5500
19           3               2           1    3.11              9.60     5400
20           3               2           1    3.11              9.60     5400
21           3               2           1    3.23              9.41     5500
22           3               2           1    3.23              9.40     5500

      price  Company
3   13950.0        6
19   6295.0        9
20   6575.0        9
21   5572.0       10
22   6377.0       10

[5 rows x 21 columns]
```

[19]:

```python
#  Data splitting

X = df.drop('price', axis=1)
y = df['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (68, 25)
Shape of X_test: (17, 25)
Shape of y_train: (68,)
Shape of y_test: (17,)
```

[20]:

```python
# Model selection and implementation

model= {
    'Linear Regression': LinearRegression()
}
for name, model in model.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f'{name}:')
    print(f'Mean Squared Error: {mse}')
    print(f'R-squared: {r2}')
    print()
```

```
Linear Regression:
Mean Squared Error: 1774475.4956749196
R-squared: 0.7140742568001623
```

ML - Task5

Classification and clustering

Problem Description

Use sklearn.datasets iris flower dataset to train your model using logistic regression. You need to figure out the accuracy of your model and use that to predict different samples in your test dataset. In iris dataset there are 150 samples containing following features,

1. Sepal Length
2. Sepal Width
3. Petal length
4. Petal width

Using above 4 features you will classify a flower in one of the three categories,

1. Setosa
2. Versicolour
3. Virginica

```
[9]: # import necessary libraries

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np
from sklearn.cluster import KMeans, AgglomerativeClustering
```

```
[10]: # Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset 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)

print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (120, 4)
Shape of X_test: (30, 4)
Shape of y_train: (120,)
Shape of y_test: (30,)
```

[11]:
```
#Train a logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

[11]:
```
LogisticRegression
LogisticRegression(max_iter=1000)
```

[12]:
```
# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, model.predict(X_test))
print("Accuracy of the logistic regression model:", accuracy)
```

```
Accuracy of the logistic regression model: 1.0
```

[13]:
```
# Step 5: Use the trained model to predict the categories of samples in the test dataset
sample1 = [[5.1, 3.5, 1.4, 0.2]]  # Sample iris data
sample2 = [[6.2, 2.8, 4.8, 1.8]]
sample3 = [[5.9, 3.0, 4.2, 1.5]]

predicted_category1 = model.predict(sample1)
predicted_category2 = model.predict(sample2)
predicted_category3 = model.predict(sample3)

print("Predicted category for sample 1:", iris.target_names[predicted_category1])
print("Predicted category for sample 2:", iris.target_names[predicted_category2])
print("Predicted category for sample 3:", iris.target_names[predicted_category3])
```

```
Predicted category for sample 1: ['setosa']
Predicted category for sample 2: ['virginica']
Predicted category for sample 3: ['versicolor']
```

[14]:
```
# Clustering

kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
kmeans_labels = kmeans.labels_
```

[16]:
```
# evaluating the clustering results
print("KMeans Clustering Accuracy:", accuracy_score(y, kmeans_labels))
```

```
KMeans Clustering Accuracy: 0.09333333333333334
```