Car Price Prediction using ANN

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
import warnings as w
import plotly.express as px
import plotly.graph_objects as go
w.filterwarnings("ignore")
plt.style.use('ggplot')
import warnings
warnings.filterwarnings("ignore")
# import tensorflow as tf
# print(tf.__version__)
# from tensorflow.keras.datasets import CarPricesData.csv
# from tensorflow.keras.models import Sequential
# from tensorflow.keras.layers import Dense, Dropout
# from sklearn.model_selection import train_test_split,GridSearchCV
# from sklearn.preprocessing import StandardScaler
# from sklearn.metrics import mean_squared_error
```

Loading Data

In [2]:
```
df = pd.read_csv(r'C:\Users\SachinR\Downloads\PersonalPy\Sia\DL\CarPricesData.csv')
```

Checking the number of rows & columns present in dataframe

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1436 entries, 0 to 1435
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Price      1436 non-null   int64
 1   Age        1434 non-null   float64
 2   KM         1436 non-null   int64
 3   FuelType   1432 non-null   object
 4   HP         1436 non-null   int64
 5   MetColor   1436 non-null   int64
 6   Automatic  1436 non-null   int64
 7   CC         1434 non-null   float64
 8   Doors      1436 non-null   int64
 9   Weight     1434 non-null   float64
dtypes: float64(3), int64(6), object(1)
memory usage: 112.3+ KB
```

Our dataset mostly consists of numerical columns.

Checking for missing values

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
Price        0
Age          2
KM           0
FuelType     4
HP           0
MetColor     0
Automatic    0
CC           2
Doors        0
Weight       2
dtype: int64
```

Null Values present in Age, FuelType, CC, and Weight

`Checking the number of rows & columns present in dataframe`

In [5]:
```
df.shape
```

Out[5]:
```
(1436, 10)
```

Our dataframe has 1436 rows with 10 attributes.

`Check the duplicate rows`

In [6]:
```
df[df.duplicated()].shape[0]
```

Out[6]:
```
1
```

Only one duplicate row in Dataframe

Drop `Duplicates` and Reset `Index`

In [7]:
```
df.drop_duplicates(keep='first', inplace=True)
df.reset_index(drop=True, inplace=True)
df.shape
```

Out[7]:
```
(1435, 10)
```

After droping 1 duplicate row, got shape of 1435 rows and 10 columns in our Dataframe.

`See top 5 rows`

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

Out[8]:

	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
0	13500	23.0	46986	Diesel	90	1	0	2000.0	3	1165.0
1	13750	23.0	72937	Diesel	90	1	0	2000.0	3	1165.0
2	13950	24.0	41711	Diesel	90	1	0	2000.0	3	1165.0
3	14950	26.0	48000	Diesel	90	0	0	2000.0	3	1165.0
4	13750	30.0	38500	Diesel	90	0	0	2000.0	3	1170.0

Basic Statistics of Numerical Columns

In [9]:
```
df.describe()
```

Out[9]:

	Price	Age	KM	HP	MetColor	Automatic	CC
count	1435.000000	1433.000000	1435.000000	1435.000000	1435.000000	1435.000000	1433.000000
mean	10720.915679	56.020237	68571.782578	101.491986	0.674564	0.055749	1566.688765
std	3608.732978	18.544948	37491.094553	14.981408	0.468701	0.229517	186.893360
min	4350.000000	1.000000	1.000000	69.000000	0.000000	0.000000	1300.000000
25%	8450.000000	44.000000	43000.000000	90.000000	0.000000	0.000000	1400.000000
50%	9900.000000	61.000000	63451.000000	110.000000	1.000000	0.000000	1600.000000
75%	11950.000000	70.000000	87041.500000	110.000000	1.000000	0.000000	1600.000000
max	32500.000000	80.000000	243000.000000	192.000000	1.000000	1.000000	2000.000000

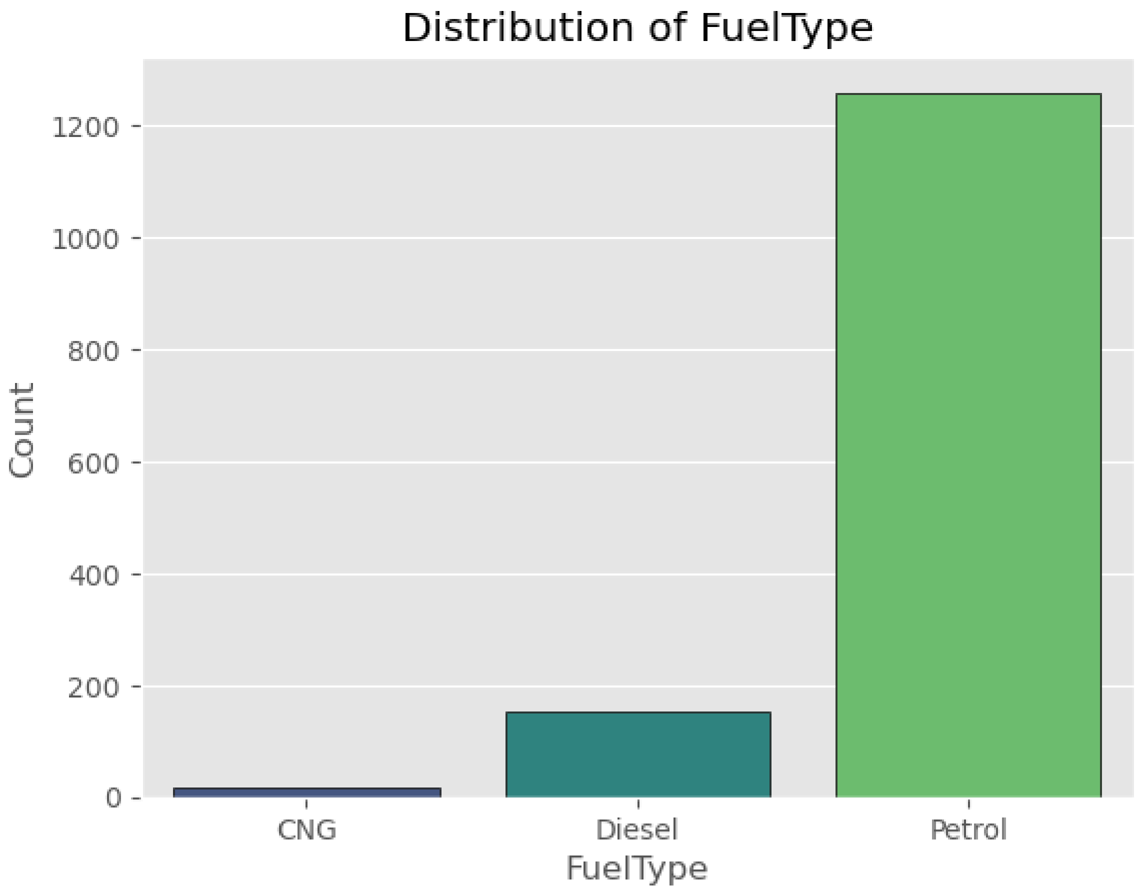
Looking at various statistics such as the mean, standard deviation, minimum, 25th percentile (Q1), median (50th percentile or Q2), 75th percentile (Q3), and maximum for each numerical column.

Visualizing distribution of all the Categorical Predictor variables in the data using bar plots

In [10]:
```
Categorical_columns = df.select_dtypes(include='O').columns
Categorical_columns
```

Out[10]:
```
Index(['FuelType'], dtype='object')
```

In [11]:
```
sns.barplot(x= Categorical_columns[0], y='Count', data=df.groupby(Categorical_columns[(
            palette='viridis', edgecolor = 'black')
plt.title(f'Distribution of {Categorical_columns[0]}')
plt.xlabel(Categorical_columns[0])
plt.show()
```



We only have FuelType column of category type. It has 3 unique values - `CNG`, `Diesel` and `Petrol`

`Visualize distribution of all the Continuous Predictor variables in the data using histograms`

In [12]:
```
continuous_columns = df.select_dtypes(include='number').columns
continuous_columns
```

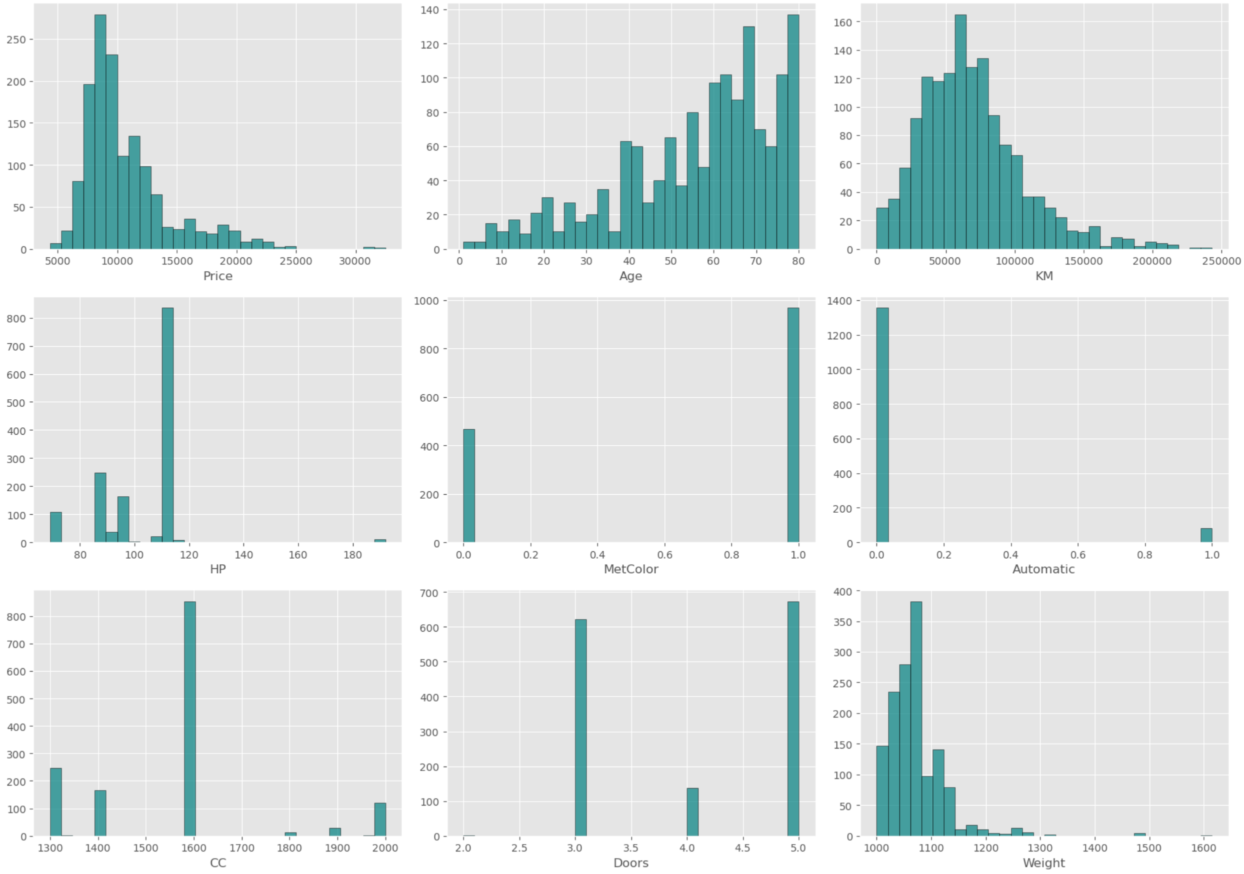
Out[12]:
```
Index(['Price', 'Age', 'KM', 'HP', 'MetColor', 'Automatic', 'CC', 'Doors',
       'Weight'],
      dtype='object')
```

In [13]:
```
plt.figure(figsize=(17, 12))

# Iterate through each continuous column and create a histogram
for i, col in enumerate(continuous_columns, 1):
    plt.subplot(3, 3, i)
    plt.hist(df[col], bins=30, color='teal', alpha=0.7, edgecolor='black')
    plt.xlabel(col)

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```



We see the distribution of numerical features in our dataframe

```
Find out the missing values
```

In [14]:
```
df.isnull().sum()
```

Out[14]:
```
Price        0
Age          2
KM           0
FuelType     4
HP           0
MetColor     0
Automatic    0
CC           2
Doors        0
Weight       2
dtype: int64
```

A few features have missing values. We will handle them in the following cells

```
Total Missing Values
```

In [15]:
```
df.isnull().sum().sum()
```

Out[15]:
```
10
```

```
Impute missing values with Median for Continuous values
```

In [16]:
```
null_continuous_cols =  ['Age','CC','Weight']
for i in null_continuous_cols:
    df[i] = df[i].fillna(df[i].median())
print(df.isnull().sum())
```

```
Price        0
Age          0
KM           0
FuelType     4
HP           0
MetColor     0
Automatic    0
CC           0
Doors        0
Weight       0
dtype: int64
```

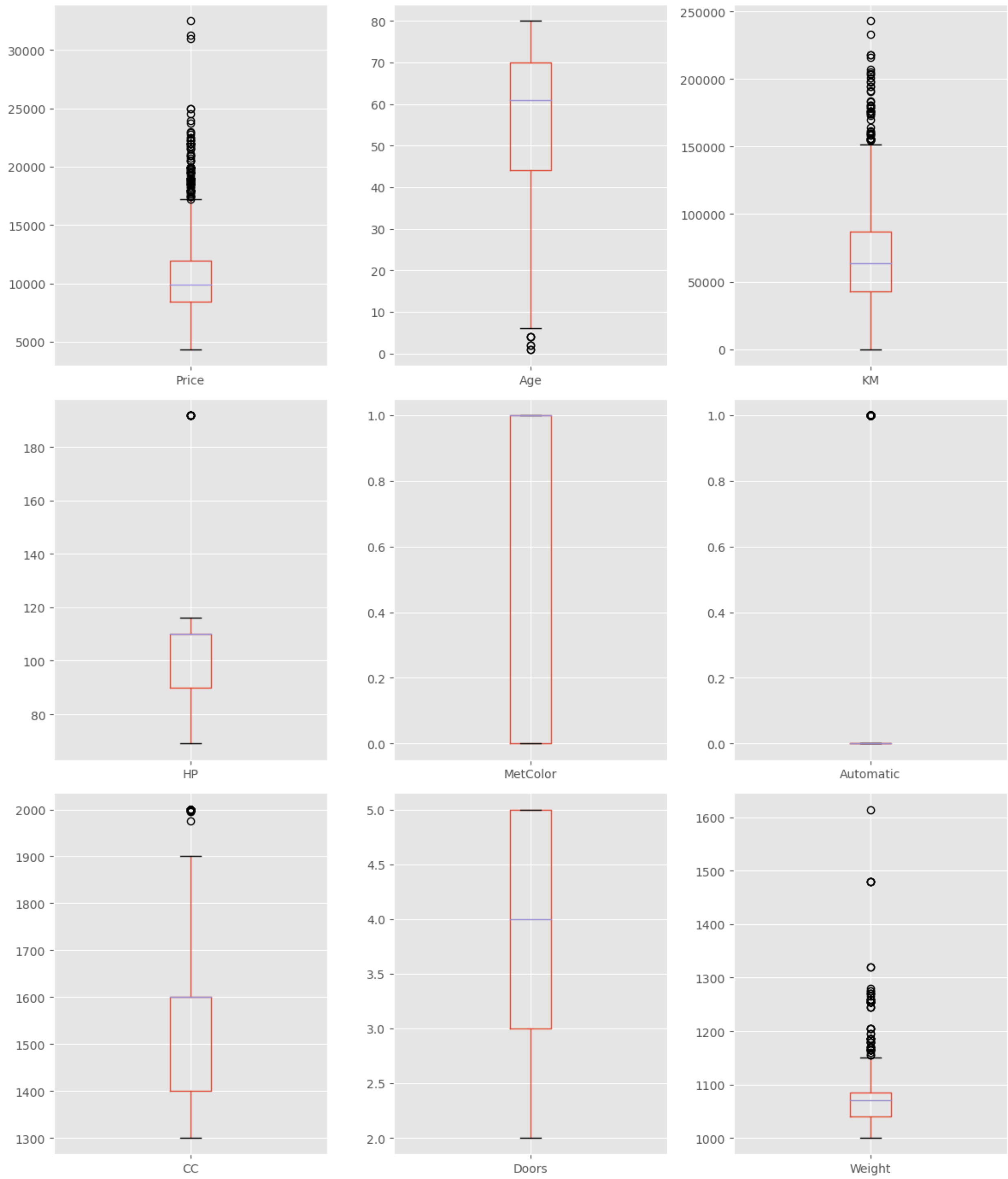
`Impute missing values with Mode for Category values`

In [17]:
```
null_category_cols =  ['FuelType']
for i in null_category_cols:
    df[i] = df[i].fillna(df[i].mode()[0])
print(df.isnull().sum())
```

```
Price        0
Age          0
KM           0
FuelType     0
HP           0
MetColor     0
Automatic    0
CC           0
Doors        0
Weight       0
dtype: int64
```

We have `eliminated` all `null` values from our dataset

In [18]:
```
# Set up subplots
fig, axes = plt.subplots(3, 3, figsize=(12, 14))
# Create boxplots for each numerical column
for ax, col in zip(axes.flatten(),continuous_columns):
    df.boxplot(column=col, ax=ax)
plt.tight_layout()
plt.show()
```



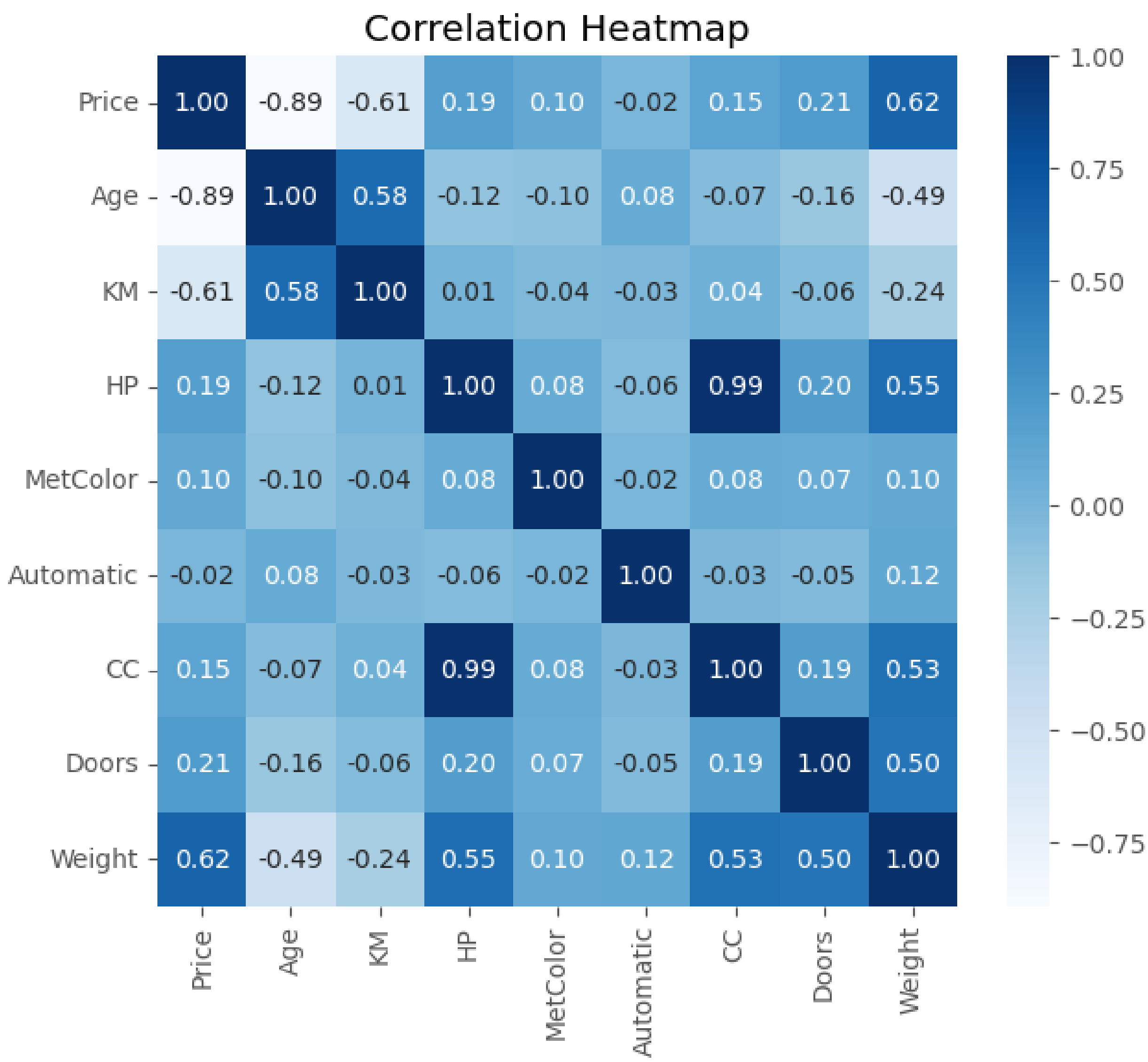
We notice that most of the numerical features have `outliers`

In [19]:
```
# Specify the IQR factor (you can adjust this based on your requirements)
iqr_factor = 1.5
outlier_columns = ['Age', 'KM', 'Weight','HP','CC']

# Remove outliers based on IQR directly
for col in outlier_columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - iqr_factor * IQR
    upper_bound = Q3 + iqr_factor * IQR
    df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
```

Removing Outliers from `Age` , `KM` , `Weight` , `HP` and `CC`

In [20]:
```
plt.figure(figsize=(7,6))
plt.title('Correlation Heatmap')
sns.heatmap(df[continuous_columns].corr(), annot=True,fmt = ".2f",cmap='Blues')
plt.show()
```



`Correlation` between all the numerical features in our dataset.

`Splitting data into features(x) and target(y)`

In [21]:
```
x = df.iloc[:,1:]
y = df['Price']
```

In [22]:
```
x.head(2)
```

Out[22]:

	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
17	24.0	21716	Petrol	110	1	0	1600.0	3	1105.0
18	24.0	25563	Petrol	110	0	0	1600.0	3	1065.0

In [23]:
```
df['FuelType'].unique()
```

Out[23]:
```
array(['Petrol', 'CNG'], dtype=object)
```

In [24]:
```
# One-Hot encoding
x = pd.get_dummies(x, columns=['FuelType'], prefix='FuelType', drop_first=True)
```

In [25]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x = sc.fit_transform(x)
```

In [26]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.20,random_state=1)
```

In [27]:
```
xtrain.shape
```

Out[27]:
```
(992, 9)
```

In [28]:
```
ytrain.shape
```

Out[28]:
```
(992,)
```

In [29]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import mean_squared_error
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import ParameterGrid
from tensorflow.keras.callbacks import EarlyStopping
```

```
WARNING:tensorflow:From C:\Users\SachinR\anaconda3\Lib\site-packages\keras\src\losses.
py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.
compat.v1.losses.sparse_softmax_cross_entropy instead.
```

Creating our ANN

In [30]:
```
model = Sequential([
    Dense(256, activation='relu', input_shape=(xtrain.shape[1],)),
    Dropout(0.3),
    Dense(128, activation='relu'),Dropout(0.3),
    Dense(64, activation = 'relu'),Dropout(0.3),
    Dense(32, activation ='relu'), Dropout(0.3),
    Dense(1)]) #Output layer for regression, no activation function
```

```
WARNING:tensorflow:From C:\Users\SachinR\anaconda3\Lib\site-packages\keras\src\backen
d.py:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_def
ault_graph instead.
```

In [31]:
```
#Compile the model
model.compile(optimizer='adam', loss='mean_absolute_error',metrics=['mae'])

#train the model
model.fit(xtrain, ytrain, epochs=100, batch_size=32, validation_split=0.2)

#Evalute the model
loss, mae = model.evaluate(xtest, ytest)
print(f"Mean Absolute Error on Test Date: {mae}")
```

```
WARNING:tensorflow:From C:\Users\SachinR\anaconda3\Lib\site-packages\keras\src\optimiz
ers\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v
1.train.Optimizer instead.

Epoch 1/100
WARNING:tensorflow:From C:\Users\SachinR\anaconda3\Lib\site-packages\keras\src\utils\t
f_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.comp
at.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\SachinR\anaconda3\Lib\site-packages\keras\src\engine
\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecate
d. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

25/25 [==============================] - 2s 15ms/step - loss: 10499.1377 - mae: 10499.
1377 - val_loss: 10270.2266 - val_mae: 10270.2266
Epoch 2/100
25/25 [==============================] - 0s 5ms/step - loss: 10460.8232 - mae: 10460.8
232 - val_loss: 10164.1602 - val_mae: 10164.1602
Epoch 3/100
25/25 [==============================] - 0s 5ms/step - loss: 10086.1074 - mae: 10086.1
074 - val_loss: 9307.8584 - val_mae: 9307.8584
Epoch 4/100
25/25 [==============================] - 0s 5ms/step - loss: 7905.8218 - mae: 7905.821
8 - val_loss: 5284.6353 - val_mae: 5284.6353
Epoch 5/100
25/25 [==============================] - 0s 6ms/step - loss: 3681.4255 - mae: 3681.425
5 - val_loss: 1842.1233 - val_mae: 1842.1233
Epoch 6/100
25/25 [==============================] - 0s 5ms/step - loss: 2615.3940 - mae: 2615.394
0 - val_loss: 1343.3903 - val_mae: 1343.3903
Epoch 7/100
25/25 [==============================] - 0s 5ms/step - loss: 2320.3457 - mae: 2320.345
7 - val_loss: 1173.7191 - val_mae: 1173.7191
Epoch 8/100
25/25 [==============================] - 0s 5ms/step - loss: 2301.9592 - mae: 2301.959
2 - val_loss: 1149.8958 - val_mae: 1149.8958
Epoch 9/100
25/25 [==============================] - 0s 5ms/step - loss: 2208.3123 - mae: 2208.312
3 - val_loss: 1052.9977 - val_mae: 1052.9977
Epoch 10/100
25/25 [==============================] - 0s 5ms/step - loss: 2197.5447 - mae: 2197.544
7 - val_loss: 1033.3560 - val_mae: 1033.3560
Epoch 11/100
25/25 [==============================] - 0s 5ms/step - loss: 2011.8353 - mae: 2011.835
3 - val_loss: 999.7230 - val_mae: 999.7230
Epoch 12/100
25/25 [==============================] - 0s 5ms/step - loss: 2175.4763 - mae: 2175.476
3 - val_loss: 1012.6705 - val_mae: 1012.6705
Epoch 13/100
25/25 [==============================] - 0s 5ms/step - loss: 2096.4419 - mae: 2096.441
9 - val_loss: 948.7883 - val_mae: 948.7883
Epoch 14/100
25/25 [==============================] - 0s 5ms/step - loss: 2034.9994 - mae: 2034.999
4 - val_loss: 996.5630 - val_mae: 996.5630
Epoch 15/100
25/25 [==============================] - 0s 5ms/step - loss: 2004.9912 - mae: 2004.991
2 - val_loss: 1154.3284 - val_mae: 1154.3284
Epoch 16/100
25/25 [==============================] - 0s 5ms/step - loss: 2014.2561 - mae: 2014.256
1 - val_loss: 972.8232 - val_mae: 972.8232
Epoch 17/100
25/25 [==============================] - 0s 5ms/step - loss: 2035.0599 - mae: 2035.059
9 - val_loss: 943.3498 - val_mae: 943.3498
Epoch 18/100
```

```
25/25 [==============================] - 0s 5ms/step - loss: 1970.4459 - mae: 1970.445
9 - val_loss: 1092.8922 - val_mae: 1092.8922
Epoch 19/100
25/25 [==============================] - 0s 5ms/step - loss: 1957.3339 - mae: 1957.333
9 - val_loss: 895.6135 - val_mae: 895.6135
Epoch 20/100
25/25 [==============================] - 0s 5ms/step - loss: 1948.6685 - mae: 1948.668
5 - val_loss: 904.8314 - val_mae: 904.8314
Epoch 21/100
25/25 [==============================] - 0s 5ms/step - loss: 1924.9805 - mae: 1924.980
5 - val_loss: 1029.1185 - val_mae: 1029.1185
Epoch 22/100
25/25 [==============================] - 0s 5ms/step - loss: 1980.3384 - mae: 1980.338
4 - val_loss: 966.0200 - val_mae: 966.0200
Epoch 23/100
25/25 [==============================] - 0s 5ms/step - loss: 1841.5446 - mae: 1841.544
6 - val_loss: 834.5682 - val_mae: 834.5682
Epoch 24/100
25/25 [==============================] - 0s 5ms/step - loss: 1915.3438 - mae: 1915.343
8 - val_loss: 993.1628 - val_mae: 993.1628
Epoch 25/100
25/25 [==============================] - 0s 5ms/step - loss: 2014.9187 - mae: 2014.918
7 - val_loss: 827.0905 - val_mae: 827.0905
Epoch 26/100
25/25 [==============================] - 0s 5ms/step - loss: 2018.0131 - mae: 2018.013
1 - val_loss: 846.4419 - val_mae: 846.4419
Epoch 27/100
25/25 [==============================] - 0s 5ms/step - loss: 1992.6108 - mae: 1992.610
8 - val_loss: 906.1006 - val_mae: 906.1006
Epoch 28/100
25/25 [==============================] - 0s 5ms/step - loss: 1862.5372 - mae: 1862.537
2 - val_loss: 908.9328 - val_mae: 908.9328
Epoch 29/100
25/25 [==============================] - 0s 5ms/step - loss: 1917.9875 - mae: 1917.987
5 - val_loss: 822.9360 - val_mae: 822.9360
Epoch 30/100
25/25 [==============================] - 0s 6ms/step - loss: 1854.5618 - mae: 1854.561
8 - val_loss: 798.5382 - val_mae: 798.5382
Epoch 31/100
25/25 [==============================] - 0s 5ms/step - loss: 1890.8124 - mae: 1890.812
4 - val_loss: 940.5406 - val_mae: 940.5406
Epoch 32/100
25/25 [==============================] - 0s 6ms/step - loss: 1822.4095 - mae: 1822.409
5 - val_loss: 839.9306 - val_mae: 839.9306
Epoch 33/100
25/25 [==============================] - 0s 5ms/step - loss: 1944.6084 - mae: 1944.608
4 - val_loss: 864.4077 - val_mae: 864.4077
Epoch 34/100
25/25 [==============================] - 0s 5ms/step - loss: 1879.0433 - mae: 1879.043
3 - val_loss: 798.6097 - val_mae: 798.6097
Epoch 35/100
25/25 [==============================] - 0s 5ms/step - loss: 1953.7437 - mae: 1953.743
7 - val_loss: 950.2793 - val_mae: 950.2793
Epoch 36/100
25/25 [==============================] - 0s 5ms/step - loss: 1829.2804 - mae: 1829.280
4 - val_loss: 882.8777 - val_mae: 882.8777
Epoch 37/100
25/25 [==============================] - 0s 5ms/step - loss: 1949.1818 - mae: 1949.181
8 - val_loss: 812.7892 - val_mae: 812.7892
Epoch 38/100
25/25 [==============================] - 0s 5ms/step - loss: 1936.0090 - mae: 1936.009
0 - val_loss: 868.9547 - val_mae: 868.9547
Epoch 39/100
25/25 [==============================] - 0s 4ms/step - loss: 1892.2592 - mae: 1892.259
```

```
2 - val_loss: 827.8629 - val_mae: 827.8629
Epoch 40/100
25/25 [==============================] - 0s 4ms/step - loss: 1902.0096 - mae: 1902.009
6 - val_loss: 883.9951 - val_mae: 883.9951
Epoch 41/100
25/25 [==============================] - 0s 5ms/step - loss: 1935.4279 - mae: 1935.427
9 - val_loss: 794.2782 - val_mae: 794.2782
Epoch 42/100
25/25 [==============================] - 0s 5ms/step - loss: 1885.4009 - mae: 1885.400
9 - val_loss: 855.1415 - val_mae: 855.1415
Epoch 43/100
25/25 [==============================] - 0s 4ms/step - loss: 1938.9952 - mae: 1938.995
2 - val_loss: 805.9748 - val_mae: 805.9748
Epoch 44/100
25/25 [==============================] - 0s 4ms/step - loss: 1874.2606 - mae: 1874.260
6 - val_loss: 899.0693 - val_mae: 899.0693
Epoch 45/100
25/25 [==============================] - 0s 5ms/step - loss: 1877.4956 - mae: 1877.495
6 - val_loss: 795.3776 - val_mae: 795.3776
Epoch 46/100
25/25 [==============================] - 0s 5ms/step - loss: 1882.6122 - mae: 1882.612
2 - val_loss: 811.5806 - val_mae: 811.5806
Epoch 47/100
25/25 [==============================] - 0s 5ms/step - loss: 1855.2283 - mae: 1855.228
3 - val_loss: 878.4367 - val_mae: 878.4367
Epoch 48/100
25/25 [==============================] - 0s 5ms/step - loss: 1868.5397 - mae: 1868.539
7 - val_loss: 872.4690 - val_mae: 872.4690
Epoch 49/100
25/25 [==============================] - 0s 5ms/step - loss: 1892.4016 - mae: 1892.401
6 - val_loss: 803.4408 - val_mae: 803.4408
Epoch 50/100
25/25 [==============================] - 0s 5ms/step - loss: 1825.7954 - mae: 1825.795
4 - val_loss: 927.1110 - val_mae: 927.1110
Epoch 51/100
25/25 [==============================] - 0s 4ms/step - loss: 1855.3641 - mae: 1855.364
1 - val_loss: 790.5375 - val_mae: 790.5375
Epoch 52/100
25/25 [==============================] - 0s 5ms/step - loss: 1853.5581 - mae: 1853.558
1 - val_loss: 772.9534 - val_mae: 772.9534
Epoch 53/100
25/25 [==============================] - 0s 4ms/step - loss: 1867.9194 - mae: 1867.919
4 - val_loss: 845.9689 - val_mae: 845.9689
Epoch 54/100
25/25 [==============================] - 0s 4ms/step - loss: 1850.9768 - mae: 1850.976
8 - val_loss: 805.7373 - val_mae: 805.7373
Epoch 55/100
25/25 [==============================] - 0s 4ms/step - loss: 1824.7787 - mae: 1824.778
7 - val_loss: 787.8149 - val_mae: 787.8149
Epoch 56/100
25/25 [==============================] - 0s 4ms/step - loss: 1857.9196 - mae: 1857.919
6 - val_loss: 803.2231 - val_mae: 803.2231
Epoch 57/100
25/25 [==============================] - 0s 4ms/step - loss: 1780.8096 - mae: 1780.809
6 - val_loss: 827.5637 - val_mae: 827.5637
Epoch 58/100
25/25 [==============================] - 0s 4ms/step - loss: 1833.2412 - mae: 1833.241
2 - val_loss: 951.9610 - val_mae: 951.9610
Epoch 59/100
25/25 [==============================] - 0s 4ms/step - loss: 1937.2950 - mae: 1937.295
0 - val_loss: 790.4266 - val_mae: 790.4266
Epoch 60/100
25/25 [==============================] - 0s 5ms/step - loss: 1783.0157 - mae: 1783.015
7 - val_loss: 789.2697 - val_mae: 789.2697
```

```
Epoch 61/100
25/25 [==============================] - 0s 4ms/step - loss: 1855.4619 - mae: 1855.461
9 - val_loss: 783.6924 - val_mae: 783.6924
Epoch 62/100
25/25 [==============================] - 0s 4ms/step - loss: 1929.5344 - mae: 1929.534
4 - val_loss: 772.5217 - val_mae: 772.5217
Epoch 63/100
25/25 [==============================] - 0s 4ms/step - loss: 1938.6144 - mae: 1938.614
4 - val_loss: 847.9369 - val_mae: 847.9369
Epoch 64/100
25/25 [==============================] - 0s 5ms/step - loss: 1824.3882 - mae: 1824.388
2 - val_loss: 820.5981 - val_mae: 820.5981
Epoch 65/100
25/25 [==============================] - 0s 4ms/step - loss: 1857.7463 - mae: 1857.746
3 - val_loss: 786.4380 - val_mae: 786.4380
Epoch 66/100
25/25 [==============================] - 0s 5ms/step - loss: 1838.5026 - mae: 1838.502
6 - val_loss: 803.4361 - val_mae: 803.4361
Epoch 67/100
25/25 [==============================] - 0s 6ms/step - loss: 1759.4623 - mae: 1759.462
3 - val_loss: 794.8010 - val_mae: 794.8010
Epoch 68/100
25/25 [==============================] - 0s 4ms/step - loss: 1869.9561 - mae: 1869.956
1 - val_loss: 860.7296 - val_mae: 860.7296
Epoch 69/100
25/25 [==============================] - 0s 4ms/step - loss: 1926.9249 - mae: 1926.924
9 - val_loss: 816.3962 - val_mae: 816.3962
Epoch 70/100
25/25 [==============================] - 0s 4ms/step - loss: 1771.9452 - mae: 1771.945
2 - val_loss: 769.8920 - val_mae: 769.8920
Epoch 71/100
25/25 [==============================] - 0s 4ms/step - loss: 1822.8070 - mae: 1822.807
0 - val_loss: 840.2377 - val_mae: 840.2377
Epoch 72/100
25/25 [==============================] - 0s 5ms/step - loss: 1926.0905 - mae: 1926.090
5 - val_loss: 790.3979 - val_mae: 790.3979
Epoch 73/100
25/25 [==============================] - 0s 5ms/step - loss: 1857.5217 - mae: 1857.521
7 - val_loss: 785.9270 - val_mae: 785.9270
Epoch 74/100
25/25 [==============================] - 0s 5ms/step - loss: 1859.4327 - mae: 1859.432
7 - val_loss: 772.3713 - val_mae: 772.3713
Epoch 75/100
25/25 [==============================] - 0s 5ms/step - loss: 1855.1957 - mae: 1855.195
7 - val_loss: 873.5983 - val_mae: 873.5983
Epoch 76/100
25/25 [==============================] - 0s 5ms/step - loss: 1914.4065 - mae: 1914.406
5 - val_loss: 813.0200 - val_mae: 813.0200
Epoch 77/100
25/25 [==============================] - 0s 4ms/step - loss: 1906.6698 - mae: 1906.669
8 - val_loss: 825.4290 - val_mae: 825.4290
Epoch 78/100
25/25 [==============================] - 0s 4ms/step - loss: 1818.4430 - mae: 1818.443
0 - val_loss: 773.0616 - val_mae: 773.0616
Epoch 79/100
25/25 [==============================] - 0s 4ms/step - loss: 1934.2328 - mae: 1934.232
8 - val_loss: 793.0047 - val_mae: 793.0047
Epoch 80/100
25/25 [==============================] - 0s 5ms/step - loss: 1870.3167 - mae: 1870.316
7 - val_loss: 820.3763 - val_mae: 820.3763
Epoch 81/100
25/25 [==============================] - 0s 4ms/step - loss: 1870.4983 - mae: 1870.498
3 - val_loss: 777.1232 - val_mae: 777.1232
Epoch 82/100
```

```
25/25 [==============================] - 0s 4ms/step - loss: 1823.4600 - mae: 1823.460
0 - val_loss: 784.5480 - val_mae: 784.5480
Epoch 83/100
25/25 [==============================] - 0s 5ms/step - loss: 1915.8079 - mae: 1915.807
9 - val_loss: 935.5740 - val_mae: 935.5740
Epoch 84/100
25/25 [==============================] - 0s 4ms/step - loss: 1777.8243 - mae: 1777.824
3 - val_loss: 805.9562 - val_mae: 805.9562
Epoch 85/100
25/25 [==============================] - 0s 4ms/step - loss: 1847.4344 - mae: 1847.434
4 - val_loss: 773.5948 - val_mae: 773.5948
Epoch 86/100
25/25 [==============================] - 0s 5ms/step - loss: 1850.2358 - mae: 1850.235
8 - val_loss: 782.7757 - val_mae: 782.7757
Epoch 87/100
25/25 [==============================] - 0s 4ms/step - loss: 1862.0864 - mae: 1862.086
4 - val_loss: 825.6231 - val_mae: 825.6231
Epoch 88/100
25/25 [==============================] - 0s 4ms/step - loss: 1804.1483 - mae: 1804.148
3 - val_loss: 817.9901 - val_mae: 817.9901
Epoch 89/100
25/25 [==============================] - 0s 5ms/step - loss: 1709.7501 - mae: 1709.750
1 - val_loss: 824.6096 - val_mae: 824.6096
Epoch 90/100
25/25 [==============================] - 0s 4ms/step - loss: 1940.6636 - mae: 1940.663
6 - val_loss: 793.1751 - val_mae: 793.1751
Epoch 91/100
25/25 [==============================] - 0s 5ms/step - loss: 1833.6851 - mae: 1833.685
1 - val_loss: 831.2734 - val_mae: 831.2734
Epoch 92/100
25/25 [==============================] - 0s 4ms/step - loss: 1727.8394 - mae: 1727.839
4 - val_loss: 789.8121 - val_mae: 789.8121
Epoch 93/100
25/25 [==============================] - 0s 5ms/step - loss: 1838.6135 - mae: 1838.613
5 - val_loss: 809.5690 - val_mae: 809.5690
Epoch 94/100
25/25 [==============================] - 0s 5ms/step - loss: 1837.0212 - mae: 1837.021
2 - val_loss: 784.2018 - val_mae: 784.2018
Epoch 95/100
25/25 [==============================] - 0s 5ms/step - loss: 1723.8143 - mae: 1723.814
3 - val_loss: 808.0723 - val_mae: 808.0723
Epoch 96/100
25/25 [==============================] - 0s 4ms/step - loss: 1826.4724 - mae: 1826.472
4 - val_loss: 790.4147 - val_mae: 790.4147
Epoch 97/100
25/25 [==============================] - 0s 4ms/step - loss: 1793.6920 - mae: 1793.692
0 - val_loss: 776.8183 - val_mae: 776.8183
Epoch 98/100
25/25 [==============================] - 0s 4ms/step - loss: 1849.4338 - mae: 1849.433
8 - val_loss: 818.0709 - val_mae: 818.0709
Epoch 99/100
25/25 [==============================] - 0s 4ms/step - loss: 1952.7972 - mae: 1952.797
2 - val_loss: 806.0855 - val_mae: 806.0855
Epoch 100/100
25/25 [==============================] - 0s 4ms/step - loss: 1883.3889 - mae: 1883.388
9 - val_loss: 763.9405 - val_mae: 763.9405
8/8 [==============================] - 0s 2ms/step - loss: 812.9025 - mae: 812.9025
Mean Absolute Error on Test Date: 812.9024658203125
```

Implementing Early Stopping

In [32]:
```
model = Sequential([
    Dense(256, activation='relu', input_shape=(xtrain.shape[1],)),
    Dropout(0.3),
```

```
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1)
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Define early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=Tr

# Train the model with early stopping
history = model.fit(xtrain, ytrain, epochs=100, batch_size=32, validation_split=0.2, ca

# Evaluate the model
loss, mae = model.evaluate(xtest, ytest, verbose=0)
print(f"Mean Absolute Error on Test Data: {mae}")
```

```
Epoch 1/100
25/25 [==============================] - 1s 11ms/step - loss: 119411160.0000 - mae: 10
497.6074 - val_loss: 112621328.0000 - val_mae: 10265.6475
Epoch 2/100
25/25 [==============================] - 0s 5ms/step - loss: 118215464.0000 - mae: 104
40.8604 - val_loss: 109409304.0000 - val_mae: 10112.7393
Epoch 3/100
25/25 [==============================] - 0s 5ms/step - loss: 107511280.0000 - mae: 993
0.5820 - val_loss: 86712496.0000 - val_mae: 8966.1484
Epoch 4/100
25/25 [==============================] - 0s 5ms/step - loss: 59412600.0000 - mae: 713
7.0254 - val_loss: 19159158.0000 - val_mae: 3950.4231
Epoch 5/100
25/25 [==============================] - 0s 5ms/step - loss: 12510900.0000 - mae: 281
7.0608 - val_loss: 2965037.2500 - val_mae: 1347.1959
Epoch 6/100
25/25 [==============================] - 0s 5ms/step - loss: 9240034.0000 - mae: 2382.
0676 - val_loss: 2797272.5000 - val_mae: 1310.4062
Epoch 7/100
25/25 [==============================] - 0s 5ms/step - loss: 7977256.5000 - mae: 2204.
1145 - val_loss: 2473574.7500 - val_mae: 1225.9386
Epoch 8/100
25/25 [==============================] - 0s 5ms/step - loss: 7413624.5000 - mae: 2120.
2783 - val_loss: 1836608.5000 - val_mae: 1034.8699
Epoch 9/100
25/25 [==============================] - 0s 5ms/step - loss: 7141352.0000 - mae: 2074.
6592 - val_loss: 1864341.3750 - val_mae: 1042.1313
Epoch 10/100
25/25 [==============================] - 0s 6ms/step - loss: 7330045.0000 - mae: 2079.
2246 - val_loss: 1926362.3750 - val_mae: 1055.9165
Epoch 11/100
25/25 [==============================] - 0s 5ms/step - loss: 6773057.5000 - mae: 2042.
1957 - val_loss: 1615634.3750 - val_mae: 955.8372
Epoch 12/100
25/25 [==============================] - 0s 5ms/step - loss: 6673770.0000 - mae: 2001.
5968 - val_loss: 1748735.0000 - val_mae: 996.9880
Epoch 13/100
25/25 [==============================] - 0s 5ms/step - loss: 6294934.0000 - mae: 1982.
6910 - val_loss: 1881274.3750 - val_mae: 1040.7844
Epoch 14/100
25/25 [==============================] - 0s 5ms/step - loss: 7247609.0000 - mae: 2104.
0312 - val_loss: 1497985.5000 - val_mae: 910.5665
Epoch 15/100
25/25 [==============================] - 0s 5ms/step - loss: 6759858.0000 - mae: 2021.
7109 - val_loss: 1461332.1250 - val_mae: 897.9520
Epoch 16/100
25/25 [==============================] - 0s 5ms/step - loss: 6405313.0000 - mae: 1946.
2003 - val_loss: 1395299.7500 - val_mae: 874.5497
Epoch 17/100
25/25 [==============================] - 0s 5ms/step - loss: 6537626.5000 - mae: 1955.
8792 - val_loss: 1489656.5000 - val_mae: 906.5704
Epoch 18/100
25/25 [==============================] - 0s 5ms/step - loss: 6241896.5000 - mae: 1965.
3885 - val_loss: 1378909.2500 - val_mae: 868.6577
Epoch 19/100
25/25 [==============================] - 0s 5ms/step - loss: 6688046.0000 - mae: 2016.
7412 - val_loss: 1466893.3750 - val_mae: 895.2790
Epoch 20/100
25/25 [==============================] - 0s 5ms/step - loss: 5944205.0000 - mae: 1891.
0287 - val_loss: 1463588.6250 - val_mae: 903.2840
Epoch 21/100
25/25 [==============================] - 0s 5ms/step - loss: 5644897.5000 - mae: 1859.
7085 - val_loss: 1433671.1250 - val_mae: 890.8448
Epoch 22/100
```

```
25/25 [==============================] - 0s 5ms/step - loss: 6051181.0000 - mae: 1900.
2448 - val_loss: 1610065.2500 - val_mae: 960.3814
Epoch 23/100
25/25 [==============================] - 0s 5ms/step - loss: 7197784.0000 - mae: 2068.
7173 - val_loss: 1380524.3750 - val_mae: 858.2697
Epoch 24/100
25/25 [==============================] - 0s 5ms/step - loss: 5594739.5000 - mae: 1875.
8280 - val_loss: 1353516.3750 - val_mae: 851.9299
Epoch 25/100
25/25 [==============================] - 0s 5ms/step - loss: 6293851.5000 - mae: 1962.
8917 - val_loss: 1513692.2500 - val_mae: 921.6259
Epoch 26/100
25/25 [==============================] - 0s 5ms/step - loss: 6120734.0000 - mae: 1893.
3400 - val_loss: 1310522.5000 - val_mae: 820.0967
Epoch 27/100
25/25 [==============================] - 0s 5ms/step - loss: 5665933.0000 - mae: 1877.
7430 - val_loss: 1249532.5000 - val_mae: 800.9068
Epoch 28/100
25/25 [==============================] - 0s 5ms/step - loss: 6196869.5000 - mae: 1913.
6704 - val_loss: 1344695.2500 - val_mae: 846.8625
Epoch 29/100
25/25 [==============================] - 0s 4ms/step - loss: 5783131.5000 - mae: 1843.
9355 - val_loss: 1418655.6250 - val_mae: 870.4410
Epoch 30/100
25/25 [==============================] - 0s 5ms/step - loss: 5993377.0000 - mae: 1906.
0834 - val_loss: 1603192.2500 - val_mae: 932.5195
Epoch 31/100
25/25 [==============================] - 0s 4ms/step - loss: 5973854.5000 - mae: 1868.
3126 - val_loss: 1280355.2500 - val_mae: 816.7831
Epoch 32/100
25/25 [==============================] - 0s 5ms/step - loss: 5865700.5000 - mae: 1914.
9628 - val_loss: 1252248.8750 - val_mae: 815.9484
Epoch 33/100
25/25 [==============================] - 0s 5ms/step - loss: 5971939.5000 - mae: 1900.
5759 - val_loss: 1273958.3750 - val_mae: 823.6008
Epoch 34/100
25/25 [==============================] - 0s 5ms/step - loss: 5647044.5000 - mae: 1835.
9297 - val_loss: 1345138.5000 - val_mae: 851.7051
Epoch 35/100
25/25 [==============================] - 0s 5ms/step - loss: 5808274.5000 - mae: 1840.
5797 - val_loss: 1529106.1250 - val_mae: 917.8580
Epoch 36/100
25/25 [==============================] - 0s 5ms/step - loss: 6347353.0000 - mae: 1928.
9335 - val_loss: 1450463.0000 - val_mae: 876.1678
Epoch 37/100
25/25 [==============================] - 0s 5ms/step - loss: 5709740.5000 - mae: 1903.
0568 - val_loss: 1265928.5000 - val_mae: 823.2996
Mean Absolute Error on Test Data: 862.5036010742188
```

`Hyperparameter Tuning` our ANN

In [33]:
```python
input_dim = xtrain.shape[1]

def create_model(num_layers=1, neurons_first_layer=64, activation='relu', output_activa
                 dropout_rate=0.2, learning_rate=0.001):
    model = Sequential()
    model.add(Dense(neurons_first_layer, activation=activation, input_shape=(input_dim

    for i in range(num_layers - 1):
        model.add(Dense(neurons_first_layer // 2, activation=activation))
        model.add(Dropout(dropout_rate))

    model.add(Dense(1, activation=output_activation))
```

```python
    model.compile(optimizer=Adam(learning_rate=learning_rate), loss='mean_absolute_erro

    return model

# Define the parameter grid
param_grid = {
    'num_layers': [1, 2, 3],
    'neurons_first_layer': [64, 128],
    'activation': ['relu'],
    'output_activation': ['linear'],
    'dropout_rate': [0.3],
    'learning_rate': [0.1],
}

# Additional parameters for training
batch_size = 128
epochs = [50, 100]

# Initialize best hyperparameters and MAE
best_hyperparameters = None
best_mae = np.inf

# Iterate over the parameter grid using ParameterGrid
for params in ParameterGrid(param_grid):
    for num_epochs in epochs:
        # Create the model with current hyperparameters
        model = create_model(**params)

        # Train the model
        model.fit(xtrain, ytrain, epochs=num_epochs, batch_size=batch_size, verbose=0)

        # Evaluate the model on the validation set
        y_pred = model.predict(xtest)
        current_mae = mean_absolute_error(ytest, y_pred)

        # Print the result for the current hyperparameters
        print(f"Hyperparameters: {params}, Epochs: {num_epochs}, Test MAE: {current_mae

        # Update best hyperparameters if the current result is better
        if current_mae < best_mae:
            best_mae = current_mae
            best_hyperparameters = {**params, 'epochs': num_epochs}

# Print the best hyperparameters and corresponding MAE
print("\n\nBest Hyperparameters:", best_hyperparameters)
print("\n\nBest Mean Absolute Error:", best_mae)
```

```
8/8 [==============================] - 0s 2ms/step
Hyperparameters: {'activation': 'relu', 'dropout_rate': 0.3, 'learning_rate': 0.1, 'ne
urons_first_layer': 64, 'num_layers': 1, 'output_activation': 'linear'}, Epochs: 50, T
est MAE: 847.5874358146422

8/8 [==============================] - 0s 2ms/step
Hyperparameters: {'activation': 'relu', 'dropout_rate': 0.3, 'learning_rate': 0.1, 'ne
urons_first_layer': 64, 'num_layers': 1, 'output_activation': 'linear'}, Epochs: 100,
Test MAE: 795.3880654611895

8/8 [==============================] - 0s 2ms/step
Hyperparameters: {'activation': 'relu', 'dropout_rate': 0.3, 'learning_rate': 0.1, 'ne
urons_first_layer': 64, 'num_layers': 2, 'output_activation': 'linear'}, Epochs: 50, T
est MAE: 949.1640093403478

8/8 [==============================] - 0s 2ms/step
Hyperparameters: {'activation': 'relu', 'dropout_rate': 0.3, 'learning_rate': 0.1, 'ne
urons_first_layer': 64, 'num_layers': 2, 'output_activation': 'linear'}, Epochs: 100,
Test MAE: 936.2749495967741

8/8 [==============================] - 0s 2ms/step
Hyperparameters: {'activation': 'relu', 'dropout_rate': 0.3, 'learning_rate': 0.1, 'ne
urons_first_layer': 64, 'num_layers': 3, 'output_activation': 'linear'}, Epochs: 50, T
est MAE: 1367.978539251512

8/8 [==============================] - 0s 2ms/step
Hyperparameters: {'activation': 'relu', 'dropout_rate': 0.3, 'learning_rate': 0.1, 'ne
urons_first_layer': 64, 'num_layers': 3, 'output_activation': 'linear'}, Epochs: 100,
Test MAE: 976.9464741368448

8/8 [==============================] - 0s 2ms/step
Hyperparameters: {'activation': 'relu', 'dropout_rate': 0.3, 'learning_rate': 0.1, 'ne
urons_first_layer': 128, 'num_layers': 1, 'output_activation': 'linear'}, Epochs: 50,
Test MAE: 822.9492207188761

8/8 [==============================] - 0s 2ms/step
Hyperparameters: {'activation': 'relu', 'dropout_rate': 0.3, 'learning_rate': 0.1, 'ne
urons_first_layer': 128, 'num_layers': 1, 'output_activation': 'linear'}, Epochs: 100,
Test MAE: 794.3550040952621

8/8 [==============================] - 0s 2ms/step
Hyperparameters: {'activation': 'relu', 'dropout_rate': 0.3, 'learning_rate': 0.1, 'ne
urons_first_layer': 128, 'num_layers': 2, 'output_activation': 'linear'}, Epochs: 50,
Test MAE: 860.0949943296371

8/8 [==============================] - 0s 2ms/step
Hyperparameters: {'activation': 'relu', 'dropout_rate': 0.3, 'learning_rate': 0.1, 'ne
urons_first_layer': 128, 'num_layers': 2, 'output_activation': 'linear'}, Epochs: 100,
Test MAE: 842.9556294102823

8/8 [==============================] - 0s 2ms/step
Hyperparameters: {'activation': 'relu', 'dropout_rate': 0.3, 'learning_rate': 0.1, 'ne
urons_first_layer': 128, 'num_layers': 3, 'output_activation': 'linear'}, Epochs: 50,
Test MAE: 922.3889632686491

8/8 [==============================] - 0s 2ms/step
Hyperparameters: {'activation': 'relu', 'dropout_rate': 0.3, 'learning_rate': 0.1, 'ne
urons_first_layer': 128, 'num_layers': 3, 'output_activation': 'linear'}, Epochs: 100,
Test MAE: 1001.4088154454386



Best Hyperparameters: {'activation': 'relu', 'dropout_rate': 0.3, 'learning_rate': 0.
1, 'neurons_first_layer': 128, 'num_layers': 1, 'output_activation': 'linear', 'epoch
```

```
s': 100}
```

```
Best Mean Absolute Error: 794.3550040952621
```

MAE of ML Models

Checking the MAE using ML models

In [34]:

```python
from sklearn.metrics import mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression


# 1. Random Forest
rf_model = RandomForestRegressor()
rf_model.fit(xtrain, ytrain)
y_pred_rf = rf_model.predict(xtest)
mae_rf = mean_absolute_error(ytest, y_pred_rf)
print("MAE (Random Forest):", mae_rf)

# 2. XGBoost
xgboost_model = XGBRegressor()
xgboost_model.fit(xtrain, ytrain)
y_pred_xgboost = xgboost_model.predict(xtest)
mae_xgboost = mean_absolute_error(ytest, y_pred_xgboost)
print("MAE (XGBoost):", mae_xgboost)

# 3. Support Vector Machines (SVM)
svm_model = SVR()
svm_model.fit(xtrain, ytrain)
y_pred_svm = svm_model.predict(xtest)
mae_svm = mean_absolute_error(ytest, y_pred_svm)
print("MAE (SVM):", mae_svm)

# 4. K-Nearest Neighbors (KNN)
knn_model = KNeighborsRegressor()
knn_model.fit(xtrain, ytrain)
y_pred_knn = knn_model.predict(xtest)
mae_knn = mean_absolute_error(ytest, y_pred_knn)
print("MAE (KNN):", mae_knn)

# 5. Linear Regression
linear_model = LinearRegression()
linear_model.fit(xtrain, ytrain)
y_pred_linear = linear_model.predict(xtest)
mae_linear = mean_absolute_error(ytest, y_pred_linear)
print("MAE (Linear Regression):", mae_linear)
```

```
MAE (Random Forest): 753.4154685099847
MAE (XGBoost): 823.7350522933468
MAE (SVM): 2242.976096560223
MAE (KNN): 874.4185483870967
MAE (Linear Regression): 864.9629800464968
```

`ML Models` outperform `ANN` because our dataset is `small` and ML models favour `less complex datasets`.