car-price-prediction-1

December 22, 2023

[2]:
```
# This Python 3 environment comes with many helpful analytics libraries␣
↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
↪docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt# data processing, CSV file I/O (e.g. pd.
↪read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
↪all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that␣
↪gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved␣
↪outside of the current session
```

[3]:
```
df = pd.read_csv("F:/499a/CarPrice_Assignment.csv")
```

[4]:
```
# Set display options to show all columns
pd.set_option('display.max_columns', None)

# Display the DataFrame
df
```

[4]:
```
     car_ID  symboling                   CarName fueltype aspiration  \
0         1          3        alfa-romero giulia      gas        std
1         2          3       alfa-romero stelvio      gas        std
```

```
2        3         1  alfa-romero Quadrifoglio      gas        std
3        4         2               audi 100 ls      gas        std
4        5         2                audi 100ls      gas        std
..      …        …                       …      …        …
200    201        -1           volvo 145e (sw)      gas        std
201    202        -1               volvo 144ea      gas      turbo
202    203        -1               volvo 244dl      gas        std
203    204        -1                 volvo 246   diesel      turbo
204    205        -1               volvo 264gl      gas      turbo

    doornumber      carbody drivewheel enginelocation  wheelbase  carlength  \
0          two  convertible        rwd          front       88.6      168.8
1          two  convertible        rwd          front       88.6      168.8
2          two    hatchback        rwd          front       94.5      171.2
3         four        sedan        fwd          front       99.8      176.6
4         four        sedan        4wd          front       99.4      176.6
..         …          …        …            …        …        …
200       four        sedan        rwd          front      109.1      188.8
201       four        sedan        rwd          front      109.1      188.8
202       four        sedan        rwd          front      109.1      188.8
203       four        sedan        rwd          front      109.1      188.8
204       four        sedan        rwd          front      109.1      188.8

     carwidth  carheight  curbweight enginetype cylindernumber  enginesize  \
0        64.1       48.8        2548       dohc           four         130
1        64.1       48.8        2548       dohc           four         130
2        65.5       52.4        2823       ohcv            six         152
3        66.2       54.3        2337        ohc           four         109
4        66.4       54.3        2824        ohc           five         136
..        …        …          …        …            …         …
200      68.9       55.5        2952        ohc           four         141
201      68.8       55.5        3049        ohc           four         141
202      68.9       55.5        3012       ohcv            six         173
203      68.9       55.5        3217        ohc            six         145
204      68.9       55.5        3062        ohc           four         141

    fuelsystem  boreratio  stroke  compressionratio  horsepower  peakrpm  \
0         mpfi       3.47    2.68               9.0         111     5000
1         mpfi       3.47    2.68               9.0         111     5000
2         mpfi       2.68    3.47               9.0         154     5000
3         mpfi       3.19    3.40              10.0         102     5500
4         mpfi       3.19    3.40               8.0         115     5500
..         …        …      …               …         …       …
200       mpfi       3.78    3.15               9.5         114     5400
201       mpfi       3.78    3.15               8.7         160     5300
202       mpfi       3.58    2.87               8.8         134     5500
203        idi       3.01    3.40              23.0         106     4800
```

```
204       mpfi      3.78    3.15              9.5         114      5400

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

[205 rows x 26 columns]
```

[5]:
```
df.columns
```

[5]:
```
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')
```

[6]:
```
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
```

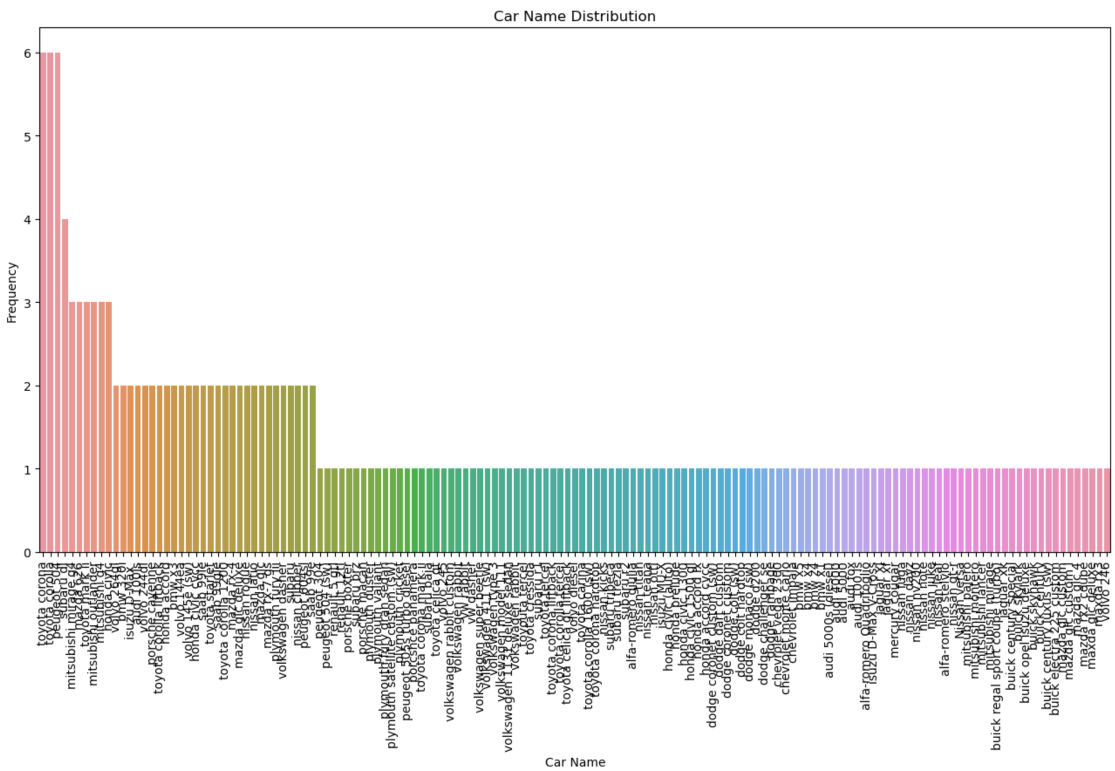
[7]:
```
df.isnull().sum()
```

[7]:
```
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
```

[8]:
```
df.duplicated().sum()
```

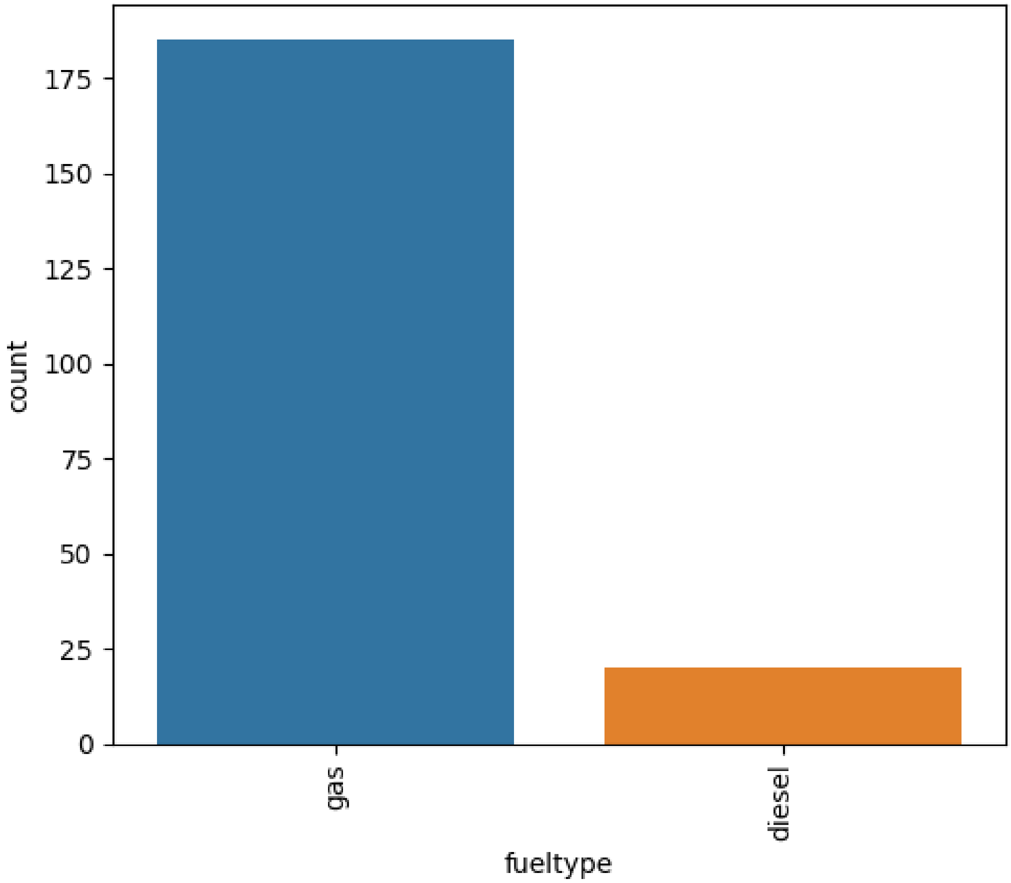
[8]: 0

[9]:
```python
plt.figure(figsize=(13, 9))  # Adjust the figure size as needed
sns.countplot(data=df, x="CarName", order=df["CarName"].value_counts().index)
plt.xticks(rotation=90, fontsize=10)  # Rotate and adjust x-axis labels

# Adding labels and title
plt.xlabel("Car Name")
plt.ylabel("Frequency")
plt.title("Car Name Distribution")

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



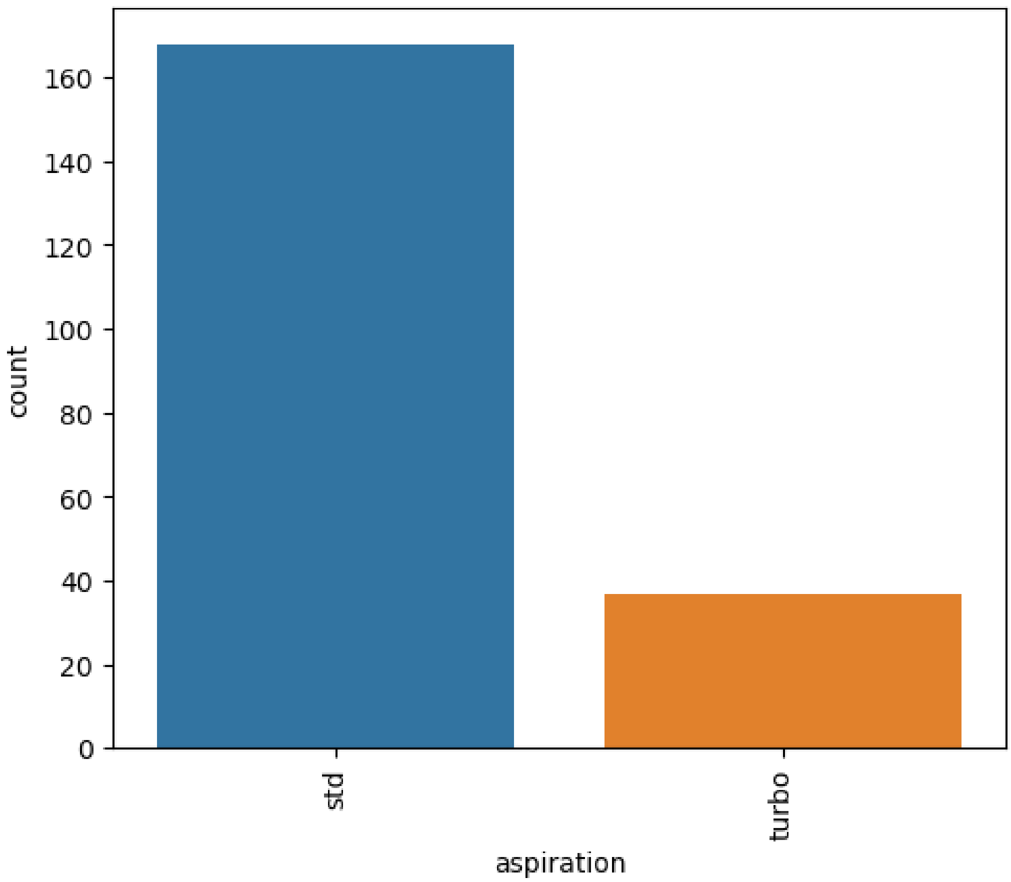
[10]:
```python
plt.figure(figsize=(6, 5))  # Adjust the figure size as needed
sns.countplot(data=df, x="fueltype", order=df["fueltype"].value_counts().index)
plt.xticks(rotation=90, fontsize=10)  # Rotate and adjust x-axis labels
```

[10]: (array([0, 1]), [Text(0, 0, 'gas'), Text(1, 0, 'diesel')])



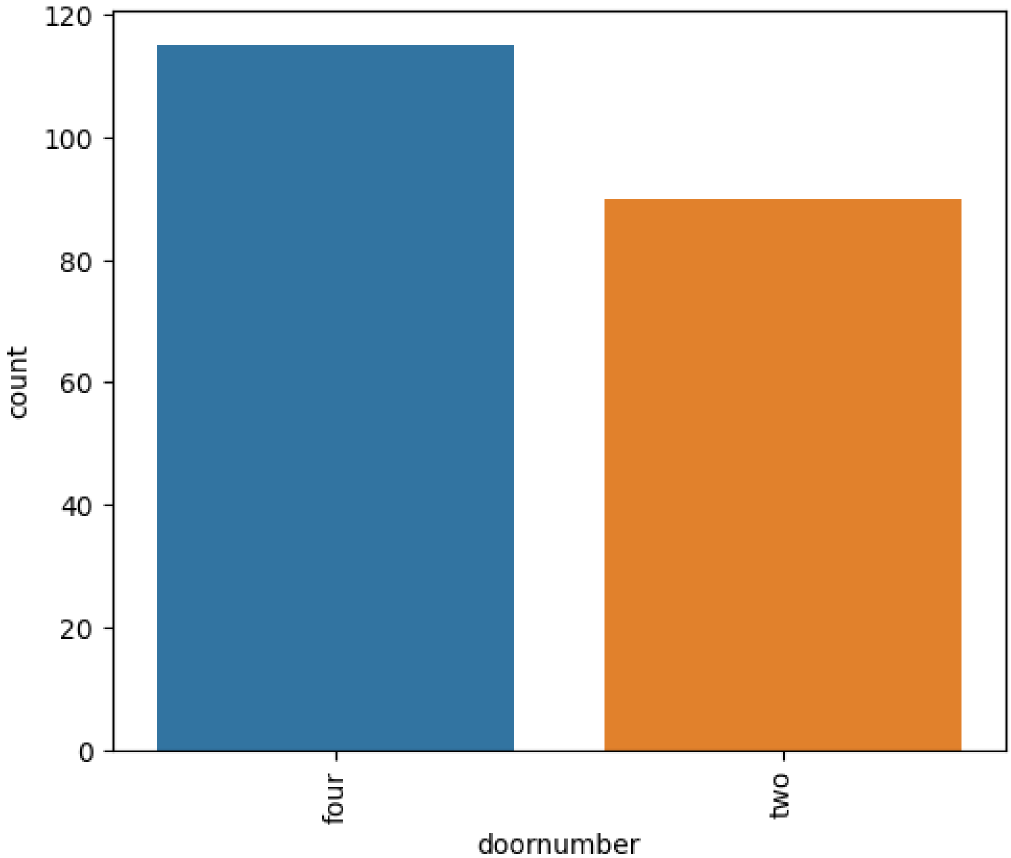
[11]:
```
plt.figure(figsize=(6, 5))  # Adjust the figure size as needed
sns.countplot(data=df, x="aspiration", order=df["aspiration"].value_counts().
 ↪index)
plt.xticks(rotation=90, fontsize=10)  # Rotate and adjust x-axis labels
```

[11]:
```
(array([0, 1]), [Text(0, 0, 'std'), Text(1, 0, 'turbo')])
```



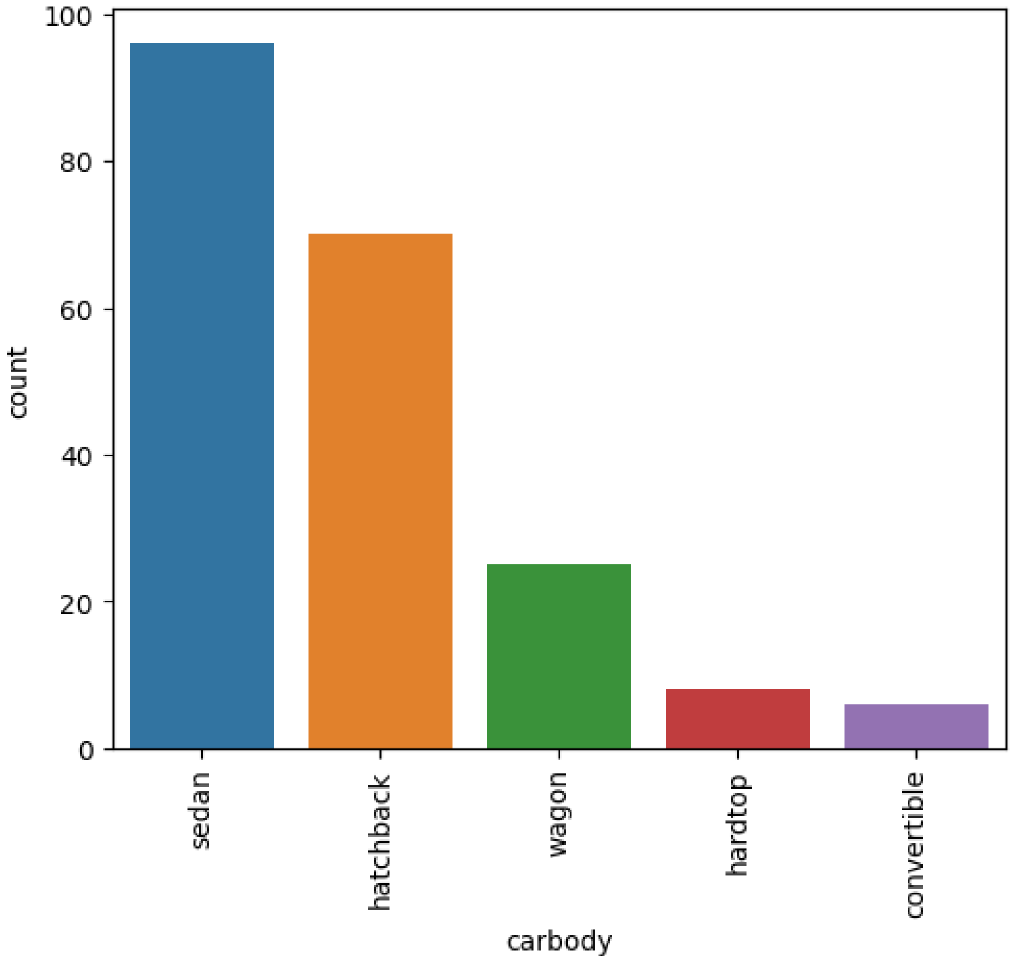
[12]:
```
plt.figure(figsize=(6, 5))  # Adjust the figure size as needed
sns.countplot(data=df, x="doornumber", order=df["doornumber"].value_counts().
 ↪index)
plt.xticks(rotation=90, fontsize=10)  # Rotate and adjust x-axis labels
```

[12]:
```
(array([0, 1]), [Text(0, 0, 'four'), Text(1, 0, 'two')])
```



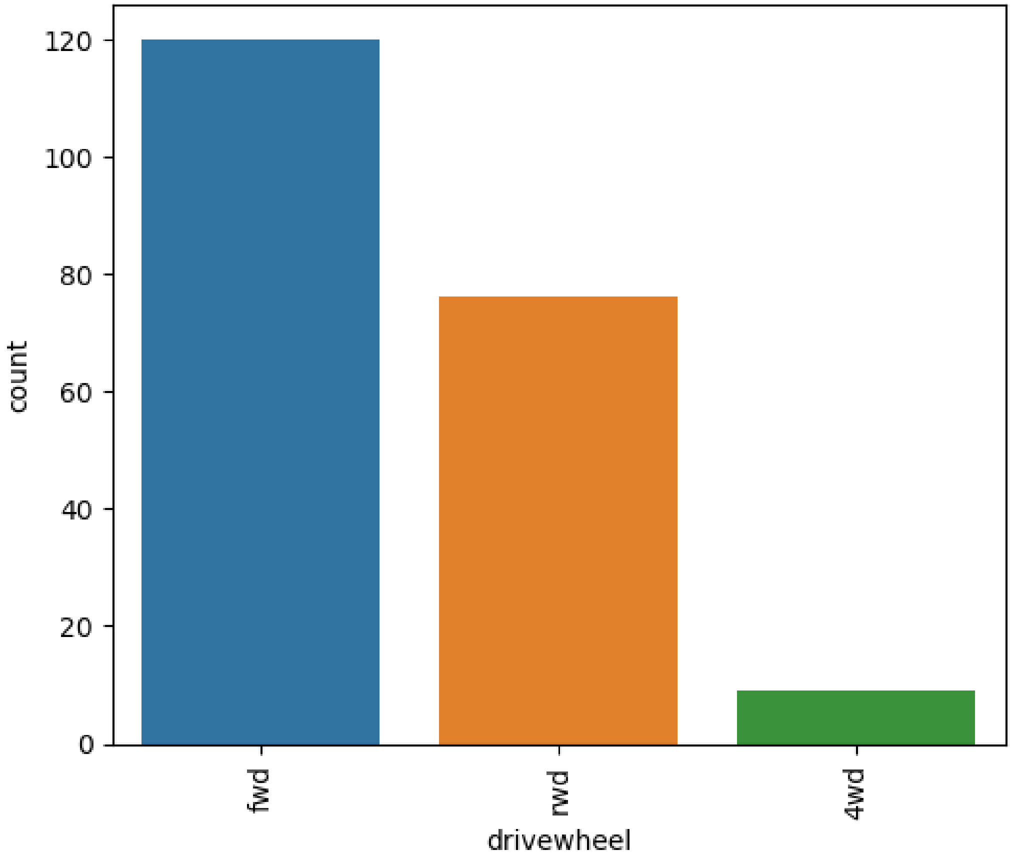

```
[13]: plt.figure(figsize=(6, 5))  # Adjust the figure size as needed
      sns.countplot(data=df, x="carbody", order=df["carbody"].value_counts().index)
      plt.xticks(rotation=90, fontsize=10)  # Rotate and adjust x-axis labels
```

```
[13]: (array([0, 1, 2, 3, 4]),
       [Text(0, 0, 'sedan'),
        Text(1, 0, 'hatchback'),
        Text(2, 0, 'wagon'),
        Text(3, 0, 'hardtop'),
        Text(4, 0, 'convertible')])
```

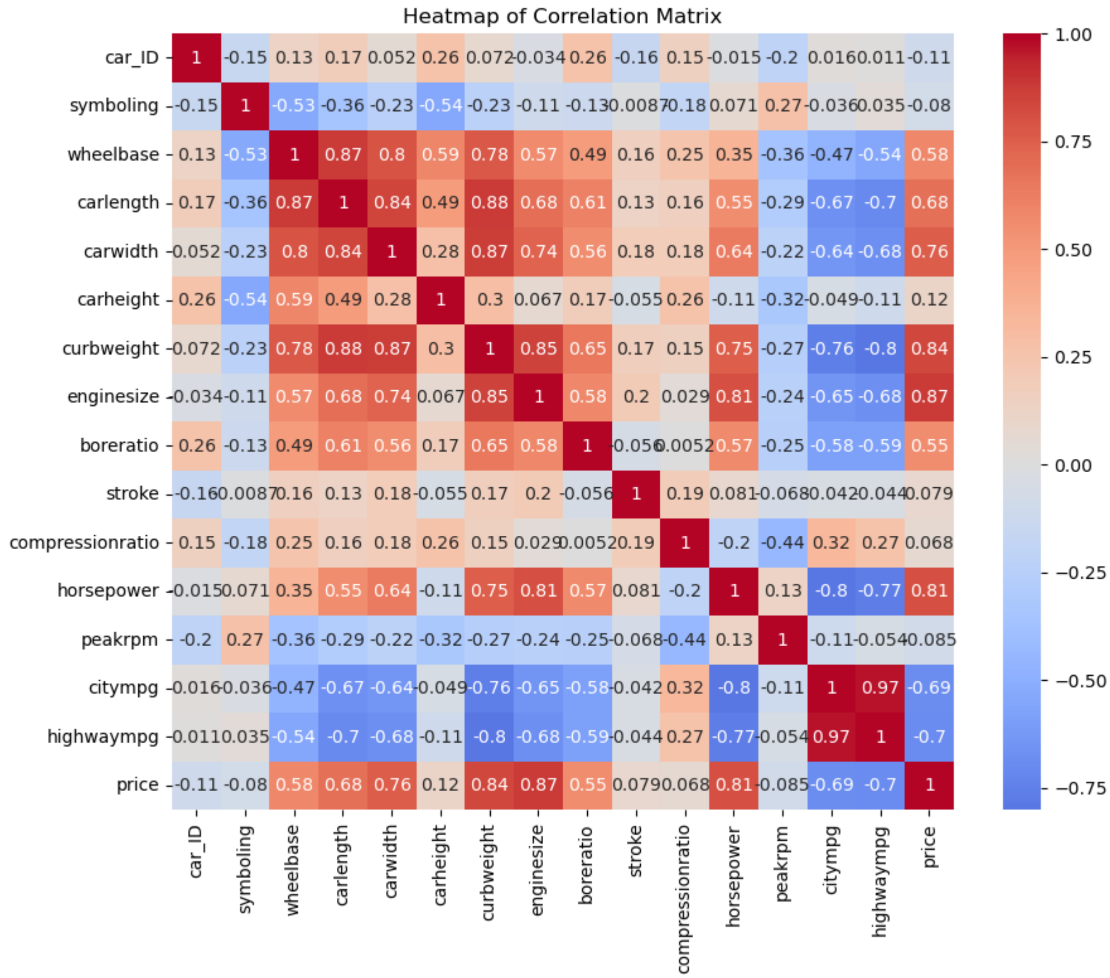


[14]:
```
plt.figure(figsize=(6, 5))  # Adjust the figure size as needed
sns.countplot(data=df, x="drivewheel", order=df["drivewheel"].value_counts().
 ↪index)
plt.xticks(rotation=90, fontsize=10)  # Rotate and adjust x-axis labels
```

[14]:
```
(array([0, 1, 2]), [Text(0, 0, 'fwd'), Text(1, 0, 'rwd'), Text(2, 0, '4wd')])
```



[15]:
```
correlation_matrix = df.corr()

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Heatmap of Correlation Matrix')
plt.show()
```



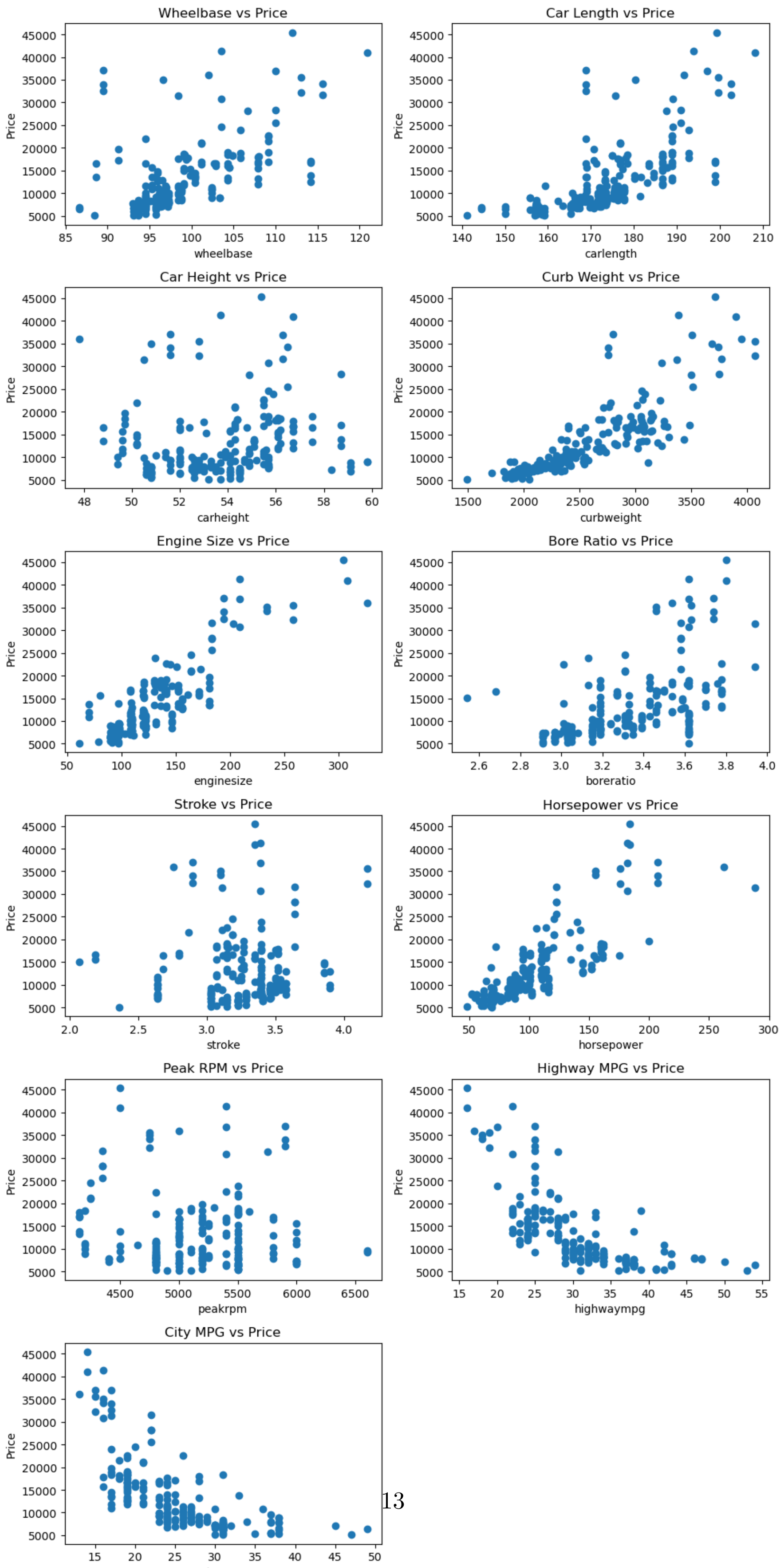
[16]:
```
df.head()
```

[16]:
```
   car_ID  symboling                   CarName fueltype aspiration doornumber  \
0       1          3        alfa-romero giulia      gas        std        two
1       2          3       alfa-romero stelvio      gas        std        two
2       3          1  alfa-romero Quadrifoglio      gas        std        two
3       4          2               audi 100 ls      gas        std       four
4       5          2                audi 100ls      gas        std       four

       carbody drivewheel enginelocation  wheelbase  carlength  carwidth  \
0  convertible        rwd          front       88.6      168.8      64.1
1  convertible        rwd          front       88.6      168.8      64.1
2    hatchback        rwd          front       94.5      171.2      65.5
3        sedan        fwd          front       99.8      176.6      66.2
4        sedan        4wd          front       99.4      176.6      66.4
```

```
   carheight  curbweight enginetype cylindernumber  enginesize fuelsystem  \
0       48.8        2548       dohc           four         130       mpfi
1       48.8        2548       dohc           four         130       mpfi
2       52.4        2823       ohcv            six         152       mpfi
3       54.3        2337        ohc           four         109       mpfi
4       54.3        2824        ohc           five         136       mpfi

   boreratio  stroke  compressionratio  horsepower  peakrpm  citympg  \
0       3.47    2.68               9.0         111     5000       21
1       3.47    2.68               9.0         111     5000       21
2       2.68    3.47               9.0         154     5000       19
3       3.19    3.40              10.0         102     5500       24
4       3.19    3.40               8.0         115     5500       18

   highwaympg    price
0          27  13495.0
1          27  16500.0
2          26  16500.0
3          30  13950.0
4          22  17450.0
```

[17]:

```python
def scatter(x, title, fig):
    plt.subplot(6, 2, fig)
    plt.scatter(df[x], df['price'])
    plt.title(title + ' vs Price')
    plt.ylabel('Price')
    plt.xlabel(x)

plt.figure(figsize=(10, 20))

scatter('wheelbase', 'Wheelbase', 1)
scatter('carlength', 'Car Length', 2)
scatter('carheight', 'Car Height', 3)
scatter('curbweight', 'Curb Weight', 4)
scatter('enginesize', 'Engine Size', 5)
scatter('boreratio', 'Bore Ratio', 6)
scatter('stroke', 'Stroke', 7)
scatter('horsepower', 'Horsepower', 8)
scatter('peakrpm', 'Peak RPM', 9)
scatter('highwaympg', 'Highway MPG', 10)
scatter('citympg', 'City MPG', 11)  # Change '11' to '1' or '2' based on your
 ↪desired position

plt.tight_layout()
plt.show()
```


Wheelbase vs Price
Price
wheelbase
Car Length vs Price
Price
carlength
Car Height vs Price
Price
carheight
Curb Weight vs Price
Price
curbweight
Engine Size vs Price
Price
enginesize
Bore Ratio vs Price
Price
boreratio
Stroke vs Price
Price
stroke
Horsepower vs Price
Price
horsepower
Peak RPM vs Price
Price
peakrpm
Highway MPG vs Price
Price
highwaympg
City MPG vs Price
Price
citympg

[18]:
```
df.drop(columns=["car_ID","peakrpm","stroke","carheight"],inplace=True)
```

[19]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

[20]:
```
en = LabelEncoder()
catCols =␣
↪['CarName','fueltype','aspiration','doornumber','carbody','drivewheel','enginelocation','en
for cols in catCols:
    df[cols] = en.fit_transform(df[cols])
```

[21]:
```
df.head()
```

[21]:
```
   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  carwidth  curbweight  enginetype  \
0               0       88.6      168.8      64.1        2548           0
1               0       88.6      168.8      64.1        2548           0
2               0       94.5      171.2      65.5        2823           5
3               0       99.8      176.6      66.2        2337           3
4               0       99.4      176.6      66.4        2824           3

   cylindernumber  enginesize  fuelsystem  boreratio  compressionratio  \
0               2         130           5       3.47               9.0
1               2         130           5       3.47               9.0
2               3         152           5       2.68               9.0
3               2         109           5       3.19              10.0
4               1         136           5       3.19               8.0

   horsepower  citympg  highwaympg    price
0         111       21          27  13495.0
1         111       21          27  16500.0
2         154       19          26  16500.0
3         102       24          30  13950.0
4         115       18          22  17450.0
```

[22]:
```
X = df.drop("price",axis = 1)
y = df["price"]
```

[23]:
```python
X.head()
```

[23]:
```
   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  carwidth  curbweight  enginetype  \
0               0       88.6      168.8      64.1        2548           0
1               0       88.6      168.8      64.1        2548           0
2               0       94.5      171.2      65.5        2823           5
3               0       99.8      176.6      66.2        2337           3
4               0       99.4      176.6      66.4        2824           3

   cylindernumber  enginesize  fuelsystem  boreratio  compressionratio  \
0               2         130           5       3.47               9.0
1               2         130           5       3.47               9.0
2               3         152           5       2.68               9.0
3               2         109           5       3.19              10.0
4               1         136           5       3.19               8.0

   horsepower  citympg  highwaympg
0         111       21          27
1         111       21          27
2         154       19          26
3         102       24          30
4         115       18          22
```

[24]:
```python
y.head()
```

[24]:
```
0    13495.0
1    16500.0
2    16500.0
3    13950.0
4    17450.0
Name: price, dtype: float64
```

[25]:
```python
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.1,random_state=4)
from sklearn.preprocessing import StandardScaler

# Create a MinMaxScaler instance
scaler = StandardScaler()

# Fit the scaler on the training data and transform it
X_train = scaler.fit_transform(X_train)
```

```
# Transform the test data using the same scaler
X_test = scaler.transform(X_test)
```

[32]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,␣
↪BaggingRegressor, ExtraTreesRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.metrics import accuracy_score
```

[36]:
```
def train_regressor(regressor, X_train, y_train, X_test, y_test):
    regressor.fit(X_train, y_train)
    y_pred = regressor.predict(X_test)

    r2 = regressor.score(X_test, y_test)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)

    return r2, mse, mae
regressors = {
    "SVR": SVR(kernel='linear', C=1),
    "KNeighborsRegressor": KNeighborsRegressor(n_neighbors=5),
    "DecisionTreeRegressor": DecisionTreeRegressor(max_depth=5),
    "Ridge": Ridge(alpha=1.0),
    "RandomForestRegressor": RandomForestRegressor(n_estimators=200,max_depth=3␣
↪,random_state=2),
    "AdaBoostRegressor": AdaBoostRegressor(n_estimators=100, random_state=2),
    "BaggingRegressor": BaggingRegressor(n_estimators=100, random_state=2),
    "ExtraTreesRegressor": ExtraTreesRegressor(n_estimators=100,␣
↪random_state=2),
    "GradientBoostingRegressor": GradientBoostingRegressor(learning_rate= 0.2, ␣
↪min_samples_leaf= 4, min_samples_split= 5, n_estimators=100),
    "XGBRegressor": XGBRegressor(n_estimators=100, random_state=2)
}
```

[48]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,␣
↪BaggingRegressor, ExtraTreesRegressor, GradientBoostingRegressor
```

```python
from xgboost import XGBRegressor
import time

def train_regressor(regressor, X_train, y_train, X_test, y_test):
    training_times = []  # List to store training times for each epoch
    for epoch in range(epochs):
        start_time = time.time()  # Record the start time of the epoch
        regressor.fit(X_train, y_train)
        end_time = time.time()  # Record the end time of the epoch
        epoch_time = end_time - start_time  # Calculate epoch training time
        training_times.append(epoch_time)

        y_pred = regressor.predict(X_test)
        r2 = regressor.score(X_test, y_test)
        mse = mean_squared_error(y_test, y_pred)
        mae = mean_absolute_error(y_test, y_pred)

        print(f"Epoch {epoch + 1} - R-squared: {r2:.2f}, MSE: {mse:.2f}, MAE: {mae:.2f}, Time: {epoch_time:.2f} seconds")

    return training_times

regressors = {
    "SVR": SVR(kernel='linear', C=1),
    "KNeighborsRegressor": KNeighborsRegressor(n_neighbors=5),
    "DecisionTreeRegressor": DecisionTreeRegressor(max_depth=5),
    "Ridge": Ridge(alpha=1.0),
    "RandomForestRegressor": RandomForestRegressor(n_estimators=200, max_depth=3, random_state=2),
    "AdaBoostRegressor": AdaBoostRegressor(n_estimators=100, random_state=2),
    "BaggingRegressor": BaggingRegressor(n_estimators=100, random_state=2),
    "ExtraTreesRegressor": ExtraTreesRegressor(n_estimators=100, random_state=2),
    "GradientBoostingRegressor": GradientBoostingRegressor(learning_rate=0.2, min_samples_leaf=4, min_samples_split=5, n_estimators=100),
    "XGBRegressor": XGBRegressor(n_estimators=100, random_state=2)
}

X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_state=4)  # Split your dataset into training and testing sets
epochs = 10  # Set the number of epochs

results = {}  # To store training times for each regressor

for regressor_name, regressor in regressors.items():
    print(f"Training {regressor_name}...")
```

```
    training_times = train_regressor(regressor, X_train, y_train, X_test,␣
↪y_test)
    results[regressor_name] = training_times

# Print the training times for each epoch for each regressor
for regressor_name, training_times in results.items():
    print(f"{regressor_name} Training Times (in seconds): {training_times}")
```

```
Training SVR…
Epoch 1 - R-squared: 0.81, MSE: 9494398.64, MAE: 2307.18, Time: 0.02 seconds
Epoch 2 - R-squared: 0.81, MSE: 9494398.64, MAE: 2307.18, Time: 0.01 seconds
Epoch 3 - R-squared: 0.81, MSE: 9494398.64, MAE: 2307.18, Time: 0.01 seconds
Epoch 4 - R-squared: 0.81, MSE: 9494398.64, MAE: 2307.18, Time: 0.01 seconds
Epoch 5 - R-squared: 0.81, MSE: 9494398.64, MAE: 2307.18, Time: 0.01 seconds
Epoch 6 - R-squared: 0.81, MSE: 9494398.64, MAE: 2307.18, Time: 0.01 seconds
Epoch 7 - R-squared: 0.81, MSE: 9494398.64, MAE: 2307.18, Time: 0.01 seconds
Epoch 8 - R-squared: 0.81, MSE: 9494398.64, MAE: 2307.18, Time: 0.01 seconds
Epoch 9 - R-squared: 0.81, MSE: 9494398.64, MAE: 2307.18, Time: 0.01 seconds
Epoch 10 - R-squared: 0.81, MSE: 9494398.64, MAE: 2307.18, Time: 0.01 seconds
Training KNeighborsRegressor…
Epoch 1 - R-squared: 0.74, MSE: 12723427.90, MAE: 2361.76, Time: 0.00 seconds
Epoch 2 - R-squared: 0.74, MSE: 12723427.90, MAE: 2361.76, Time: 0.00 seconds
Epoch 3 - R-squared: 0.74, MSE: 12723427.90, MAE: 2361.76, Time: 0.00 seconds
Epoch 4 - R-squared: 0.74, MSE: 12723427.90, MAE: 2361.76, Time: 0.00 seconds
Epoch 5 - R-squared: 0.74, MSE: 12723427.90, MAE: 2361.76, Time: 0.00 seconds
Epoch 6 - R-squared: 0.74, MSE: 12723427.90, MAE: 2361.76, Time: 0.00 seconds
Epoch 7 - R-squared: 0.74, MSE: 12723427.90, MAE: 2361.76, Time: 0.00 seconds
Epoch 8 - R-squared: 0.74, MSE: 12723427.90, MAE: 2361.76, Time: 0.00 seconds
Epoch 9 - R-squared: 0.74, MSE: 12723427.90, MAE: 2361.76, Time: 0.00 seconds
Epoch 10 - R-squared: 0.74, MSE: 12723427.90, MAE: 2361.76, Time: 0.00 seconds
Training DecisionTreeRegressor…
Epoch 1 - R-squared: 0.88, MSE: 6163271.96, MAE: 1892.47, Time: 0.00 seconds
Epoch 2 - R-squared: 0.88, MSE: 5995462.40, MAE: 1830.23, Time: 0.00 seconds
Epoch 3 - R-squared: 0.87, MSE: 6577179.04, MAE: 1941.35, Time: 0.00 seconds
Epoch 4 - R-squared: 0.85, MSE: 7309910.35, MAE: 2018.61, Time: 0.00 seconds
Epoch 5 - R-squared: 0.88, MSE: 6163271.96, MAE: 1892.47, Time: 0.00 seconds
Epoch 6 - R-squared: 0.87, MSE: 6455411.05, MAE: 1941.35, Time: 0.00 seconds
Epoch 7 - R-squared: 0.85, MSE: 7278440.34, MAE: 2002.93, Time: 0.00 seconds
Epoch 8 - R-squared: 0.85, MSE: 7263868.78, MAE: 1956.37, Time: 0.00 seconds
Epoch 9 - R-squared: 0.88, MSE: 6163271.96, MAE: 1892.47, Time: 0.00 seconds
Epoch 10 - R-squared: 0.87, MSE: 6455411.05, MAE: 1941.35, Time: 0.00 seconds
Training Ridge…
Epoch 1 - R-squared: 0.83, MSE: 8196426.81, MAE: 2170.84, Time: 0.00 seconds
Epoch 2 - R-squared: 0.83, MSE: 8196426.81, MAE: 2170.84, Time: 0.00 seconds
Epoch 3 - R-squared: 0.83, MSE: 8196426.81, MAE: 2170.84, Time: 0.00 seconds
Epoch 4 - R-squared: 0.83, MSE: 8196426.81, MAE: 2170.84, Time: 0.00 seconds
Epoch 5 - R-squared: 0.83, MSE: 8196426.81, MAE: 2170.84, Time: 0.00 seconds
```

```
Epoch 6 - R-squared: 0.83, MSE: 8196426.81, MAE: 2170.84, Time: 0.00 seconds
Epoch 7 - R-squared: 0.83, MSE: 8196426.81, MAE: 2170.84, Time: 0.00 seconds
Epoch 8 - R-squared: 0.83, MSE: 8196426.81, MAE: 2170.84, Time: 0.00 seconds
Epoch 9 - R-squared: 0.83, MSE: 8196426.81, MAE: 2170.84, Time: 0.00 seconds
Epoch 10 - R-squared: 0.83, MSE: 8196426.81, MAE: 2170.84, Time: 0.00 seconds
Training RandomForestRegressor…
Epoch 1 - R-squared: 0.87, MSE: 6582210.91, MAE: 1914.52, Time: 0.36 seconds
Epoch 2 - R-squared: 0.87, MSE: 6582210.91, MAE: 1914.52, Time: 0.30 seconds
Epoch 3 - R-squared: 0.87, MSE: 6582210.91, MAE: 1914.52, Time: 0.30 seconds
Epoch 4 - R-squared: 0.87, MSE: 6582210.91, MAE: 1914.52, Time: 0.28 seconds
Epoch 5 - R-squared: 0.87, MSE: 6582210.91, MAE: 1914.52, Time: 0.28 seconds
Epoch 6 - R-squared: 0.87, MSE: 6582210.91, MAE: 1914.52, Time: 0.28 seconds
Epoch 7 - R-squared: 0.87, MSE: 6582210.91, MAE: 1914.52, Time: 0.29 seconds
Epoch 8 - R-squared: 0.87, MSE: 6582210.91, MAE: 1914.52, Time: 0.28 seconds
Epoch 9 - R-squared: 0.87, MSE: 6582210.91, MAE: 1914.52, Time: 0.29 seconds
Epoch 10 - R-squared: 0.87, MSE: 6582210.91, MAE: 1914.52, Time: 0.28 seconds
Training AdaBoostRegressor…
Epoch 1 - R-squared: 0.89, MSE: 5259872.25, MAE: 1933.86, Time: 0.18 seconds
Epoch 2 - R-squared: 0.89, MSE: 5259872.25, MAE: 1933.86, Time: 0.15 seconds
Epoch 3 - R-squared: 0.89, MSE: 5259872.25, MAE: 1933.86, Time: 0.17 seconds
Epoch 4 - R-squared: 0.89, MSE: 5259872.25, MAE: 1933.86, Time: 0.16 seconds
Epoch 5 - R-squared: 0.89, MSE: 5259872.25, MAE: 1933.86, Time: 0.16 seconds
Epoch 6 - R-squared: 0.89, MSE: 5259872.25, MAE: 1933.86, Time: 0.18 seconds
Epoch 7 - R-squared: 0.89, MSE: 5259872.25, MAE: 1933.86, Time: 0.16 seconds
Epoch 8 - R-squared: 0.89, MSE: 5259872.25, MAE: 1933.86, Time: 0.14 seconds
Epoch 9 - R-squared: 0.89, MSE: 5259872.25, MAE: 1933.86, Time: 0.19 seconds
Epoch 10 - R-squared: 0.89, MSE: 5259872.25, MAE: 1933.86, Time: 0.15 seconds
Training BaggingRegressor…
Epoch 1 - R-squared: 0.91, MSE: 4226915.74, MAE: 1486.30, Time: 0.38 seconds
Epoch 2 - R-squared: 0.91, MSE: 4226915.74, MAE: 1486.30, Time: 0.43 seconds
Epoch 3 - R-squared: 0.91, MSE: 4226915.74, MAE: 1486.30, Time: 0.53 seconds
Epoch 4 - R-squared: 0.91, MSE: 4226915.74, MAE: 1486.30, Time: 0.47 seconds
Epoch 5 - R-squared: 0.91, MSE: 4226915.74, MAE: 1486.30, Time: 0.39 seconds
Epoch 6 - R-squared: 0.91, MSE: 4226915.74, MAE: 1486.30, Time: 0.37 seconds
Epoch 7 - R-squared: 0.91, MSE: 4226915.74, MAE: 1486.30, Time: 0.37 seconds
Epoch 8 - R-squared: 0.91, MSE: 4226915.74, MAE: 1486.30, Time: 0.35 seconds
Epoch 9 - R-squared: 0.91, MSE: 4226915.74, MAE: 1486.30, Time: 0.35 seconds
Epoch 10 - R-squared: 0.91, MSE: 4226915.74, MAE: 1486.30, Time: 0.36 seconds
Training ExtraTreesRegressor…
Epoch 1 - R-squared: 0.92, MSE: 4074228.75, MAE: 1627.64, Time: 0.21 seconds
Epoch 2 - R-squared: 0.92, MSE: 4074228.75, MAE: 1627.64, Time: 0.23 seconds
Epoch 3 - R-squared: 0.92, MSE: 4074228.75, MAE: 1627.64, Time: 0.22 seconds
Epoch 4 - R-squared: 0.92, MSE: 4074228.75, MAE: 1627.64, Time: 0.20 seconds
Epoch 5 - R-squared: 0.92, MSE: 4074228.75, MAE: 1627.64, Time: 0.21 seconds
Epoch 6 - R-squared: 0.92, MSE: 4074228.75, MAE: 1627.64, Time: 0.21 seconds
Epoch 7 - R-squared: 0.92, MSE: 4074228.75, MAE: 1627.64, Time: 0.22 seconds
Epoch 8 - R-squared: 0.92, MSE: 4074228.75, MAE: 1627.64, Time: 0.21 seconds
Epoch 9 - R-squared: 0.92, MSE: 4074228.75, MAE: 1627.64, Time: 0.21 seconds
```

```
Epoch 10 - R-squared: 0.92, MSE: 4074228.75, MAE: 1627.64, Time: 0.23 seconds
Training GradientBoostingRegressor…
Epoch 1 - R-squared: 0.93, MSE: 3420535.61, MAE: 1464.40, Time: 0.12 seconds
Epoch 2 - R-squared: 0.93, MSE: 3410006.06, MAE: 1463.99, Time: 0.11 seconds
Epoch 3 - R-squared: 0.93, MSE: 3501758.93, MAE: 1482.52, Time: 0.13 seconds
Epoch 4 - R-squared: 0.93, MSE: 3415039.28, MAE: 1453.30, Time: 0.12 seconds
Epoch 5 - R-squared: 0.93, MSE: 3424994.32, MAE: 1465.81, Time: 0.13 seconds
Epoch 6 - R-squared: 0.93, MSE: 3406715.27, MAE: 1459.56, Time: 0.15 seconds
Epoch 7 - R-squared: 0.93, MSE: 3542737.97, MAE: 1495.00, Time: 0.12 seconds
Epoch 8 - R-squared: 0.93, MSE: 3427623.50, MAE: 1466.08, Time: 0.15 seconds
Epoch 9 - R-squared: 0.93, MSE: 3419481.91, MAE: 1454.70, Time: 0.12 seconds
Epoch 10 - R-squared: 0.93, MSE: 3558609.02, MAE: 1489.47, Time: 0.12 seconds
Training XGBRegressor…
Epoch 1 - R-squared: 0.92, MSE: 3711713.11, MAE: 1324.63, Time: 0.10 seconds
Epoch 2 - R-squared: 0.92, MSE: 3711713.11, MAE: 1324.63, Time: 0.09 seconds
Epoch 3 - R-squared: 0.92, MSE: 3711713.11, MAE: 1324.63, Time: 0.12 seconds
Epoch 4 - R-squared: 0.92, MSE: 3711713.11, MAE: 1324.63, Time: 0.09 seconds
Epoch 5 - R-squared: 0.92, MSE: 3711713.11, MAE: 1324.63, Time: 0.09 seconds
Epoch 6 - R-squared: 0.92, MSE: 3711713.11, MAE: 1324.63, Time: 0.08 seconds
Epoch 7 - R-squared: 0.92, MSE: 3711713.11, MAE: 1324.63, Time: 0.09 seconds
Epoch 8 - R-squared: 0.92, MSE: 3711713.11, MAE: 1324.63, Time: 0.09 seconds
Epoch 9 - R-squared: 0.92, MSE: 3711713.11, MAE: 1324.63, Time: 0.09 seconds
Epoch 10 - R-squared: 0.92, MSE: 3711713.11, MAE: 1324.63, Time: 0.09 seconds
SVR Training Times (in seconds): [0.015958786010742188, 0.013962268829345703,
0.012964725494384766, 0.013965368270874023, 0.0139617919921875,
0.013960838317871094, 0.01302337646484375, 0.012023448944091797,
0.012023448944091797, 0.013007402420043945]
KNeighborsRegressor Training Times (in seconds): [0.0020248889923095703,
0.0022466182708740234, 0.0009970664978027344, 0.0009970664978027344,
0.001994609832763672, 0.0009970664978027344, 0.000997304916381836,
0.0009968280792236328, 0.0009965896606445312, 0.0019936561584472656]
DecisionTreeRegressor Training Times (in seconds): [0.0019941329956054688,
0.001994609832763672, 0.0029840469360351562, 0.0019948482513427734,
0.001994609832763672, 0.001993894577026367, 0.001994609832763672,
0.001994609832763672, 0.002992868423461914, 0.002991914749145508]
Ridge Training Times (in seconds): [0.0019958019256591797,
0.0019943714141845703, 0.001995086669921875, 0.001994609832763672,
0.0019953250885009766, 0.001994609832763672, 0.0019943714141845703,
0.002006053924560547, 0.0019943714141845703, 0.0009975433349609375]
RandomForestRegressor Training Times (in seconds): [0.35610437393188477,
0.2971975803375244, 0.29517292976379395, 0.282275915145874, 0.2792532444000244,
0.28029561042785645, 0.28623199462890625, 0.28224754333496094,
0.29026293754577637, 0.27829551696777344]
AdaBoostRegressor Training Times (in seconds): [0.18350958824157715,
0.15259027481079102, 0.16655611991882324, 0.16161036491394043,
0.1585242748260498, 0.17547845840454102, 0.15953278541564941,
0.14162087440490723, 0.18550562858581543, 0.15255141258239746]
BaggingRegressor Training Times (in seconds): [0.3809685707092285,
```

```
0.43088865280151367, 0.5305945873260498, 0.474729061126709, 0.38999199867248535,
0.3660435676574707, 0.3700551986694336, 0.35400986671447754,
0.35301899909973145, 0.36305880546569824]
ExtraTreesRegressor Training Times (in seconds): [0.20939874649047852,
0.23342037200927734, 0.21642446517944336, 0.20345544815063477,
0.2064762115478515б, 0.20644760131835938, 0.22042274475097656,
0.20648550987243652, 0.21342992782592773, 0.22838807106018066]
GradientBoostingRegressor Training Times (in seconds): [0.11667943000793457,
0.11170077323913574, 0.133683443069458, 0.12067842483520508,
0.12865281105041504, 0.14561057090759277, 0.12462282180786133,
0.14760732650756836, 0.12462782859802246, 0.12366008758544922]
XGBRegressor Training Times (in seconds): [0.0967404842376709,
0.08976054191589355, 0.1176862716674804 7, 0.09076642990112305,
0.09474587440490723, 0.0827779769897461, 0.09275102615356445,
0.09075665473937988, 0.08676767349243164, 0.09075760841369629]
```

[37]:
```python
# Lists to store metrics
algorithm_names = []
r2_scores = []
mse_scores = []
mae_scores = []

# Train and evaluate each regressor
for name, regressor in regressors.items():
    r2, mse, mae= train_regressor(regressor, X_train, y_train, X_test, y_test)

    algorithm_names.append(name)
    r2_scores.append(r2)
    mse_scores.append(mse)
    mae_scores.append(mae)


# Create a DataFrame to store the results
results_df = pd.DataFrame({
    'Algorithm': algorithm_names,
    'R2 Score': r2_scores,
    'Mean Squared Error': mse_scores,
    'Mean Absolute Error': mae_scores
})

display(results_df)
```

```
                Algorithm  R2 Score  Mean Squared Error  \
0                     SVR -0.121589        5.598468e+07
1     KNeighborsRegressor  0.857828        7.096604e+06
2   DecisionTreeRegressor  0.847318        7.621191e+06
3                   Ridge  0.846768        7.648635e+06
4   RandomForestRegressor  0.833496        8.311119e+06
```

```
5          AdaBoostRegressor  0.899744        5.004332e+06
6           BaggingRegressor  0.904408        4.771540e+06
7        ExtraTreesRegressor  0.911473        4.418865e+06
8  GradientBoostingRegressor  0.941728        2.908671e+06
9               XGBRegressor  0.913827        4.301364e+06

   Mean Absolute Error
0          5266.610617
1          2063.990476
2          2020.238631
3          2220.268551
4          2027.620255
5          1684.996219
6          1305.606825
7          1565.243414
8          1262.223506
9          1173.013346
```

[39]:

```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score,␣
↪accuracy_score
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,␣
↪BaggingRegressor, ExtraTreesRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
import numpy as np

# Define your training and testing data: X_train, y_train, X_test, y_test

threshold = 0.5  # You can adjust the threshold as needed

for model_name, regressor in regressors.items():
    regressor.fit(X_train, y_train)
    y_pred = regressor.predict(X_test)

    # Binarize the predictions based on the threshold
    y_pred_binary = (y_pred >= threshold).astype(int)

    # Calculate regression metrics
    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)

    # Calculate accuracy based on binarized predictions
    accuracy = accuracy_score(y_test, y_pred_binary)
```

```
    print(f"{model_name} - R-squared: {r2}, MSE: {mse}, MAE: {mae}, Accuracy:␣
↪{accuracy}")
```

```
SVR - R-squared: -0.12158881504099561, MSE: 55984678.84228565, MAE:
5266.61061658763, Accuracy: 0.0
KNeighborsRegressor - R-squared: 0.8578276878993255, MSE: 7096603.60952381, MAE:
2063.9904761904763, Accuracy: 0.0
DecisionTreeRegressor - R-squared: 0.853439936266576, MSE: 7315620.474443436,
MAE: 1906.5522954796409, Accuracy: 0.0
Ridge - R-squared: 0.8467683775631772, MSE: 7648634.736335464, MAE:
2220.268551296449, Accuracy: 0.0
RandomForestRegressor - R-squared: 0.8334962668337302, MSE: 8311118.925534684,
MAE: 2027.6202545102465, Accuracy: 0.0
AdaBoostRegressor - R-squared: 0.8997439443063169, MSE: 5004332.251476664, MAE:
1684.9962185038569, Accuracy: 0.0
BaggingRegressor - R-squared: 0.9044076645171308, MSE: 4771540.273961137, MAE:
1305.606825238095, Accuracy: 0.0
ExtraTreesRegressor - R-squared: 0.911473104993635, MSE: 4418865.202088281, MAE:
1565.243413809524, Accuracy: 0.0
GradientBoostingRegressor - R-squared: 0.9382325041087549, MSE:
3083156.121022084, MAE: 1297.3170644570703, Accuracy: 0.0
XGBRegressor - R-squared: 0.9138271000295217, MSE: 4301364.336964516, MAE:
1173.0133463541667, Accuracy: 0.0
```

[41]:
```
import pandas as pd

# Lists to store metrics
algorithm_names = []
r2_scores = []
mse_scores = []
mae_scores = []
accuracy_scores = []  # Updated this variable name to avoid conflicts

# Train and evaluate each regressor
for model_name, regressor in regressors.items():
    r2, mse, mae = train_regressor(regressor, X_train, y_train, X_test, y_test)

    algorithm_names.append(model_name)  # Corrected variable name
    r2_scores.append(r2)
    mse_scores.append(mse)
    mae_scores.append(mae)
    accuracy_scores.append(accuracy)  # You need to calculate accuracy here

results_df = pd.DataFrame({
    'Algorithm': algorithm_names,
    'R2 Score': r2_scores,
```

```
    'Mean Squared Error': mse_scores,
    'Mean Absolute Error': mae_scores,
    'Accuracy': accuracy_scores
})

# Display the DataFrame
display(results_df)
```

```
                   Algorithm  R2 Score  Mean Squared Error  \
0                        SVR -0.121589        5.598468e+07
1        KNeighborsRegressor  0.857828        7.096604e+06
2      DecisionTreeRegressor  0.847318        7.621191e+06
3                      Ridge  0.846768        7.648635e+06
4      RandomForestRegressor  0.833496        8.311119e+06
5          AdaBoostRegressor  0.899744        5.004332e+06
6           BaggingRegressor  0.904408        4.771540e+06
7        ExtraTreesRegressor  0.911473        4.418865e+06
8  GradientBoostingRegressor  0.946082        2.691336e+06
9               XGBRegressor  0.913827        4.301364e+06

   Mean Absolute Error Accuracy
0          5266.610617       []
1          2063.990476       []
2          2020.238631       []
3          2220.268551       []
4          2027.620255       []
5          1684.996219       []
6          1305.606825       []
7          1565.243414       []
8          1180.561497       []
9          1173.013346       []
```