used-car-price-preiction-2

September 15, 2024

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
pd.set_option("display.max_columns", 200)
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
```

[ ]:
```
data = pd.read_csv("/content/extended_data.csv")
data.head()
```

[ ]:

	model_year	brand	model	type	miles_per_gallon	premium_version	msrp	collection_car
0	2016	Toyota	Land Cruiser Base	SUV	13.0	1	84900.0	0
1	2014	RAM	ProMaster 2500 Window Van High Roof	Van	15.0	0	35000.0	0
2	2002	Ford	Mustang GT	Coupe	16.0	0	26250.0	0
3	2012	BMW	428 Gran Coupe i xDrive	Sedan	27.0	1	45000.0	0
4	2008	Mercedes-Benz	SL-Class SL500 Roadster	Convertible	18.0	1	100000.0	1

[ ]:
```
data.isnull().sum()
```

[ ]:
```
model_year           0
brand                0
model                0
type                 0
miles_per_gallon    17
premium_version      0
```

```
msrp                17
collection_car       0
dtype: int64
```

[ ]:
```
data.shape
```

[ ]:
```
(28143, 8)
```

[ ]:
```
data.columns
```

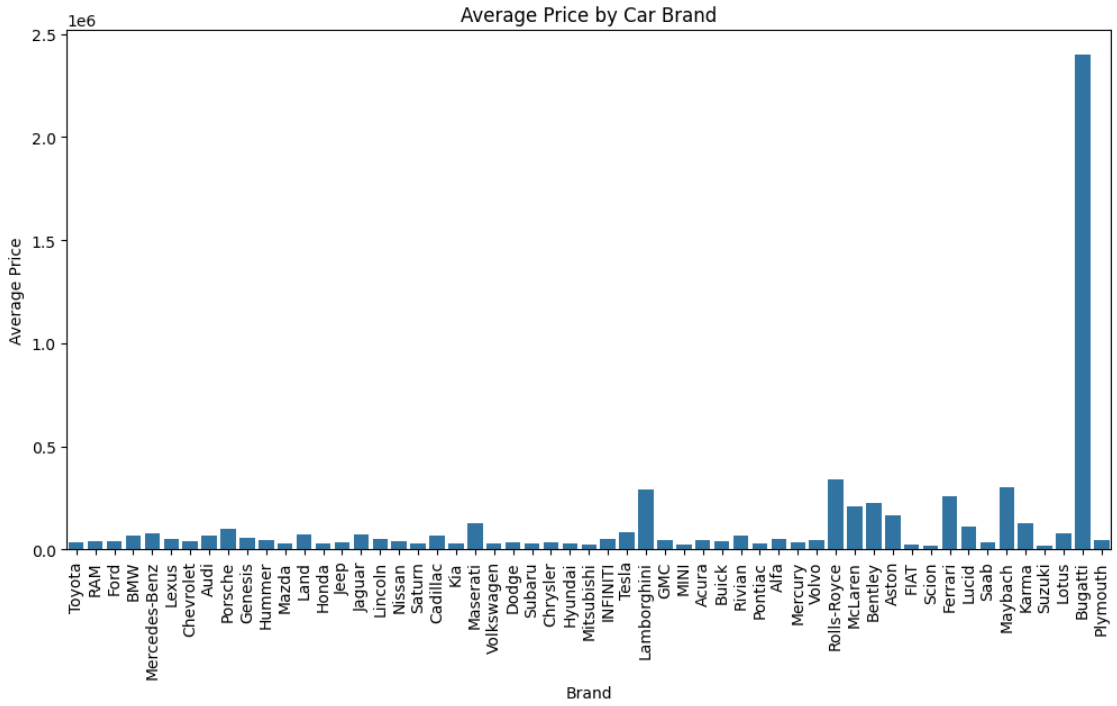
[ ]:
```
Index(['model_year', 'brand', 'model', 'type', 'miles_per_gallon',
       'premium_version', 'msrp', 'collection_car'],
      dtype='object')
```

[ ]:
```
data.dropna(inplace=True)
data.isnull().sum()
```

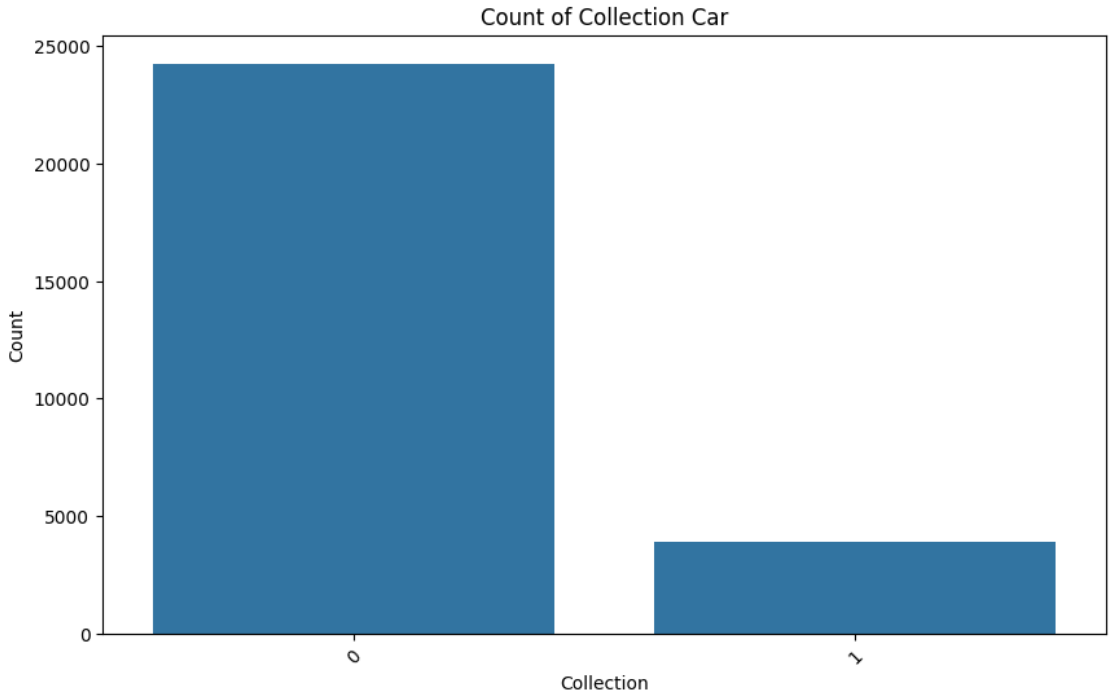
[ ]:
```
model_year          0
brand               0
model               0
type                0
miles_per_gallon    0
premium_version     0
msrp                0
collection_car      0
dtype: int64
```

[ ]:
```
data = data.rename(columns = {"msrp":"price"})
```

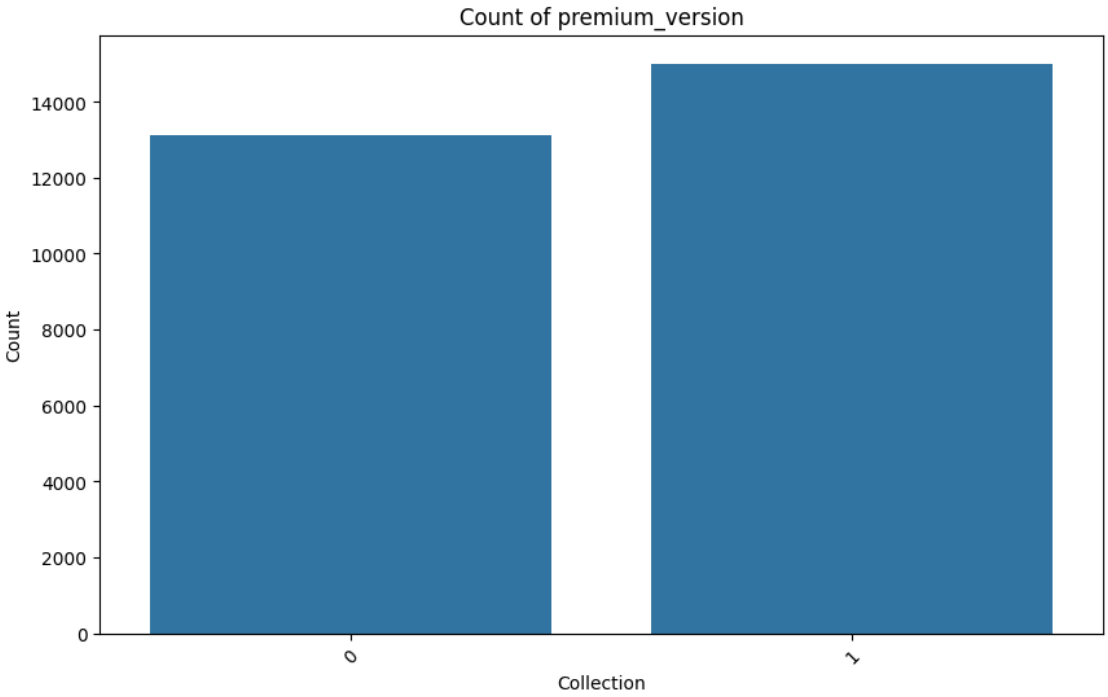
[ ]:
```
plt.figure(figsize=(12, 6))
sns.barplot(x='brand', y='price', data=data[:10000], errorbar=None)
plt.title('Average Price by Car Brand')
plt.xlabel('Brand')
plt.ylabel('Average Price')
plt.xticks(rotation=90)
plt.show()
```



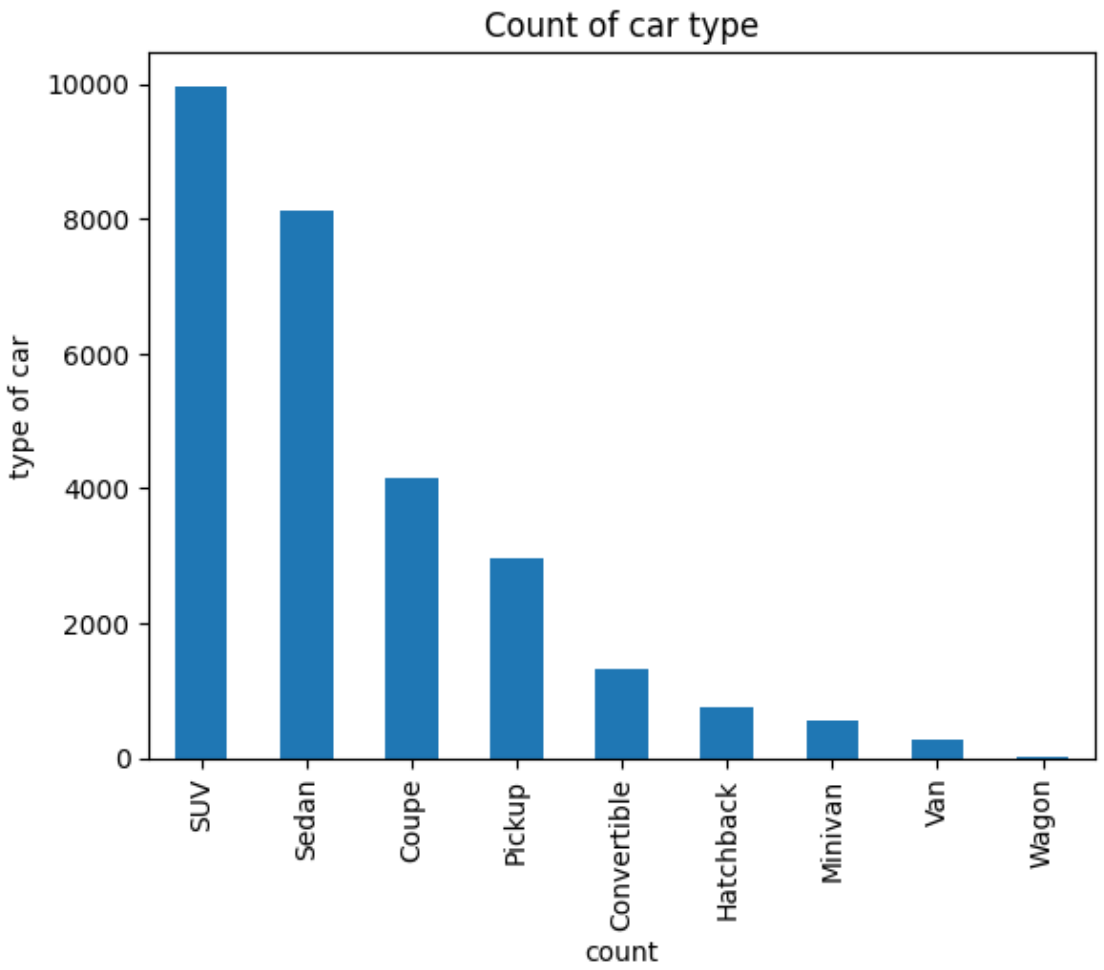
[ ]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='collection_car', data=data)
plt.title('Count of Collection Car')
plt.xlabel('Collection')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



[ ]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='premium_version', data=data)
plt.title('Count of premium_version')
plt.xlabel('Collection')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



[ ]:
```
data["type"].value_counts().plot.bar()
plt.title("Count of car type")
plt.xlabel("count")
plt.ylabel("type of car")
plt.show()
```



[ ]:
```
data.head()
```

[ ]:
```
   model_year          brand                                  model  \
0        2016         Toyota                     Land Cruiser Base
1        2014            RAM  ProMaster 2500 Window Van High Roof
2        2002           Ford                            Mustang GT
3        2012            BMW                428 Gran Coupe i xDrive
4        2008  Mercedes-Benz                SL-Class SL500 Roadster

           type  miles_per_gallon  premium_version     price  collection_car
0           SUV              13.0                1   84900.0               0
1           Van              15.0                0   35000.0               0
2        Coupe               16.0                0   26250.0               0
3        Sedan               27.0                1   45000.0               0
4  Convertible               18.0                1  100000.0               1
```

[ ]:
```
data["type"].unique()
```

[ ]:
```
array(['SUV', 'Van', 'Coupe', 'Sedan', 'Convertible', 'Pickup', 'Minivan',
       'Hatchback', 'Wagon'], dtype=object)
```

[ ]:
```
data['car_age'] = 2024 - data['model_year']
```

[ ]:
```
data["type"].unique()
```

[ ]:
```
array(['SUV', 'Van', 'Coupe', 'Sedan', 'Convertible', 'Pickup', 'Minivan',
       'Hatchback', 'Wagon'], dtype=object)
```

[ ]:
```
#removing outliers

Q1 = data["miles_per_gallon"].quantile(0.25)
Q3 = data["miles_per_gallon"].quantile(0.75)
IQR = Q3 - Q1

upper_limit = Q3 + 1.5 * IQR
lower_limit = Q1 - 1.5 * IQR
```

[ ]:
```
print("UPPER LIMIT :",upper_limit)
print("LOWER LIMIT :",lower_limit)
print("IQR", IQR)
```

```
UPPER LIMIT : 34.5
LOWER LIMIT : 6.5
IQR 7.0
```

[ ]:
```
IQR
```

[ ]:
```
7.0
```

[ ]:
```
data["miles_per_gallon"].describe()
```

[ ]:
```
count    28126.000000
mean        21.769242
std         13.472177
min         -1.000000
25%         17.000000
50%         20.000000
75%         24.000000
max        234.000000
Name: miles_per_gallon, dtype: float64
```

[ ]:
```
data[data["miles_per_gallon"] > upper_limit]
```

[ ]:

	model_year	brand	model	type	\
160	2017	Tesla	Model S 85D	Sedan	
170	2022	Tesla	Model S Long Range Plus	Sedan	
191	2022	Rivian	R1S Adventure Package	SUV	
253	2021	Tesla	Model Y Long Range	SUV	
258	2023	Tesla	Model Y Performance	SUV	
…	…	…	…	…	
28085	2012	Toyota	Mirai Base	Sedan	
28105	2006	Ford	Fusion Hybrid Base	Sedan	
28123	2011	Tesla	Leaf SL	Sedan	
28136	2010	Lexus	CT 200h Premium	Hatchback	
28140	2018	Ford	Model X P100D	SUV	

	miles_per_gallon	premium_version	price	collection_car	car_age
160	90.0	1	75000.0	0	7
170	120.0	1	79990.0	0	2
191	74.0	1	74900.0	0	2
253	131.0	1	53990.0	0	3
258	110.0	1	66990.0	0	1
…	…	…	…	…	…
28085	62.0	0	57800.0	0	12
28105	36.0	0	24995.0	0	18
28123	99.0	0	35000.0	0	13
28136	43.0	1	29900.0	0	14
28140	94.0	1	199000.0	1	6

[1058 rows x 9 columns]

[ ]:
```
data[data["miles_per_gallon"] < lower_limit]
```

[ ]:

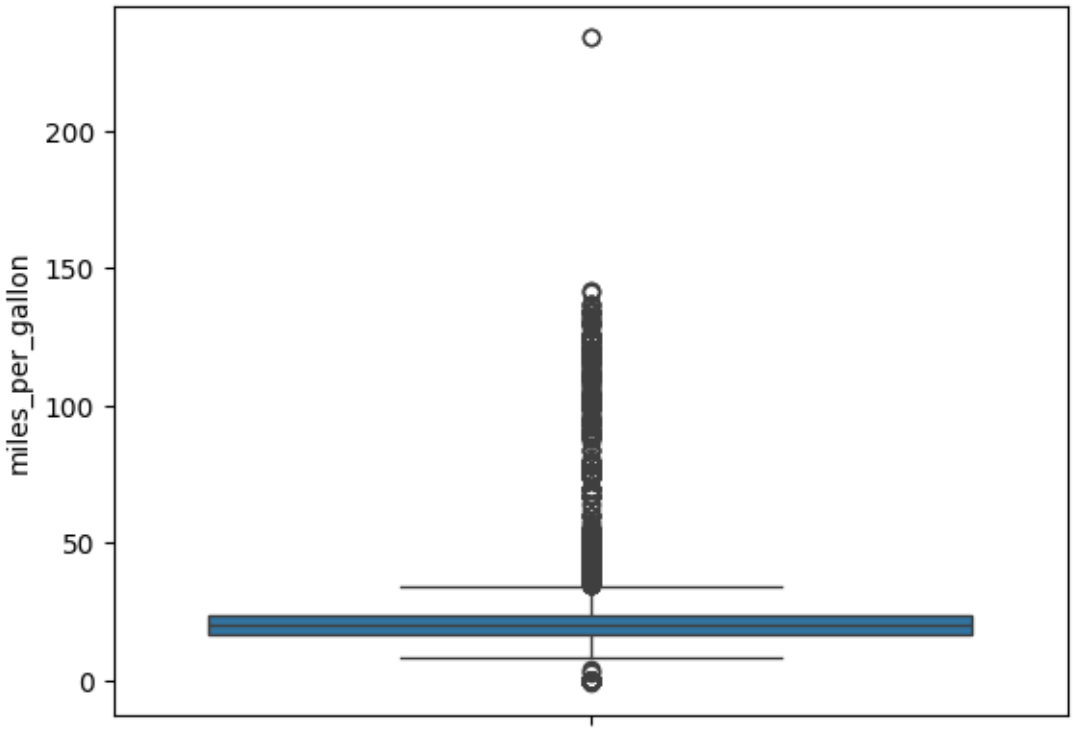
	model_year	brand	model	type	\
167	2007	Ford	E250 Cargo	Van	
635	2019	Ford	Mustang Mach-E Premium	SUV	
778	2018	Lincoln	AMG C 43 Base 4MATIC	Sedan	
830	2015	Tesla	Model S P100D	Sedan	
1271	2020	Rivian	R1S Adventure Package	SUV	
…	…	…	…	…	
27976	2009	Kia	Seltos S	SUV	
27983	2023	Acura	R1S Adventure Package	SUV	
27992	2009	Toyota	350Z Touring	Coupe	
28051	2013	Chevrolet	Bolt EV LT	Hatchback	
28106	2016	Polestar	2 Launch Edition	Sedan	

	miles_per_gallon	premium_version	price	collection_car	car_age
167	0.0	0	0.0	0	17
635	0.0	1	60000.0	0	5
778	0.0	0	0.0	0	6

```
830          0.0              1  134500.0              0        9
1271         0.0              1   70000.0              0        4
…            …                …        …              …        …
27976        0.0              0       0.0              0       15
27983        0.0              0       0.0              0        1
27992        0.0              0       0.0              0       15
28051        0.0              0   34995.0              0       11
28106        0.0              1       0.0              1        8

[447 rows x 9 columns]
```

[ ]:
```
sns.boxplot(data["miles_per_gallon"])
```

[ ]:
```
<Axes: ylabel='miles_per_gallon'>
```

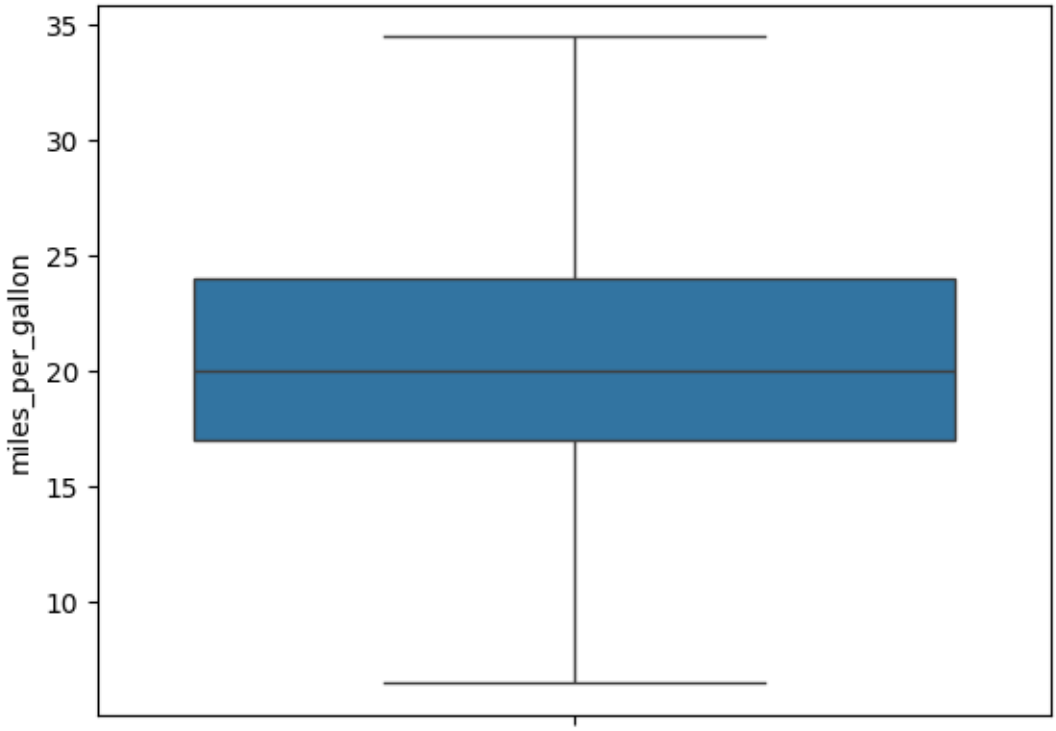


[ ]:
```
new_df_cap = data.copy()

new_df_cap['miles_per_gallon'] = np.where(
    new_df_cap['miles_per_gallon'] > upper_limit,
    upper_limit,
    np.where(
        new_df_cap['miles_per_gallon'] < lower_limit,
        lower_limit,
```

```
        new_df_cap['miles_per_gallon']
    )
)
```

[ ]:
```
sns.boxplot(new_df_cap["miles_per_gallon"])
```

[ ]:
```
<Axes: ylabel='miles_per_gallon'>
```



[ ]:
```
final_data =␣
 ↪new_df_cap[["brand","type","miles_per_gallon","premium_version","collection_car","car_age",
final_data.head()
```

[ ]:

	brand	type	miles_per_gallon	premium_version	collection_car	car_age	price
0	Toyota	SUV	13.0	1	0	8	84900.0
1	RAM	Van	15.0	0	0	10	35000.0
2	Ford	Coupe	16.0	0	0	22	26250.0
3	BMW	Sedan	27.0	1			
4	Mercedes-Benz	Convertible	18.0	1			

```
3                0        12   45000.0
4                1        16  100000.0
```

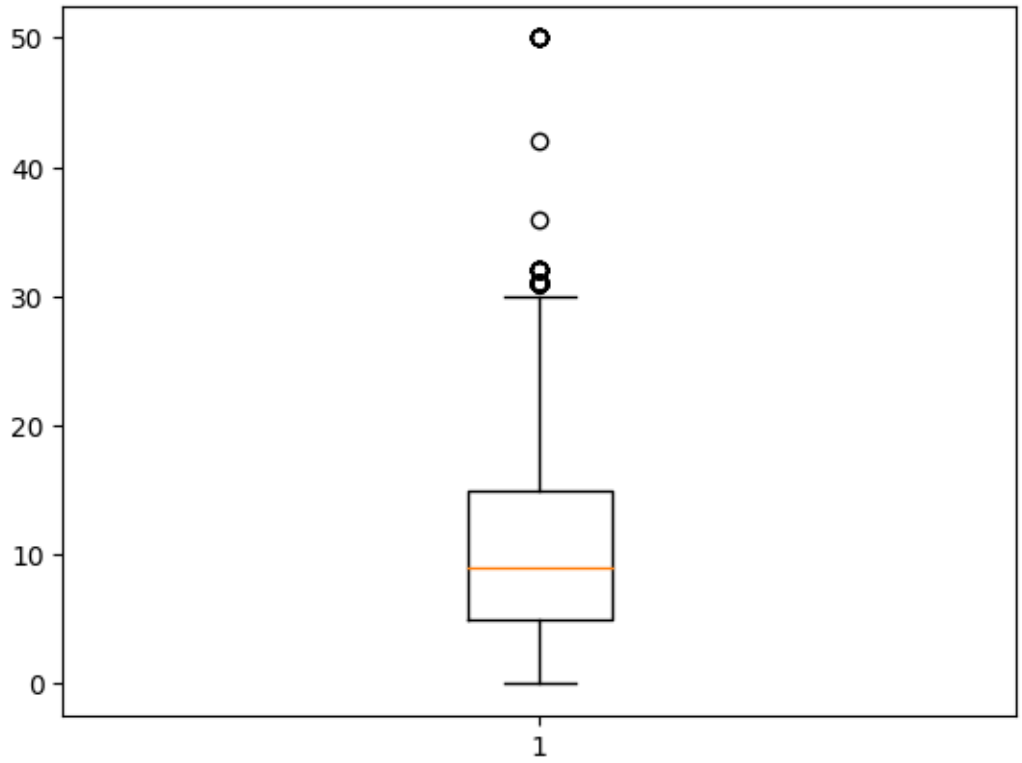
[ ]:
```
label_encoder = LabelEncoder()

final_data['brand'] = label_encoder.fit_transform(final_data['brand'])
final_data['type'] = label_encoder.fit_transform(final_data['type'])
```

```
<ipython-input-139-5cbdd1b59029>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  final_data['brand'] = label_encoder.fit_transform(final_data['brand'])
<ipython-input-139-5cbdd1b59029>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  final_data['type'] = label_encoder.fit_transform(final_data['type'])
```

[ ]:
```
final_data.head()
```

[ ]:
```
   brand  type  miles_per_gallon  premium_version  collection_car  car_age  \
0     53     5              13.0                1               0        8
1     44     7              15.0                0               0       10
2     14     1              16.0                0               0       22
3      4     6              27.0                1               0       12
4     36     0              18.0                1               1       16

      price
0   84900.0
1   35000.0
2   26250.0
3   45000.0
4  100000.0
```

[ ]:
```
plt.boxplot(final_data["car_age"])
plt.show()
```

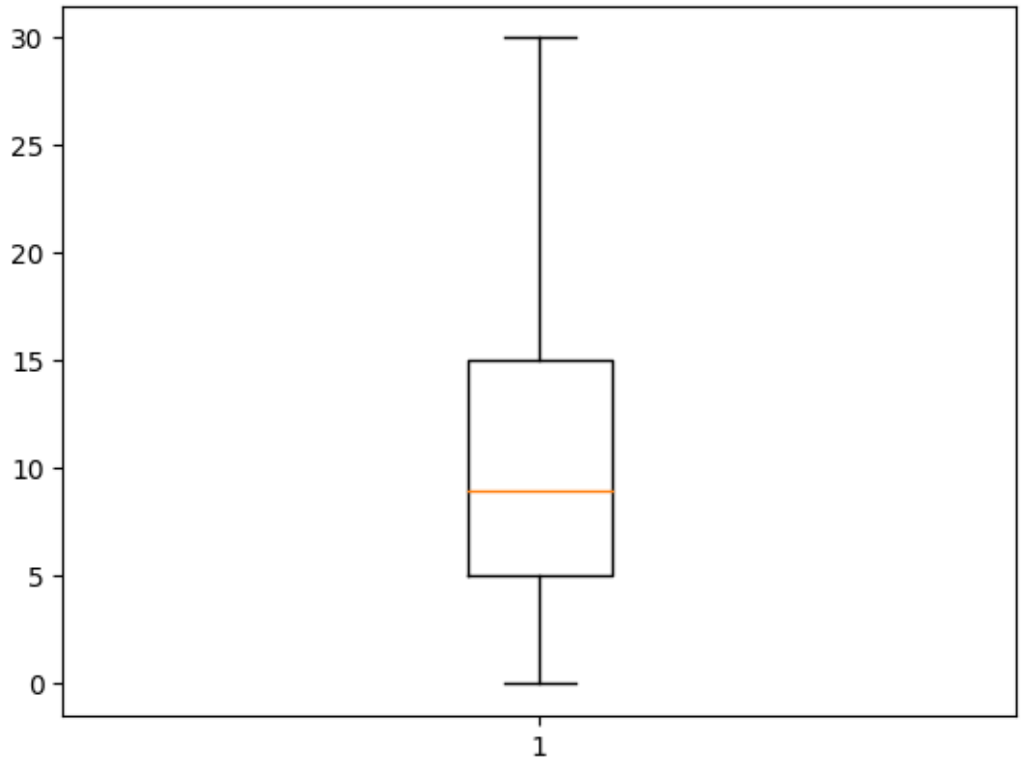


[ ]:
```
Q1 = final_data["car_age"].quantile(0.25)
Q3 = final_data["car_age"].quantile(0.75)
IQR = Q3 - Q1

upper_limit = Q3 + 1.5 * IQR
lower_limit = Q1 - 1.5 * IQR

final_data['car_age'] = np.where(
    final_data['car_age'] > upper_limit,
    upper_limit,
    np.where(
        final_data['car_age'] < lower_limit,
        lower_limit,
        final_data['car_age']
    )
)
```

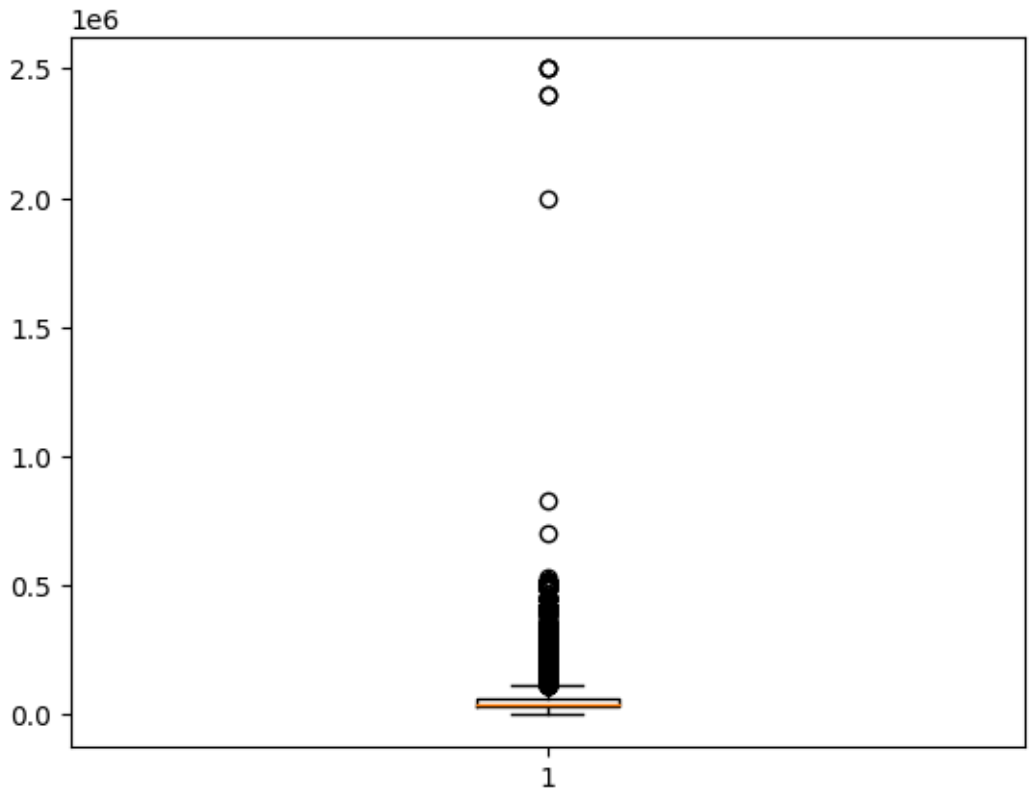
```
<ipython-input-142-a877fec76e8c>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  final_data['car_age'] = np.where(
```

[ ]:
```
plt.boxplot(final_data["car_age"])
plt.show()
```



[ ]:
```
plt.boxplot(final_data["price"])
plt.show()
```



[ ]:
```
X = final_data.iloc[:,:-1]
y = final_data.iloc[:,-1]
```

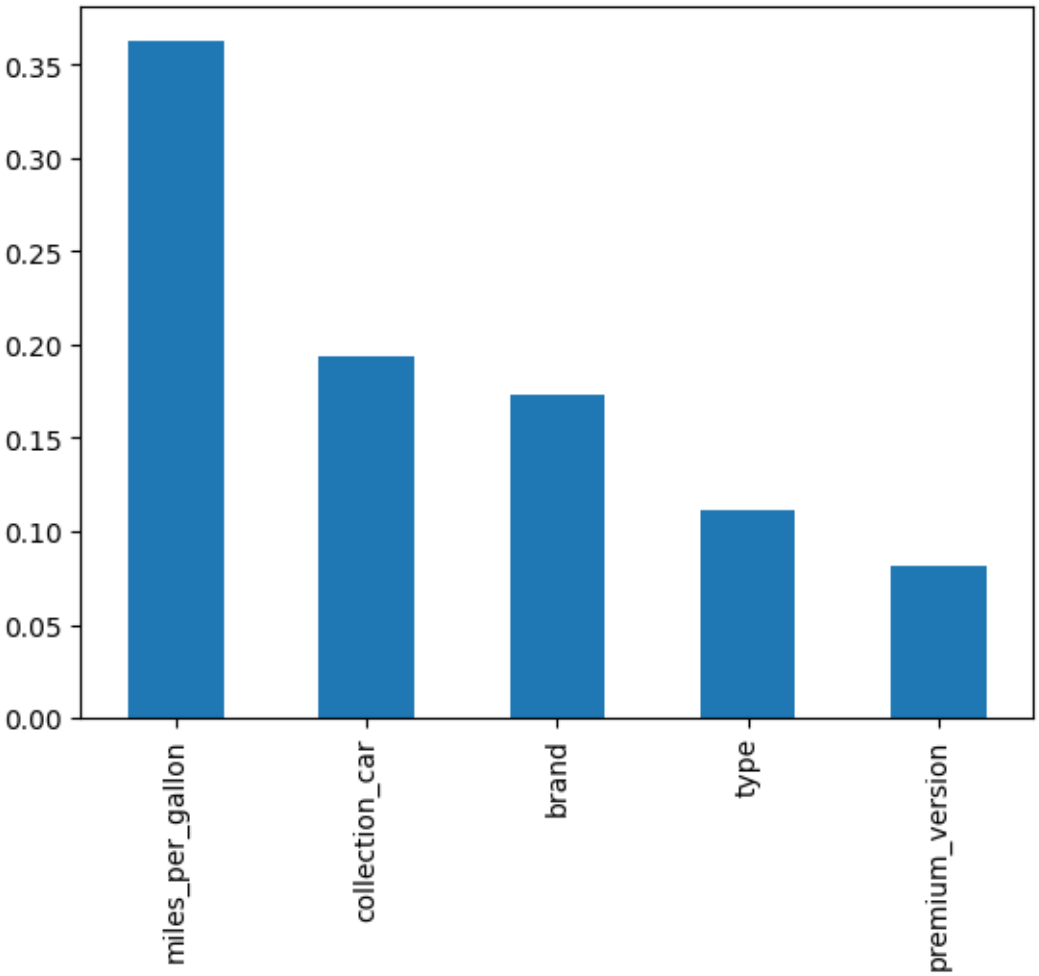
[ ]:
```
from sklearn.ensemble import ExtraTreesRegressor

e_tree_reg = ExtraTreesRegressor()
e_tree_reg.fit(X, y)
```

[ ]:
```
ExtraTreesRegressor()
```

[ ]:
```
print(e_tree_reg.feature_importances_)
```

```
[0.17356152 0.11162471 0.36279878 0.08095634 0.19336636 0.07769229]
```

[ ]:
```
#Identify important features in our data
f_imp = pd.Series(e_tree_reg.feature_importances_, index=X.columns)
f_imp.nlargest(5).plot(kind="bar")
plt.show()
```



[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

model = RandomForestRegressor()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)

acc = r2_score(y_test, y_pred)

print(f"Accuracy Score: {acc}")
```

0.7853381823327864

[ ]:
```
def data_preprocessing(data, upper_limit, lower_limit):
    data.dropna(inplace=True)
```

```
    data = data.rename(columns={"msrp":"suggested_price"})

    data['car_age'] = 2024 - data['model_year']

    new_df_cap = data.copy()

    new_df_cap['miles_per_gallon'] = np.where(
        new_df_cap['miles_per_gallon'] > upper_limit,
        upper_limit,
        np.where(
            new_df_cap['miles_per_gallon'] < lower_limit,
            lower_limit,
            new_df_cap['miles_per_gallon']
        )
    )

    final_data = new_df_cap[["brand", "type", "miles_per_gallon",␣
 ↪"premium_version", "collection_car", "car_age", "suggested_price"]]

    label_encoder = LabelEncoder()
    final_data['brand'] = label_encoder.fit_transform(final_data['brand'].
 ↪values)
    final_data['type'] = label_encoder.fit_transform(final_data['type'].values)

    return final_data

def model_selection(final_data, threshold=0.3):
    X = final_data.iloc[:, :-1].values
    y = final_data.iloc[:, -1].values

    X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪test_size=threshold, random_state=42)
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
    return X_train, X_test, y_train, y_test

def model_prediction(X_train, X_test, y_train, y_test):
    rf = RandomForestRegressor()

    n_estimators = [int(x) for x in np.linspace(start=100, stop=1200, num=12)]
    max_features = ['auto', 'sqrt']
    max_depth = [int(x) for x in np.linspace(start=5, stop=30, num=6)]
    min_samples_split = [2, 5, 10, 15,          100]
    min_samples_leaf = [1, 2, 5, 10]
```

```
    random_grid = {
        "n_estimators": n_estimators,
        "max_features": max_features,
        "max_depth": max_depth,
        "min_samples_split": min_samples_split,
        "min_samples_leaf": min_samples_leaf
    }

    rf_random = RandomizedSearchCV(estimator=rf,␣
 ↪param_distributions=random_grid,
                                   scoring="neg_mean_squared_error", n_iter=10,␣
 ↪cv=5, verbose=2, random_state=42)

    rf_random.fit(X_train, y_train)

    best_rf = rf_random.best_estimator_

    y_pred = best_rf.predict(X_test)

    accuracy = r2_score(y_test, y_pred)

    return accuracy, y_pred

if __name__ == "__main__":
    data = pd.read_csv("extended_data.csv")

    Q1 = data["miles_per_gallon"].quantile(0.75)
    Q3 = data["miles_per_gallon"].quantile(0.25)

    IQR = Q3-Q1

    upper_limit = Q3 + 1.5 * IQR
    lower_limit = Q1 - 1.5 * IQR

    data = data_preprocessing(data, upper_limit, lower_limit)

    X_train1, X_test1, y_train1, y_test1 = model_selection(data)

    accuracy, y_pred = model_prediction(X_train1, X_test1, y_train1, y_test1)

    print(accuracy)
```

```
<ipython-input-151-96a751c7f009>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  final_data['brand'] = label_encoder.fit_transform(final_data['brand'].values)
<ipython-input-151-96a751c7f009>:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  final_data['type'] = label_encoder.fit_transform(final_data['type'].values)

Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5,
min_samples_split=5, n_estimators=900; total time=   6.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5,
min_samples_split=5, n_estimators=900; total time=   5.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5,
min_samples_split=5, n_estimators=900; total time=   6.3s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5,
min_samples_split=5, n_estimators=900; total time=   5.3s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5,
min_samples_split=5, n_estimators=900; total time=   6.3s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2,
min_samples_split=10, n_estimators=1100; total time=   9.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2,
min_samples_split=10, n_estimators=1100; total time=  10.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2,
min_samples_split=10, n_estimators=1100; total time=   9.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2,
min_samples_split=10, n_estimators=1100; total time=   9.3s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2,
min_samples_split=10, n_estimators=1100; total time=  11.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5,
min_samples_split=100, n_estimators=300; total time=   0.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5,
min_samples_split=100, n_estimators=300; total time=   0.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5,
min_samples_split=100, n_estimators=300; total time=   0.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5,
min_samples_split=100, n_estimators=300; total time=   0.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5,
min_samples_split=100, n_estimators=300; total time=   0.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5,
min_samples_split=5, n_estimators=400; total time=   0.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5,
min_samples_split=5, n_estimators=400; total time=   0.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5,
min_samples_split=5, n_estimators=400; total time=   0.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5,
```

```
min_samples_split=5, n_estimators=400; total time=   0.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5,
min_samples_split=5, n_estimators=400; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10,
min_samples_split=5, n_estimators=700; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10,
min_samples_split=5, n_estimators=700; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10,
min_samples_split=5, n_estimators=700; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10,
min_samples_split=5, n_estimators=700; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10,
min_samples_split=5, n_estimators=700; total time=   0.0s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=1000; total time=  14.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=1000; total time=  14.8s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=1000; total time=  14.7s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=1000; total time=  14.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=1000; total time=  14.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10,
min_samples_split=15, n_estimators=1100; total time=   4.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10,
min_samples_split=15, n_estimators=1100; total time=   4.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10,
min_samples_split=15, n_estimators=1100; total time=   5.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10,
min_samples_split=15, n_estimators=1100; total time=   4.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10,
min_samples_split=15, n_estimators=1100; total time=   4.3s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1,
min_samples_split=15, n_estimators=300; total time=   3.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1,
min_samples_split=15, n_estimators=300; total time=   2.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1,
min_samples_split=15, n_estimators=300; total time=   2.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1,
min_samples_split=15, n_estimators=300; total time=   2.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1,
min_samples_split=15, n_estimators=300; total time=   2.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=10, n_estimators=700; total time=   3.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=10, n_estimators=700; total time=   2.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
```

```
min_samples_split=10, n_estimators=700; total time=   2.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=10, n_estimators=700; total time=   2.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=10, n_estimators=700; total time=   3.3s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=15, n_estimators=700; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=15, n_estimators=700; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=15, n_estimators=700; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=15, n_estimators=700; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=15, n_estimators=700; total time=   0.0s

/usr/local/lib/python3.10/dist-
packages/sklearn/model_selection/_validation.py:425: FitFailedWarning:
20 fits failed out of a total of 50.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
20 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-
packages/sklearn/model_selection/_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1145, in
wrapper
    estimator._validate_params()
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 638, in
_validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.10/dist-
packages/sklearn/utils/_param_validation.py", line 96, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of RandomForestRegressor must be an int in the range [1, inf), a float
in the range (0.0, 1.0], a str among {'sqrt', 'log2'} or None. Got 'auto'
instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:979:
```

```
UserWarning: One or more of the test scores are non-finite: [-2.33552584e+09
-2.12914880e+09             nan             nan
             nan -2.08839383e+09 -2.67947501e+09 -2.00869776e+09
 -2.67034990e+09             nan]
  warnings.warn(

Accuracy: 0.6212086746532638
```

Thank you