

```
In [1]: import pandas as pd
import numpy as np
from xgboost import XGBRegressor
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split, RandomizedSearchCV, GridSearchCV
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
import time
```

```
In [2]: # Read the dataset
file_path = r"D:\CV things\ML projects\audi.csv"
df = pd.read_csv(file_path)
print(df.shape)
df
```

(10668, 9)

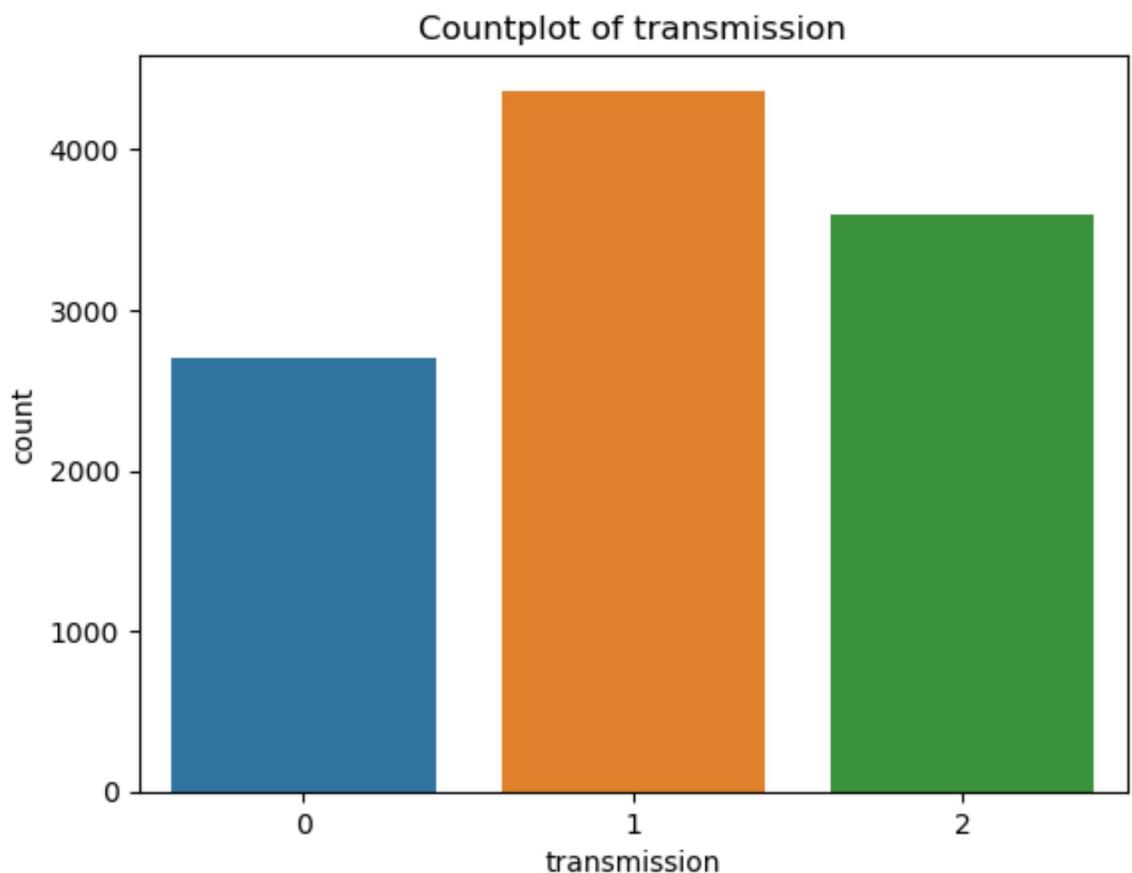
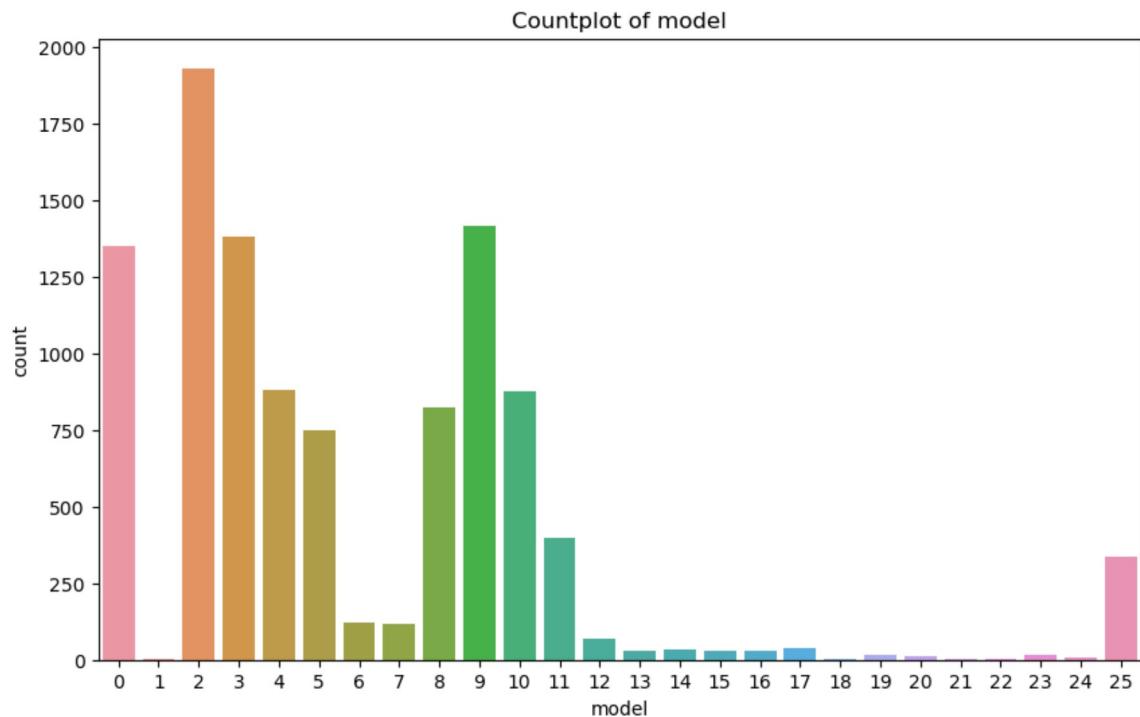
```
Out[2]:   model  year  price  transmission  mileage  fuelType  tax  mpg  engineSize
0      A1  2017  12500       Manual  15735  Petrol  150  55.4      1.4
1      A6  2016  16500    Automatic  36203  Diesel   20  64.2      2.0
2      A1  2016  11000       Manual  29946  Petrol   30  55.4      1.4
3      A4  2017  16800    Automatic  25952  Diesel  145  67.3      2.0
4      A3  2019  17300       Manual  1998  Petrol  145  49.6      1.0
...
10663     A3  2020  16999       Manual  4018  Petrol  145  49.6      1.0
10664     A3  2020  16999       Manual  1978  Petrol  150  49.6      1.0
10665     A3  2020  17199       Manual   609  Petrol  150  49.6      1.0
10666     Q3  2017  19499    Automatic  8646  Petrol  150  47.9      1.4
10667     Q3  2016  15999       Manual  11855  Petrol  150  47.9      1.4
```

10668 rows × 9 columns

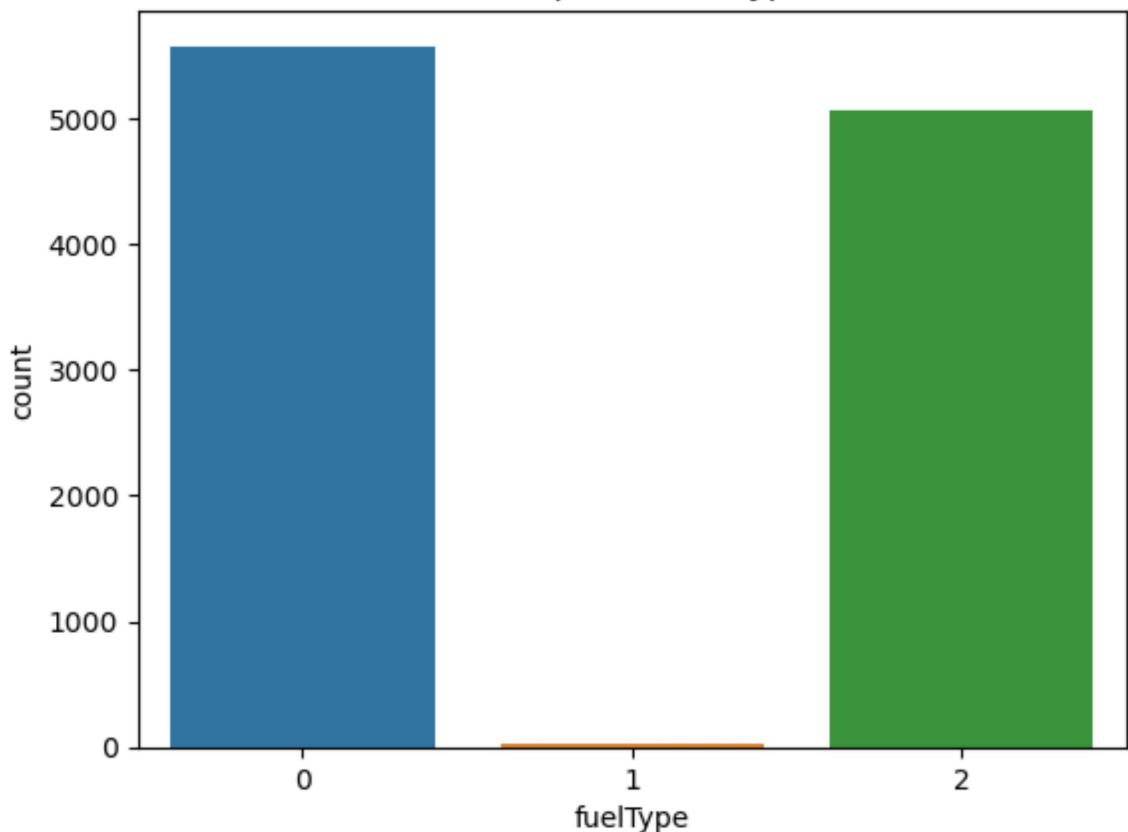
```
In [3]: # Label Encoding for categorical features
categorical_features = ['model', 'transmission', 'fuelType']
le = LabelEncoder()
df[categorical_features] = df[categorical_features].apply(lambda col: le.fit_transform(col))
df.head()
```

```
Out[3]:   model  year  price  transmission  mileage  fuelType  tax  mpg  engineSize
0      0  2017  12500       1  15735       2  150  55.4      1.4
1      5  2016  16500       0  36203       0   20  64.2      2.0
2      0  2016  11000       1  29946       2   30  55.4      1.4
3      3  2017  16800       0  25952       0  145  67.3      2.0
4      2  2019  17300       1  1998        2  145  49.6      1.0
```

```
In [4]: # Countplot for categorical variables
plt.figure(figsize=(10, 6))
for col in categorical_features:
    sns.countplot(data=df, x=col)
    plt.title(f'Countplot of {col}')
    plt.show()
```



Countplot of fuelType



```
In [5]: # Preprocessing
X = df.drop('price', axis=1)
y = df['price']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

```
In [6]: # R2 Score before Feature Engineering
xgb = XGBRegressor()

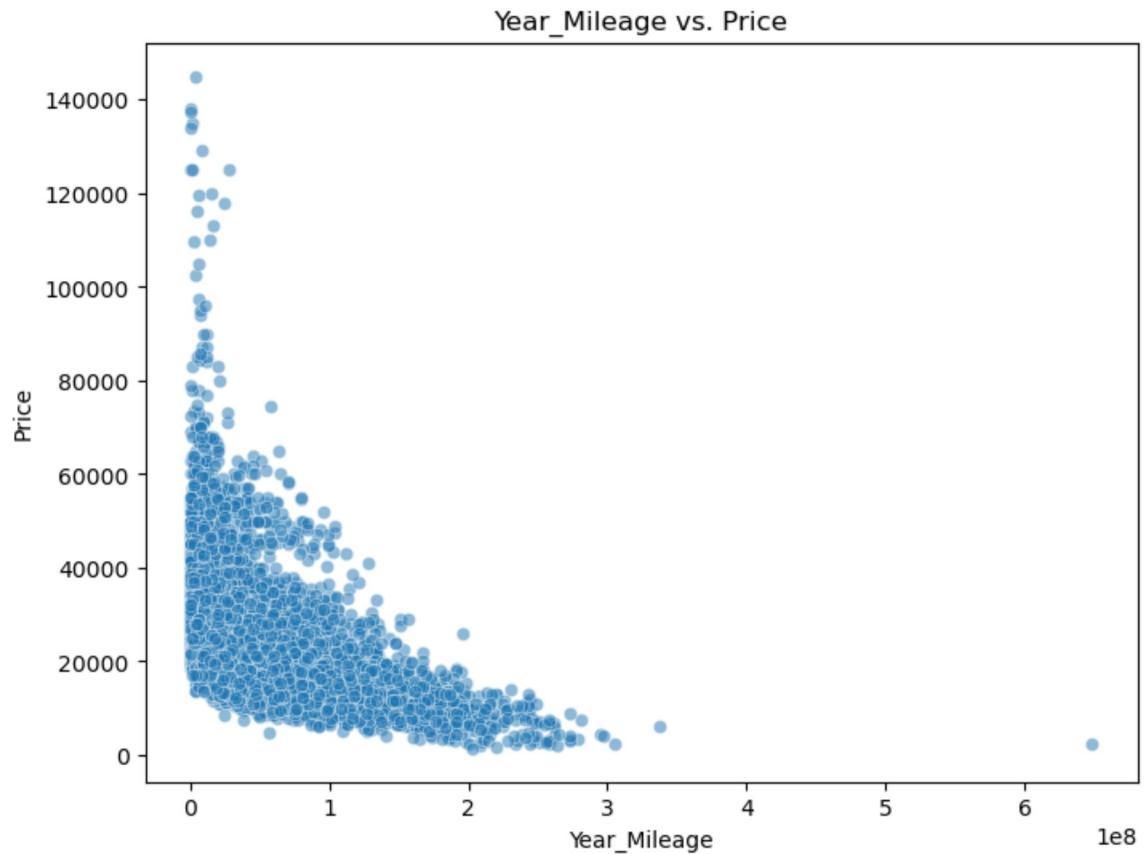
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
r2_base = r2_score(y_test, y_pred)
print(f"R2 Score before feature engineering: {r2_base}")
```

R2 Score before feature engineering: 0.9624305988307729

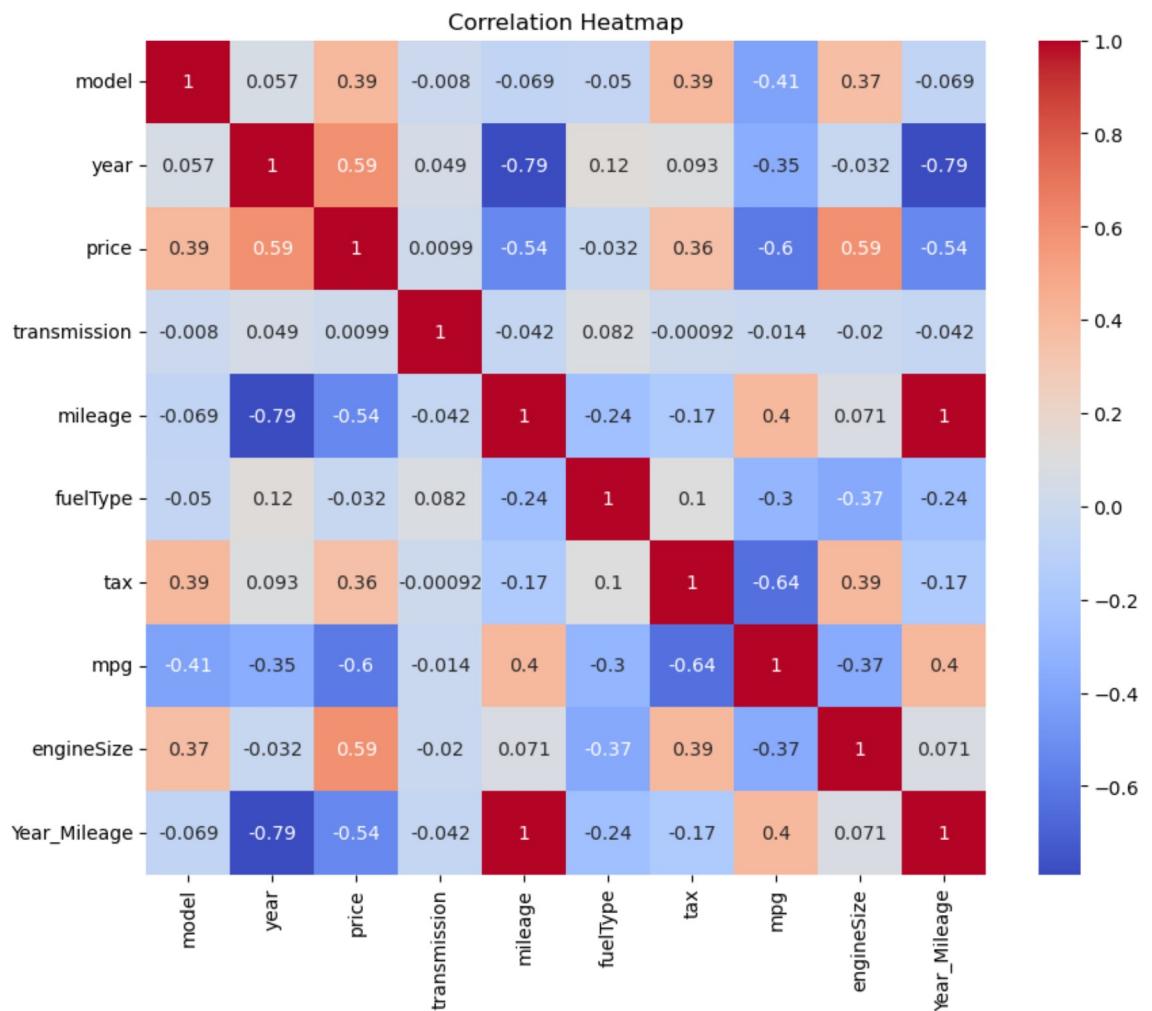
```
In [7]: # Feature Engineering - Creating a new feature 'Year_Mileage'
df['Year_Mileage'] = df['year'] * df['mileage']
df.head()
```

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize	Year_Mileage	
0	0	2017	12500		1	15735	2	150	55.4	1.4	31737495
1	5	2016	16500		0	36203	0	20	64.2	2.0	72985248
2	0	2016	11000		1	29946	2	30	55.4	1.4	60371136
3	3	2017	16800		0	25952	0	145	67.3	2.0	52345184
4	2	2019	17300		1	1998	2	145	49.6	1.0	4033962

```
In [8]: # Scatterplot showing Year_Mileage vs. Price after feature engineering
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='Year_Mileage', y='price', alpha=0.5, palette='viridis')
plt.title('Year_Mileage vs. Price')
plt.xlabel('Year_Mileage')
plt.ylabel('Price')
plt.show()
```



```
In [9]: # Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



```
In [10]: # Preprocessing
X = df.drop('price', axis=1)
y = df['price']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# R2 Score after Feature Engineering
xgb = XGBRegressor()
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
r2_base = r2_score(y_test, y_pred)

print(f"R2 Score after feature engineering: {r2_base}")
```

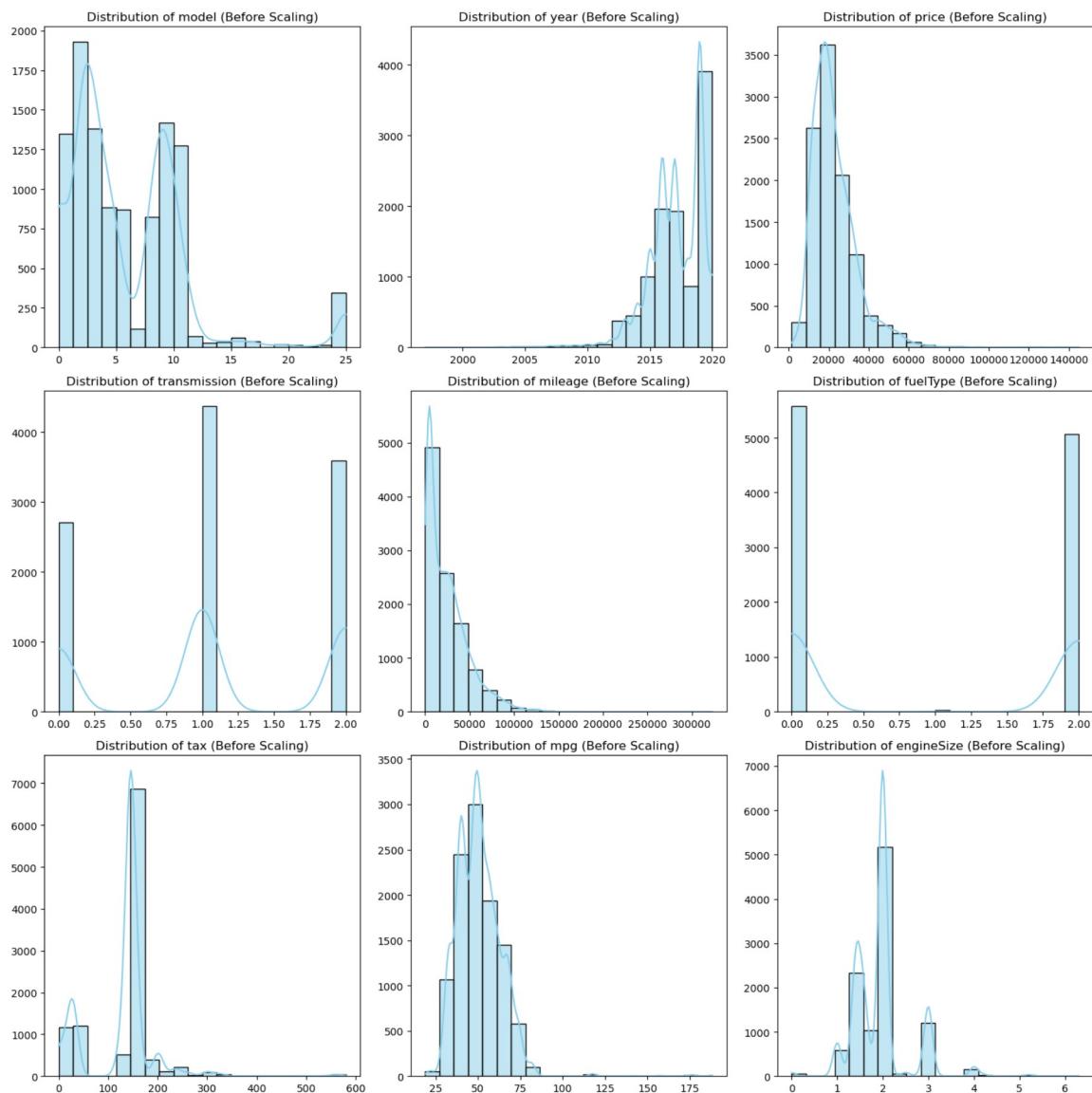
R2 Score after feature engineering: 0.9613934545410823

```
In [11]: # Get numerical columns and set the ones to display
numerical_columns = df.select_dtypes(include='number').columns.tolist()
num_cols_to_display = 9
num_cols = numerical_columns[:num_cols_to_display]

# Visualize distributions before scaling for each numerical feature separately
plt.figure(figsize=(15, 15))

for i, col in enumerate(num_cols):
    plt.subplot(3, 3, i+1)
    sns.histplot(df[col], bins=20, kde=True, color='skyblue')
    plt.title(f"Distribution of {col} (Before Scaling)")
    plt.xlabel('')
    plt.ylabel('')

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



```
In [12]: # Assuming df contains your dataset and numerical_columns are the columns you want to scale
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[numerical_columns])

# Create a DataFrame for scaled features
X_scaled_df = pd.DataFrame(X_scaled, columns=numerical_columns)
X_scaled_df['price'] = df['price'] # Include the target variable if needed

X = X_scaled_df.drop('price', axis=1)
y = X_scaled_df['price']

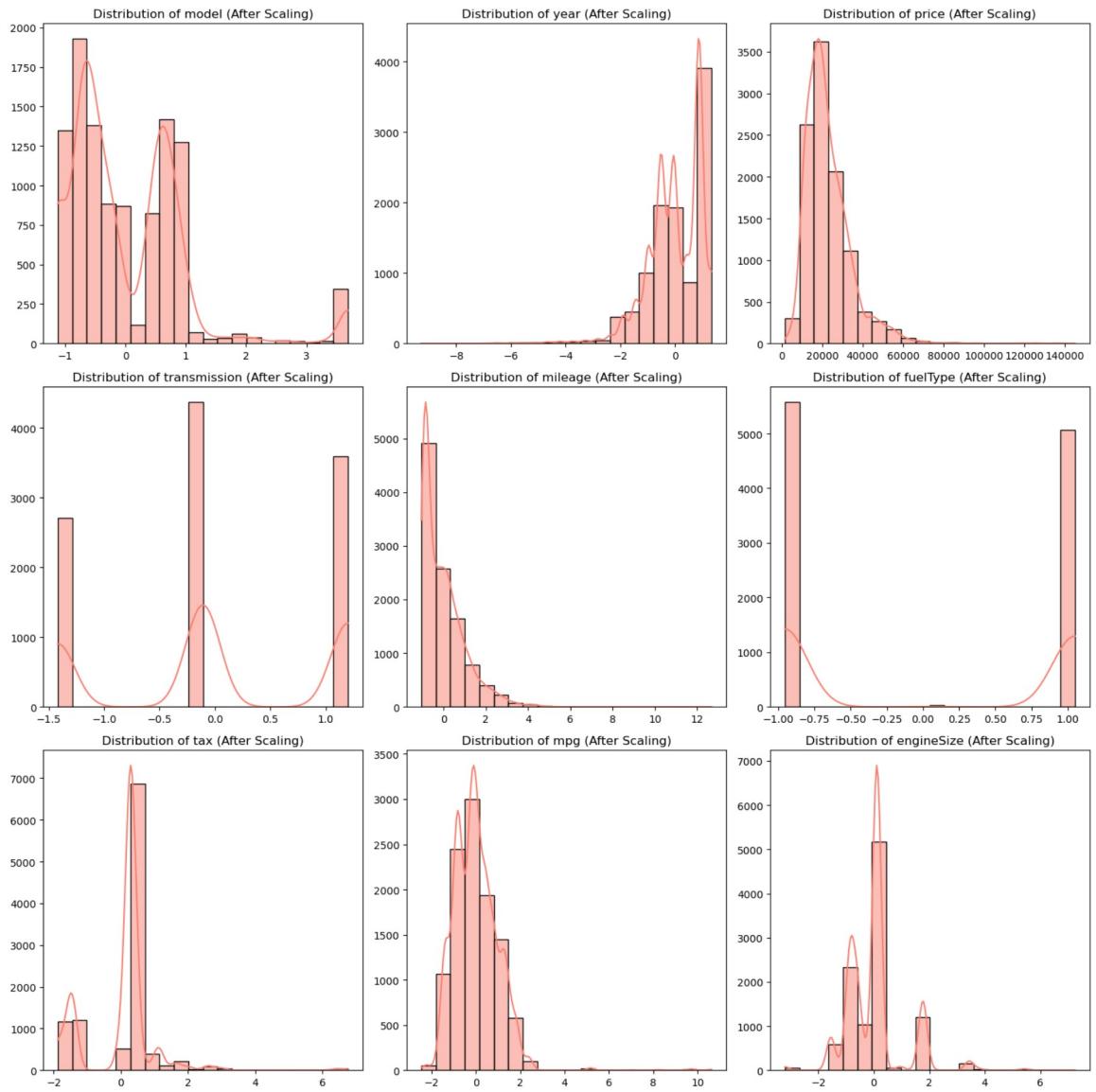
X_scaled_df.head()
```

```
Out[12]:   model      year    price transmission mileage fuelType      tax     mpg engine
0 -1.123544 -0.046450  12500     -0.108347 -0.386836  1.050783  0.357147  0.357550 -0.880
1 -0.160831 -0.507834  16500     -1.417348  0.483989 -0.954181 -1.578323  1.037130  0.110
2 -1.123544 -0.507834  11000     -0.108347  0.217781  1.050783 -1.429440  0.357550 -0.880
3 -0.545916 -0.046450  16800     -1.417348  0.047853 -0.954181  0.282706  1.276528  0.110
4 -0.738459  0.876318  17300     -0.108347 -0.971285  1.050783  0.282706 -0.090355 -1.540
```

```
In [13]: # Visualize distributions after scaling for each numerical feature separately
plt.figure(figsize=(15, 15))

for i, col in enumerate(num_cols):
    plt.subplot(3, 3, i+1)
    sns.histplot(X_scaled_df[col], bins=20, kde=True, color='salmon') # Adjust color
    plt.title(f"Distribution of {col} (After Scaling)")
    plt.xlabel('')
    plt.ylabel('')

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



```
In [14]: # Assuming you've performed the train-test split previously
X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled = train_test_split(
```

```
In [15]: # R2 Score after scaling
scaled_xgb = XGBRegressor()
scaled_xgb.fit(X_train_scaled, y_train_scaled)
y_pred_scaled = scaled_xgb.predict(X_test_scaled)
r2_scaled = r2_score(y_test_scaled, y_pred_scaled)
print(f"R2 Score after scaling: {r2_scaled}")
```

R2 Score after scaling: 0.9613934545410823

```
In [16]: # Hyperparameter tuning
param_dist = {
    'n_estimators': range(100, 300),
    'learning_rate': [0.01, 0.1, 0.2, 0.3],
    'max_depth': range(3, 8)
}

# Initialize XGBoost regressor and RandomizedSearchCV
xgb = XGBRegressor()

# RandomizedSearchCV
start_time_random = time.time()

random_search = RandomizedSearchCV(estimator=xgb, param_distributions=param_dist)

# Perform RandomizedSearchCV on the scaled training data
random_search.fit(X_train_scaled, y_train_scaled)

# Get the best estimator
best_xgb_random = random_search.best_estimator_

end_time_random = time.time()
random_search_time = end_time_random - start_time_random

# Predictions using the best model
y_pred_random = best_xgb_random.predict(X_test_scaled)
```

```
In [17]: # Print mean scores and standard deviations for different hyperparameter combinations
cv_results = random_search.cv_results_
for mean_score, std_score, params in zip(
    cv_results["mean_test_score"],
    cv_results["std_test_score"],
    cv_results["params"]
):
    print(f"Mean R2: {mean_score:.4f}, Std: {std_score:.4f} for {params}")
```

```
Mean R2: 0.8895, Std: 0.0035 for {'n_estimators': 151, 'max_depth': 7, 'learning_rate': 0.01}
Mean R2: 0.9585, Std: 0.0036 for {'n_estimators': 294, 'max_depth': 6, 'learning_rate': 0.3}
Mean R2: 0.8792, Std: 0.0027 for {'n_estimators': 228, 'max_depth': 3, 'learning_rate': 0.01}
Mean R2: 0.9394, Std: 0.0018 for {'n_estimators': 237, 'max_depth': 7, 'learning_rate': 0.01}
Mean R2: 0.9610, Std: 0.0044 for {'n_estimators': 166, 'max_depth': 4, 'learning_rate': 0.3}
Mean R2: 0.9591, Std: 0.0036 for {'n_estimators': 228, 'max_depth': 6, 'learning_rate': 0.3}
Mean R2: 0.9617, Std: 0.0028 for {'n_estimators': 124, 'max_depth': 6, 'learning_rate': 0.1}
Mean R2: 0.8467, Std: 0.0033 for {'n_estimators': 152, 'max_depth': 4, 'learning_rate': 0.01}
Mean R2: 0.9542, Std: 0.0023 for {'n_estimators': 114, 'max_depth': 4, 'learning_rate': 0.1}
Mean R2: 0.9599, Std: 0.0021 for {'n_estimators': 231, 'max_depth': 4, 'learning_rate': 0.1}
Mean R2: 0.9002, Std: 0.0019 for {'n_estimators': 210, 'max_depth': 4, 'learning_rate': 0.01}
Mean R2: 0.9297, Std: 0.0019 for {'n_estimators': 208, 'max_depth': 7, 'learning_rate': 0.01}
Mean R2: 0.9606, Std: 0.0033 for {'n_estimators': 176, 'max_depth': 4, 'learning_rate': 0.2}
Mean R2: 0.8640, Std: 0.0030 for {'n_estimators': 204, 'max_depth': 3, 'learning_rate': 0.01}
Mean R2: 0.9539, Std: 0.0022 for {'n_estimators': 213, 'max_depth': 3, 'learning_rate': 0.1}
Mean R2: 0.9620, Std: 0.0024 for {'n_estimators': 208, 'max_depth': 5, 'learning_rate': 0.1}
Mean R2: 0.9608, Std: 0.0032 for {'n_estimators': 165, 'max_depth': 5, 'learning_rate': 0.3}
Mean R2: 0.9573, Std: 0.0023 for {'n_estimators': 163, 'max_depth': 4, 'learning_rate': 0.1}
Mean R2: 0.9610, Std: 0.0034 for {'n_estimators': 204, 'max_depth': 4, 'learning_rate': 0.2}
Mean R2: 0.9619, Std: 0.0030 for {'n_estimators': 263, 'max_depth': 6, 'learning_rate': 0.2}
```

```
In [18]: # Evaluate the model
r2_random = r2_score(y_test_scaled, y_pred_random)
mae = mean_absolute_error(y_test_scaled, y_pred_random)
mse = mean_squared_error(y_test_scaled, y_pred_random)
rmse = np.sqrt(mse)

print("Best XGBoost Model Parameters:", random_search.best_params_)
```

```
Best XGBoost Model Parameters: {'n_estimators': 208, 'max_depth': 5, 'learning_rate': 0.1}
```

```
In [19]: print(f"R2 Score after RandomizedSearchCV: {r2_random}")

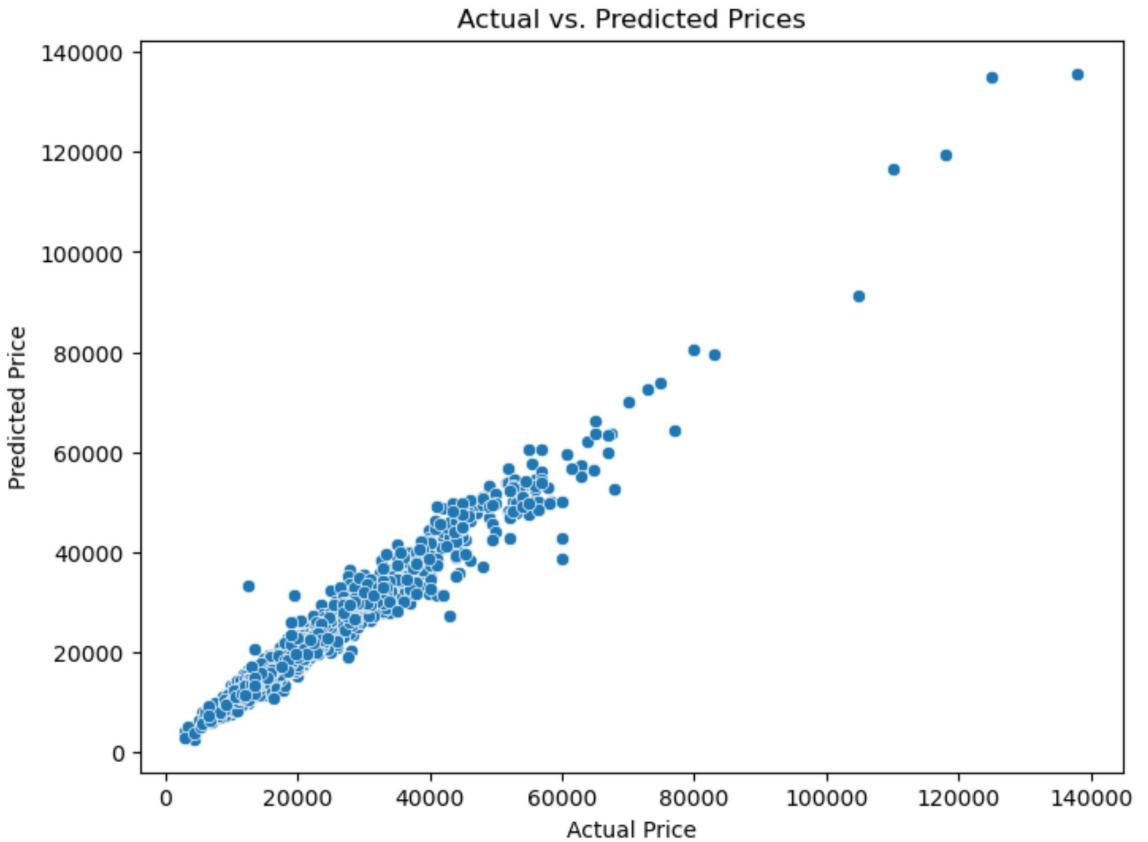
print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
```

```
R2 Score after RandomizedSearchCV: 0.9609913901632462
MAE: 1554.2503148430178
MSE: 5369702.057036524
RMSE: 2317.2617584201666
```

```
In [20]: # Residual plot
residuals = y_test_scaled - y_pred_random
plt.figure(figsize=(8, 6))
sns.residplot(x=y_pred_scaled, y=residuals, lowess=True, color='skyblue')
plt.title('Residual Plot')
plt.xlabel('Predicted')
plt.ylabel('Residuals')
plt.show()
```



```
In [21]: # Scatterplot for actual vs. predicted prices
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test_scaled, y=y_pred_random)
plt.title('Actual vs. Predicted Prices')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.show()
```



```
In [22]: # Define the parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100, 150, 200, 250, 300],
    'learning_rate': [0.01, 0.1, 0.2, 0.3],
    'max_depth': [3, 4, 5, 6, 7]
}

# Initialize XGBoost regressor
xgb = XGBRegressor()

# GridSearchCV
start_time_grid = time.time()

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=5, scoring=''

# Perform GridSearchCV on the scaled training data
grid_search.fit(X_train_scaled, y_train_scaled)

end_time_grid = time.time()
grid_search_time = end_time_grid - start_time_grid

# Get the best estimator
best_xgb_grid = grid_search.best_estimator_

# Predictions using the best model from GridSearchCV
y_pred_grid = best_xgb_grid.predict(X_test_scaled)
```

```
In [23]: # Print mean scores and standard deviations for different hyperparameter combinations
cv_results = grid_search.cv_results_
for mean_score, std_score, params in zip(
    cv_results["mean_test_score"],
    cv_results["std_test_score"],
    cv_results["params"]
):
    print(f"Mean R2: {mean_score:.4f}, Std: {std_score:.4f} for {params}")
```

Mean R2: 0.6913, Std: 0.0096 for {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100}
Mean R2: 0.8061, Std: 0.0051 for {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 150}
Mean R2: 0.8609, Std: 0.0031 for {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 200}
Mean R2: 0.8900, Std: 0.0025 for {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 250}
Mean R2: 0.9077, Std: 0.0025 for {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 300}
Mean R2: 0.7381, Std: 0.0053 for {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 100}
Mean R2: 0.8439, Std: 0.0035 for {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 150}
Mean R2: 0.8939, Std: 0.0019 for {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 200}
Mean R2: 0.9181, Std: 0.0017 for {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 250}
Mean R2: 0.9304, Std: 0.0018 for {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 300}
Mean R2: 0.7700, Std: 0.0045 for {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 100}
Mean R2: 0.8673, Std: 0.0028 for {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 150}
Mean R2: 0.9114, Std: 0.0025 for {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 200}
Mean R2: 0.9316, Std: 0.0028 for {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 250}
Mean R2: 0.9413, Std: 0.0030 for {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 300}
Mean R2: 0.7878, Std: 0.0069 for {'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 100}
Mean R2: 0.8813, Std: 0.0038 for {'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 150}
Mean R2: 0.9208, Std: 0.0022 for {'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 200}
Mean R2: 0.9393, Std: 0.0020 for {'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 250}
Mean R2: 0.9482, Std: 0.0022 for {'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 300}
Mean R2: 0.7973, Std: 0.0068 for {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 100}
Mean R2: 0.8884, Std: 0.0036 for {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 150}
Mean R2: 0.9261, Std: 0.0020 for {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 200}
Mean R2: 0.9424, Std: 0.0018 for {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 250}
Mean R2: 0.9502, Std: 0.0025 for {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 300}
Mean R2: 0.9439, Std: 0.0020 for {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}
Mean R2: 0.9498, Std: 0.0021 for {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 150}
Mean R2: 0.9533, Std: 0.0021 for {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}
Mean R2: 0.9552, Std: 0.0022 for {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 250}
Mean R2: 0.9569, Std: 0.0022 for {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 300}
Mean R2: 0.9530, Std: 0.0024 for {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 100}
Mean R2: 0.9566, Std: 0.0024 for {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 150}

```
        ators': 150}
Mean R2: 0.9590, Std: 0.0022 for {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 200}
Mean R2: 0.9605, Std: 0.0022 for {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 250}
Mean R2: 0.9615, Std: 0.0023 for {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 300}
Mean R2: 0.9582, Std: 0.0029 for {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100}
Mean R2: 0.9603, Std: 0.0026 for {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 150}
Mean R2: 0.9618, Std: 0.0024 for {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200}
Mean R2: 0.9626, Std: 0.0024 for {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 250}
Mean R2: 0.9630, Std: 0.0022 for {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 300}
Mean R2: 0.9608, Std: 0.0028 for {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 100}
Mean R2: 0.9621, Std: 0.0028 for {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 150}
Mean R2: 0.9628, Std: 0.0027 for {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 200}
Mean R2: 0.9630, Std: 0.0026 for {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 250}
Mean R2: 0.9630, Std: 0.0026 for {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 300}
Mean R2: 0.9601, Std: 0.0028 for {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 100}
Mean R2: 0.9608, Std: 0.0030 for {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 150}
Mean R2: 0.9610, Std: 0.0030 for {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 200}
Mean R2: 0.9607, Std: 0.0030 for {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 250}
Mean R2: 0.9604, Std: 0.0031 for {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 300}
Mean R2: 0.9519, Std: 0.0032 for {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 100}
Mean R2: 0.9554, Std: 0.0032 for {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 150}
Mean R2: 0.9573, Std: 0.0034 for {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 200}
Mean R2: 0.9586, Std: 0.0035 for {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 250}
Mean R2: 0.9596, Std: 0.0036 for {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 300}
Mean R2: 0.9575, Std: 0.0033 for {'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 100}
Mean R2: 0.9598, Std: 0.0032 for {'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 150}
Mean R2: 0.9609, Std: 0.0035 for {'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 200}
Mean R2: 0.9617, Std: 0.0033 for {'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 250}
Mean R2: 0.9620, Std: 0.0032 for {'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 300}
Mean R2: 0.9608, Std: 0.0027 for {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 100}
Mean R2: 0.9619, Std: 0.0027 for {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 150}
Mean R2: 0.9623, Std: 0.0025 for {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 200}
```

Mean R2: 0.9624, Std: 0.0025 for {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 250}
Mean R2: 0.9622, Std: 0.0026 for {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 300}
Mean R2: 0.9623, Std: 0.0030 for {'learning_rate': 0.2, 'max_depth': 6, 'n_estimators': 100}
Mean R2: 0.9626, Std: 0.0030 for {'learning_rate': 0.2, 'max_depth': 6, 'n_estimators': 150}
Mean R2: 0.9623, Std: 0.0030 for {'learning_rate': 0.2, 'max_depth': 6, 'n_estimators': 200}
Mean R2: 0.9619, Std: 0.0030 for {'learning_rate': 0.2, 'max_depth': 6, 'n_estimators': 250}
Mean R2: 0.9616, Std: 0.0031 for {'learning_rate': 0.2, 'max_depth': 6, 'n_estimators': 300}
Mean R2: 0.9600, Std: 0.0038 for {'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 100}
Mean R2: 0.9596, Std: 0.0039 for {'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 150}
Mean R2: 0.9592, Std: 0.0039 for {'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 200}
Mean R2: 0.9586, Std: 0.0041 for {'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 250}
Mean R2: 0.9581, Std: 0.0042 for {'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 300}
Mean R2: 0.9548, Std: 0.0026 for {'learning_rate': 0.3, 'max_depth': 3, 'n_estimators': 100}
Mean R2: 0.9581, Std: 0.0025 for {'learning_rate': 0.3, 'max_depth': 3, 'n_estimators': 150}
Mean R2: 0.9597, Std: 0.0025 for {'learning_rate': 0.3, 'max_depth': 3, 'n_estimators': 200}
Mean R2: 0.9604, Std: 0.0028 for {'learning_rate': 0.3, 'max_depth': 3, 'n_estimators': 250}
Mean R2: 0.9610, Std: 0.0027 for {'learning_rate': 0.3, 'max_depth': 3, 'n_estimators': 300}
Mean R2: 0.9592, Std: 0.0043 for {'learning_rate': 0.3, 'max_depth': 4, 'n_estimators': 100}
Mean R2: 0.9607, Std: 0.0045 for {'learning_rate': 0.3, 'max_depth': 4, 'n_estimators': 150}
Mean R2: 0.9612, Std: 0.0046 for {'learning_rate': 0.3, 'max_depth': 4, 'n_estimators': 200}
Mean R2: 0.9615, Std: 0.0046 for {'learning_rate': 0.3, 'max_depth': 4, 'n_estimators': 250}
Mean R2: 0.9615, Std: 0.0045 for {'learning_rate': 0.3, 'max_depth': 4, 'n_estimators': 300}
Mean R2: 0.9603, Std: 0.0035 for {'learning_rate': 0.3, 'max_depth': 5, 'n_estimators': 100}
Mean R2: 0.9608, Std: 0.0031 for {'learning_rate': 0.3, 'max_depth': 5, 'n_estimators': 150}
Mean R2: 0.9606, Std: 0.0030 for {'learning_rate': 0.3, 'max_depth': 5, 'n_estimators': 200}
Mean R2: 0.9604, Std: 0.0030 for {'learning_rate': 0.3, 'max_depth': 5, 'n_estimators': 250}
Mean R2: 0.9601, Std: 0.0030 for {'learning_rate': 0.3, 'max_depth': 5, 'n_estimators': 300}
Mean R2: 0.9603, Std: 0.0036 for {'learning_rate': 0.3, 'max_depth': 6, 'n_estimators': 100}
Mean R2: 0.9600, Std: 0.0033 for {'learning_rate': 0.3, 'max_depth': 6, 'n_estimators': 150}
Mean R2: 0.9594, Std: 0.0035 for {'learning_rate': 0.3, 'max_depth': 6, 'n_estimators': 200}
Mean R2: 0.9589, Std: 0.0036 for {'learning_rate': 0.3, 'max_depth': 6, 'n_estimators': 250}
Mean R2: 0.9584, Std: 0.0036 for {'learning_rate': 0.3, 'max_depth': 6, 'n_estimators': 300}

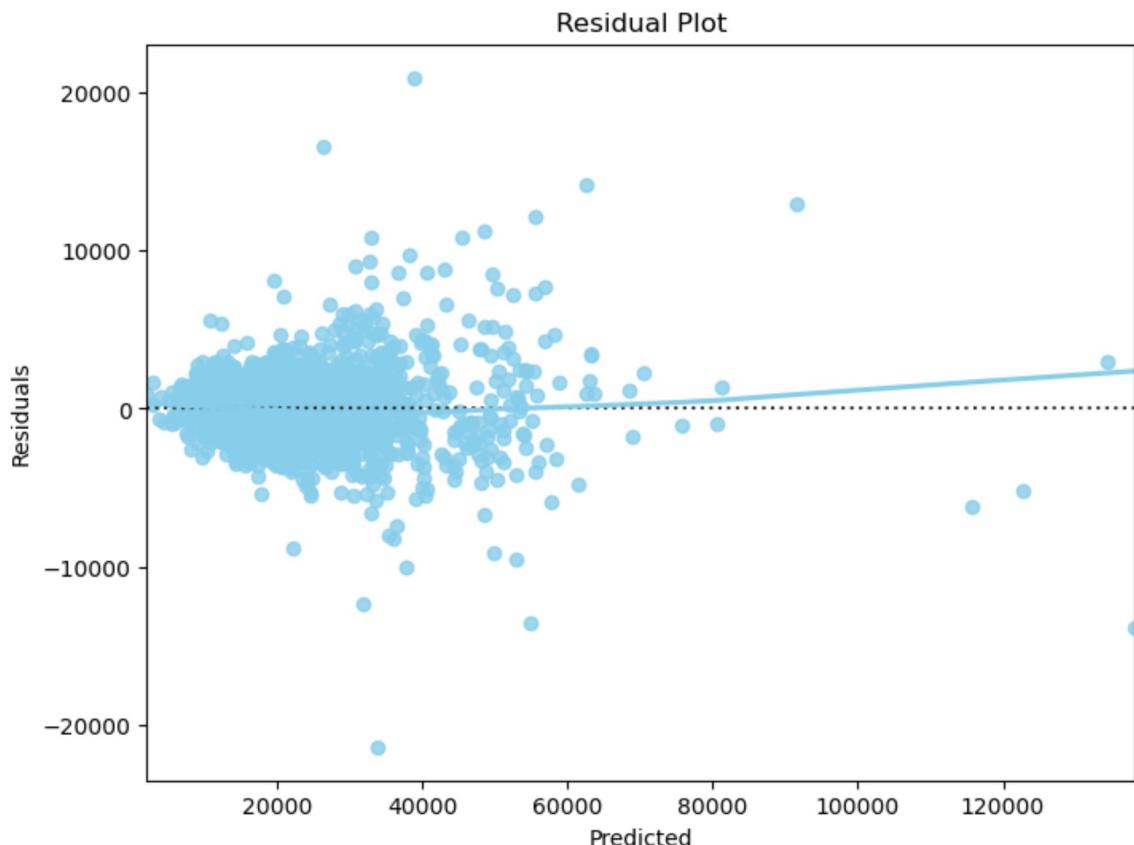
```
        ators': 300}
Mean R2: 0.9582, Std: 0.0042 for {'learning_rate': 0.3, 'max_depth': 7, 'n_estimators': 100}
Mean R2: 0.9573, Std: 0.0043 for {'learning_rate': 0.3, 'max_depth': 7, 'n_estimators': 150}
Mean R2: 0.9565, Std: 0.0044 for {'learning_rate': 0.3, 'max_depth': 7, 'n_estimators': 200}
Mean R2: 0.9560, Std: 0.0044 for {'learning_rate': 0.3, 'max_depth': 7, 'n_estimators': 250}
Mean R2: 0.9555, Std: 0.0045 for {'learning_rate': 0.3, 'max_depth': 7, 'n_estimators': 300}
```

```
In [24]: # Evaluate the model
r2_grid = r2_score(y_test_scaled, y_pred_grid)

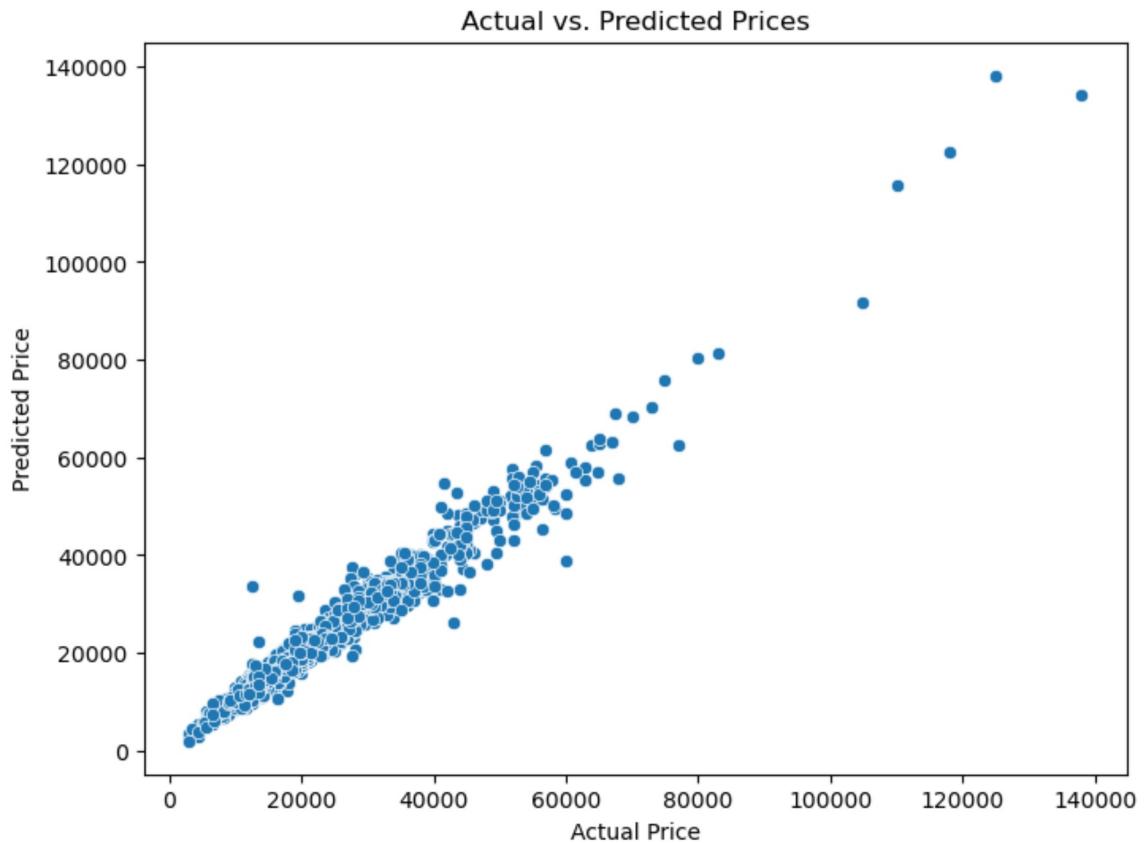
print("Best XGBoost Model Parameters (GridSearchCV):", grid_search.best_params_)
print(f'R2 Score after GridSearchCV: {r2_grid}')

Best XGBoost Model Parameters (GridSearchCV): {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 300}
R2 Score after GridSearchCV: 0.963084519548073
```

```
In [25]: # Residual plot
residuals = y_test_scaled - y_pred_grid
plt.figure(figsize=(8, 6))
sns.residplot(x=y_pred_grid, y=residuals, lowess=True, color='skyblue')
plt.title('Residual Plot')
plt.xlabel('Predicted')
plt.ylabel('Residuals')
plt.show()
```



```
In [26]: # Scatterplot for actual vs. predicted prices
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test_scaled, y=y_pred_grid)
plt.title('Actual vs. Predicted Prices')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.show()
```

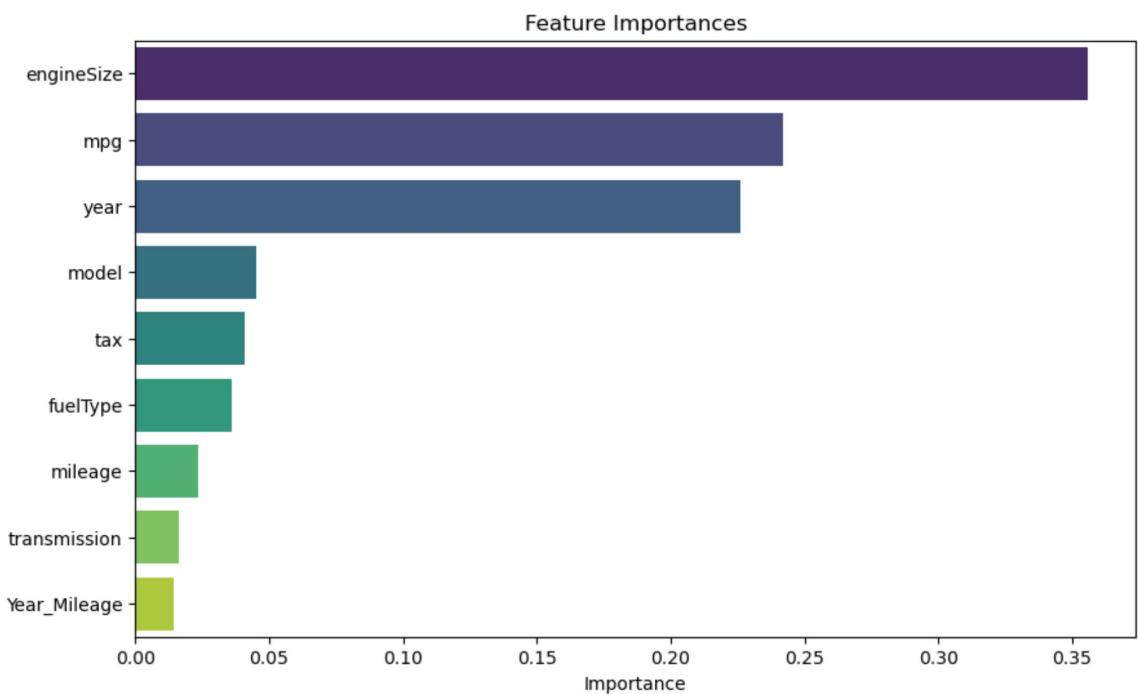


```
In [27]: print(f"RandomizedSearchCV took {random_search_time} seconds.")
print(f"GridSearchCV took {grid_search_time} seconds.")
```

RandomizedSearchCV took 10.409626483917236 seconds.
GridSearchCV took 53.03264260292053 seconds.

```
In [28]: # Display feature importance for the best model in decreasing order and different
plt.figure(figsize=(10, 6))
feat_importances = pd.Series(best_xgb_grid.feature_importances_, index=X.columns)
feat_importances = feat_importances.sort_values(ascending=False) # Sort in decr
sns.barplot(x=feat_importances.values, y=feat_importances.index, palette='viridi
plt.title('Feature Importances')
plt.xlabel('Importance')
```

```
Out[28]: Text(0.5, 0, 'Importance')
```



```
In [29]: # Select the top 5 features based on importance
top_features = feat_importances.index[:5]

# Extract only the top 5 features from the original dataset
X_top5 = X[top_features]

# Train-test split with the selected features
X_train_top5, X_test_top5, y_train_top5, y_test_top5 = train_test_split(X_top5,

# Initialize XGBoost regressor
xgb_top5 = XGBRegressor()

# Fit the model using the top 5 features
xgb_top5.fit(X_train_top5, y_train_top5)

# Make predictions
y_pred_top5 = xgb_top5.predict(X_test_top5)

# Evaluate the model with the top 5 features
r2_top5 = r2_score(y_test_top5, y_pred_top5)

print(f"R2 Score with the top 5 features: {r2_top5}")
```

R2 Score with the top 5 features: 0.9569746615504151