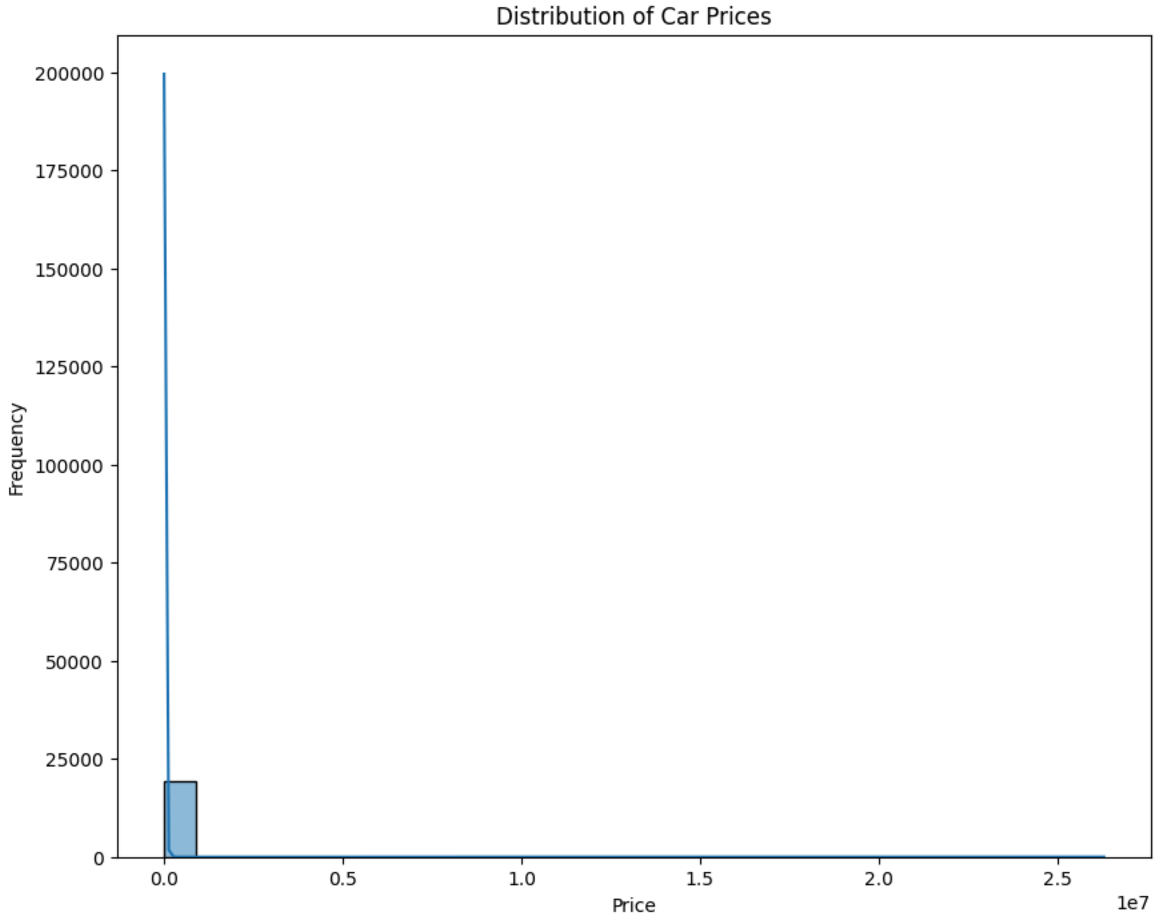
```
import pandas as pd

# Load the data
data = pd.read_csv('/content/car_price_prediction.xlsx - Sheet1.csv')
print(data.head())
print(data.info())
print(data.describe())
```

```
         ID  Price Manufacturer    Model  Prod. year   Category  \
0  45654403  13328        LEXUS   RX 450        2010       Jeep
1  44731507  16621    CHEVROLET  Equinox        2011       Jeep
2  45774419   8467        HONDA      FIT        2006  Hatchback
3  45769185   3607         FORD   Escape        2011       Jeep
4  45809263  11726        HONDA      FIT        2014  Hatchback

  Leather interior Fuel type Engine volume    Mileage  Cylinders  \
0              Yes    Hybrid           3.5  186005 km          6
1               No    Petrol             3  192000 km          6
2               No    Petrol           1.3  200000 km          4
3              Yes    Hybrid           2.5  168966 km          4
4              Yes    Petrol           1.3   91901 km          4

  Gear box type Drive wheels   Doors             Wheel   Color  Airbags
0     Automatic          4x4  04-May        Left wheel  Silver       12
1     Tiptronic          4x4  04-May        Left wheel   Black        8
2      Variator        Front  04-May  Right-hand drive   Black        2
3     Automatic          4x4  04-May        Left wheel   White        0
4     Automatic        Front  04-May        Left wheel  Silver        4
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19237 entries, 0 to 19236
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ID                19237 non-null  int64
 1   Price             19237 non-null  int64
 2   Manufacturer      19237 non-null  object
 3   Model             19237 non-null  object
 4   Prod. year        19237 non-null  int64
 5   Category          19237 non-null  object
 6   Leather interior  19237 non-null  object
 7   Fuel type         19237 non-null  object
 8   Engine volume     19237 non-null  object
 9   Mileage           19237 non-null  object
 10  Cylinders         19237 non-null  int64
 11  Gear box type     19237 non-null  object
 12  Drive wheels      19237 non-null  object
 13  Doors             19237 non-null  object
 14  Wheel             19237 non-null  object
 15  Color             19237 non-null  object
 16  Airbags           19237 non-null  int64
dtypes: int64(5), object(12)
memory usage: 2.5+ MB
None
                 ID         Price    Prod. year     Cylinders       Airbags
count  1.923700e+04  1.923700e+04  19237.000000  19237.000000  19237.000000
mean   4.557654e+07  1.855593e+04   2010.912824      4.582991      6.582627
std    9.365914e+05  1.905813e+05      5.668673      1.199933      4.320168
min    2.074688e+07  1.000000e+00   1939.000000      1.000000      0.000000
25%    4.569837e+07  5.331000e+03   2009.000000      4.000000      4.000000
50%    4.577231e+07  1.317200e+04   2012.000000      4.000000      6.000000
75%    4.580204e+07  2.207500e+04   2015.000000      4.000000     12.000000
max    4.581665e+07  2.630750e+07   2020.000000     16.000000     16.000000
```

Univariate Analysis

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 8))
sns.histplot(data['Price'], bins=30, kde=True)
plt.title('Distribution of Car Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```



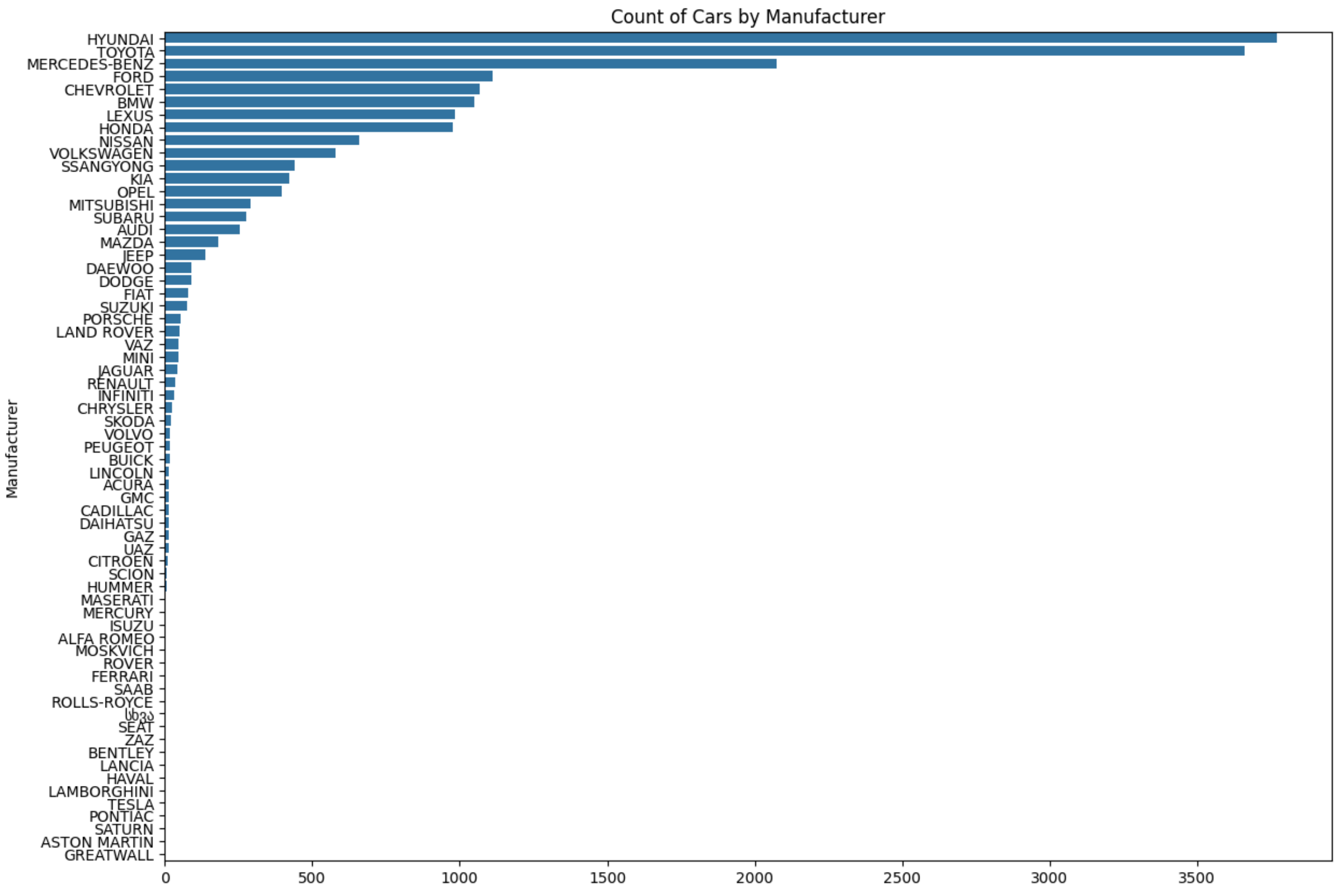
numerical features

```
# Summary statistics for numerical features
print(data[['Prod. year', 'Mileage']].describe())
```

```
        Prod. year
count  19237.000000
mean    2010.912824
std        5.668673
min     1939.000000
25%     2009.000000
50%     2012.000000
75%     2015.000000
max     2020.000000
```

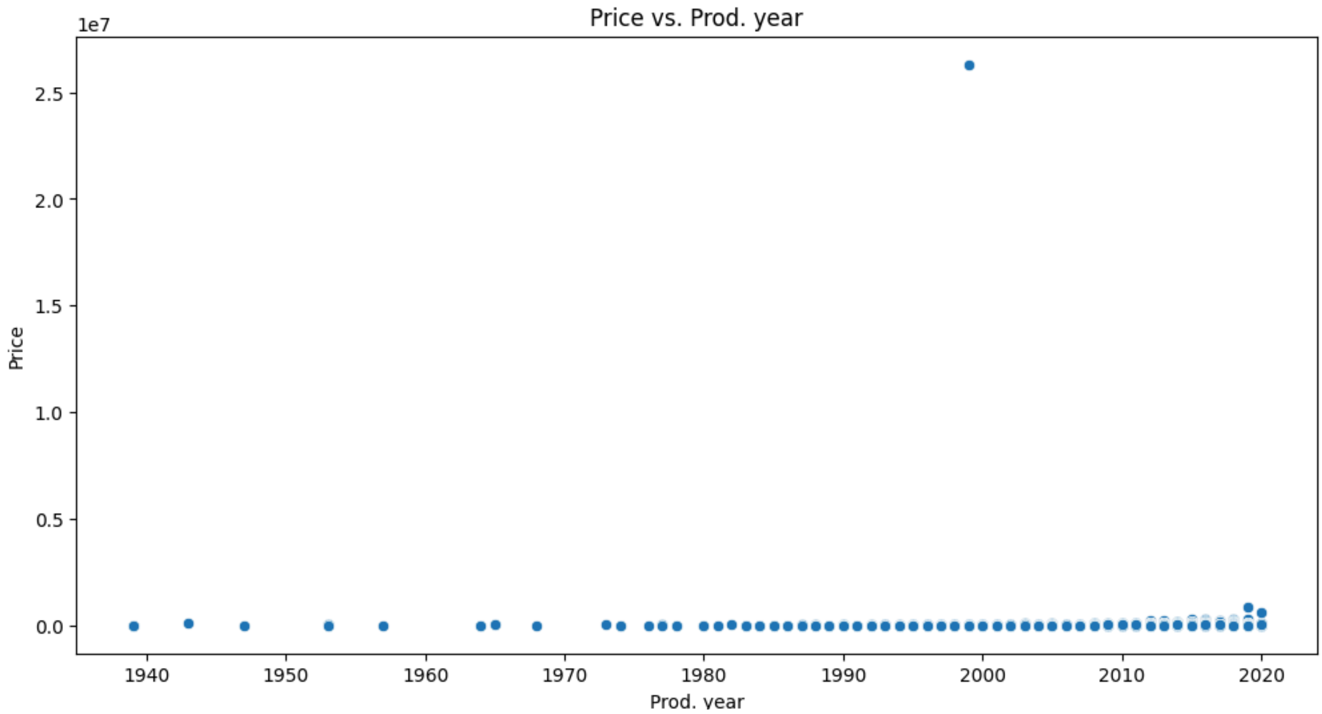
Categorical Features

```
# Countplot for categorical features
plt.figure(figsize=(14, 10))
sns.countplot(y='Manufacturer', data=data, order=data['Manufacturer'].value_counts().index)
plt.title('Count of Cars by Manufacturer')
plt.show()
```



Bivariate Analysis

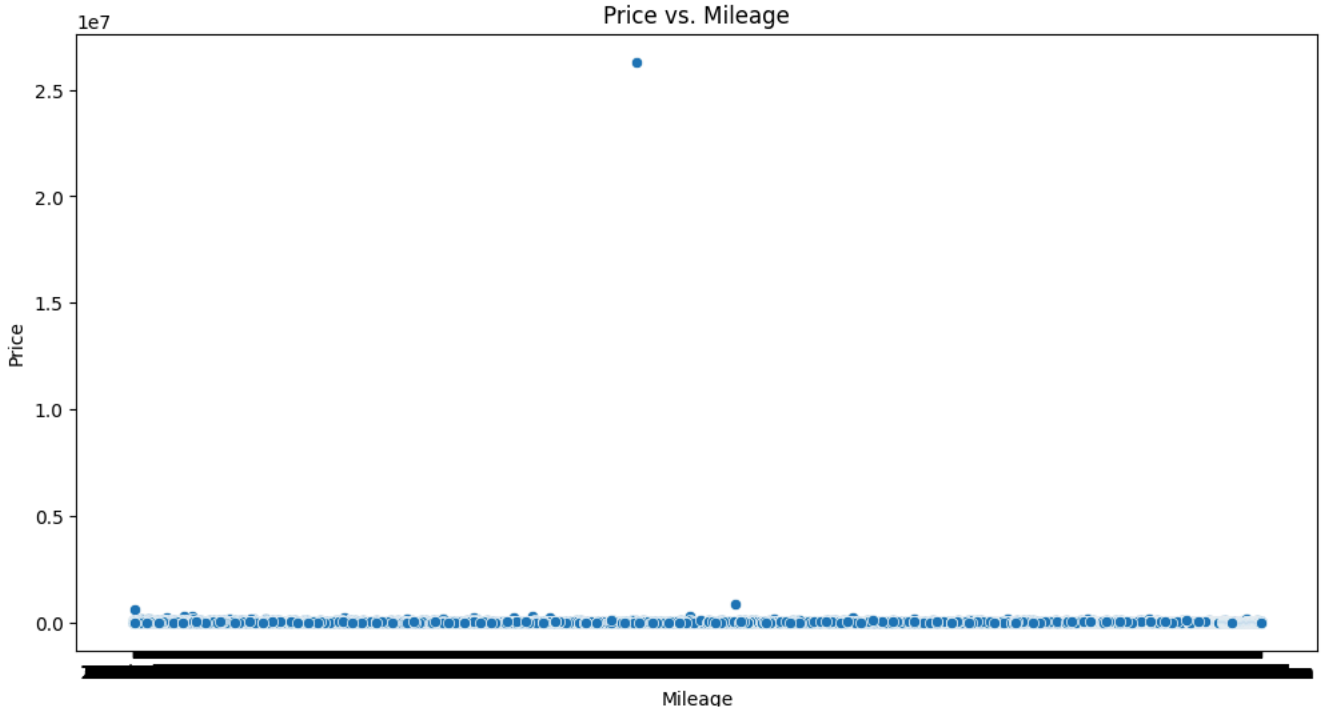
```
#price vs year
plt.figure(figsize=(12, 6))
sns.scatterplot(x='Prod. year', y='Price', data=data)
plt.title('Price vs. Prod. year')
plt.xlabel('Prod. year')
plt.ylabel('Price')
plt.show()
```



Price vs. Mileage

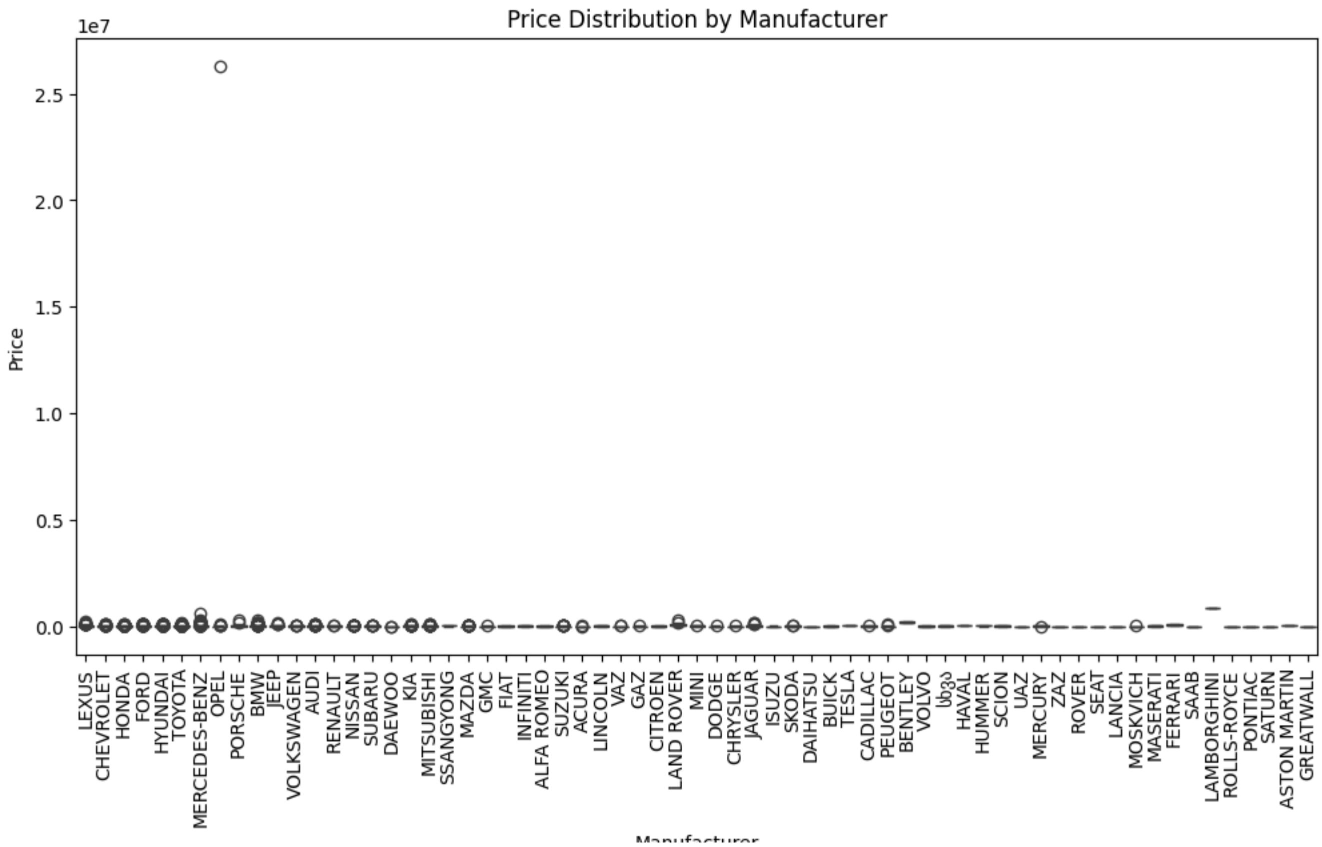
```
plt.figure(figsize=(12, 6))
sns.scatterplot(x='Mileage', y='Price', data=data)
plt.title('Price vs. Mileage')
```

```
plt.xlabel('Mileage')
plt.ylabel('Price')
plt.show()
```



Price vs. Brand

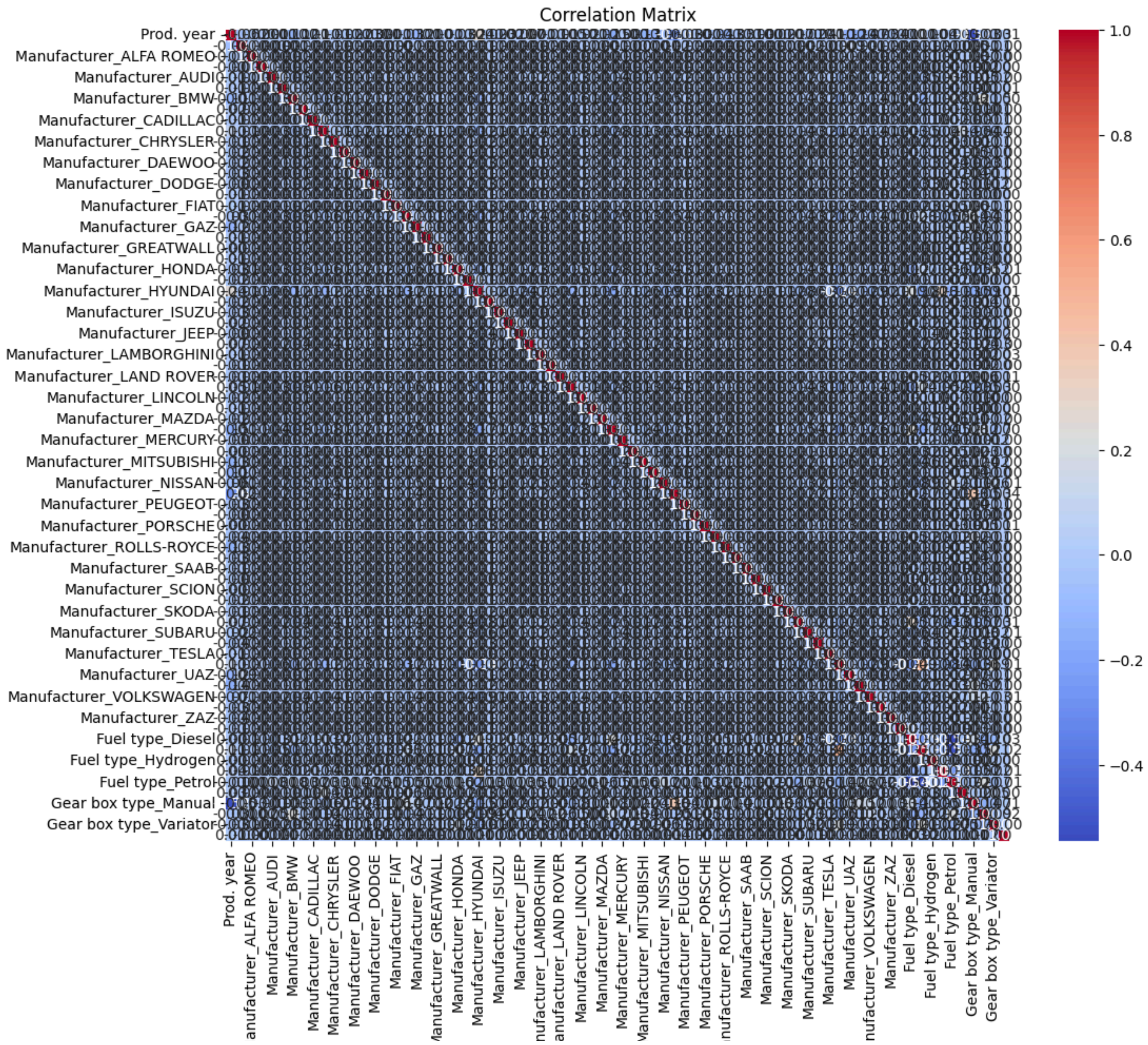
```
plt.figure(figsize=(12, 6))
sns.boxplot(x='Manufacturer', y='Price', data=data)
plt.title('Price Distribution by Manufacturer')
plt.xticks(rotation=90)
plt.show()
```



Multivariate Analysis Correlation Matrix

```
# Convert categorical features to numerical for correlation matrix
data_encoded = pd.get_dummies(data[['Prod. year', 'Mileage', 'Manufacturer', 'Fuel type', 'Gear box type']], drop_first=True)
```

```
data_encoded['Price'] = data['Price']

correlation_matrix = data_encoded.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



Data Preprocessing Handling Missing Values

```
# Check for missing values
print(data.isnull().sum())

# Example: Impute missing values if necessary
# Remove ' km' and convert to numeric before calculating the median
data['Mileage'] = data['Mileage'].str.replace(' km', '').astype(float)
data['Mileage'].fillna(data['Mileage'].median(), inplace=True)
```

```
ID                  0
Price               0
Manufacturer        0
Model               0
Prod. year          0
Category            0
Leather interior    0
Fuel type           0
Engine volume       0
Mileage             0
Cylinders           0
Gear box type       0
Drive wheels        0
Doors               0
Wheel               0
Color               0
```

```
Airbags             0
dtype: int64
```

Encoding Categorical Variables

```
# One-hot encoding
data_encoded = pd.get_dummies(data, columns=['Manufacturer', 'Fuel type', 'Gear box type'], drop_first=True)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data_encoded[['Prod. year', 'Mileage']] = scaler.fit_transform(data_encoded[['Prod. year', 'Mileage']])
```

Feature Engineering

```
# Example: Age of the car
data_encoded['Car_Age'] = 2024 - data_encoded['Prod. year']
data_encoded.drop(columns=['Prod. year'], inplace=True)
```

Model Building Train-Test Split

```
from sklearn.model_selection import train_test_split

X = data_encoded.drop(columns=['Price'])
y = data_encoded['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Train Models Linear Regression

```
# Identify categorical columns
categorical_cols = X_train.select_dtypes(include=['object']).columns

# Apply one-hot encoding
X_train = pd.get_dummies(X_train, columns=categorical_cols)
X_test = pd.get_dummies(X_test, columns=categorical_cols)

# Ensure both train and test sets have the same columns after encoding
X_train, X_test = X_train.align(X_test, join='outer', axis=1, fill_value=0)

# Create and train the model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Make predictions and evaluate the model
y_pred_lr = lr_model.predict(X_test)
print("Linear Regression RMSE:", mean_squared_error(y_test, y_pred_lr, squared=False))
print("Linear Regression R^2:", r2_score(y_test, y_pred_lr))
```

```
Linear Regression RMSE: 48157.473964406
Linear Regression R^2: -6.442771164243821
```

Random Forest

```
from sklearn.ensemble import RandomForestRegressor

rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

print("Random Forest RMSE:", mean_squared_error(y_test, y_pred_rf, squared=False))
print("Random Forest R^2:", r2_score(y_test, y_pred_rf))
```

```
Random Forest RMSE: 9323.21433902538
Random Forest R^2: 0.7210421374440481
```

Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(estimator=RandomForestRegressor(random_state=42),
                           param_grid=param_grid,
                           cv=3,
                           scoring='neg_mean_squared_error',
                           n_jobs=-1)
grid_search.fit(X_train, y_train)

#print("Best Parameters:", grid_search.best_params_)
#best_model = grid_search.best_estimator_
#y_pred_best = best_model.predict(X_test)

#print("Best Model RMSE:", mean_squared_error(y_test, y_pred_best, squared=False))
#print("Best Model R^2:", r2_score(y_test, y_pred_best))
```

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV

# Define the model
model = GradientBoostingRegressor()

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5],
    'learning_rate': [0.01, 0.1]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=model,
                           param_grid=param_grid,
                           cv=3,
                           scoring='neg_mean_squared_error',
                           n_jobs=-1)

# Fit the model
grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_

# Evaluate the best model
y_pred_best = best_model.predict(X_test)
print("Best Model RMSE:", mean_squared_error(y_test, y_pred_best, squared=False))
print("Best Model R^2:", r2_score(y_test, y_pred_best))
```