Project Report:

Used Car Price Prediction

Team Members:

Nicy Prince (C0924777): Model Experimentation, and Report Writing

Sona Stephan (C0928473): Data Visualization and Interactive Dashboard Creation

Vishnu Chandrasekharan (C0924468): Model Selection

Joel George (C0927062): Data Selection, Performance Analysis and Presentation

Tarun Samuel (C0928591): Front-End Development and Cloud Deployment

Introduction

Project Overview

This project aims to develop a comprehensive machine learning solution using a dataset with over 20,000 observations. The primary objective is to analyse the dataset, create visualizations, experiment with multiple machine learning models, develop a front-end interface, and deploy the solution on a cloud platform. The final output is intended to be a user-friendly application that provides valuable insights and predictions.

Data Selection and Preparation

Dataset Overview

The dataset is a car sales dataset containing 852,122 rows and 8 columns, with the primary goal of predicting the price of a car based on various features. The dataset includes the following columns:

1. Price (Target Variable): The car's selling price.
2. Year: The year the car was manufactured.
3. Mileage: The car's total distance travelled in kilometres.
4. City: The city where the car was sold.
5. State: The state where the car was sold.
6. Vin: The Vehicle Identification Number, unique to each car.
7. Make: The car's manufacturer (e.g., Toyota, Ford, Honda).
8. Model: The specific model of the car.

In [83]:

```
import pandas as pd
```

Load the dataset from a CSV file and preview the first few rows to understand its structure and contents.

In [84]:

```
df=pd.read_csv("C:/Users/vishn/Downloads/dataset_processed.csv")
df.head()
```

Out[84]:

	Price	Year	Mileage	City	State	Vin	Make	Model
0	29833	2016	55	Tinley Park	IL	04WT3N56GG0646582	Buick	CascadaPremium
1	29833	2016	56	Tinley Park	IL	04WT3N59GG1261202	Buick	CascadaPremium
2	67488	2002	84310	Peoria	AZ	137FA84322E198163	HUMMER	H14-Passenger
3	77995	2004	51651	Hampstead	MD	137FA84374E208897	HUMMER	H14-Passenger
4	89999	2001	49100	Houston	TX	137FA843X1E195221	AM	General

Initial Exploration

The initial exploration involved loading the dataset and previewing its structure using df.head(), df.describe(), and df.info(). Key observations included:

In [85]:

```
#Initial exploration to understand the data structure, types of variables
df.describe()
```

Out[85]:

	Price	Year	Mileage
count	852075.000000	852075.000000	8.520750e+05
mean	21464.278569	2013.289087	5.250859e+04
std	13596.183552	3.415039	4.198948e+04
min	1500.000000	1997.000000	5.000000e+00
25%	13000.000000	2012.000000	2.383600e+04
50%	18500.000000	2014.000000	4.025600e+04
75%	26995.000000	2016.000000	7.218700e+04
max	499500.000000	2018.000000	2.856196e+06

```
In [86]:
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 852075 entries, 0 to 852074
Data columns (total 8 columns):
 #   Column   Non-Null Count   Dtype 
---  ------   --------------   ----- 
 0   Price    852075 non-null  int64 
 1   Year     852075 non-null  int64 
 2   Mileage  852075 non-null  int64 
 3   City     852075 non-null  object
 4   State    852075 non-null  object
 5   Vin      852075 non-null  object
 6   Make     852075 non-null  object
 7   Model    852075 non-null  object
dtypes: int64(3), object(5)
memory usage: 52.0+ MB
```

- **No Missing Values**: The dataset had no missing values across all columns.

  ```
  In [87]:
  ```

  ```
  # Identifying missing values
  df.isna().sum()
  ```

  ```
  Out[87]:
  Price      0
  Year       0
  Mileage    0
  City       0
  State      0
  Vin        0
  Make       0
  Model      0
  dtype: int64
  ```

  ```
  In [88]:
  ```

  ```
  # No Missing Values found
  ```

- **High Dimensionality**: The Vin column had unique values for all rows, making it a high-cardinality categorical feature.

```
In [89]:

df.nunique()

Out[89]:
Price       47124
Year           22
Mileage    158836
City         2553
State          59
Vin        852075
Make           58
Model        2736
dtype: int64
```

- **Target Distribution**: The Price column varied significantly, with values ranging from 1,500 to 499,500 units.

Data Cleaning

Identifying Duplicates

Since each Vin represents a unique car, duplicate entries likely correspond to the same car being listed multiple times. The dataset was sorted by Vin and Year to ensure that only the most recent record for each car was retained:

In [90]:

```
# Correcting this error effectively

df = df.sort_values(by=['Vin', 'Year'], ascending=[True, False])
df = df.drop_duplicates(subset=['Vin'], keep='first')

# Reset the index
df = df.reset_index(drop=True)

df
```

Out[90]:

	Price	Year	Mileage	City	State	Vin	Make	Model
0	29833	2016	55	Tinley Park	IL	04WT3N56GG0646582	Buick	CascadaPremium
1	29833	2016	56	Tinley Park	IL	04WT3N59GG1261202	Buick	CascadaPremium
2	67488	2002	84310	Peoria	AZ	137FA84322E198163	HUMMER	H14-Passenger
3	77995	2004	51651	Hampstead	MD	137FA84374E208897	HUMMER	H14-Passenger
4	89999	2001	49100	Houston	TX	137FA843X1E195221	AM	General
...	...	...	...	...	...	...	...	...
852070	81875	2017	2220	Oakhurst	NJ	ZN661YUL8HX230445	Maserati	LevanteS
852071	89995	2017	3701	San Antonio	TX	ZN661YUS6HX235084	Maserati	LevanteS
852072	82687	2017	1632	Gaithersburg	MD	ZN661YUS7HX230573	Maserati	LevanteS
852073	82782	2017	2182	Gaithersburg	MD	ZN661YUS8HX230520	Maserati	LevanteS
852074	84995	2017	12687	Austin	TX	ZN661YUS9HX226976	Maserati	LevanteS

852075 rows × 8 columns

Result: The dataset was reduced to 852,075 rows after removing duplicates.

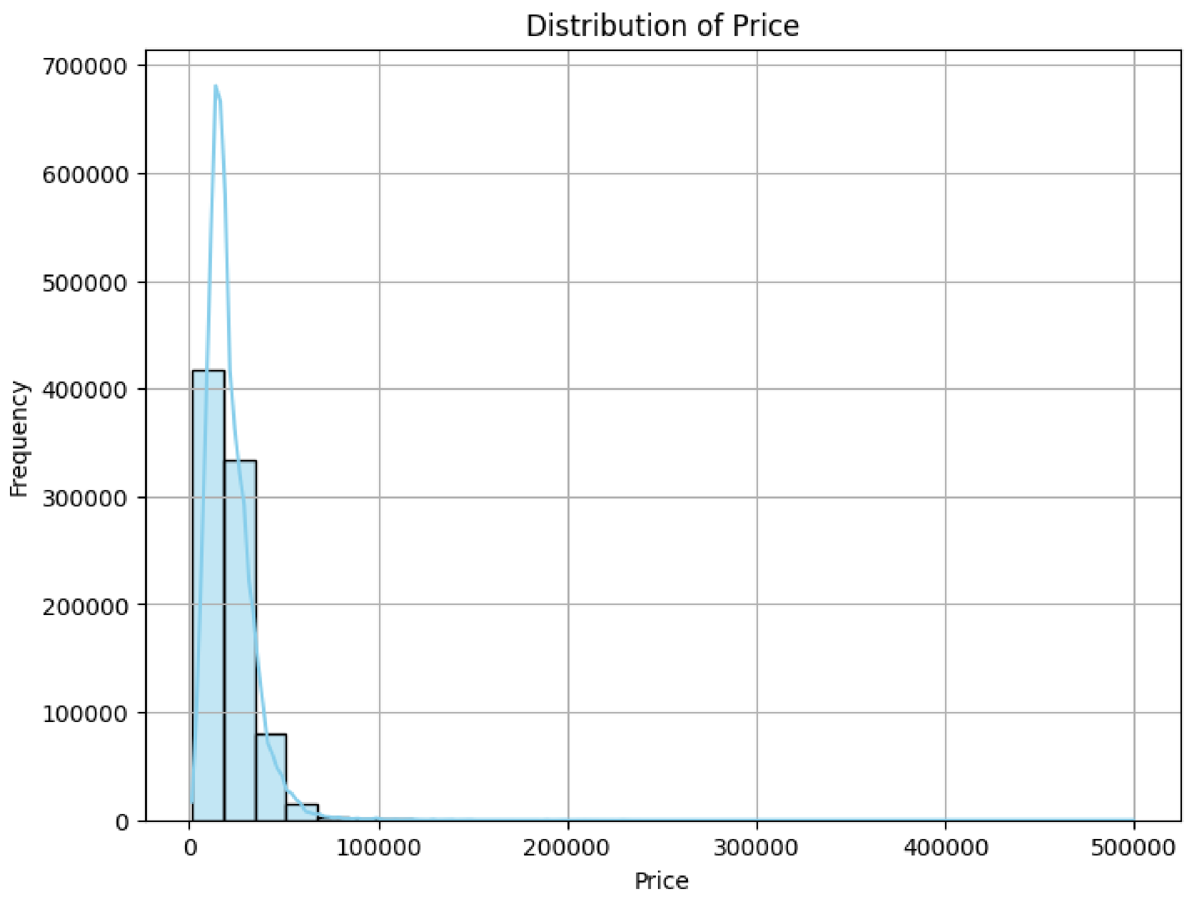
Data Visualization

1. Histogram of Car Prices

The histogram of car prices provides insight into the distribution of the Price variable:

In [95]:

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

# Histogram of Price
plt.figure(figsize=(8, 6))
sns.histplot(df_new['Price'], bins=30, kde=True, color='skyblue')
plt.title('Distribution of Price')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

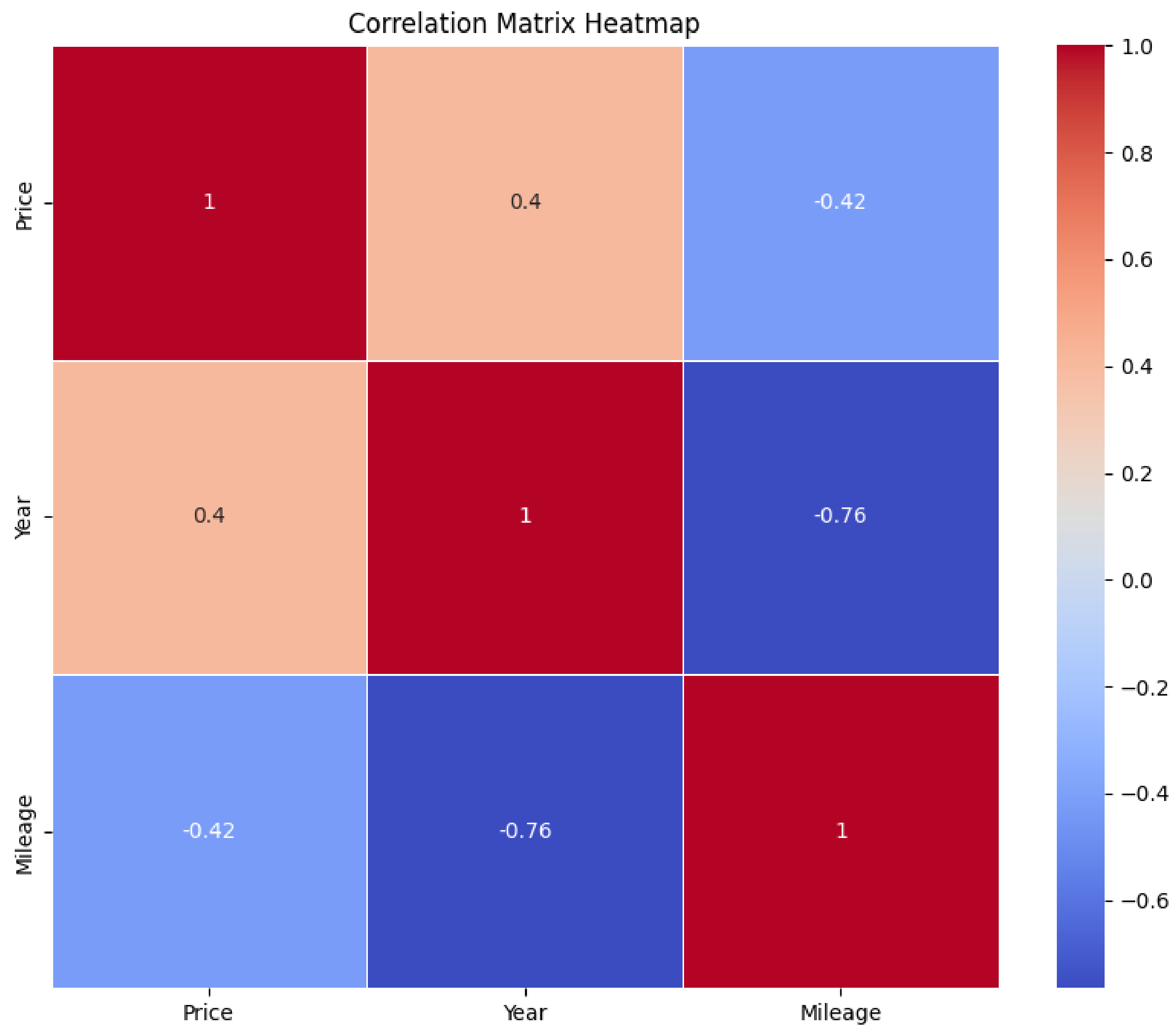


Insight: The car prices are right-skewed, with most prices clustered in the lower range, indicating a market where most cars are sold for less than 30,000 units.

2. Correlation Heatmap

The correlation heatmap of the numeric variables reveals relationships between features:

```
In [96]:

# Select only numeric columns
numeric_columns = df_new.select_dtypes(include='number')

# Plot correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(numeric_columns.corr(), annot=True, cmap='coolwarm', linewidths=.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



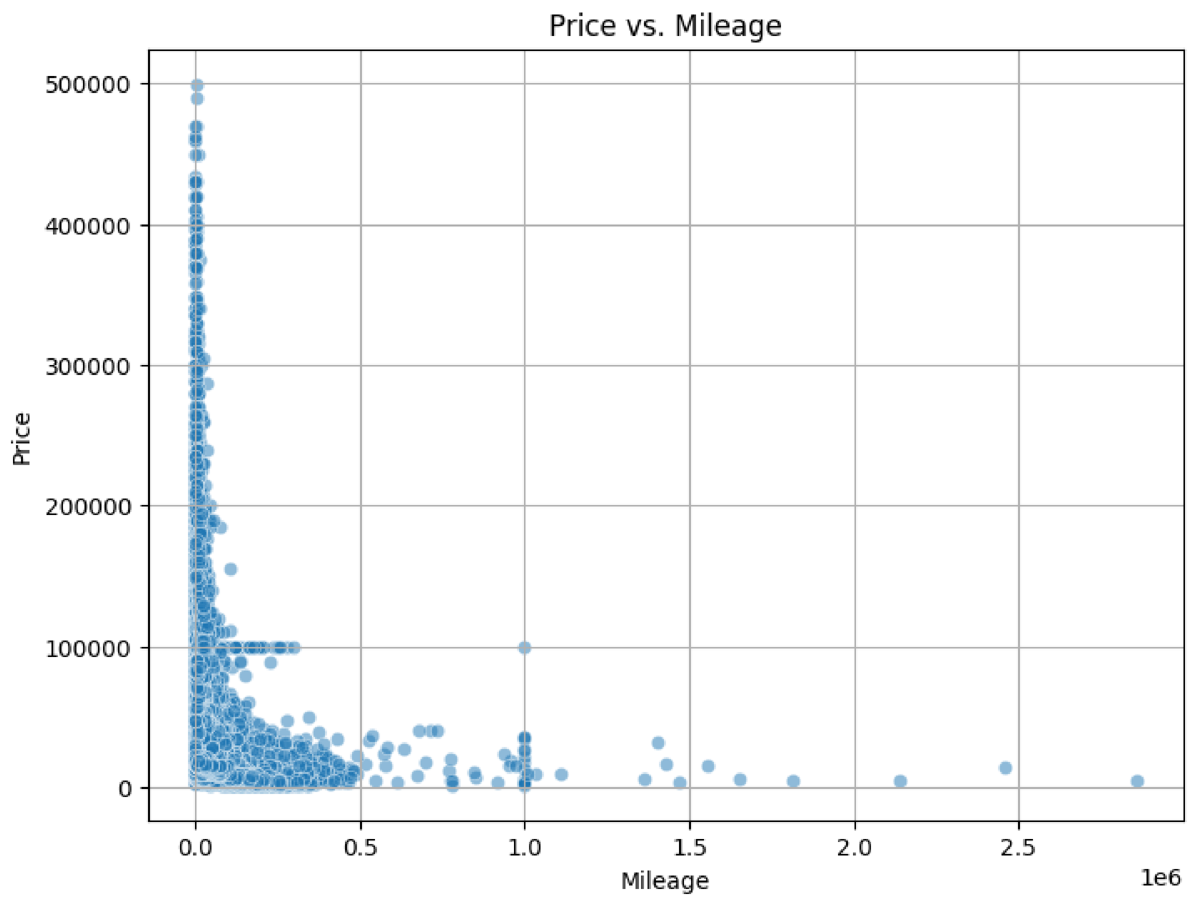
Insight: The Price has a moderate positive correlation with Year (0.36) and a moderate negative correlation with Mileage (-0.41). Newer cars with lower mileage tend to have higher prices.

3. Scatter Plot of Price vs. Mileage

The scatter plot shows the relationship between Price and Mileage:

```
In [97]:
```

```
# Scatter Plot of Price vs. Mileage
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Mileage', y='Price', data=df_new, alpha=0.5)
plt.title('Price vs. Mileage')
plt.xlabel('Mileage')
plt.ylabel('Price')
plt.grid(True)
plt.show()
```



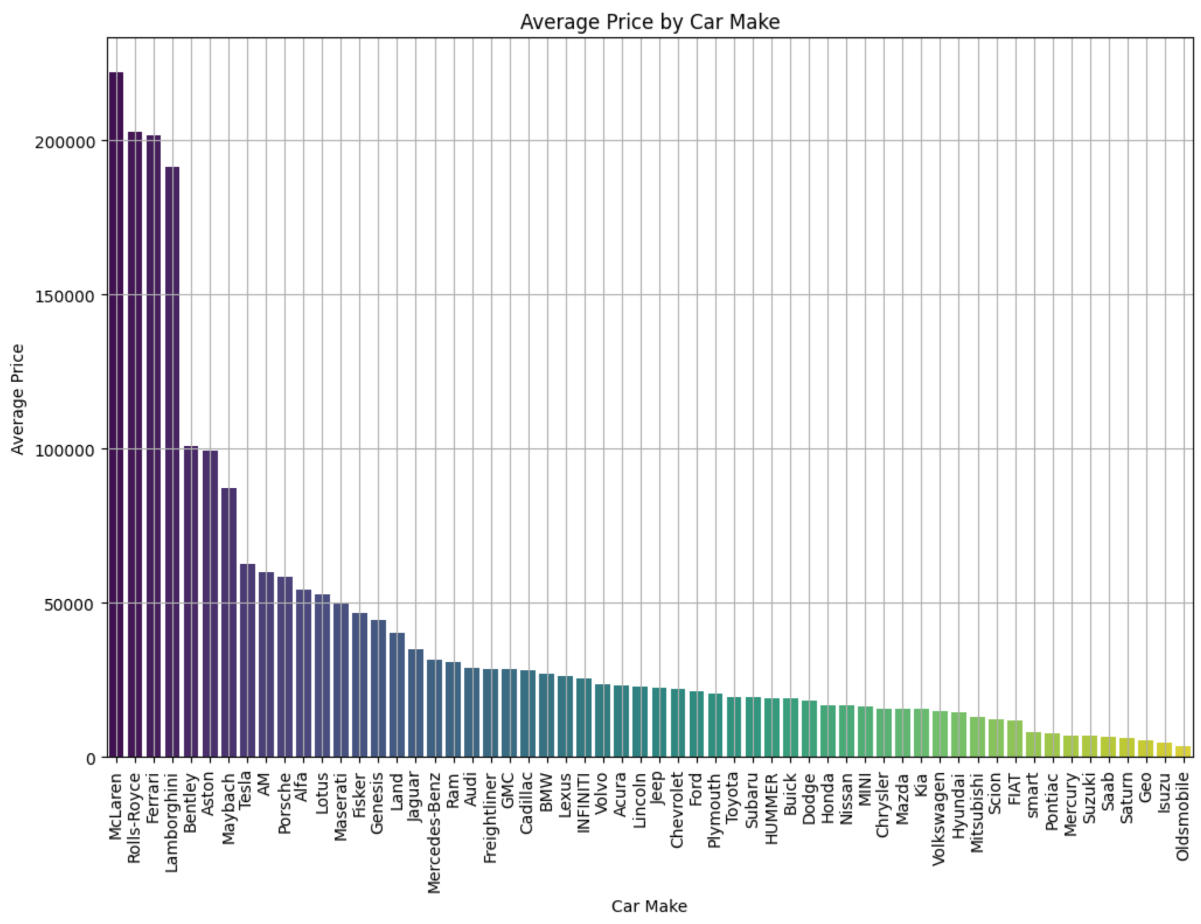
Insight: As mileage increases, the price generally decreases, reflecting the depreciation in value as a car is driven more.

4. Bar Plot of Average Price by Car Make

The bar plot compares the average prices across different car makes:

In [98]:

```
#Bar Plot of Average Price by Car Make
plt.figure(figsize=(12, 8))
avg_price_by_make = df_new.groupby('Make')['Price'].mean().sort_values(ascending=False)
sns.barplot(x=avg_price_by_make.index, y=avg_price_by_make.values, palette='viridis')
plt.title('Average Price by Car Make')
plt.xlabel('Car Make')
plt.ylabel('Average Price')
plt.xticks(rotation=90)
plt.grid(True)
plt.show()
```

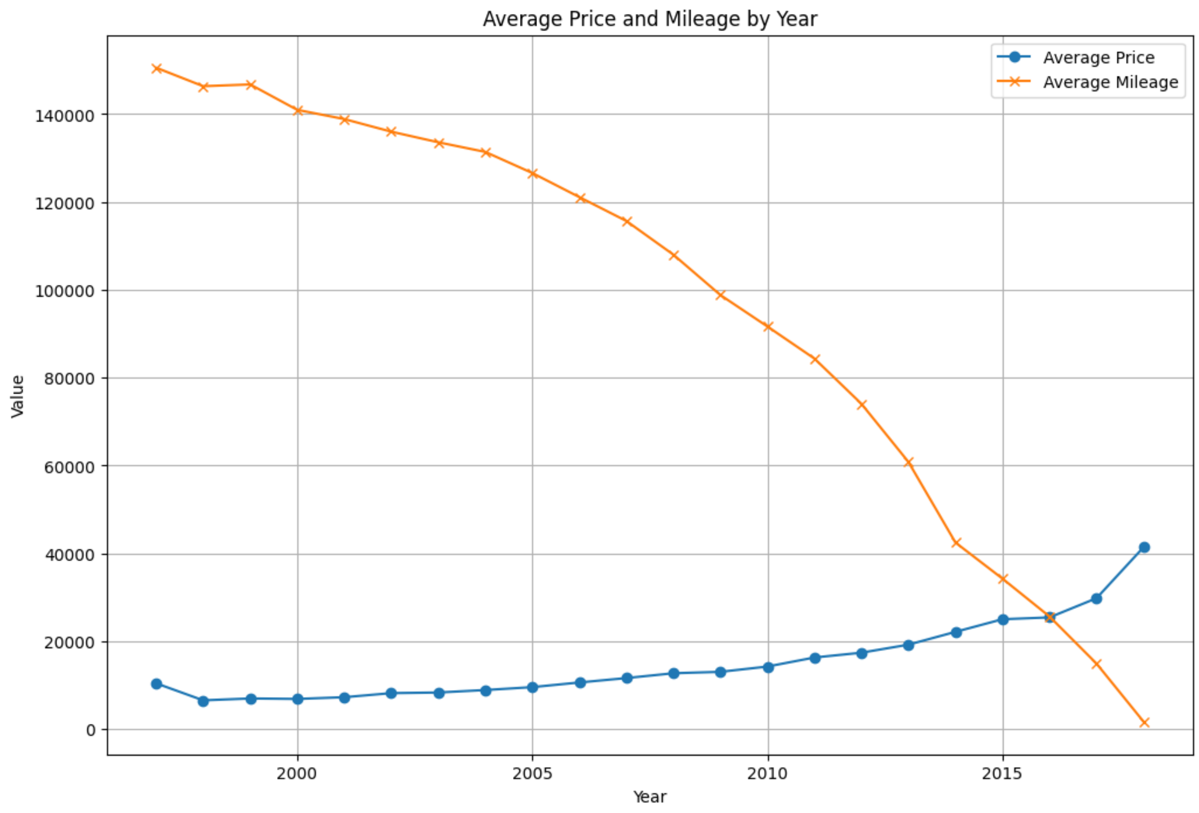


Insight: Luxury brands like Ferrari and Rolls-Royce have significantly higher average prices, while economy brands like Geo have much lower prices.

5. Line Graph of Average Price and Mileage by Year

The line graph tracks trends in average price and mileage over time:

In [99]:
```
# line graph of average Price and Mileage by Year
avg_price_by_year = df_new.groupby('Year')['Price'].mean()
avg_mileage_by_year = df_new.groupby('Year')['Mileage'].mean()

# Plot the line chart
plt.figure(figsize=(12, 8))

plt.plot(avg_price_by_year.index, avg_price_by_year.values, label='Average Price', marker='o')
plt.plot(avg_mileage_by_year.index, avg_mileage_by_year.values, label='Average Mileage', marker=

plt.title('Average Price and Mileage by Year')
plt.xlabel('Year')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()
```



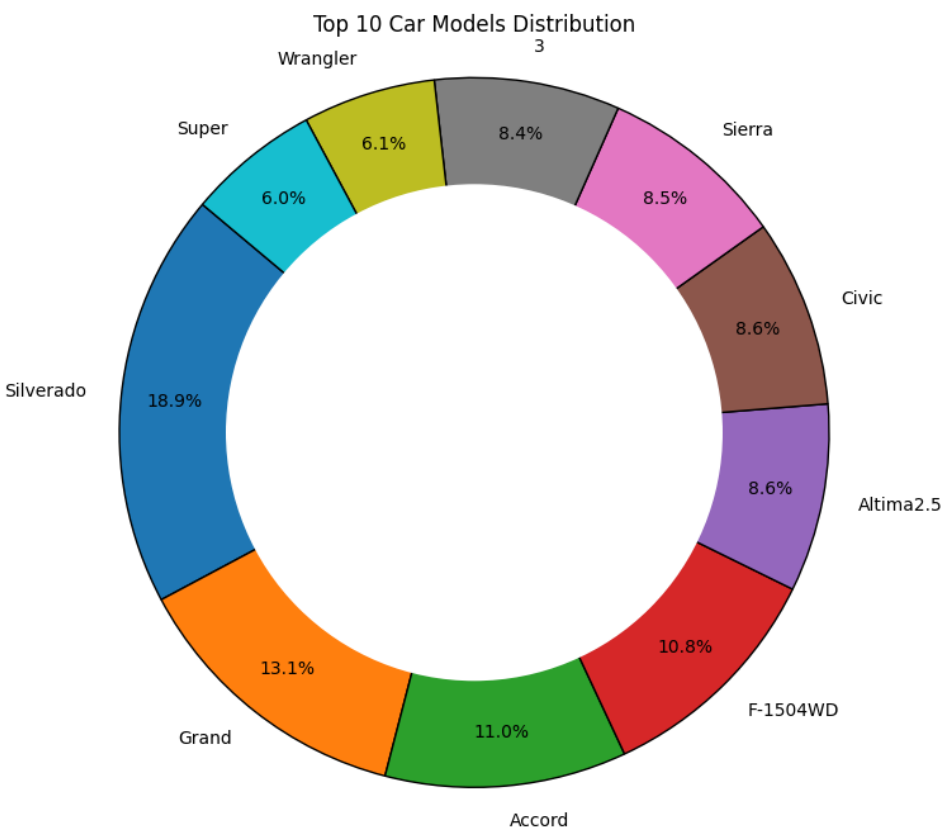
Insight: Average prices have increased steadily since 2010, while average mileage has decreased, suggesting a market preference for newer, lower-mileage vehicles.

6. Donut Chart of Car Models

The donut chart illustrates the distribution of the top 10 car models:

```
In [100]:
```

```python
#Donut Chart of Car Models

# Calculate the distribution of car models
model_counts = df_new['Model'].value_counts().head(10)  # Showing top 10 models for clarity

# Plot the pie chart
plt.figure(figsize=(12, 8))
plt.pie(model_counts, labels=model_counts.index, autopct='%1.1f%%', startangle=140, pctdistance=

# Draw a circle at the center of pie to make it look like a donut
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.title('Top 10 Car Models Distribution')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



Insight: The Chevrolet Silverado is the most common model in the dataset, followed by the Jeep Grand and Honda Accord.

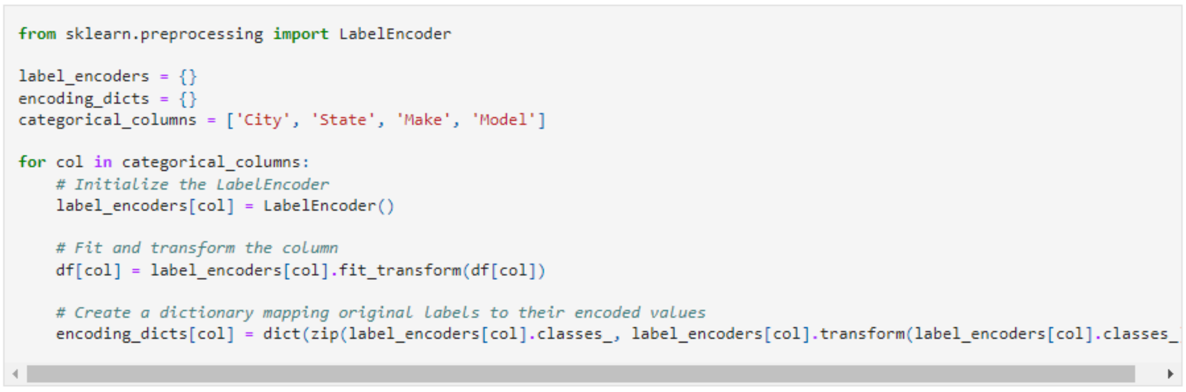
Feature Engineering

1. Car Age: Created as 2024 - Year to capture the effect of age on price.
2. Mileage_Age_Interaction: Created as Mileage * Car_Age to capture the combined effect of mileage and age.
3. Log_Price: Applied log transformation to the target variable to handle skewness.

In [101]:
```
import numpy as np
df=df_new
# Feature Engineering: Create new features
df['Car_Age'] = 2024 - df['Year']  # Assuming current year is 2024

# Interaction features
df['Mileage_Age_Interaction'] = df['Mileage'] * df['Car_Age']

# Log transformation of target variable to handle skewness
df['Log_Price'] = np.log1p(df['Price'])
```

Data Preparation

- Encoding Categorical Variables: Label encoding was applied to City, State, Make, and Model due to high cardinality.



```python
from sklearn.preprocessing import LabelEncoder

label_encoders = {}
encoding_dicts = {}
categorical_columns = ['City', 'State', 'Make', 'Model']

for col in categorical_columns:
    # Initialize the LabelEncoder
    label_encoders[col] = LabelEncoder()

    # Fit and transform the column
    df[col] = label_encoders[col].fit_transform(df[col])

    # Create a dictionary mapping original labels to their encoded values
    encoding_dicts[col] = dict(zip(label_encoders[col].classes_, label_encoders[col].transform(label_encoders[col].classes_
```

In [105]:

```python
import pickle

# Export the dictionary to a Pickle file
with open('label_encodings.pkl', 'wb') as f:
    pickle.dump(encoding_dicts, f)
```

Model Selection

Train-Test Split:

The data was split into training and testing sets using a 80-20 ratio.

In [106]:

```python
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler


# Splitting data into features (X) and target (y)
X = df.drop(columns=['Log_Price'])  # Use 'Log_Price' as target
y = df['Log_Price']
```

```python
# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

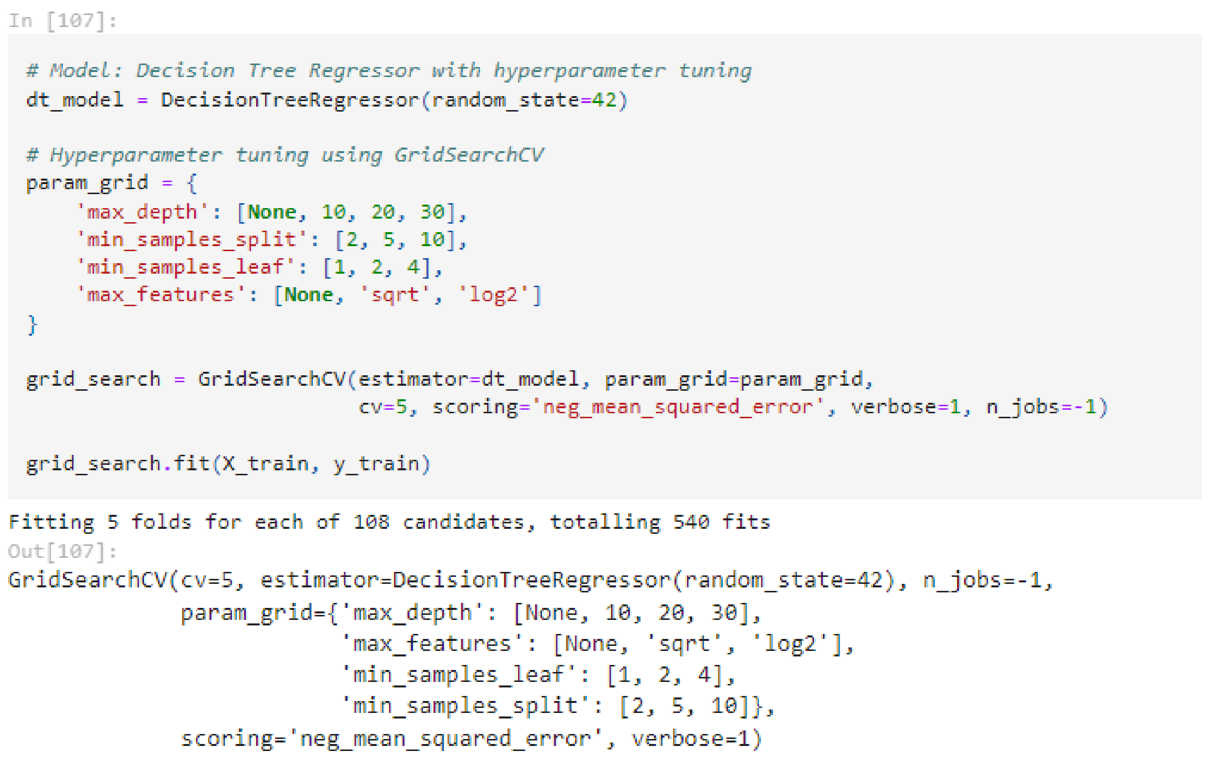
Decision Tree Regressor:

A Decision Tree Regressor was selected for initial modeling due to its ability to capture non-linear relationships.

Hyperparameter Tuning:

GridSearchCV was used for tuning, with the following parameters:

- max_depth: [None, 10, 20, 30]
- min_samples_split: [2, 5, 10]
- min_samples_leaf: [1, 2, 4]
- max_features: [None, 'sqrt', 'log2']



```
In [107]:
# Model: Decision Tree Regressor with hyperparameter tuning
dt_model = DecisionTreeRegressor(random_state=42)

# Hyperparameter tuning using GridSearchCV
param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2']
}

grid_search = GridSearchCV(estimator=dt_model, param_grid=param_grid,
                           cv=5, scoring='neg_mean_squared_error', verbose=1, n_jobs=-1)

grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Out[107]:
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(random_state=42), n_jobs=-1,
             param_grid={'max_depth': [None, 10, 20, 30],
                         'max_features': [None, 'sqrt', 'log2'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10]},
             scoring='neg_mean_squared_error', verbose=1)
```

Results:

The best parameters found were:

- max_depth: 20
- min_samples_split: 5
- min_samples_leaf: 2
- max_features: 'sqrt'

Model Evaluation

Predictions: The model's predictions were transformed back to the original price scale using the exponential function.

```
In [108]:

# Best parameters
print("Best parameters found: ", grid_search.best_params_)

# Predict on test set using the best model
best_model = grid_search.best_estimator_
y_pred_log = best_model.predict(X_test)

# Inverse the log transformation to get the original price scale
y_test_original = np.expm1(y_test)
y_pred_original = np.expm1(y_pred_log)


# Evaluate the model
rmse = np.sqrt(mean_squared_error(y_test_original, y_pred_original))
mae = mean_absolute_error(y_test_original, y_pred_original)
r2 = r2_score(y_test_original, y_pred_original)

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R² Score: {r2:.2f}")

# Cross-validation score
cv_scores = cross_val_score(best_model, X_train, y_train, cv=5, scoring='r2')
print(f"Cross-validated R² Score: {cv_scores.mean():.2f} ± {cv_scores.std():.2f}")
```

```
Best parameters found:  {'max_depth': 20, 'max_features': None, 'min_samples_leaf': 4, 'min_samp
les_split': 10}
Root Mean Squared Error (RMSE): 4843.50
Mean Absolute Error (MAE): 2475.55
R² Score: 0.87
Cross-validated R² Score: 0.89 ± 0.00
```

Performance Metrics:

- RMSE: The root mean squared error (RMSE) was 4843.50 units.
- MAE: The mean absolute error (MAE) was 2475.55 units.
- R²: The R² score was 0.87, indicating that the model explains 87% of the variance in the target variable.

Conclusions and Insights

Feature Importance:

- Mileage and Year were the most influential features, highlighting the importance of a car's condition and age in determining its price.
- The interaction between Mileage and Age further refined predictions by capturing their combined impact.

Market Trends:

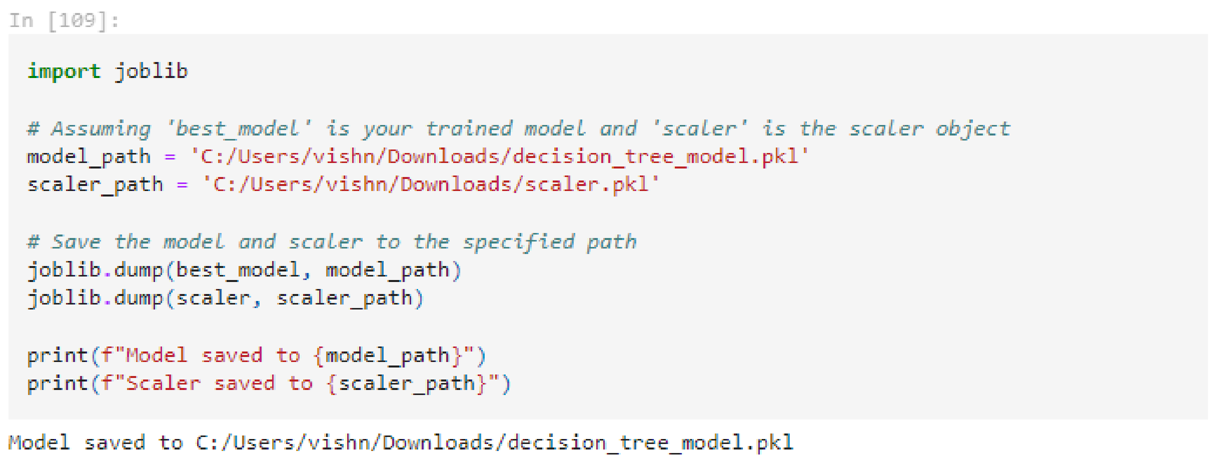
- The dominance of luxury brands in high-price categories suggests a clear market segmentation.
- The declining trend in average mileage and increasing average price over time reflects consumer preferences for newer cars.

Recommendations:

- For Buyers: Focus on newer, low-mileage cars to get the best value for money.
- For Sellers: Emphasize the age and mileage of vehicles in listings to attract buyers.

Front-End Development

1. **Import the joblib Library**: joblib is a Python library used for efficiently serializing Python objects, especially large numpy arrays or machine learning models.
2. **Define Paths for Saving the Model and Scaler**: model_path and scaler_path specify the file paths where the model and scaler objects will be saved. These paths are customizable and can be changed according to your directory structure.
3. **Save the Model and Scaler:** joblib.dump() is used to save the best_model and scaler objects to the specified paths in the form of .pkl files. These files can later be loaded to use the saved model and scaler without retraining.
4. **Confirmation Message:** These print statements confirm that the model and scaler have been successfully saved to the specified paths.

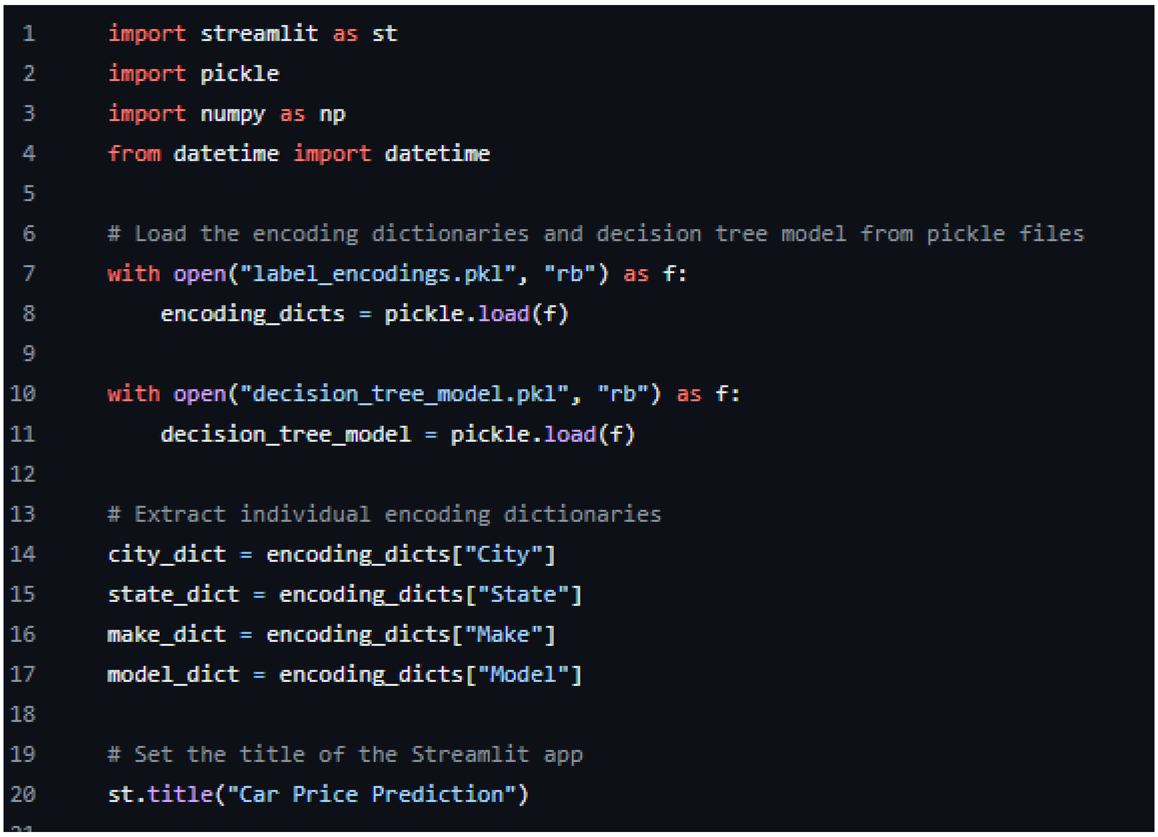


```
In [109]:
import joblib

# Assuming 'best_model' is your trained model and 'scaler' is the scaler object
model_path = 'C:/Users/vishn/Downloads/decision_tree_model.pkl'
scaler_path = 'C:/Users/vishn/Downloads/scaler.pkl'

# Save the model and scaler to the specified path
joblib.dump(best_model, model_path)
joblib.dump(scaler, scaler_path)

print(f"Model saved to {model_path}")
print(f"Scaler saved to {scaler_path}")
```

```
Model saved to C:/Users/vishn/Downloads/decision_tree_model.pkl
```

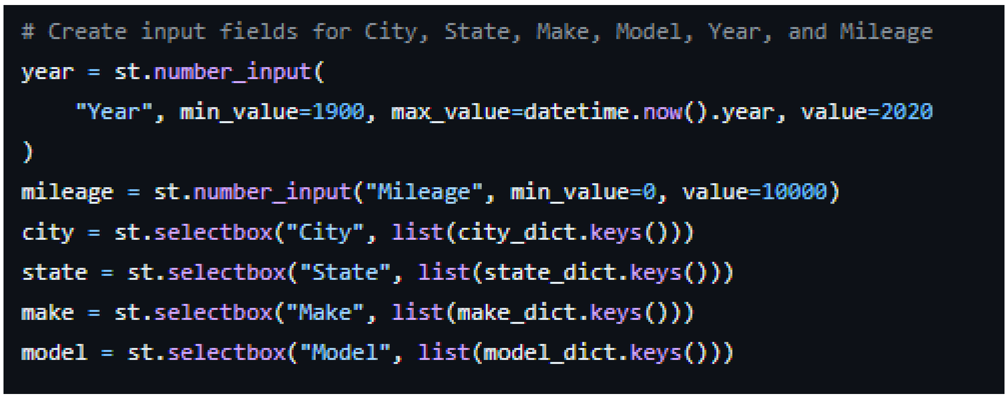
Streamlit application that predicts the price of a used car based on user input

1. **Loading Pre-trained Model and Encodings:**
   - The code loads pre-trained label encoding dictionaries and a decision tree model from pickle files. These are used to encode categorical input features and make predictions.



```python
import streamlit as st
import pickle
import numpy as np
from datetime import datetime

# Load the encoding dictionaries and decision tree model from pickle files
with open("label_encodings.pkl", "rb") as f:
    encoding_dicts = pickle.load(f)

with open("decision_tree_model.pkl", "rb") as f:
    decision_tree_model = pickle.load(f)

# Extract individual encoding dictionaries
city_dict = encoding_dicts["City"]
state_dict = encoding_dicts["State"]
make_dict = encoding_dicts["Make"]
model_dict = encoding_dicts["Model"]

# Set the title of the Streamlit app
st.title("Car Price Prediction")
```

2. **User Input:**
   - The app allows users to input the car's details: City, State, Make, Model, Year, and Mileage. These inputs are taken through various Streamlit input widgets like number_input and selectbox.



```python
# Create input fields for City, State, Make, Model, Year, and Mileage
year = st.number_input(
    "Year", min_value=1900, max_value=datetime.now().year, value=2020
)
mileage = st.number_input("Mileage", min_value=0, value=10000)
city = st.selectbox("City", list(city_dict.keys()))
state = st.selectbox("State", list(state_dict.keys()))
make = st.selectbox("Make", list(make_dict.keys()))
model = st.selectbox("Model", list(model_dict.keys()))
```

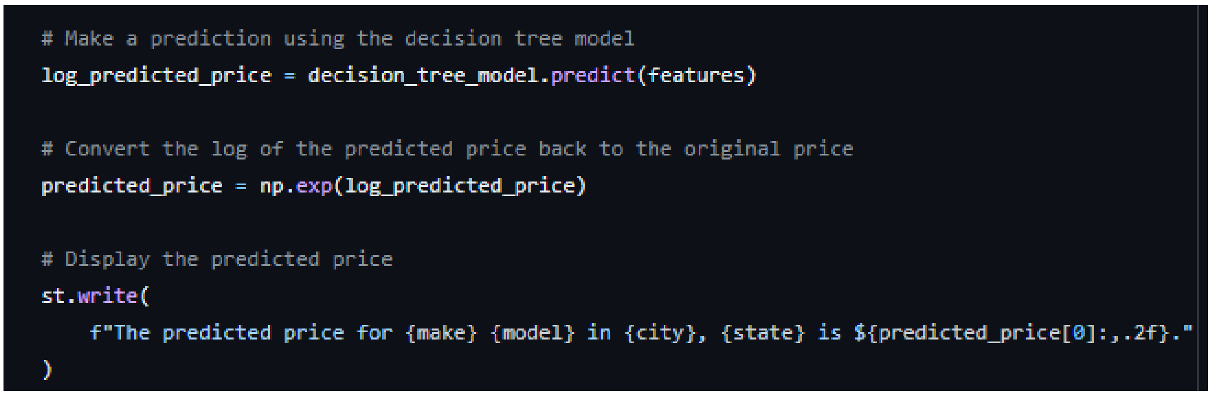
3. **Feature Engineering:**
   - The code calculates the age of the car and the mileage_age_interaction, which is the product of the car's mileage and its age. This interaction term is used as an additional feature in the prediction model.

```
# Output a prediction based on the inputs
if st.button("Predict"):
    # Calculate mileage_age_interaction
    current_year = datetime.now().year
    age = current_year - year
    mileage_age_interaction = mileage * age
```

4. **Encoding and Prediction:**
   - The input categorical values are encoded using the previously loaded dictionaries.
   - These encoded values, along with Year, Mileage, and mileage_age_interaction, are combined into a feature array.
   - The decision tree model predicts the log of the car's price using this feature array.

```
# Encode the inputs using the dictionaries
encoded_city = city_dict[city]
encoded_state = state_dict[state]
encoded_make = make_dict[make]
encoded_model = model_dict[model]

# Create a feature array for the model
features = np.array(
    [
        [
            year,
            mileage,
            encoded_city,
            encoded_state,
            encoded_make,
            encoded_model,
            mileage_age_interaction,
        ]
    ]
)
```

5. **Display Predicted Price:**
   - The predicted log price is then converted back to the original price scale using the exponential function.
   - The app displays the predicted price to the user in a formatted string.



```
# Make a prediction using the decision tree model
log_predicted_price = decision_tree_model.predict(features)

# Convert the log of the predicted price back to the original price
predicted_price = np.exp(log_predicted_price)

# Display the predicted price
st.write(
    f"The predicted price for {make} {model} in {city}, {state} is ${predicted_price[0]:,.2f}."
)
```

This app provides a simple yet effective interface for users to estimate the price of a used car based on various factors

Appendices

GitHub Link: https://github.com/tarunsamuel7/car-price-prediction/tree/main

UI URL: https://car-price-prediction-nrrj2dzcwnzhj8jmygenqy.streamlit.app/