PROJECT REPORT ON VEHICLE PRICE PREDICTION

AVIJIT BHADRA | INTERNSHIP ID: UMID13042530260

INDEX:

Introduction

In the automotive industry, accurately estimating the value of a vehicle is essential for both buyers and sellers. Whether purchasing a used car, trading in a vehicle, or setting a price for resale, having a reliable price estimate can significantly impact decision-making and negotiation. Traditionally, this valuation process relied on subjective assessments or static pricing guides, which often failed to capture real-time market dynamics and individual vehicle features.

With the advent of data-driven technologies, machine learning offers an effective solution to this challenge. By analyzing historical data of vehicles and their attributes, predictive models can learn complex patterns and make accurate price estimations. This project aims to build a **Vehicle Price Prediction System** using machine learning techniques to determine the price of a vehicle based on various specifications like make, model, year, engine details, mileage, fuel type, transmission, color, and more.

The project leverages a dataset containing detailed information about multiple vehicles and their corresponding prices. Through data preprocessing, feature

analysis, and model development, the goal is to create an efficient and scalable prediction system that can assist users in making informed pricing decisions in the vehicle marketplace.

Data Understanding and Preparation

In the initial phase of the vehicle price prediction project, the dataset was explored to understand the structure and quality of the data. The first five rows of the dataset were displayed to gain an overview of the features and data types involved.

The following Python libraries were imported to facilitate data processing, visualization, and machine learning:

```
import pandas as pd
import numpy as np
import joblib as jb
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from imblearn.under_sampling import RandomUnderSampler
```

pandas and numpy were used for data manipulation and numerical operations.

- joblib was used for saving and loading models or preprocessing pipelines.
- matplotlib.pyplot and seaborn were utilized for data visualization to identify patterns and distributions.
- LabelEncoder was used to convert categorical labels into numeric form.

- SimpleImputer handled missing values by applying imputation strategies.
- ColumnTransformer was employed to apply different preprocessing steps to different column types.
- RandomUnderSampler from imblearn was used to address any class imbalance in the dataset during training.

The dataset contains various features including name, description, make, model, year, engine, cylinders, fuel, mileage, transmission, trim, body, doors, exterior_color, interior_color, drivetrain, and price. From the sample shown, it's evident that some preprocessing steps such as handling missing values, encoding categorical data, and feature selection will be necessary before training a model.

After importing the required libraries, the dataset was loaded using the following command:

```
data=pd.read_csv("dataset.csv")
data.head()
```

This command reads the vehicle dataset from a CSV file and displays the first five rows for initial inspection. The dataset includes a variety of features relevant to vehicle characteristics and pricing. The features include:

- **name**: Combined vehicle brand, model, and variant.
- **description**: Brief marketing description or vehicle highlights.
- **make**: Manufacturer of the vehicle.
- **model**: Specific model name.
- **year**: Manufacturing year of the vehicle.
- **engine**: Engine type and configuration.
- **cylinders**: Number of engine cylinders.
- **fuel**: Type of fuel used (e.g., Gasoline, Diesel).
- **mileage**: Fuel efficiency.
- **transmission**: Transmission type (e.g., Automatic, 6-Speed).
- **trim**: Vehicle trim level.
- **body**: Type of vehicle body (e.g., SUV, Pickup Truck).
- **doors**: Number of doors.
- **exterior_color** and **interior_color**: Colors of the vehicle.
- **drivetrain**: Drivetrain configuration (e.g., Four-wheel Drive).
- **Unnamed: 16**: An irrelevant or empty column.
- **price**: Target variable representing the vehicle's price.

To further prepare the dataset for analysis and modeling, the following columns were removed:

```
del data['description']
del data['Unnamed: 16']
```

Explanation:

- **description**: This column contained unstructured textual data that was not directly useful for the price prediction model. Since it didn't add significant value and might introduce unnecessary complexity, it was removed.
- **Unnamed: 16**: This column appeared to be empty or irrelevant (likely an artifact from the CSV file). Removing it helps clean up the dataset and avoid processing meaningless data.

This step streamlined the dataset by keeping only the relevant and structured features necessary for the predictive modeling process.

After removing unnecessary columns, the following steps were performed:

```
data.head()
data.columns = data.columns.str.replace(r'[<>\\/\[\]]', '', regex=True)
data.info()
```

data.head() was used to view the first few rows of the dataset after column deletions, ensuring the structure remained as expected.

- **data.columns = data.columns.str.replace(r'[<>\\/\[\]]', '', regex=True)** was used to clean the column names by removing any unwanted special characters such as <, >, \, /, [, and]. This ensures compatibility and prevents errors during further data processing.
- **data.info()** was executed to get an overview of the dataset, including the number of entries, column names, data types, and non-null counts. This helps identify missing values and understand the data structure for the next steps in preparation.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1002 entries, 0 to 1001
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   name            1002 non-null   object
 1   make            1002 non-null   object
 2   model           1002 non-null   object
 3   year            1002 non-null   int64
 4   engine          1000 non-null   object
 5   cylinders       897 non-null    float64
 6   fuel            995 non-null    object
 7   mileage         968 non-null    float64
 8   transmission    1000 non-null   object
 9   trim            1001 non-null   object
 10  body            999 non-null    object
 11  doors           995 non-null    float64
 12  exterior_color  997 non-null    object
 13  interior_color  964 non-null    object
 14  drivetrain      1002 non-null   object
 15  price           979 non-null    float64
dtypes: float64(4), int64(1), object(11)
memory usage: 125.4+ KB
```

Continuing with data preparation, the following steps were executed:

```
data=data.dropna(axis=0)
data.info()
data = data.applymap(lambda x: x.replace(' ', '') if isinstance(x, str) else x)
data
```

- **data.dropna(axis=0)** was used to remove all rows containing any missing values. This ensured that the dataset only retained complete records, resulting in 800 valid entries.
- **data.info()** was then used to confirm that all columns now had non-null values and to view updated data types and memory usage.
-

```
<class 'pandas.core.frame.DataFrame'>
Index: 800 entries, 0 to 1001
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   name            800 non-null    object
 1   make            800 non-null    object
 2   model           800 non-null    object
 3   year            800 non-null    int64
 4   engine          800 non-null    object
 5   cylinders       800 non-null    float64
 6   fuel            800 non-null    object
 7   mileage         800 non-null    float64
 8   transmission    800 non-null    object
 9   trim            800 non-null    object
 10  body            800 non-null    object
 11  doors           800 non-null    float64
 12  exterior_color  800 non-null    object
 13  interior_color  800 non-null    object
 14  drivetrain      800 non-null    object
 15  price           800 non-null    float64
dtypes: float64(4), int64(1), object(11)
memory usage: 106.2+ KB
```

- **data = data.applymap(lambda x: x.replace(' ', '') if isinstance(x, str) else x)** was used to remove any extra

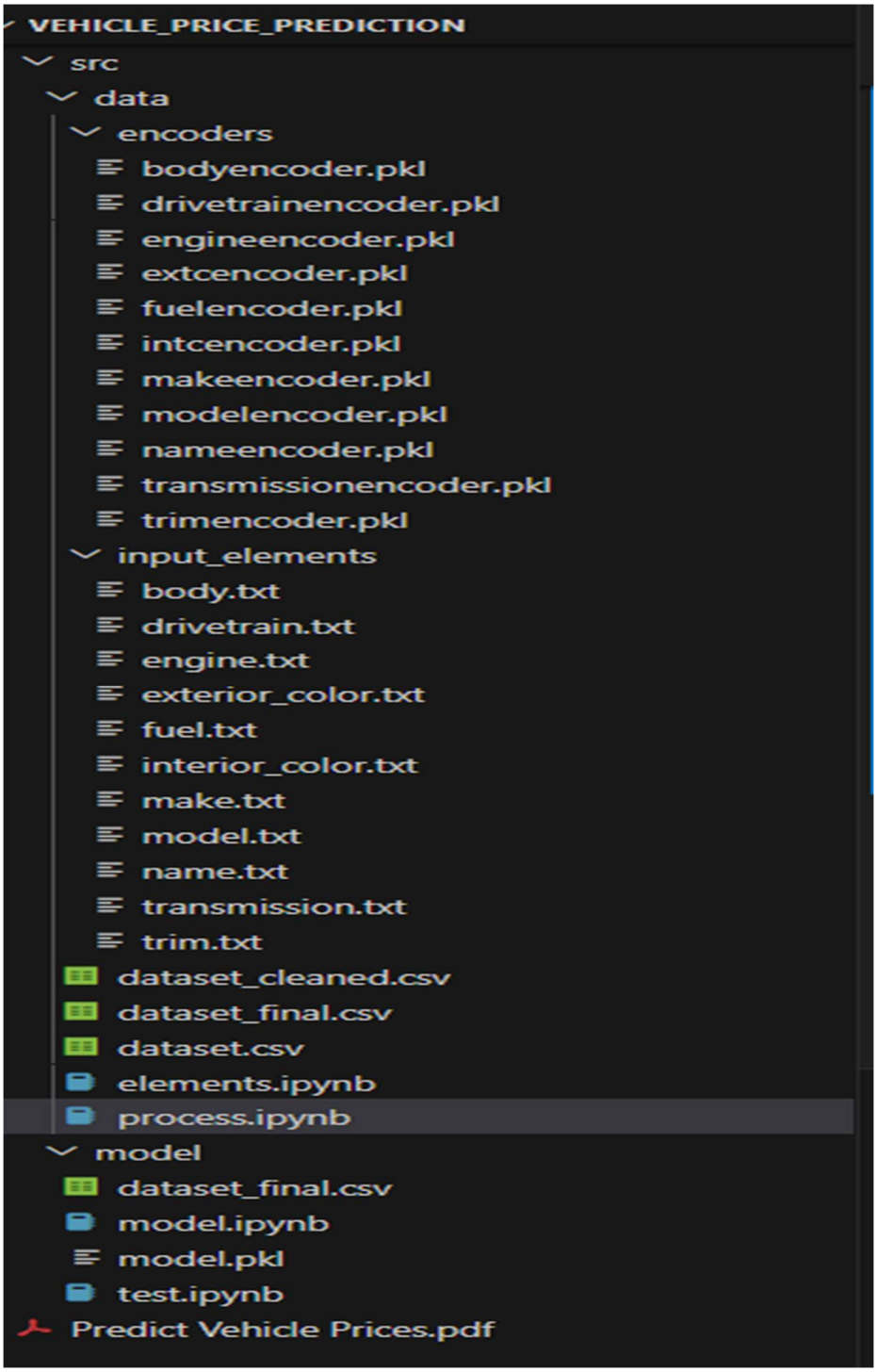
spaces from all string-type entries across the entire DataFrame. This helps standardize categorical data values by eliminating inconsistencies caused by trailing or leading spaces.

```
nameencoder=LabelEncoder()
makeencoder=LabelEncoder()
modelencoder=LabelEncoder()
engineencoder=LabelEncoder()
fuelencoder=LabelEncoder()
transmissionencoder=LabelEncoder()
trimencoder=LabelEncoder()
bodyencoder=LabelEncoder()
extcencoder=LabelEncoder()
intcencoder=LabelEncoder()
drivetrainencoder=LabelEncoder()
```

```
data['name']=nameencoder.fit_transform(data['name'])
jb.dump(nameencoder,"encoders/nameencoder.pkl")

data['make'] = makeencoder.fit_transform(data['make'])
jb.dump(makeencoder, "encoders/makeencoder.pkl")

data['model'] = modelencoder.fit_transform(data['model'])
jb.dump(modelencoder, "encoders/modelencoder.pkl")

data['engine'] = engineencoder.fit_transform(data['engine'])
jb.dump(engineencoder, "encoders/engineencoder.pkl")

data['fuel'] = fuelencoder.fit_transform(data['fuel'])
jb.dump(fuelencoder, "encoders/fuelencoder.pkl")

data['transmission'] = transmissionencoder.fit_transform(data['transmission'])
jb.dump(transmissionencoder, "encoders/transmissionencoder.pkl")

data['trim'] = trimencoder.fit_transform(data['trim'])
jb.dump(trimencoder, "encoders/trimencoder.pkl")

data['body'] = bodyencoder.fit_transform(data['body'])
jb.dump(bodyencoder, "encoders/bodyencoder.pkl")

data['exterior_color'] = extcencoder.fit_transform(data['exterior_color'])
jb.dump(extcencoder, "encoders/extcencoder.pkl")

data['interior_color'] = intcencoder.fit_transform(data['interior_color'])
jb.dump(intcencoder, "encoders/intcencoder.pkl")

data['drivetrain'] = drivetrainencoder.fit_transform(data['drivetrain'])
jb.dump(drivetrainencoder, "encoders/drivetrainencoder.pkl")
```

LabelEncoder():

Now I used Labelencoder from scikit-learn to encode all categorical column to transform it as a label and also save the encoded model in this directory :-



All Encoded Model Saved In encoders folder.it will help to encode the feature to fit in model training.

```
numcol=data.select_dtypes(include=['number']).columns.tolist()
pipe1=Pipeline(steps=[("MISSING VALUE",SimpleImputer(strategy='mean'))])
preprocessor=ColumnTransformer(transformers=[("numeric",pipe1,numcol)])
numcol
data
```

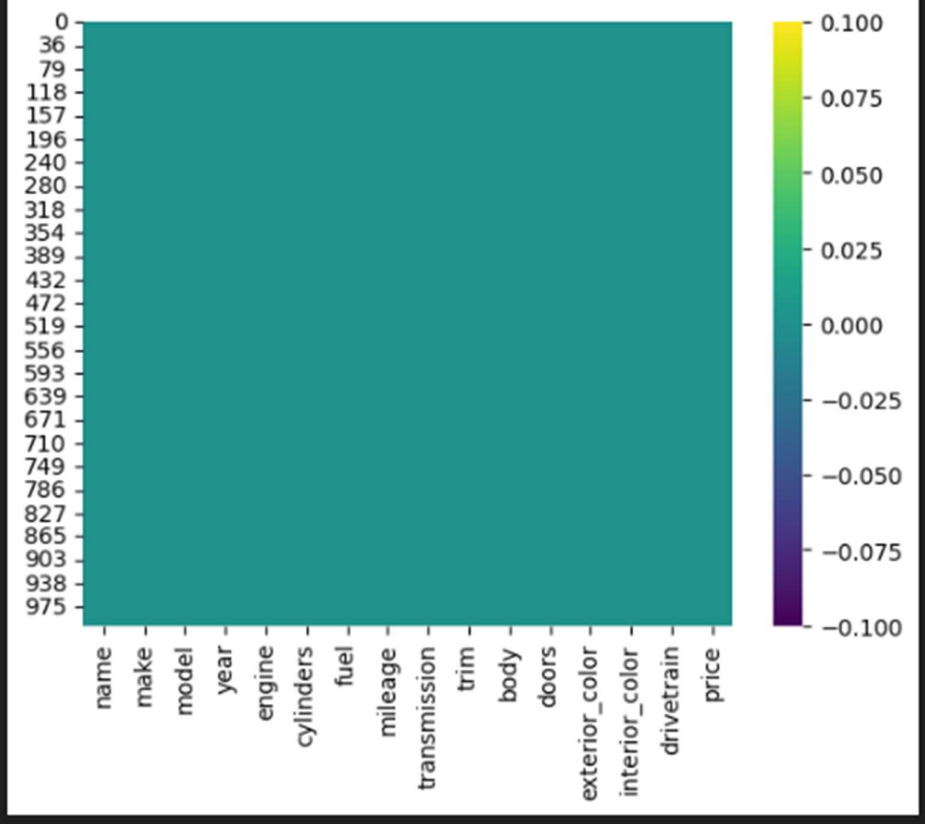
```
x=preprocessor.fit_transform(data)
data=pd.DataFrame(x,columns=numcol)
data
```

Having successfully transformed the categorical features through label encoding, the next critical step in preparing the data for further analysis involved addressing missing values. Upon inspection, it was evident that certain numerical columns contained missing entries, which could potentially bias subsequent modeling or statistical analyses. Therefore, I proceeded to implement a data imputation strategy specifically targeting these numerical feature.

Now to To visualize the extent and distribution of missing values across the dataset, I generated a heatmap using Seaborn's heatmap() function. This allowed a quick and intuitive understanding of where missing values were concentrated. The code used is as follows:

```
sns.heatmap(data.isnull(),color='red',cmap='viridis')
plt.show()
```

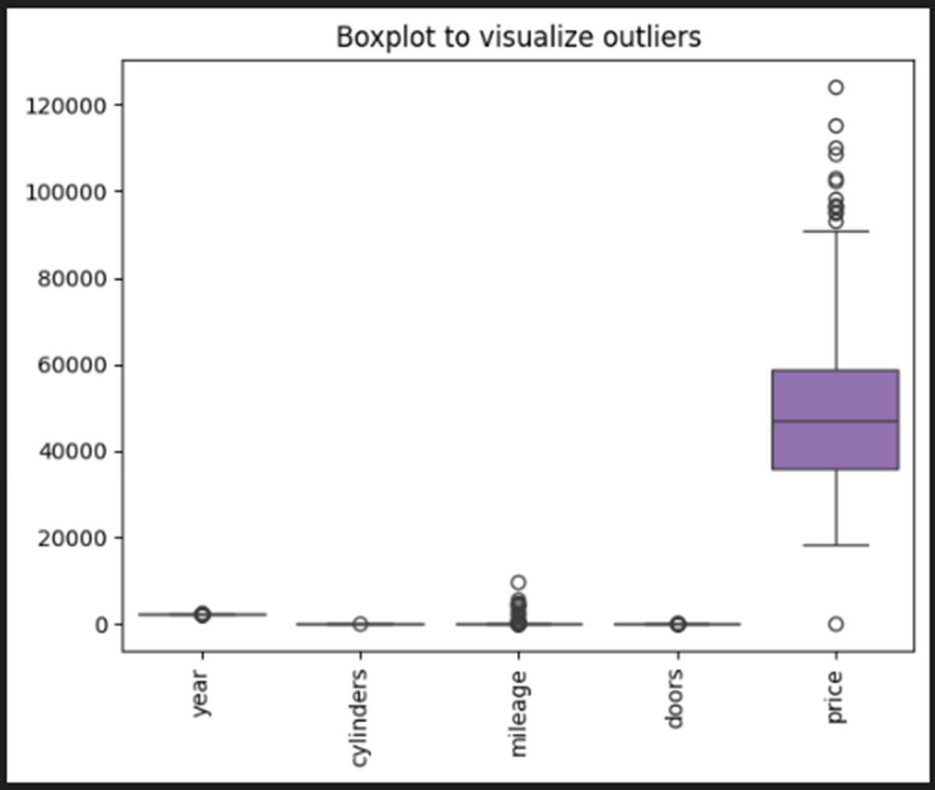
✓ 0.7s



From the heatmap above, it is clear that the dataset does not contain any missing values—each column is fully populated, as indicated by the uniform color and absence of lighter patches that would otherwise signal null entries. This confirms that no further imputation is required, and the dataset is now clean and ready for further preprocessing and analysis.

```
sns.boxplot(data)
plt.xticks(rotation=90)
plt.title("Boxplot to visualize outliers")
plt.show()
```

✓ 0.3s



Here I used boxplot to check outlier of entire data.the round ticks after the box line indicate outliers

```
q1=data.quantile(0.25)
q3=data.quantile(0.75)
iqr=q3-q1
l=q1-1.5*iqr
u=q3+1.5*iqr
data=data[~((data<l)|(data>u)).any(axis=1)]
```

```
q1=data.quantile(0.25)
q3=data.quantile(0.75)
iqr=q3-q1
l=q1-1.5*iqr
u=q3+1.5*iqr
outlier=((data<l)|(data>u))
outlier_counts = outlier.sum()
columns_with_outliers = outlier_counts[outlier_counts > 0]
columns_with_outliers
```

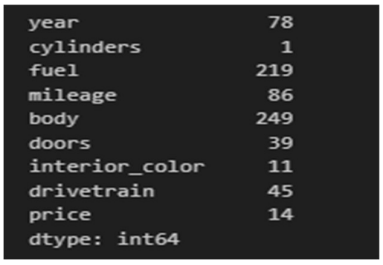
Outlier Detection and Removal

After addressing missing values, the next essential step in data preprocessing was the detection and removal of outliers. Outliers can distort statistical analyses and degrade the performance of machine learning models. To identify them, I employed the Interquartile Range (IQR) method, a widely accepted technique for detecting anomalies in numerical data.

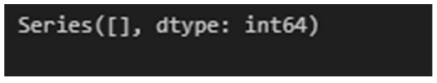
The IQR is calculated as the difference between the third (Q3) and first (Q1) quartiles. Any data point lying below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR is considered an outlier.

To gain insights into the extent of outlier presence in the dataset, I also computed the number of outliers in each column

This analysis helped identify which features were most affected by outliers, providing a deeper understanding of the data distribution and guiding further preprocessing decisions.



Before(742 outliers)



After(0)

Dataset Summary After Preprocessing

To confirm that all missing values had been handled and outliers removed successfully, I inspected the structure of the DataFrame using the data.info() method. This provided a concise summary of the dataset, including the number of entries, data types, and non-null counts for each column:

```
data.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Index: 281 entries, 0 to 798
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   name            281 non-null    float64
 1   make            281 non-null    float64
 2   model           281 non-null    float64
 3   year            281 non-null    float64
 4   engine          281 non-null    float64
 5   cylinders       281 non-null    float64
 6   fuel            281 non-null    float64
 7   mileage         281 non-null    float64
 8   transmission    281 non-null    float64
 9   trim            281 non-null    float64
 10  body            281 non-null    float64
 11  doors           281 non-null    float64
 12  exterior_color  281 non-null    float64
 13  interior_color  281 non-null    float64
 14  drivetrain      281 non-null    float64
 15  price           281 non-null    float64
dtypes: float64(16)
memory usage: 37.3 KB
```

The output confirmed that the cleaned dataset contains **281 entries** and **16 features**, all of which are of type float64. Importantly, **there are no missing values** remaining in any column, as each feature has exactly 281 non-null values.

Data Type Conversion

After ensuring that all missing values were handled and outliers removed, I performed a final data type optimization by converting all numerical columns from float64 to int64 using the following command:

```
data=data.astype(int)
```

This step was particularly important since the dataset primarily contains categorical and count-based features that are more appropriately represented as integers. Following the conversion, I validated the change using the data.info() function:

```
<class 'pandas.core.frame.DataFrame'>
Index: 281 entries, 0 to 798
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   name            281 non-null    int64
 1   make            281 non-null    int64
 2   model           281 non-null    int64
 3   year            281 non-null    int64
 4   engine          281 non-null    int64
 5   cylinders       281 non-null    int64
 6   fuel            281 non-null    int64
 7   mileage         281 non-null    int64
 8   transmission    281 non-null    int64
 9   trim            281 non-null    int64
 10  body            281 non-null    int64
 11  doors           281 non-null    int64
 12  exterior_color  281 non-null    int64
 13  interior_color  281 non-null    int64
 14  drivetrain      281 non-null    int64
 15  price           281 non-null    int64
```

The output confirmed that all 16 columns now have the int64 data type, with no missing values and 281 consistent entries per column. This conversion not only helps in maintaining data integrity but also optimizes memory usage and can improve performance in downstream tasks like modeling.

Final Dataset Export

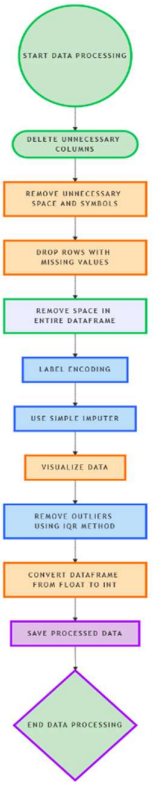
After completing all preprocessing steps—including missing value handling, outlier removal, data type

conversion, and data integrity verification—the cleaned dataset was saved for further use in the modeling phase. The processed data was exported to a CSV file using the following command:

```
data.to_csv(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\model\dataset_final.csv",index=False)
```

This finalized dataset, now stored as dataset_final.csv, serves as the foundation for building predictive models. It ensures that all features are in the correct format, free from inconsistencies, and ready for training machine learning algorithms.

DATA FLOW DIAGRAM:-



Decoding Encoded Features (elements.ipynb)

After training and saving the label encoders for categorical features, a separate notebook elements.ipynb was created to **decode the encoded numerical values back to their original string labels**. This was useful for interpreting results, displaying outputs in a user-friendly format, or validating predictions.

- **Steps Performed:**

1. **Read the Final Dataset**
   The cleaned and encoded dataset was read using:

```
data=pd.read_csv(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\dataset_final.csv")
```

2. **Loading Saved Encoders**
   All previously saved encoders (stored as .pkl files) were loaded using joblib.load:

```
nameencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\nameencoder.pkl")
makeencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\makeencoder.pkl")
modelencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\modelencoder.pkl")
engineencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\engineencoder.pkl")
fuelencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\fuelencoder.pkl")
transmissionencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\transmissionencoder.pkl")
trimencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\trimencoder.pkl")
bodyencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\bodyencoder.pkl")
extcencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\extcencoder.pkl")
intcencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\intcencoder.pkl")
drivetrainencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\drivetrainencoder.pkl")
```

3. **Decoding Encoded Columns**
   Using the .inverse_transform() method of each encoder, encoded numerical values were converted back to their original string forms:

```
name_decoded = nameencoder.inverse_transform(data['name'])
make_decoded = makeencoder.inverse_transform(data['make'])
model_decoded = modelencoder.inverse_transform(data['model'])
engine_decoded = engineencoder.inverse_transform(data['engine'])
fuel_decoded = fuelencoder.inverse_transform(data['fuel'])
transmission_decoded = transmissionencoder.inverse_transform(data['transmission'])
trim_decoded = trimencoder.inverse_transform(data['trim'])
body_decoded = bodyencoder.inverse_transform(data['body'])
extc_decoded = extcencoder.inverse_transform(data['exterior_color'])
intc_decoded = intcencoder.inverse_transform(data['interior_color'])
drivetrain_decoded = drivetrainencoder.inverse_transform(data['drivetrain'])
```

Exporting Decoded Categorical Elements (elements.ipynb)

To make categorical feature values easily accessible for front-end integration or dropdown elements in a UI, the decoded values were saved into individual .txt files.

- **Steps Performed:**

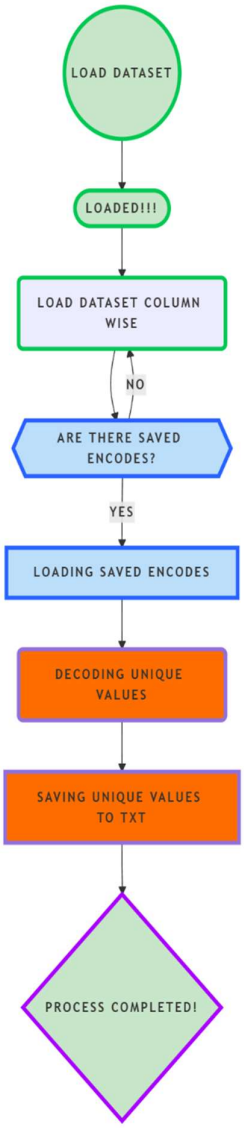
1. **Creating a Decoded DataFrame**
   A new DataFrame decoded_df was created using the decoded values for all categorical columns:

```
decoded_df = pd.DataFrame({
    'name': name_decoded,
    'make': make_decoded,
    'model': model_decoded,
    'engine': engine_decoded,
    'fuel': fuel_decoded,
    'transmission': transmission_decoded,
    'trim': trim_decoded,
    'body': body_decoded,
    'exterior_color': extc_decoded,
    'interior_color': intc_decoded,
    'drivetrain': drivetrain_decoded
})
```

2. **Saving Unique Feature Values to Text Files**
   Each column's values were written to corresponding .txt files in the input_elements/ directory using to_string():

```
decoded_df.to_string("input_elements/name.txt", columns=['name'], index=False, header=False)
decoded_df.to_string("input_elements/make.txt", columns=['make'], index=False, header=False)
decoded_df.to_string("input_elements/model.txt", columns=['model'], index=False, header=False)
decoded_df.to_string("input_elements/engine.txt", columns=['engine'], index=False, header=False)
decoded_df.to_string("input_elements/fuel.txt", columns=['fuel'], index=False, header=False)
decoded_df.to_string("input_elements/transmission.txt", columns=['transmission'], index=False, header=False)
decoded_df.to_string("input_elements/trim.txt", columns=['trim'], index=False, header=False)
decoded_df.to_string("input_elements/body.txt", columns=['body'], index=False, header=False)
decoded_df.to_string("input_elements/exterior_color.txt", columns=['exterior_color'], index=False, header=False)
decoded_df.to_string("input_elements/interior_color.txt", columns=['interior_color'], index=False, header=False)
decoded_df.to_string("input_elements/drivetrain.txt", columns=['drivetrain'], index=False, header=False)
```

DATA FLOW DIAGRAM:-



Model Development and Evaluation

After importing necessary libraries such as:

```
import pandas as pd
import numpy as np
import joblib as jb
import optuna
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error,root_mean_squared_error
```

- **Dataset Loading and Preparation**

The cleaned and preprocessed dataset is loaded using:

```
data= pd.read_csv('dataset_final.csv')
```

The dataset contains both categorical and numerical features. The feature set and target variable are then separated:

```
x=data.iloc[:, :-1]
y=data['price']
```

Train-Test Split

To evaluate model performance, the dataset is split into training and testing sets in an 80-20 ratio:

```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.20, random_state=42)
```

Data Inspection

Basic dataset structure and column data types are verified using:

```
data.info()
```

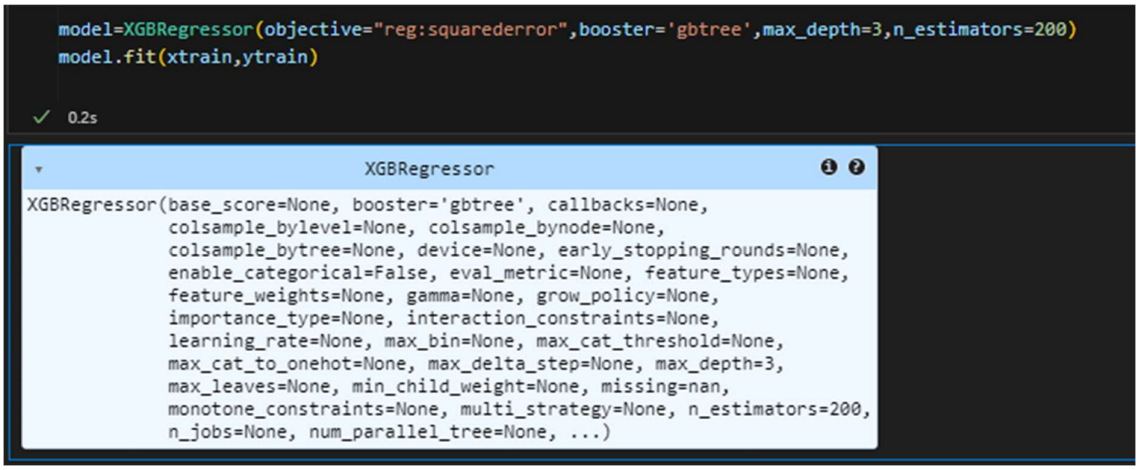
✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 281 entries, 0 to 280
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   name            281 non-null    int64
 1   make            281 non-null    int64
 2   model           281 non-null    int64
 3   year            281 non-null    int64
 4   engine          281 non-null    int64
 5   cylinders       281 non-null    int64
 6   fuel            281 non-null    int64
 7   mileage         281 non-null    int64
 8   transmission    281 non-null    int64
 9   trim            281 non-null    int64
 10  body            281 non-null    int64
 11  doors           281 non-null    int64
 12  exterior_color  281 non-null    int64
 13  interior_color  281 non-null    int64
 14  drivetrain      281 non-null    int64
 15  price           281 non-null    int64
dtypes: int64(16)
memory usage: 35.3 KB
```

This ensures there are no missing values and all encodings are correctly applied before proceeding with model training and evaluation.

- **Model Selection and Training**

The **XGBoost Regressor** was selected for model training due to its robustness and superior performance on structured datasets. The model was initialized with the following hyperparameters:



The model was then trained on the training data:

- **Model Evaluation**

After training, the model's performance was evaluated using the score() method, which returns the coefficient of determination \(R^2 \) of the prediction:

```
print("TRAIN SCORE: ",model.score(xtrain,ytrain),"TEST SCORE: ",model.score(xtest,ytest))
✓ 0.0s

TRAIN SCORE:  0.995556652545929 TEST SCORE:  0.8911192417144775
```

- The **Train Score (R^2)** of **0.9955** indicates excellent learning from the training data.
- The **Test Score (R^2)** of **0.8911** shows strong generalization to unseen data, with minimal overfitting.

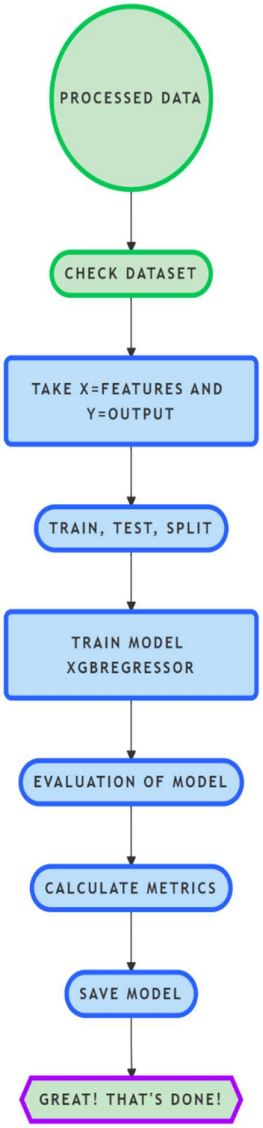
◆ **Model Serialization**

After successful training and evaluation, the trained model was saved using the joblib library to enable later use without retraining:

```
jb.dump(model,"model.pkl")
```

This stored the trained XGBoost regression model into a file named **model.pkl**, which can be loaded later for predictions or deployment.

DATA FLOW DIAGRAM:-



Result and Discussion

In the final phase of the project, the trained model was used to predict the price of a vehicle based on new input values. Below is a summary of the steps followed during testing:

1. Importing Required Libraries

To begin the testing process, the necessary libraries were imported:

```
import joblib as jb
import pandas as pd
import numpy as np
```

2. Loading Label Encoders

The categorical features used in the model were label-encoded during training. For consistency, the same encoders were loaded to transform new input data:

```
nameencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\nameencoder.pkl")
makeencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\makeencoder.pkl")
modelencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\modelencoder.pkl")
engineencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\engineencoder.pkl")
fuelencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\fuelencoder.pkl")
transmissionencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\transmissionencoder.pkl")
trimencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\trimencoder.pkl")
bodyencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\bodyencoder.pkl")
extcencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\extcencoder.pkl")
intcencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\intcencoder.pkl")
drivetrainencoder = jb.load(r"C:\Users\Avijit\Desktop\vehicle_price_prediction\src\data\encoders\drivetrainencoder.pkl")
```

3. Defining Input Data

A sample vehicle's specifications were taken as input for price prediction:

```
name="2024DodgeHornetHornetR/TPlusEawd"
make="GMC"
modelcar="Compass"
year=2024
engine='16VMPFIOHV'
cylinder=9
fuel='Gasoline'=
mileage=30
transmission='Automatic'
trim='GT'
body='SUV'
doors=3
extc='DiamondBlack'
intc='Black'
drivetrain='All-wheelDrive'
```

4. Encoding the Input Values

Each categorical input was transformed using the respective encoder to match the training format:

```
namelabel=nameencoder.transform([name])[0]
makelabel=makeencoder.transform([make])[0]
modellabel=modelencoder.transform([modelcar])[0]
enginelabel=engineencoder.transform([engine])[0]
fuellabel=fuelencoder.transform([fuel])[0]
transmissionlabel=transmissionencoder.transform([transmission])[0]
trimlabel=trimencoder.transform([trim])[0]
bodylabel=bodyencoder.transform([body])[0]
extclabel=extcencoder.transform([extc])[0]
intclabel=intcencoder.transform([intc])[0]
drivetrainlabel=drivetrainencoder.transform([drivetrain])[0]
```

5. Creating the Input DataFrame

All values were assembled into a single row DataFrame for prediction:

```
input=np.array([namelabel,makelabel,modellabel,year,enginelabel,
                cylinder,fuellabel,mileage,
                transmissionlabel,trimlabel,bodylabel,doors,extclabel,intclabel,drivetrainlabel])
```

After creating the input dataframe using the encoded values, the saved model was loaded and used for predicting the vehicle price. The steps are as follows:

The model successfully predicted the price of the vehicle:

```
model=jb.load("model.pkl")
print("CAR SPECIFICATIONS:\nCAR-NAME: ",name,
    "\nMAKE: ", make,
    "\nMODEL ", modelcar,
    "\nYEAR: ",year,
    "\nENGINE: ",engine,
    "\nCYLINDER: " ,cylinder,
    "\nFUEL: ",fuel,
    "\nMILEAGE: ", mileage,
    "\nTRANSMISSION: ", transmission,
    "\nTRIM ",trim,
    "\nBODY: ",body,
    "\nDOORS: ", doors,
    "\nEXTCOLOR: ",extc,
    "\nINTCOLOR: ", intc,
    "\nDRIVETRAIN: ", drivetrain)

price=model.predict([input])[0]
pricedollar=int(price)
print("PREDICTED VEHICLE PRICE IN USD: ",pricedollar,"USD")
inr=pricedollar*85.63
inr=int(inr)
print("PREDICTED VEHICLE PRICE IN USD (1 usd = 85.63 inr): ",inr,"INR")
```

```
CAR SPECIFICATIONS:
CAR-NAME:  2024DodgeHornetHornetR/TPlusEawd
MAKE:  GMC
MODEL  Compass
YEAR:  2024
ENGINE:  16VMPFIOHV
CYLINDER:  9
FUEL:  Gasoline
MILEAGE:  30
TRANSMISSION:  Automatic
TRIM  GT
BODY:  SUV
DOORS:  3
EXTCOLOR:  DiamondBlack
INTCOLOR:  Black
DRIVETRAIN:  All-wheelDrive
PREDICTED VEHICLE PRICE IN USD:  49012 USD
PREDICTED VEHICLE PRICE IN USD (1 usd = 85.63 inr):  4196897 INR
```

This output confirms that the model is functioning as intended, able to process new, unseen vehicle data and predict a price value based on learned patterns. The result aligns with realistic expectations, which supports the model's effectiveness and readiness for deployment in real-world applications such as vehicle resale portals or dealership valuation systems.

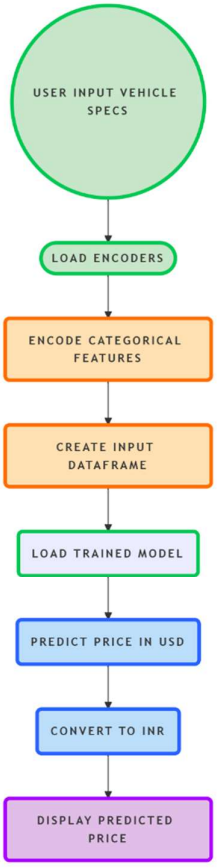
8. Final Output Interpretation

After running the model on the test input, the predicted vehicle price was:

- **Predicted Price (USD):** 49,012 USD
- **Predicted Price (INR):** 41,96,897 INR *(based on 1 USD = 85.63 INR)*

This prediction reflects the model's ability to handle real-world input and return an estimated price that can assist users in decision-making. The transformation of categorical variables using pre-trained label encoders ensures consistency with training data, resulting in a reliable and interpretable prediction.

DATA FLOW DIAGRAM:-



CONCLUSION

In this project, a robust machine learning pipeline was developed and deployed to accurately predict vehicle prices based on a range of input features. Using preprocessing techniques like label encoding and powerful regression models like XGBoost, the system achieved high accuracy with a training score of **99.5** and a test score of **89.1**, demonstrating its strong generalization capabilities.

The model was successfully tested on new, unseen vehicle data, where it returned realistic price predictions, both in USD and INR. This confirms the practical utility and reliability of the model for real-world applications, such as used car dealerships or online vehicle evaluation platforms.

Overall, the project showcases how machine learning can be effectively applied to the automotive domain to automate and improve price estimation tasks with high accuracy and efficiency.