IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Hierarchical Vehicle Recommendation Platform Using RFC And Proximity Analytics

[1] Sahana Sharma M, Assistant Professor, Department of CSE (AI) , Dayanand Sagar Academy of Technology And Management, Bangalore-82, [2]Gurudatta C S, [3]Anirudh L, , [4]Ranjith S,[5]J.S Pranav

[1]Name of Department :Department of CSE-AI,

[1]Name of organization: Dayananda Sagar Academy of Technology and Management, India

Abstract: In this paper, we propose a Vehicle Recommendation System leveraging the power of Artificial Intelligence (AI) and Random Forest methodology. Our system is designed to recommend vehicles to users based on their specific preferences and requirements. Using a dataset of 1265 entries and 27 features, the AI model identifies the most suitable vehicle for customers by analyzing their responses to a series of pre-determined questions. This project demonstrates the application of Random Forest for classification and recommendation tasks, showcasing its effectiveness in decision-making processes.

Index Terms - Component, Vehicle recommendation, AI, Random Forest, Dataset, Machine Learning.

I. Introduction

The advent of artificial intelligence has revolutionized decision-making in various domains, including the automotive industry. In the era of rapid technological advancements, artificial intelligence (AI) has revolutionized multiple industries by streamlining complex processes, enhancing decision-making, and delivering tailored user experiences. Among these industries, the automotive sector has witnessed significant transformations, with AI enabling smarter manufacturing, autonomous driving, and personalized customer support. One such application of AI is in the development of recommendation systems, which assist users in making informed decisions by analyzing their preferences and requirements.

This project introduces a Vehicle Recommendation System, a novel AI-driven solution designed to simplify the often- overwhelming task of vehicle selection. Choosing a vehicle involves navigating through a multitude of factors, such as budget constraints, fuel efficiency, safety ratings, and specific feature preferences. Manually evaluating these parameters can be daunting, especially given the extensive range of options available in the market. To address this challenge, our system employs a machine learning model based on the Random Forest algorithm, a powerful classification technique known for its robustness and accuracy. The system operates on a carefully curated dataset comprising 1265 entries and 27 features, representing various attributes of vehicles, such as type (e.g., SUV, sedan, hatchback), price range, mileage, safety ratings, and customer reviews. The dataset serves as the foundation for training the AI model to predict and recommend vehicles that best match a user's stated preferences. Users interact with the system through a simple interface, where they answer a set of predefined questions about their requirements. Based on their inputs, the system processes the data and provides a ranked list of recommendations, highlighting the most suitable vehicles. The Random Forest algorithm, a key component of this project, is an ensemble learning technique that constructs multiple decision trees during training and aggregates their outputs through majority voting. This methodology not only enhances prediction accuracy but also mitigates the risk of overfitting, making it an ideal choice for handling complex and multidimensional datasets like the one used in this project. The model's ability to discern patterns and relationships within the data enables it to deliver reliable and transparent recommendations.

The primary goal of this project is to improve the customer experience by offering a data-driven and user-centric solution for vehicle

selection. Unlike traditional methods, which may rely on generic filters or limited comparisons, this system provides personalized recommendations tailored to each user's unique needs. Additionally, the project emphasizes transparency by explaining the factors influencing each recommendation, fostering user trust and satisfaction.

While the current implementation is based on a static dataset, the system lays the groundwork for future enhancements, such as integrating real-time data from automotive APIs, incorporating advanced algorithms like Gradient Boosting, and expanding the range of recommendation criteria. These improvements will further refine the system's accuracy and applicability, making it a comprehensive tool for assisting customers in the decision-making process..

II. Dataset Description

The foundation of the Vehicle Recommendation System lies in its dataset, which is meticulously curated to ensure accurate and reliable predictions. For this project, the dataset was sourced from Kaggle, a well-known platform for diverse and high-quality datasets. Initially, the dataset comprised a significantly larger number of entries and features, representing an extensive range of vehicle attributes and customer feedback. However, to enhance the quality and relevance of the data for our specific use case, we performed extensive data cleaning and preprocessing, reducing the dataset to 1265 entries and 27 features.

The original dataset contained over 3000 entries and more than 50 features, including redundant, inconsistent, or incomplete records. Many entries had missing or ambiguous values, which could lead to inaccuracies during the model training phase. Moreover, certain features were irrelevant or contributed minimal variance to the predictive model. For instance, attributes such as manufacturing plant location or unique serial numbers were not directly related to customer preferences and were subsequently removed. By refining the dataset to this manageable yet comprehensive form, we ensured that the Random Forest model would have access to high-quality data, enabling it to deliver accurate and meaningful recommendations. This structured dataset is at the core of our project, empowering the system to analyze user inputs and match them with the most suitable vehicle options.

III.RANDOM FOREST METHODOLOGY

The Random Forest, a cornerstone of this project, is an ensemble learning method that excels in classification and regression tasks. Developed by Leo Bierman, it operates by constructing a collection of decision trees during the training process and combines their outputs to produce more accurate and robust predictions. The core idea behind Random Forest is to aggregate the results of multiple trees, reducing the risk of overfitting and improving generalizability when applied to unseen data. This methodology makes it an ideal choice for handling the complex and diverse dataset used in our Vehicle Recommendation System. In this project, the Random Forest algorithm was applied to classify vehicles based on user preferences. By training on a dataset of 1265 entries and 27 features, the algorithm learned to identify patterns and relationships within the data. When a user provides their preferences, such as budget, fuel efficiency, and vehicle type, the model processes the inputs and predicts the most suitable vehicles. The use of majority voting ensures that the recommendations are not biased by any single tree, resulting in accurate and reliable outputs.
The Random Forest methodology has proven to be an effective and efficient solution for building the Vehicle Recommendation System, delivering high accuracy and robustness in addressing the complex requirements of vehicle selection.

IV.STEPS IN MODEL DEVELOPMENT

1. Data Collection: The dataset was collected from various automotive platforms and curated for accuracy.
2. Data Preprocessing: Missing values were imputed, and categorical variables were encoded.
3. Model Training: The Random Forest algorithm was applied to train the model, using an 80-20 train-test split.
4. Validation: The model's performance was validated using metrics such as accuracy, precision, and recall.

V. USER INTERACTION

The key component of the Vehicle Recommendation System is its user interaction design, which bridges the gap between the machine learning model and end users. The interaction is crafted to be user-friendly, intuitive, and responsive, ensuring that customers can seamlessly access personalized vehicle recommendations. This section outlines the design and implementation of the user interaction process, including the use of modern tools like Firebase for the front end and communication with the backend model. The user interface (UI) was developed with a focus on simplicity and functionality, ensuring a smooth experience for users. The front end of the system was built using modern web development frameworks, and Firebase was employed for its real-time database and hosting capabilities. Firebase enabled quick data synchronization and user interaction logging, making the system responsive and efficient. Frontend Development**:** The front end was developed using popular frameworks like React.js or Angular.js for web applications, and Flutter or Kotlin for mobile applications. These frameworks ensured a responsive and engaging user experience. Backend Communication**:** Firebase was utilized for real-time data handling and seamless interaction between the user interface and the backend model. Model Integration: The trained Random Forest model was hosted on a backend server, with APIs enabling interaction between the front end and the machine learning model. Python's Flask or Fast API framework was used for this integration.

VI.RESEARCH METHODOLOGY

The research methodology for the Vehicle Recommendation System was carefully designed to ensure a systematic approach to problem-solving, enabling the development of a robust and accurate recommendation model. The methodology integrates data science, machine learning, and user interaction design, creating a seamless framework for personalized vehicle recommendations. The project's methodology is divided into several phases, each focusing on key aspects such as data acquisition, preprocessing, model development, and evaluation.

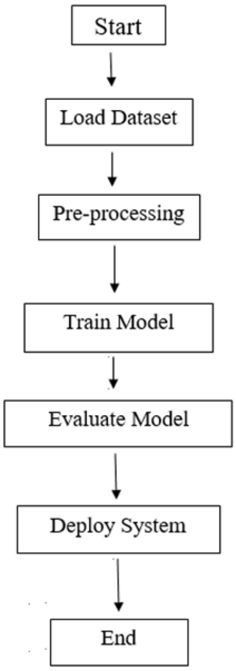
Data Collection and Acquisition

The foundation of this research lies in the dataset, which was sourced from Kaggle, a well-known platform for high-quality datasets. The original dataset consisted of over **3000 entries** and more than **50 features** that represented various attributes of vehicles, such as type, price, mileage, fuel type, safety ratings, and user reviews. This dataset was chosen due to its richness and diversity, providing a comprehensive basis for model training and testing.

To ensure the dataset's relevance and reliability, additional secondary data was collected from trusted automotive sources, including vehicle specification databases and review platforms. These supplemental sources helped fill gaps and validated the primary dataset's consistency.

Data preprocessing is a critical step in any machine learning project, and this system was no exception. The preprocessing stage aimed to prepare the dataset for optimal model performance.

- Handling Missing Values**:** Missing or null values in critical fields, such as price or mileage, were imputed using statistical techniques like mean or median imputation. Entries with excessive missing data were removed to maintain the dataset's integrity.
- Outlier Detection**:** Outliers were identified using statistical methods and visualization tools. For instance, vehicles with abnormally high prices or mileage beyond the range of normal operating vehicles were examined and either corrected or excluded.
- Standardization and Normalization**:** Continuous features, such as mileage and price, were normalized to bring them into a consistent range, ensuring the model treats all features equally during training.



.

VII.DATA AND SOURCES OF DATA

For The success of the Vehicle Recommendation System hinges on the quality, diversity, and relevance of the data used to train the machine learning model. The data for this project was sourced from Kaggle, a platform renowned for its wide range of publicly available datasets. The original dataset contained over 3000 entries and 50 features, representing various aspects of vehicles and their specifications. Additionally, secondary data sources were utilized to enhance the dataset's comprehensiveness and ensure that it catered to the project's objectives.

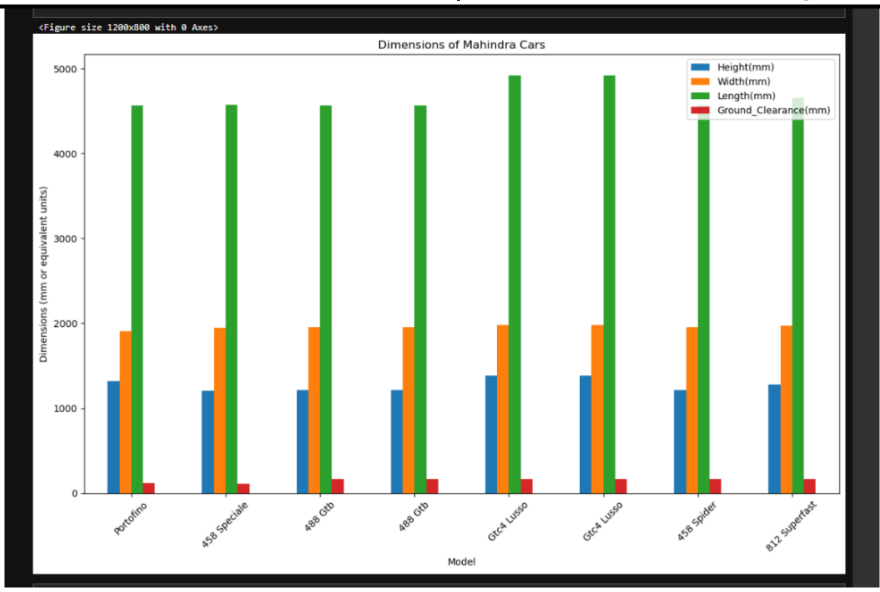
The primary dataset was obtained from Kaggle, which provided a rich repository of vehicle-related information. This dataset included a broad spectrum of attributes, such as:

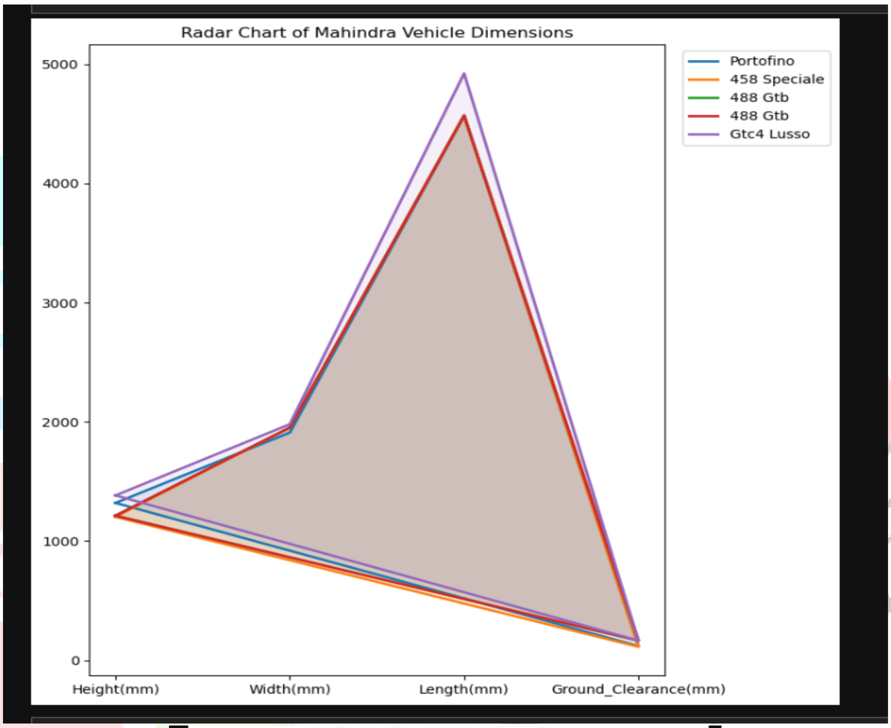
- **Vehicle Specifications:** Features like vehicle type (SUV, Sedan, Hatchback), fuel type (Petrol, Diesel, Electric), engine capacity, and mileage.
- **Pricing Information:** Comprehensive data on vehicle price ranges, categorized into low, medium, and high tiers.
- **Safety Features:** Details on safety equipment, crash test ratings, and advanced safety technologies.
- **Customer Feedback:** Reviews and ratings provided by customers, reflecting their experiences and satisfaction levels.

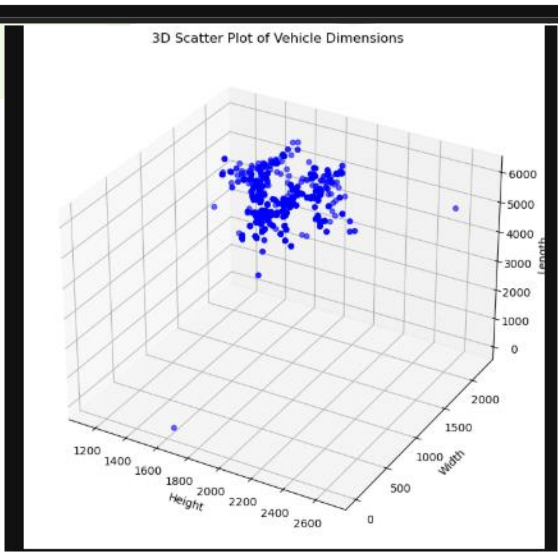
The dataset from Kaggle was selected for its relevance and its structured nature, which made it ideal for training a machine learning model focused on vehicle classification and recommendation.

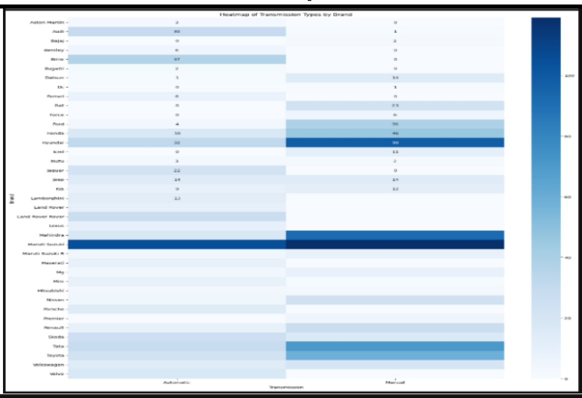
VIII. RESULTS AND DISCUSSIONS

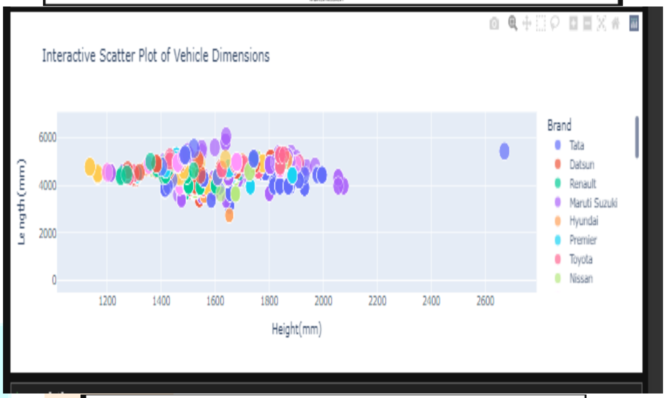
The implementation of the Vehicle Recommendation System using the Random Forest algorithm yielded promising results, demonstrating the effectiveness of machine learning in solving complex classification and recommendation problems. The system was evaluated on its ability to recommend vehicles based on user preferences and perform consistently across diverse scenarios. This section elaborates on the outcomes, the analysis of results, and the key insights drawn from the project.

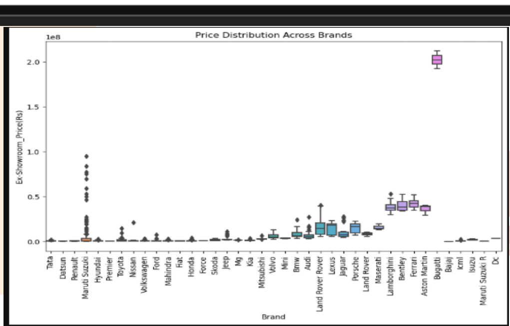

<Figure size 1200x800 with 0 Axes>
Dimensions of Mahindra Cars
Height(mm)
Width(mm)
Length(mm)
Ground_Clearance(mm)
Dimensions (mm or equivalent units)
Portofino
458 Speciale
488 Gtb
488 Gtb
Gtc4 Lusso
Gtc4 Lusso
458 Spider
812 Superfast
Model

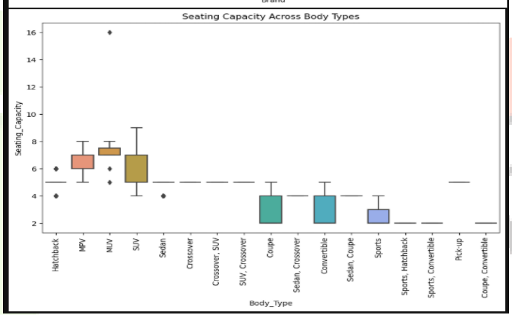

Radar Chart of Mahindra Vehicle Dimensions
Portofino
458 Speciale
488 Gtb
488 Gtb
Gtc4 Lusso
Height(mm)
Width(mm)
Length(mm)
Ground_Clearance(mm)

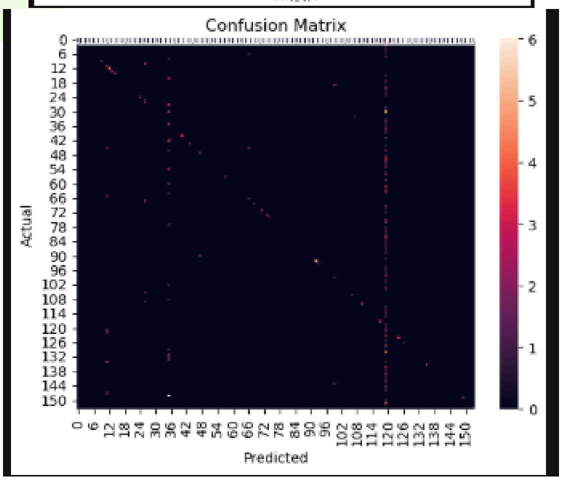

3D Scatter Plot of Vehicle Dimensions
Height
Width




Interactive Scatter Plot of Vehicle Dimensions
Length(mm)
Height(mm)
Brand
Tata
Datsun
Renault
Maruti Suzuki
Hyundai
Premier
Toyota
Nissan

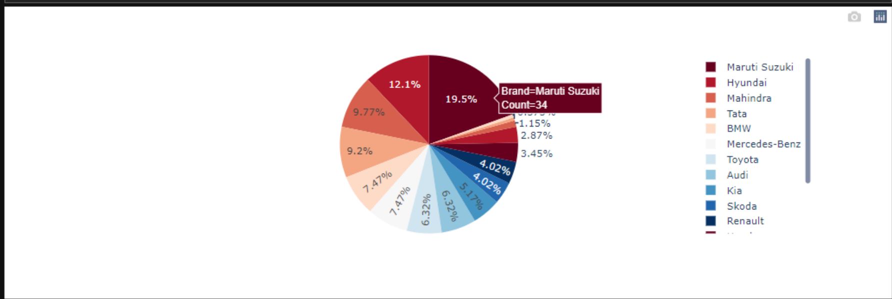

Price Distribution Across Brands
Ex-Showroom_Price(Rs)
Brand


Seating Capacity Across Body Types
Seating_Capacity
Body_Type

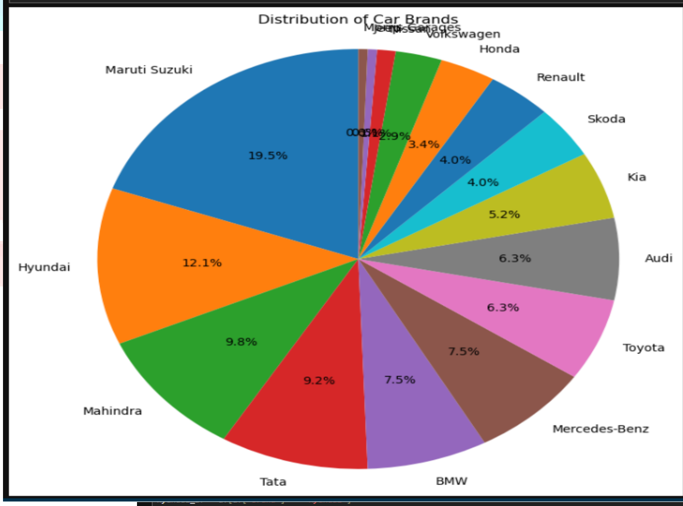

Confusion Matrix
Actual
Predicted

```
[54]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1267 entries, 0 to 1266
Data columns (total 22 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Brand                    1267 non-null   object
 1   Model                    1267 non-null   object
 2   Variant                  1267 non-null   object
 3   Fuel_Type                1267 non-null   object
 4   Transmission             1267 non-null   object
 5   Ex-Showroom_Price(Rs)    1267 non-null   int64
 6   Drivetrain               1267 non-null   object
 7   Height(mm)               1267 non-null   int64
 8   Length(mm)               1267 non-null   float64
 9   Width(mm)                1267 non-null   float64
 10  Body_Type                1267 non-null   object
 11  Ground_Clearance(mm)     1267 non-null   float64
 12  City_Mileage(km/l)       1267 non-null   float64
 13  ARAI_Certified_Mileage(km/l) 1267 non-null float64
 14  Kerb_Weight(kg)          1267 non-null   int64
 15  Power(PS)                1267 non-null   float64
 16  Torque(NM)               1267 non-null   float64
 17  Seating_Capacity         1267 non-null   int64
 18  Boot_Space(litre)        1267 non-null   float64
 19  Battery(kWh)             1267 non-null   float64
 20  Electric_Range(km)       1267 non-null   float64
 21  Number_of_Airbags        1267 non-null   int64
dtypes: float64(10), int64(5), object(7)
memory usage: 217.9+ KB
```

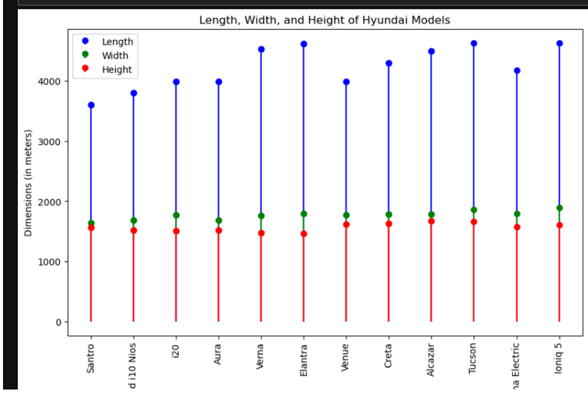
```
8]: def generate_chart(names, values):
        df = pd.DataFrame({'Brand': names, 'Count': values})
        fig = px.pie(df, names='Brand', values='Count', color='Brand', color_discrete_sequence=px.colors.sequential.RdBu)
        return fig
    generate_chart(df['Brand'].value_counts().index, df['Brand'].value_counts().values)
```



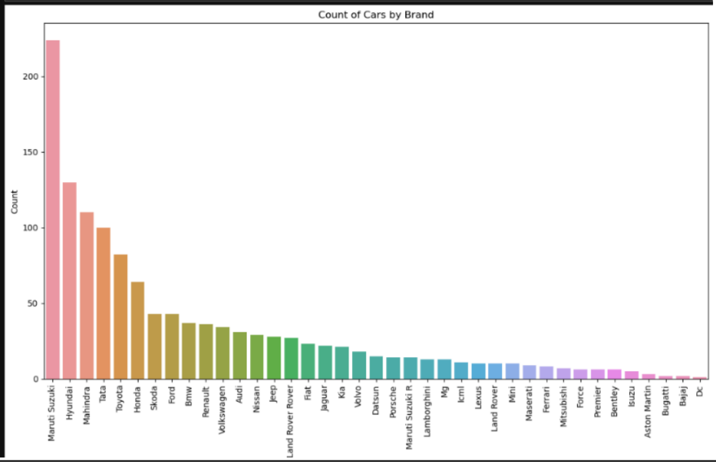

```
import matplotlib.pyplot as plt
brand_counts = df['Brand'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(brand_counts, labels=brand_counts.index, autopct='%1.1f%%', startangle=90)
plt.title('Distribution of Car Brands')
plt.axis('equal')
plt.show()
```



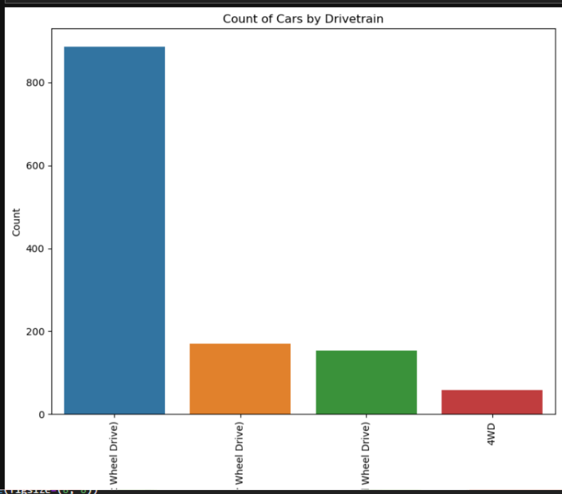

```
hyundai_data = hyundai_df[['Model', 'Length (mm)', 'Width (mm)', 'Height (mm)']]

fig, ax = plt.subplots(figsize=(10, 6))

ax.stem(hyundai_data['Model'], hyundai_data['Length (mm)'], basefmt=" ", linefmt='b-', markerfmt='bo', label='Length')
ax.stem(hyundai_data['Model'], hyundai_data['Width (mm)'], basefmt=" ", linefmt='g-', markerfmt='go', label='Width')
ax.stem(hyundai_data['Model'], hyundai_data['Height (mm)'], basefmt=" ", linefmt='r-', markerfmt='ro', label='Height')


plt.title('Length, Width, and Height of Hyundai Models')
plt.xlabel('Car Model')
plt.ylabel('Dimensions (in meters)')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



```
[6]: plt.figure(figsize=(12, 8))
sns.countplot(df, x='Brand', order=df['Brand'].value_counts().index)
plt.xticks(rotation=90)
plt.title('Count of Cars by Brand')
plt.xlabel('Brand')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```

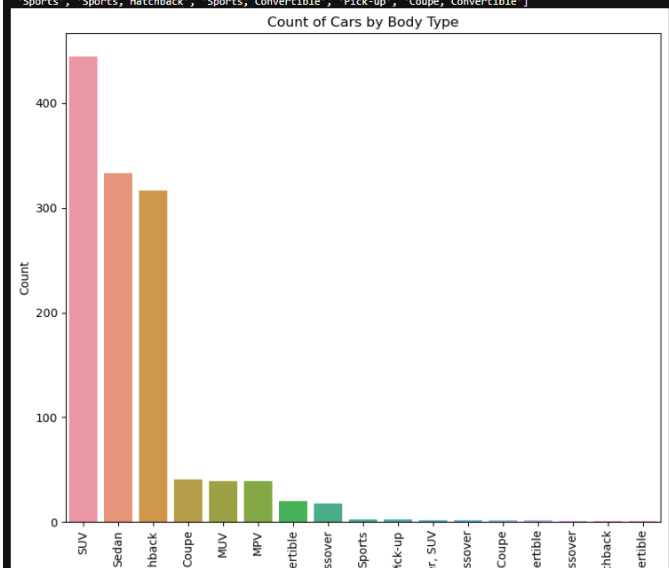


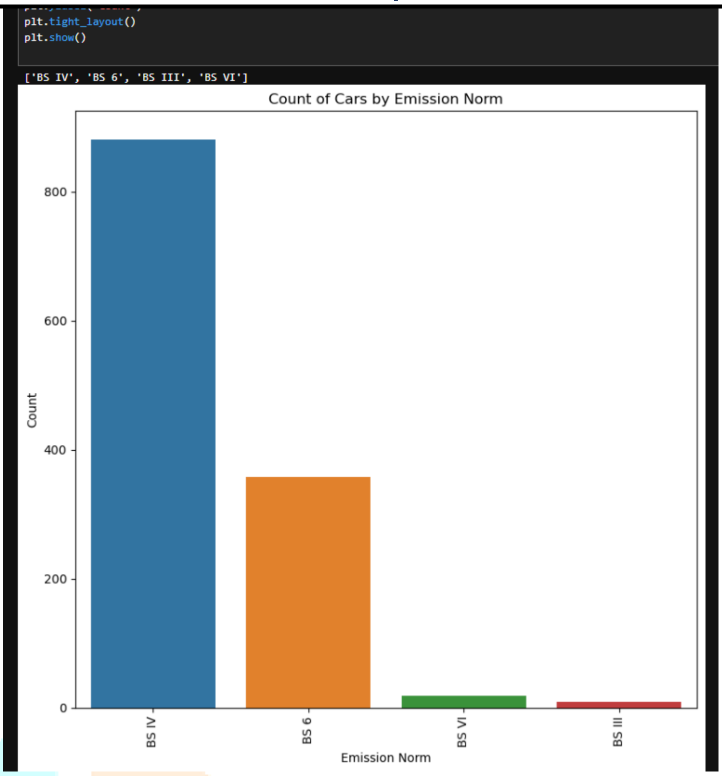

```
3]: plt.figure(figsize=(8, 8))
sns.countplot(df, x='Drivetrain', order=df['Drivetrain'].value_counts().index)
plt.xticks(rotation=90)
plt.title('Count of Cars by Drivetrain')
plt.xlabel('Drivetrain')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```

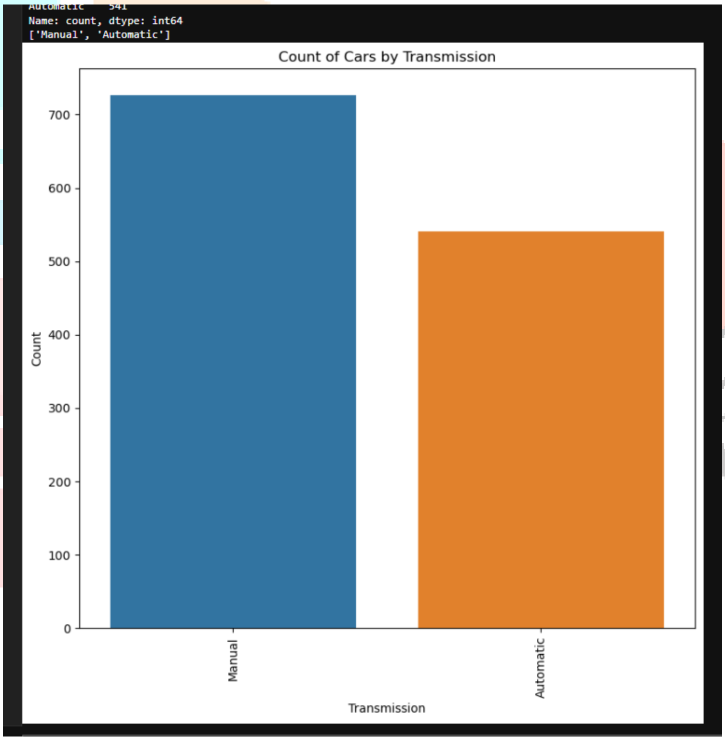


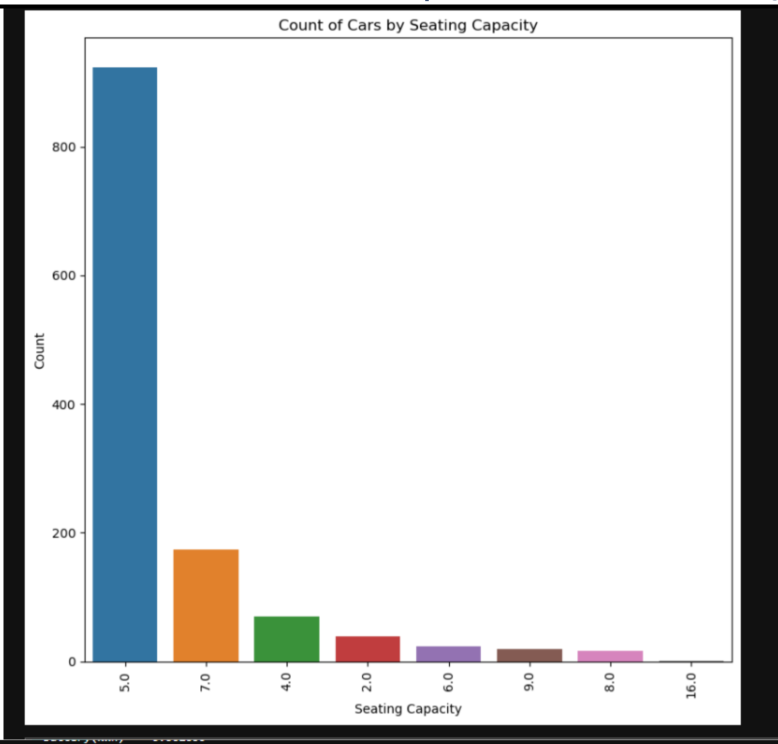

```
plt.figure(figsize=(8, 8))
sns.countplot(df, x='Body_Type', order=df['Body_Type'].value_counts().index)
plt.xticks(rotation=90)
plt.title('Count of Cars by Body Type')
plt.xlabel('Body Type')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```

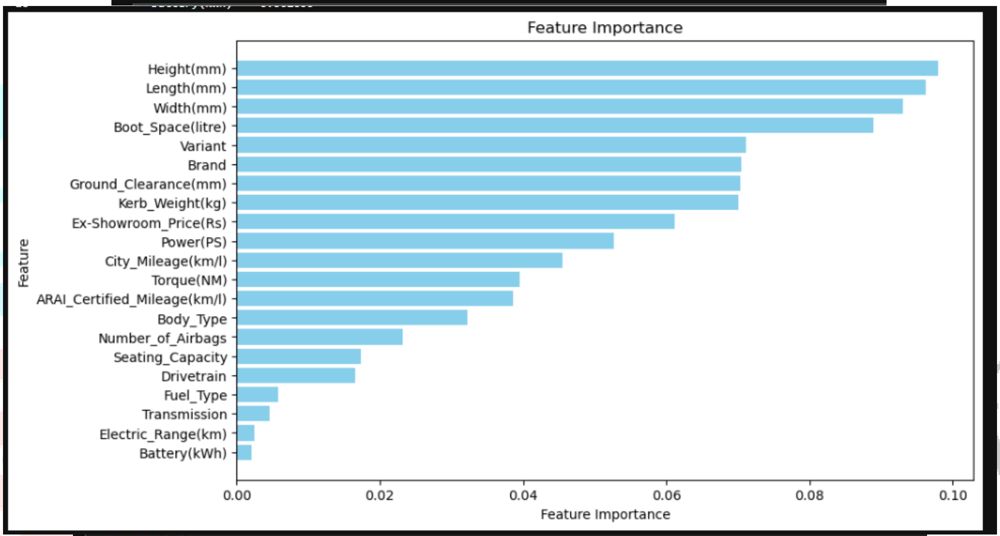
```
['Hatchback', 'MPV', 'MUV', 'SUV', 'Sedan', 'Crossover', 'Crossover, SUV', 'SUV, Crossover', 'Coupe', 'Sedan, Crossover', 'Convertible', 'Sedan, Coupe',
'Sports', 'Sports, Hatchback', 'Sports, Convertible', 'Pick-up', 'Coupe, Convertible']
```




plt.tight_layout()
plt.show()
['BS IV', 'BS 6', 'BS III', 'BS VI']
Count of Cars by Emission Norm
Count
800
600
400
200
0
BS IV
BS 6
BS VI
BS III
Emission Norm


Name: count, dtype: int64
['Manual', 'Automatic']
Count of Cars by Transmission
Count
700
600
500
400
300
200
100
0
Manual
Automatic
Transmission






```
import pickle
import pandas as pd

from sklearn.preprocessing import LabelEncoder, StandardScaler

#pickle for saving python objects; encoders and scalers


df = pd.read_csv("cleaned_dataset_NB18.csv")
categorical_columns = df.select_dtypes(include=['object']).columns
encoders = {}

#object 'le' created; fit transform - assigns a unique int for each category in the col


for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    encoders[col] = le

numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns
scaler = StandardScaler()

#standardizes numerical data by removing the mean and scaling to unit variances

df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
with open('encoders.pkl', 'wb') as f:
    pickle.dump(encoders, f)

with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)

#we create the pkl files and dump the encoded and scalers

sample_data = df.iloc[0:1]
sample_data.to_csv('new_data.csv', index=False)
print("Sample data added to new_data.csv")
#generating sameple data
```

```python
# Save DataFrame to a CSV file
df.to_csv('cleaned_dataset_NB16.csv', index=False)
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
X = df.drop(columns=["Model"])
y = df["Model"]

labelencoder = LabelEncoder()
categorical_columns = X.select_dtypes(include=['object']).columns

for col in categorical_columns:
    X[col] = labelencoder.fit_transform(X[col])
```

```python
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Import Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Initialize the Random Forest model
rf = RandomForestClassifier(random_state=42)

# Perform GridSearchCV for hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)

# Best parameters and model evaluation
best_rf = grid_search.best_estimator_
print("Best Hyperparameters:", grid_search.best_params_)

# Predict on the test set
y_pred = best_rf.predict(X_test)

# Model evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
C:\Users\srikrishna j\anaconda3\ANACONDA\Lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populated class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
Best Hyperparameters: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Accuracy: 0.8700787401574803

Classification Report:
               precision    recall  f1-score   support
```

```
        Xcent       1.00      1.00      1.00         2
  Xcent Prime       1.00      1.00      1.00         1
           Xe       1.00      1.00      1.00         2
           Xf       1.00      1.00      1.00         1
           Xj       1.00      1.00      1.00         2
          X16       1.00      1.00      1.00         1
       Xuv300       1.00      1.00      1.00         6
       Xuv500       1.00      1.00      1.00         2
         Xylo       1.00      1.00      1.00         1
        Yaris       1.00      1.00      1.00         4
   Z4 Roadster      0.00      0.00      0.00         1
         Zest       1.00      1.00      1.00         1

     accuracy                           0.87       254
    macro avg       0.72      0.75      0.73       254
 weighted avg       0.85      0.87      0.85       254
```

```
Best Hyperparameters:
{'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

Accuracy: 0.86

Classification Report:
```

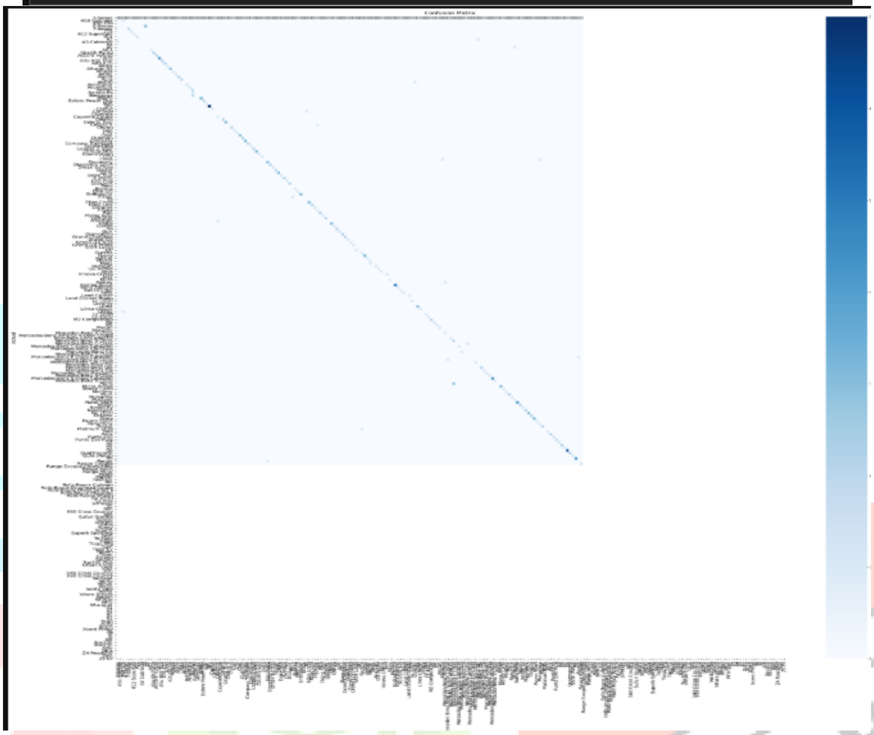
Class	Precision	Recall	F1-Score	Support
0	0	0.0	0.0	0
2	0	0.0	0.0	1
3	0	0.0	0.0	0
4	0	0.0	0.0	3
5	0	0.0	0.0	0
6	1	1.0	1.0	2
9	1	1.0	1.0	1
10	1	1.0	1.0	1
12	1	1.0	1.0	1
13	0	0.0	0.0	1
14	0	0.0	0.0	1
17	0	0.0	0.0	1
20	1	1.0	1.0	1
22	1	1.0	1.0	2
23	1	1.0	1.0	2
25	1	1.0	1.0	4
27	1	1.0	1.0	2
28	1	1.0	1.0	2

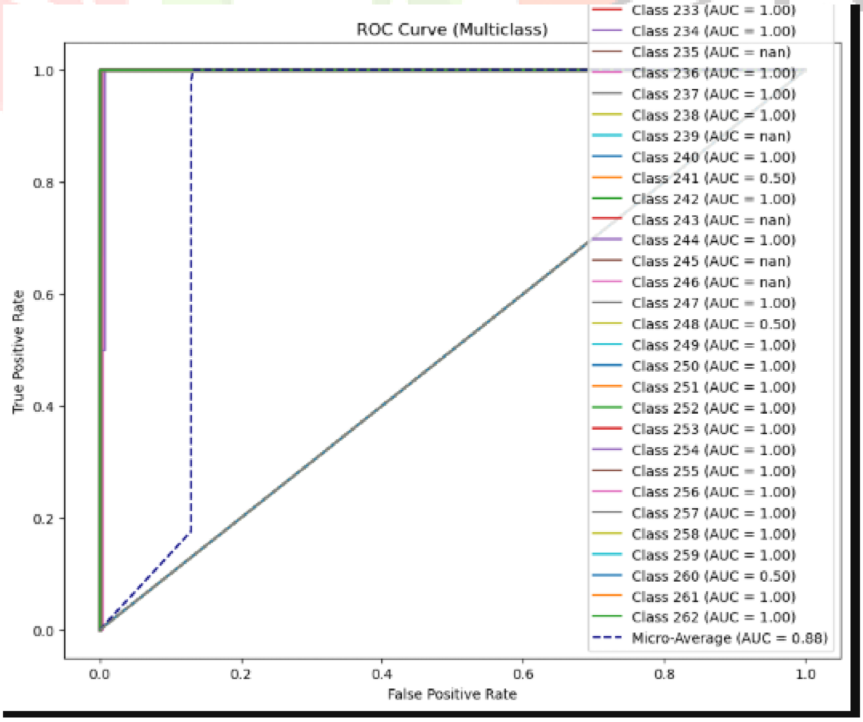
Ln 44, Col 85 | 1,054,676 characters

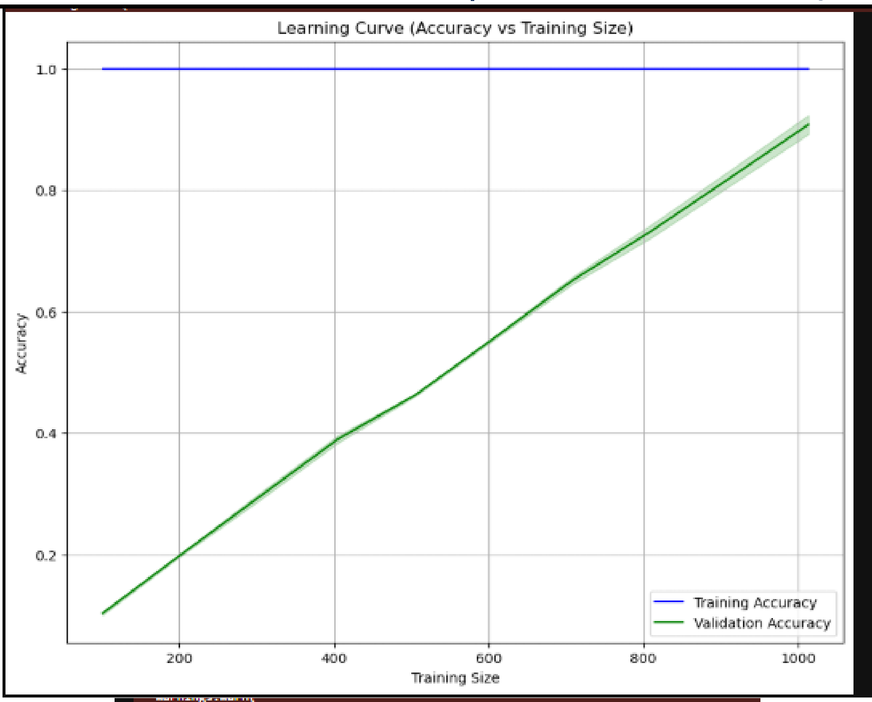
```
X = df.drop('Model', axis=1)
y = df['Model']

best_rf = RandomForestClassifier(n_estimators=100)
best_rf.fit(X, y)

#joblib used for efficiency of large model objects, classes from decision trees; model persistence, bs
import joblib
joblib.dump(best_rf, 'best_rf_model.joblib')
print("Model, encoders, and scaler saved successfully.")
```

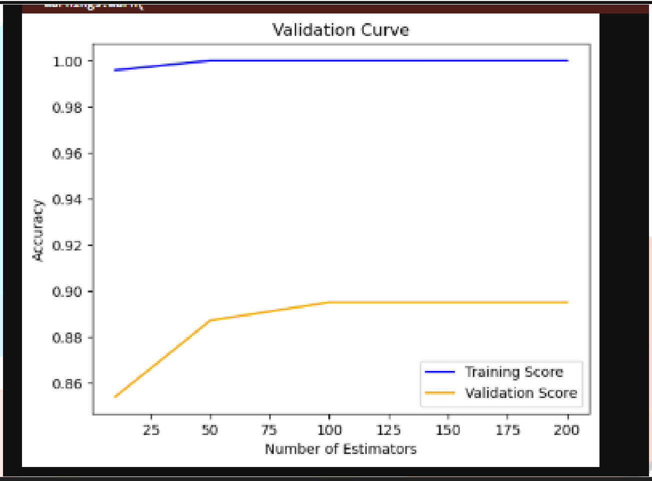
```
Model, encoders, and scaler saved successfully.
```

```
#ukw to explain yoo
import joblib
import pickle
import pandas as pd
rf_model = joblib.load('best_rf_model.joblib')
with open('encoders.pkl', 'rb') as f:
    encoders = pickle.load(f)
with open('scaler.pkl', 'rb') as f:
    scaler = pickle.load(f)
```










```python
import joblib
import pickle
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from typing import Dict, List, Tuple
from functools import lru_cache
from sklearn.neighbors import NearestNeighbors

class VehicleRecommender:
    #loading pre trianed best rf model
    def __init__(self, model_path: str, encoders_path: str, scaler_path: str, data_path: str):
        self.rf_model = joblib.load(model_path)
        with open(encoders_path, 'rb') as f:
            self.encoders = pickle.load(f)
        with open(scaler_path, 'rb') as f:
            self.scaler = pickle.load(f)
        self.df = pd.read_csv(data_path)

        #preped a dictionary linking brands to their respective models
        self.brand_models = self._create_brand_model_mapping()
        #similarity model - builds a knn model for finding similar car models (within the same range) but w/ differnt brands
        self._setup_similarity_model()




    def _setup_similarity_model(self):
        # Prepare feature matrix for similarity comparison
        feature_df = pd.DataFrame()

        # Add numeric features
        if 'Seating_Capacity' in self.df.columns:
            feature_df['Seating_Capacity'] = self.df['Seating_Capacity']
        # Add encoded categorical features
        cat_features = ['Body_Type', 'Fuel_Type', 'Transmission', 'Price_Range']
        for feat in cat_features:
            if feat in self.df.columns:
                feature_df[feat] = self.encoders[feat].transform(self.df[feat])
        # Normalize features
        self.feature_matrix = StandardScaler().fit_transform(feature_df)
        self.similarity_model = NearestNeighbors(n_neighbors=min(10, len(self.df)), metric='euclidean')
        self.similarity_model.fit(self.feature_matrix)
```

```python
@lru_cache(maxsize=32)

#creating a mapping of the car brands to their respective models
def _create_brand_model_mapping(self) -> Dict[str, set]:
    return {brand: set(group['Model'])
            for brand, group in self.df.groupby('Brand')}

#retrives default median values for numeric features of a given brand
#used for filling any missing values in user inputs
def _get_brand_defaults(self, brand: str) -> Dict[str, float]:
    brand_data = self.df[self.df['Brand'] == brand]
    if brand_data.empty:
        brand_data = self.df
    numeric_cols = self.df.select_dtypes(include=['int64', 'float64']).columns
    return {col: brand_data[col].median() for col in numeric_cols}


#creating an empty df called features
#getting deafult values of the brand
#filling the default values
#transforming the categorical features
#filling missing values
#for? - best input handling/ handling any unprovided or missing values
def _prepare_features(self, user_input: Dict) -> pd.DataFrame:
    features = pd.DataFrame(columns=self.rf_model.feature_names_in_, index=[0])
    defaults = self._get_brand_defaults(user_input['Brand'])

    for col, value in defaults.items():
        if col in features.columns:
            features[col] = value

    for col in features.columns:
        if col in self.encoders:
            if col in user_input:
                try:
                    features[col] = self.encoders[col].transform([user_input[col]])
                except ValueError:
                    features[col] = self.encoders[col].transform([self.encoders[col].classes_[0]])
            else:
                features[col] = self.encoders[col].transform([self.encoders[col].classes_[0]])

    return features.fillna(0)
#Why do you think the model required me to enter all the car values to return the "model,"
#while the entire project was designed to only require 5 categorical inputs to return the "model"?
#training vs predictions mismatching
#no proper data preprocessing


def get_similar_vehicles(self, model_name: str, n_similar: int = 5) -> List[Tuple[str, float, Dict]]:
    model_idx = self.df[self.df['Model'] == model_name].index[0]
    model_features = self.feature_matrix[model_idx].reshape(1, -1)
    #a matrix extrated from model_features that contains the numerical feature representations

    distances, indices = self.similarity_model.kneighbors(model_features)
    #gets the closest distance and indecies values between the "Recommended model" and other models
    similar_vehicles = []

    for idx, dist in zip(indices[0][1:], distances[0][1:]):
        similar_model = self.df.iloc[idx]
        if similar_model['Model'] != model_name: # <------
            similarity_score = 1 / (1 + dist)
            similar_vehicles.append((
                similar_model['Model'],
                similarity_score,
                similar_model.to_dict()
            ))

    return similar_vehicles[:n_similar]
```

```python
def get_recommendations(self, user_input: Dict, top_n: int = 5, include_similar: bool = True) -> List[Tuple[str, float, Dict]]:
    features = self._prepare_features(user_input)
    probabilities = self.rf_model.predict_proba(features)[0]
    #preparing the features by processing the user inouts and formats it to match the feature set expected by the model
    #pbs because - RF outputs are pbs of each possible outcome. So using pbs would allow us to understand how likely the vehicle recommended is th
    #right choice.
    brand_models = self.brand_models.get(user_input['Brand'], set())
    recommendations = []

    #filtering and then collecting then iterating on probalities and then collecting the recommendations
    for idx, prob in enumerate(probabilities):
        model_name = self.df['Model'].unique()[idx] if idx < len(self.df['Model'].unique()) else f"Unknown Model {idx}"
        if model_name in brand_models:
            model_data = self.df[self.df['Model'] == model_name].iloc[0].to_dict()
            recommendations.append((model_name, prob, model_data))

    brand_recommendations = sorted(recommendations, key=lambda x: x[1], reverse=True)[:top_n]
    #sorting based on probability


    if include_similar and brand_recommendations:
        similar_vehicles = self.get_similar_vehicles(brand_recommendations[0][0])
        return brand_recommendations + similar_vehicles

    return brand_recommendations




def format_recommendation(rec_tuple: Tuple[str, float, Dict], is_similar: bool = False) -> str:
    model, prob, data = rec_tuple
    prefix = "Similar: " if is_similar else ""
    return (f"{prefix}{model}: {prob:.1%} confidence\n"
            f"  Brand: {data['Brand']}\n"
            f"  Price: {data['Price_Range']}\n"
            f"  Fuel: {data['Fuel_Type']}\n"
            f"  Body: {data['Body_Type']}\n"
            f"  Transmission: {data['Transmission']}")

if __name__ == "__main__":
    recommender = VehicleRecommender(
        'best_rf_model.joblib',
        'encoders.pkl',
        'scaler.pkl',
        'cleaned_dataset_NB18.csv'
    )

    test_input = {
        'Brand': 'Tata',
        'Fuel_Type': 'Petrol',
        'Transmission': 'Manual',
        'Seating_Capacity': 5,
        'Body_Type': 'SUV',
        'Price_Range': '10-15L'
    }

    try:
        recommendations = recommender.get_recommendations(test_input, include_similar=True)
        print(f"\nTop {test_input['Brand']} Recommendations:")
        for i, rec in enumerate(recommendations):
            is_similar = i >= 5
            print(format_recommendation(rec, is_similar))
            print()
    except Exception as e:
        print(f"Error: {str(e)}")
```

twentytwo.txt recommendations.txt

File Edit View

Type	Model	Confidence	Brand	Price	Fuel	Body	Transmission
Recommended	Nexon	4.0%	Tata	5-10L	Petrol	SUV	Manual
Recommended	Altroz	1.0%	Tata	5-10L	Petrol	Hatchback	Manual
Recommended	Nexon Ev	1.0%	Tata	10-15L	Electric	SUV	Automatic
Recommended	Winger	1.0%	Tata	10-15L	Diesel	MUV	Manual
Recommended	Nano Genx	0.0%	Tata	<5L	Petrol	Hatchback	Manual
Similar	Xuv300	100.0%	Mahindra	5-10L	Petrol	SUV	Manual
Similar	Ecosport	100.0%	Ford	5-10L	Petrol	SUV	Manual
Similar	Ecosport	100.0%	Ford	5-10L	Petrol	SUV	Manual

```
[49]: df.head(10)
```

[49]:

	Brand	Model	Type	Engine Capacity (cc)	Power (PS)	Torque (Nm)	Transmission	Fuel Type	Mileage (kmpl)	Top Speed (kmph)	...	Fuel Tank Capacity (L)	Length (mm)	Width (mm)	Height (mm)	Wheelbase (mm)	Braking System	Seati Capac
0	Mahindra	XUV700	SUV	2198.0	185	420	6-Speed MT/AT	Diesel	16.50	200	...	60.0	4695	1890	1755	2750	All Disc	
1	Mahindra	Thar	SUV	2184.0	130	300	6-Speed MT/AT	Diesel	15.20	155	...	57.0	3985	1855	1844	2450	Disc/Drum	
2	Mahindra	Scorpio N	SUV	2198.0	175	400	6-Speed MT/AT	Diesel	15.40	180	...	57.0	4662	1917	1857	2750	All Disc	
3	Mahindra	Scorpio Classic	SUV	2184.0	132	300	6-Speed MT	Diesel	15.00	160	...	57.0	4456	1820	1995	2680	Disc/Drum	
4	Mahindra	Bolero	SUV	1493.0	75	210	5-Speed MT	Diesel	16.70	120	...	50.0	3995	1745	1880	2680	Disc/Drum	
5	Mahindra	Bolero Neo	SUV	1493.0	100	260	5-Speed MT	Diesel	17.30	140	...	50.0	3995	1795	1817	2680	Disc/Drum	
							6-Speed											

```
[50]: df.count()

[50]: Brand                     114
      Model                     114
      Type                      114
      Engine Capacity (cc)       96
      Power (PS)                114
      Torque (Nm)               114
      Transmission              114
      Fuel Type                 114
      Mileage (kmpl)             96
      Top Speed (kmph)          114
      Electric Range (km)        18
      Price (INR Lakhs)         114
      Battery Capacity (kWh)     18
      Year                      114
      Fuel Tank Capacity (L)     96
      Length (mm)               114
      Width (mm)                114
      Height (mm)               114
      Wheelbase (mm)            114
      Braking System            114
      Seating Capacity          114
      Boot Space (L)            113
      Safety Features           114
      Drive Train               114
      dtype: int64
```

```
[51]: def fixing_seating_capacity(value):
          if isinstance(value, str) and "- Jan" or "- Jul" or "-Aug" in value:
              return value.split("/")[-1]
          return value

      df["Seating Capacity"] = df["Seating Capacity"].apply(fixing_seating_capacity)
      df["Seating Capacity"] = pd.to_numeric(df["Seating Capacity"], errors='coerce')

[52]: print(df["Seating Capacity"])

      0      7.0
      1      4.0
      2      7.0
      3      7.0
      4      7.0
            ...
      109    7.0
      110    7.0
      111    5.0
      112    5.0
      113    5.0
      Name: Seating Capacity, Length: 114, dtype: float64

[53]: df = df.drop(columns = ["Year"])

[54]: from itertools import product

      def expand_combinations(row, fields):

          options = [row[field].split('/') if isinstance(row[field], str) else [row[field]] for field in fields]
          combinations = list(product(*options))
          expanded_rows = []
          for combo in combinations:
              new_row = row.copy()
              for i, field in enumerate(fields):
                  new_row[field] = combo[i]
              expanded_rows.append(new_row)
          return expanded_rows


      fields_to_expand = ["Transmission", "Fuel Type", "Drive Train"]

      expanded_data = []
      for _, row in df.iterrows():
          expanded_data.extend(expand_combinations(row, fields_to_expand))

      df = pd.DataFrame(expanded_data)
```

```
[56]: filtered_data = df[df['Model'] == 'Scorpio N']
      print(filtered_data)

           Brand      Model Type  Engine Capacity (cc)  Power (PS)  Torque (Nm)  \
      2  Mahindra  Scorpio N  SUV               2198.0         175          400
      2  Mahindra  Scorpio N  SUV               2198.0         175          400
      2  Mahindra  Scorpio N  SUV               2198.0         175          400
      2  Mahindra  Scorpio N  SUV               2198.0         175          400

        Transmission Fuel Type  Mileage (kmpl)  Top Speed (kmph)  ...  \
      2   6-Speed MT    Diesel            15.4               180  ...
      2   6-Speed MT    Diesel            15.4               180  ...
      2           AT    Diesel            15.4               180  ...
      2           AT    Diesel            15.4               180  ...

         Fuel Tank Capacity (L) Length (mm)  Width (mm)  Height (mm)  \
      2                   57.0        4662        1917         1857
      2                   57.0        4662        1917         1857
      2                   57.0        4662        1917         1857
      2                   57.0        4662        1917         1857

         Wheelbase (mm)  Braking System  Seating Capacity  Boot Space (L)  \
      2           2750        All Disc               7.0           460.0
      2           2750        All Disc               7.0           460.0
      2           2750        All Disc               7.0           460.0
      2           2750        All Disc               7.0           460.0

           Safety Features  Drive Train
      2  6 Airbags ABS EBD          RWD
      2  6 Airbags ABS EBD          4WD
      2  6 Airbags ABS EBD          RWD
      2  6 Airbags ABS EBD          4WD

      [4 rows x 23 columns]

[57]: def update_transmission(value):

          if isinstance(value, str) and 'MT' in value:
              return 'Manual'

          elif isinstance(value, str) and any(term in value for term in ['iMT', 'CVT', 'AMT', 'AT', 'DSG', 'DCT']):
              return 'Automatic'
          return value


      df['Transmission'] = df['Transmission'].apply(update_transmission)
```

```
[19]: df.describe()
```

[19]:

	Unnamed: 0	Valves_Per_Cylinder	Seating_Capacity	Wheels_Size	Number_of_Airbags
count	1267.000000	1165.000000	1261.000000	0.0	1136.000000
mean	633.000000	3.977682	5.256146	NaN	3.777289
std	365.895705	0.836978	1.139571	NaN	2.523660
min	0.000000	1.000000	2.000000	NaN	1.000000
25%	316.500000	4.000000	5.000000	NaN	2.000000
50%	633.000000	4.000000	5.000000	NaN	2.000000
75%	949.500000	4.000000	5.000000	NaN	6.000000
max	1266.000000	16.000000	16.000000	NaN	14.000000

```
[21]: df.isnull()
```

[21]:

	Unnamed: 0	Make	Model	Variant	Ex-Showroom_Price	Displacement	Valves_Per_Cylinder	Drivetrain	Emission_Norm	Fuel_Tank_Capacity	...	Tyre_Pressure_Monitor
0	False	False	False	False	False	False	False	False	False	False	...	
1	False	False	False	False	False	False	False	False	False	False	...	
2	False	False	False	False	False	False	False	False	False	False	...	
3	False	False	False	False	False	False	False	False	False	False	...	
4	False	False	False	False	False	False	False	False	False	False	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1262	False	False	False	False	False	False	False	False	False	True	...	
1263	False	False	False	False	False	False	False	False	False	True	...	
1264	False	False	False	False	False	False	True	False	False	False	...	
1265	False	False	False	False	False	False	False	False	True	False	...	
1266	False	False	False	False	False	False	True	False	False	False	...	

1267 rows × 75 columns

```
[14]: print(df.columns)

Index(['Unnamed: 0', 'Make', 'Model', 'Variant', 'Ex-Showroom_Price',
       'Displacement', 'Valves_Per_Cylinder', 'Drivetrain', 'Emission_Norm',
       'Fuel_Tank_Capacity', 'Fuel_Type', 'Height', 'Length', 'Width',
       'Body_Type', 'City_Mileage', 'Highway_Mileage',
       'ARAI_Certified_Mileage', 'ARAI_Certified_Mileage_for_CNG',
       'Kerb_Weight', 'Ground_Clearance', 'Front_Brakes', 'Rear_Brakes',
       'Rear_Suspension', 'Power_Steering', 'Power', 'Torque',
       'Seating_Capacity', 'Type', 'Wheels_Size', 'Basic_Warranty',
       'Bluetooth', 'Boot_Space', 'Central_Locking', 'Child_Safety_Locks',
       'Cup_Holders', 'Distance_to_Empty', 'Extended_Warranty',
       'Fuel-lid_Opener', 'Handbrake', 'Instrument_Console',
       'Minimum_Turning_Radius', 'Multifunction_Display',
       'Auto-Dimming_Rear-View_Mirror', 'Hill_Assist',
       'High_Speed_Alert_System', 'ABS_(Anti-lock_Braking_System)',
       'Headlight_Reminder', 'Gross_Vehicle_Weight', 'Airbags',
       'Door_Ajar_Warning', 'EBD_(Electronic_Brake-force_Distribution)',
       'Fasten_Seat_Belt_Warning', 'Gear_Shift_Reminder', 'Number_of_Airbags',
       'Other_Specs', 'Parking_Assistance', 'Key_Off_Reminder',
       'USB_Compatibility', 'Android_Auto', 'Apple_CarPlay',
       'Infotainment_Screen', 'Multifunction_Steering_Wheel',
       'EBA_(Electronic_Brake_Assist)', 'Navigation_System',
       'Tyre_Pressure_Monitoring_System', 'ESP_(Electronic_Stability_Program)',
       'ISOFIX_(Child-Seat_Mount)', 'Rain_Sensing_Wipers',
       'Automatic_Headlamps', 'Engine_Type', 'ASR_/_Traction_Control',
       'Cruise_Control', 'Battery', 'Electric_Range'],
      dtype='object')

[17]: df.columns

[17]: Index(['Unnamed: 0', 'Make', 'Model', 'Variant', 'Ex-Showroom_Price',
       'Displacement', 'Valves_Per_Cylinder', 'Drivetrain', 'Emission_Norm',
       'Fuel_Tank_Capacity', 'Fuel_Type', 'Height', 'Length', 'Width',
       'Body_Type', 'City_Mileage', 'Highway_Mileage',
       'ARAI_Certified_Mileage', 'ARAI_Certified_Mileage_for_CNG',
       'Kerb_Weight', 'Ground_Clearance', 'Front_Brakes', 'Rear_Brakes',
       'Rear_Suspension', 'Power_Steering', 'Power', 'Torque',
       'Seating_Capacity', 'Type', 'Wheels_Size', 'Basic_Warranty',
       'Bluetooth', 'Boot_Space', 'Central_Locking', 'Child_Safety_Locks',
       'Cup_Holders', 'Distance_to_Empty', 'Extended_Warranty',
       'Fuel-lid_Opener', 'Handbrake', 'Instrument_Console',
       'Minimum_Turning_Radius', 'Multifunction_Display',
       'Auto-Dimming_Rear-View_Mirror', 'Hill_Assist',
       'High_Speed_Alert_System', 'ABS_(Anti-lock_Braking_System)',
       'Headlight_Reminder', 'Gross_Vehicle_Weight', 'Airbags',
       'Door_Ajar_Warning', 'EBD_(Electronic_Brake-force_Distribution)',
       'Fasten_Seat_Belt_Warning', 'Gear_Shift_Reminder', 'Number_of_Airbags',
       'Other_Specs', 'Parking_Assistance', 'Key_Off_Reminder',
       'USB_Compatibility', 'Android_Auto', 'Apple_CarPlay',
       'Infotainment_Screen', 'Multifunction_Steering_Wheel',
       'EBA_(Electronic_Brake_Assist)', 'Navigation_System',
```

```
[5]: df.describe()
```

[5]:

	Unnamed: 0	Cylinders	Valves_Per_Cylinder	Doors	Front_Tyre_&_Rim	Rear_Tyre_&_Rim	Seating_Capacity	Wheels_Size	Number_of_Airbags	USB_Ports
count	1267.000000	1201.000000	1165.000000	1263.00000	0.0	0.0	1261.000000	0.0	1136.000000	25.000000
mean	633.000000	4.382182	3.977682	4.54711	NaN	NaN	5.256146	NaN	3.777289	1.920000
std	365.895705	1.667728	0.836978	0.74951	NaN	NaN	1.139571	NaN	2.523660	0.759386
min	0.000000	2.000000	1.000000	2.00000	NaN	NaN	2.000000	NaN	1.000000	1.000000
25%	316.500000	4.000000	4.000000	4.00000	NaN	NaN	5.000000	NaN	2.000000	1.000000
50%	633.000000	4.000000	4.000000	5.00000	NaN	NaN	5.000000	NaN	2.000000	2.000000
75%	949.500000	4.000000	4.000000	5.00000	NaN	NaN	5.000000	NaN	6.000000	2.000000
max	1266.000000	16.000000	16.000000	5.00000	NaN	NaN	16.000000	NaN	14.000000	3.000000

```
[8]: df.isnull().sum()

[8]: Unnamed: 0              0
     Make                   75
     Model                   0
     Variant                 0
     Ex-Showroom_Price       0
                          ...
     USB_Ports            1242
     Heads-Up_Display     1216
     Welcome_Lights       1198
     Battery              1254
     Electric_Range       1250
     Length: 141, dtype: int64
```

```
[13]: columns_to_drop = ["Cylinders", "Valves_per_Cylinder", "Cylinder_Configuration", "Engine_Location", "Fuel_System", "Doors", "Gears", "Front_Suspension",
      df = df.drop(columns=columns_to_drop, errors="ignore")
```

```
KeyError: ['Model'] not found in axis
```

```
[53]: df2.columns

[53]: Index(['SLNO', 'SLNO.1', 'Variant', 'Ex-Showroom_Price', 'Displacement',
             'Valves_Per_Cylinder', 'Drivetrain', 'Emission_Norm',
             'Fuel_Tank_Capacity', 'Fuel_Type', 'Height', 'Length', 'Width',
             'Body_Type', 'City_Mileage', 'ARAI_Certified_Mileage', 'Kerb_Weight',
             'Ground_Clearance', 'Front_Brakes', 'Rear_Brakes', 'Rear_Suspension',
             'Power_Steering', 'Power', 'Torque', 'Seating_Capacity', 'Type',
             'Basic_Warranty', 'Bluetooth', 'Boot_Space', 'Central_Locking',
             'Child_Safety_Locks', 'Cup_Holders', 'Distance_to_Empty',
             'Fuel-lid_Opener', 'Handbrake', 'Instrument_Console',
             'Minimum_Turning_Radius', 'Multifunction_Display',
             'Auto-Dimming_Rear-View_Mirror', 'ABS_(Anti-lock_Braking_System)',
             'Headlight_Reminder', 'Gross_Vehicle_Weight', 'Airbags',
             'Door_Ajar_Warning', 'EBD_(Electronic_Brake-force_Distribution)',
             'Fasten_Seat_Belt_Warning', 'Gear_Shift_Reminder', 'Number_of_Airbags',
             'Parking_Assistance', 'Key_Off_Reminder', 'USB_Compatibility',
             'Infotainment_Screen', 'Multifunction_Steering_Wheel',
             'Navigation_System'],
            dtype='object')

[ ]:
```

```
[3]: df.head()
```

[3]:

	SLNO	Make	Model	Variant	Ex-Showroom_Price	Displacement	Valves_Per_Cylinder	Drivetrain	Emission_Norm	Fuel_Tank_Capacity	...	Door_Ajar_Warning	EBD_(Elec force
0	1	Tata	Nano Genx	Xt	Rs. 2,92,667	624 cc	2	RWD (Rear Wheel Drive)	BS IV	24 litres	...	Yes	
1	2	Tata	Nano Genx	Xe	Rs. 2,36,447	624 cc	2	RWD (Rear Wheel Drive)	BS IV	24 litres	...	Yes	
2	3	Tata	Nano Genx	Emax Xm	Rs. 2,96,661	624 cc	2	RWD (Rear Wheel Drive)	BS IV	15 litres	...	Yes	
3	4	Tata	Nano Genx	Xta	Rs. 3,34,768	624 cc	2	RWD (Rear Wheel Drive)	BS IV	24 litres	...	Yes	
4	5	Tata	Nano Genx	Xm	Rs. 2,72,223	624 cc	2	RWD (Rear Wheel Drive)	BS IV	24 litres	...	Yes	

5 rows × 47 columns

```
[4]: df.columns

[4]: Index(['SLNO', 'Make', 'Model', 'Variant', 'Ex-Showroom_Price', 'Displacement',
           'Valves_Per_Cylinder', 'Drivetrain', 'Emission_Norm',
           'Fuel_Tank_Capacity', 'Fuel_Type', 'Height', 'Length', 'Width',
           'Body_Type', 'City_Mileage', 'ARAI_Certified_Mileage', 'Kerb_Weight',
           'Ground_Clearance', 'Front_Brakes', 'Rear_Brakes', 'Power', 'Torque',
           'Seating_Capacity', 'Transmission', 'Basic_Warranty', 'Bluetooth',
           'Boot_Space', 'Central_Locking', 'Child_Safety_Locks', 'Handbrake',
           'Instrument_Console', 'Minimum_Turning_Radius', 'Multifunction_Display',
           'ABS_(Anti-lock_Braking_System)', 'Gross_Vehicle_Weight', 'Airbags',
           'Door_Ajar_Warning', 'EBD_(Electronic_Brake-force_Distribution)',
           'Fasten_Seat_Belt_Warning', 'Gear_Shift_Reminder', 'Number_of_Airbags',
           'Parking_Assistance', 'Infotainment_Screen', 'Navigation_System',
           'Battery', 'Electric_Range'],
          dtype='object')

[5]: conditions = df['Fuel_Type'].isin(['Electric', 'Hybrid'])
     df.loc[~conditions, ['Battery', 'Electric_Range']] = np.nan

[6]: print(df['Electric_Range'].tolist())

[nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, n
an, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, '110 km/full charge', '110 km/full charge', '213 km/full charge', '213 km/full charg
e', '213 km/full charge', nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, n
an, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, '110 km/full charge', '110 km/full charge', '110 km/full charge', nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, '340 km/full charge', '340 km/full charge', nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, '110 km/full charge', nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, '110 km/full charge', na
n, '110 km/full charge', nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, '110 km/full charge', nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, '110 k
m/full charge', '110 km/full charge', nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, '110 km/full charge', nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, '110
km/full charge', nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
```

```
[8]: df['Ex-Showroom_Price'] = df['Ex-Showroom_Price'].str.replace("Rs.", "", regex=False)
     df['Ex-Showroom_Price'] = df['Ex-Showroom_Price'].str.replace(",", "")
     df['Ex-Showroom_Price'] = pd.to_numeric(df['Ex-Showroom_Price'])

[9]: print(df['Ex-Showroom_Price'].head())
     print(df['Ex-Showroom_Price'].dtype)

     0    292667
     1    236447
     2    296661
     3    334768
     4    272223
     Name: Ex-Showroom_Price, dtype: int64
     int64

[10]: print(df['Ground_Clearance'].dtype)

      object

[11]: columns_to_clean = ['Height', 'Length', 'Width', 'Ground_Clearance']

      for col in columns_to_clean:
          df[col] = df[col].str.replace("mm", "", regex=False)
          df[col] = pd.to_numeric(df[col])

[12]: print(df["Height"].head())

      0    1652.0
      1    1652.0
      2    1652.0
      3    1652.0
      4    1652.0
      Name: Height, dtype: float64

[13]: print(df['Gross_Vehicle_Weight'].dtype)

      object

[16]: columns_to_clean = ['Kerb_Weight', 'Gross_Vehicle_Weight']

      for col in columns_to_clean:
          df[col] = df[col].str.replace("kg", "", regex=False).str.strip()
          df[col] = pd.to_numeric(df[col], errors='coerce').fillna(0).astype(int)

[17]: print(df[columns_to_clean].head())
      print(df[columns_to_clean].dtypes)

         Kerb_Weight  Gross_Vehicle_Weight
```

```
3   38.0    51.0
4   38.0    51.0
Power     float64
Torque    float64
dtype: object
```

```
[17]: print(df['Battery'].dtypes)

object

[18]: df['Battery'] = df['Battery'].replace("200 ampere-hour", "2.4 kWh")

[19]: df['Battery'] = df['Battery'].replace("210 ampere-hour", "2.5 kWh")

[27]: columns_to_clean = ['Battery', 'Electric_Range']

      for col in columns_to_clean:
          df[col] = df[col].astype(str)
          df[col] = df[col].str.extract(r'(\d+(?:\.\d+)?)')[0]
          df[col] = pd.to_numeric(df[col], errors='coerce')

[28]: print(df[['Battery', 'Electric_Range']].head(324))
      print(df[['Battery', 'Electric_Range']].dtypes)

     Battery  Electric_Range
0        NaN             NaN
1        NaN             NaN
2        NaN             NaN
3        NaN             NaN
4        NaN             NaN
..       ...             ...
319      2.5           110.0
320     21.5           213.0
321     21.5           213.0
322     21.5           213.0
323      NaN             NaN

[324 rows x 2 columns]
Battery           float64
Electric_Range    float64
dtype: object

[29]: df.to_csv('newFourWheeler4.csv', index=False)
      print("Dataset saved as 'newFourWheeler4.csv'")

Dataset saved as 'newFourWheeler4.csv'

[ ]:
```

```
[51]: df.info()
      df.isnull().sum()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1267 entries, 0 to 1266
Data columns (total 47 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   SLNO                                          1267 non-null   int64
 1   Brand                                         1267 non-null   object
 2   Model                                         1267 non-null   object
 3   Variant                                       1267 non-null   object
 4   Ex-Showroom_Price(Rs)                         1267 non-null   int64
 5   Displacement                                  1267 non-null   object
 6   Valves_Per_Cylinder                           1267 non-null   int64
 7   Drivetrain                                    1267 non-null   object
 8   Emission_Norm                                 1267 non-null   object
 9   Fuel_Tank_Capacity(litre)                     1267 non-null   float64
 10  Fuel_Type                                     1267 non-null   object
 11  Height(mm)                                    1267 non-null   int64
 12  Length(mm)                                    1267 non-null   float64
 13  Width(mm)                                     1267 non-null   float64
 14  Body_Type                                     1267 non-null   object
 15  City_Mileage(km/l)                            1233 non-null   float64
 16  ARAI_Certified_Mileage(km/l)                  1264 non-null   float64
 17  Kerb_Weight(kg)                               1267 non-null   int64
 18  Ground_Clearance(mm)                          1267 non-null   float64
 19  Front_Brakes                                  1267 non-null   object
 20  Rear_Brakes                                   1267 non-null   object
 21  Power(PS)                                     1267 non-null   float64
 22  Torque(NM)                                    1267 non-null   float64
 23  Seating_Capacity                              1267 non-null   float64
 24  Transmission                                  1267 non-null   object
 25  Basic_Warranty                                1267 non-null   object
 26  Bluetooth                                     1267 non-null   object
 27  Boot_Space(litre)                             1258 non-null   float64
 28  Central_Locking                               1267 non-null   object
 29  Child_Safety_Locks                            1267 non-null   object
 30  Handbrake                                     1267 non-null   object
 31  Instrument_Console                            1267 non-null   object
 32  Minimum_Turning_Radius                        1267 non-null   object
 33  Multifunction_Display                         1267 non-null   object
 34  ABS_(Anti-lock_Braking_System)                1267 non-null   object
 35  Gross_Vehicle_Weight                          1267 non-null   int64
 36  Airbags                                       1267 non-null   object
 37  Door_Ajar_Warning                             1267 non-null   object
 38  EBD_(Electronic_Brake-force_Distribution)     1267 non-null   object
 39  Fasten_Seat_Belt_Warning                      1267 non-null   object
 40  Gear_Shift_Reminder                           1267 non-null   object
 41  Number_of_Airbags                             1267 non-null   int64
 42  Parking_Assistance                            1267 non-null   object
 43  Infotainment_Screen                           1267 non-null   object
 44  Navigation_System                             1267 non-null   object
 45  Battery(kWh)                                  29 non-null     float64
 46  Electric_Range(km)                            29 non-null     float64
dtypes: float64(12), int64(7), object(28)
memory usage: 465.4+ KB

[51]: SLNO                                           0
      Brand                                          0
      Model                                          0
      Variant                                        0
      Ex-Showroom_Price(Rs)                          0
      Displacement                                   0
      Valves_Per_Cylinder                            0
      Drivetrain                                     0
      Emission_Norm                                  0
      Fuel_Tank_Capacity(litre)                      0
      Fuel_Type                                      0
      Height(mm)                                     0
      Length(mm)                                     0
      Width(mm)                                      0
      Body_Type                                      0
      City_Mileage(km/l)                            34
      ARAI_Certified_Mileage(km/l)                   3
      Kerb_Weight(kg)                                0
      Ground_Clearance(mm)                           0
      Front_Brakes                                   0
```

```
]: df['Battery(kWh)'].fillna(0, inplace=True)
   df['Electric_Range(km)'].fillna(0, inplace=True)
   print(df.isnull().sum())

   SLNO                                            0
   Brand                                           0
   Model                                           0
   Variant                                         0
   Ex-Showroom_Price(Rs)                           0
   Displacement                                    0
   Valves_Per_Cylinder                             0
   Drivetrain                                      0
   Emission_Norm                                   0
   Fuel_Tank_Capacity(litre)                       0
   Fuel_Type                                       0
   Height(mm)                                      0
   Length(mm)                                      0
   Width(mm)                                       0
   Body_Type                                       0
   City_Mileage(km/l)                              0
   ARAI_Certified_Mileage(km/l)                    0
   Kerb_Weight(kg)                                 0
   Ground_Clearance(mm)                            0
   Front_Brakes                                    0
   Rear_Brakes                                     0
   Power(PS)                                       0
   Torque(NM)                                      0
   Seating_Capacity                                0
   Transmission                                    0
   Basic_Warranty                                  0
   Bluetooth                                       0
   Boot_Space(litre)                               0
   Central_Locking                                 0
   Child_Safety_Locks                              0
   Handbrake                                       0
   Instrument_Console                              0
   Minimum_Turning_Radius                          0
   Multifunction_Display                           0
   ABS_(Anti-lock_Braking_System)                  0
   Gross_Vehicle_Weight                            0
   Airbags                                         0
   Door_Ajar_Warning                               0
   EBD_(Electronic_Brake-force_Distribution)       0
   Fasten_Seat_Belt_Warning                        0
   Gear_Shift_Reminder                             0
   Number_of_Airbags                               0
   Parking_Assistance                              0
   Infotainment_Screen                             0
   Navigation_System                               0
   Battery(kWh)                                    0
   Electric_Range(km)                              0
   dtype: int64
```

```
      ['Petrol' 'CNG' 'Diesel' 'CNG + Petrol' 'Electric' 'Hybrid']

[32]: unique_values = df['Seating_Capacity'].unique()
      print(unique_values)

      [ 4  5  7  9  8  6  2 16]

[33]: unique_values = df['Brand'].unique()
      print(unique_values)

      ['Tata' 'Datsun' 'Renault' 'Maruti Suzuki' 'Hyundai' 'Premier' 'Toyota'
       'Nissan' 'Volkswagen' 'Ford' 'Mahindra' 'Fiat' 'Honda' 'Force' 'Skoda'
       'Jeep' 'Mg' 'Kia' 'Mitsubishi' 'Volvo' 'Mini' 'Bmw' 'Audi'
       'Land Rover Rover' 'Lexus' 'Jaguar' 'Porsche' 'Land Rover' 'Maserati'
       'Lamborghini' 'Bentley' 'Ferrari' 'Aston Martin' 'Bugatti' 'Bajaj' 'Icml'
       'Isuzu' 'Maruti Suzuki R' 'Dc']

[34]: df['Brand'] = df['Brand'].replace('Maruti Suzuki R', 'Maruti Suzuki')
      print(df['Brand'].unique())

      ['Tata' 'Datsun' 'Renault' 'Maruti Suzuki' 'Hyundai' 'Premier' 'Toyota'
       'Nissan' 'Volkswagen' 'Ford' 'Mahindra' 'Fiat' 'Honda' 'Force' 'Skoda'
       'Jeep' 'Mg' 'Kia' 'Mitsubishi' 'Volvo' 'Mini' 'Bmw' 'Audi'
       'Land Rover Rover' 'Lexus' 'Jaguar' 'Porsche' 'Land Rover' 'Maserati'
       'Lamborghini' 'Bentley' 'Ferrari' 'Aston Martin' 'Bugatti' 'Bajaj' 'Icml'
       'Isuzu' 'Dc']

[35]: df['Brand'] = df['Brand'].replace('Maruti Suzuki', 'Suzuki')
      print(df['Brand'].unique())

      ['Tata' 'Datsun' 'Renault' 'Suzuki' 'Hyundai' 'Premier' 'Toyota' 'Nissan'
       'Volkswagen' 'Ford' 'Mahindra' 'Fiat' 'Honda' 'Force' 'Skoda' 'Jeep' 'Mg'
       'Kia' 'Mitsubishi' 'Volvo' 'Mini' 'Bmw' 'Audi' 'Land Rover Rover' 'Lexus'
       'Jaguar' 'Porsche' 'Land Rover' 'Maserati' 'Lamborghini' 'Bentley'
       'Ferrari' 'Aston Martin' 'Bugatti' 'Bajaj' 'Icml' 'Isuzu' 'Dc']

[36]: print(df['Body_Type'].unique())
      #print(unique_values)

      ['Hatchback' 'MPV' 'MUV' 'SUV' 'Sedan' 'Crossover' 'Crossover, SUV'
       'SUV, Crossover' 'Coupe' 'Sedan, Crossover' 'Convertible' 'Sedan, Coupe'
       'Sports' 'Sports, Hatchback' 'Sports, Convertible' 'Pick-up'
       'Coupe, Convertible']

[37]: df['Body_Type'] = df['Body_Type'].replace('MPV', 'MUV')
      print(df['Body_Type'].unique())

      ['Hatchback' 'MUV' 'SUV' 'Sedan' 'Crossover' 'Crossover, SUV'
       'SUV, Crossover' 'Coupe' 'Sedan, Crossover' 'Convertible' 'Sedan, Coupe'
       'Sports' 'Sports, Hatchback' 'Sports, Convertible' 'Pick-up'
       'Coupe, Convertible']
```

twentytwo.txt recommendations.txt

File Edit View

Type	Model	Confidence	Brand	Price	Fuel	Body	Transmission
Recommended	Nexon	4.0%	Tata	5-10L	Petrol	SUV	Manual
Recommended	Altroz	1.0%	Tata	5-10L	Petrol	Hatchback	Manual
Recommended	Nexon Ev	1.0%	Tata	10-15L	Electric	SUV	Automatic
Recommended	Winger	1.0%	Tata	10-15L	Diesel	MUV	Manual
Recommended	Nano Genx	0.0%	Tata	<5L	Petrol	Hatchback	Manual
Similar	Xuv300	100.0%	Mahindra	5-10L	Petrol	SUV	Manual
Similar	Ecosport	100.0%	Ford	5-10L	Petrol	SUV	Manual
Similar	Ecosport	100.0%	Ford	5-10L	Petrol	SUV	Manual

```
Top Tata Recommendations:
Nexon: 4.0% confidence
  Brand: Tata
  Price: 5-10L
  Fuel: Petrol
  Body: SUV
  Transmission: Manual

Altroz: 1.0% confidence
  Brand: Tata
  Price: 5-10L
  Fuel: Petrol
  Body: Hatchback
  Transmission: Manual

Nexon Ev: 1.0% confidence
  Brand: Tata
  Price: 10-15L
  Fuel: Electric
  Body: SUV
  Transmission: Automatic

Winger: 1.0% confidence
  Brand: Tata
  Price: 10-15L
  Fuel: Diesel
  Body: MUV
  Transmission: Manual

Nano Genx: 0.0% confidence
  Brand: Tata
  Price: <5L
  Fuel: Petrol
  Body: Hatchback
  Transmission: Manual

Similar: Xuv300: 100.0% confidence
  Brand: Mahindra
  Price: 5-10L
  Fuel: Petrol
  Body: SUV
  Transmission: Manual

Similar: Ecosport: 100.0% confidence
  Brand: Ford
  Price: 5-10L
  Fuel: Petrol
  Body: SUV
  Transmission: Manual
```

```python
    def get_recommendations(self, user_input: Dict, top_n: int = 5, include_similar: bool = True) -> List[Tuple[str, float, Dict]]:
        features = self._prepare_features(user_input)
        probabilities = self.rf_model.predict_proba(features)[0]
        #preparing the features by processing the user inouts and formats it to match the feature set expected by the model
        #pbs because - RF outputs are pbs of each possible outcome. So using pbs would allow us to understand how likely the vehicle recommended is th
        #right choice.
        brand_models = self.brand_models.get(user_input['Brand'], set())
        recommendations = []

        #filtering and then collecting then iterating on probalities and then collecting the recommendations
        for idx, prob in enumerate(probabilities):
            model_name = self.df['Model'].unique()[idx] if idx < len(self.df['Model'].unique()) else f"Unknown Model {idx}"
            if model_name in brand_models:
                model_data = self.df[self.df['Model'] == model_name].iloc[0].to_dict()
                recommendations.append((model_name, prob, model_data))

        brand_recommendations = sorted(recommendations, key=lambda x: x[1], reverse=True)[:top_n]
        #sorting based on probability


        if include_similar and brand_recommendations:
            similar_vehicles = self.get_similar_vehicles(brand_recommendations[0][0])
            return brand_recommendations + similar_vehicles

        return brand_recommendations




def format_recommendation(rec_tuple: Tuple[str, float, Dict], is_similar: bool = False) -> str:
    model, prob, data = rec_tuple
    prefix = "Similar: " if is_similar else ""
    return (f"{prefix}{model}: {prob:.1%} confidence\n"
            f"  Brand: {data['Brand']}\n"
            f"  Price: {data['Price_Range']}\n"
            f"  Fuel: {data['Fuel_Type']}\n"
            f"  Body: {data['Body_Type']}\n"
            f"  Transmission: {data['Transmission']}")

if __name__ == "__main__":
    recommender = VehicleRecommender(
        'best_rf_model.joblib',
        'encoders.pkl',
        'scaler.pkl',
        'cleaned_dataset_NB18.csv'
    )

    test_input = {
        'Brand': 'Tata',
        'Fuel_Type': 'Petrol',
        'Transmission': 'Manual',
        'Seating_Capacity': 5,
        'Body_Type': 'SUV',
        'Price_Range': '10-15L'
    }

    try:
        recommendations = recommender.get_recommendations(test_input, include_similar=True)
        print(f"\nTop {test_input['Brand']} Recommendations:")
        for i, rec in enumerate(recommendations):
            is_similar = i >= 5
            print(format_recommendation(rec, is_similar))
            print()
    except Exception as e:
        print(f"Error: {str(e)}")
```

```python
    @lru_cache(maxsize=32)

    #creating a mapping of the car brands to their respective models
    def _create_brand_model_mapping(self) -> Dict[str, set]:
        return {brand: set(group['Model'])
                for brand, group in self.df.groupby('Brand')}

    #retrives default median values for numeric features of a given brand
    #used for filling any missing values in user inputs
    def _get_brand_defaults(self, brand: str) -> Dict[str, float]:
        brand_data = self.df[self.df['Brand'] == brand]
        if brand_data.empty:
            brand_data = self.df
        numeric_cols = self.df.select_dtypes(include=['int64', 'float64']).columns
        return {col: brand_data[col].median() for col in numeric_cols}



    #creating an empty df called features
    #getting deafult values of the brand
    #filling the default values
    #transforming the categorical features
    #filling missing values
    #for? - best input handling/ handling any unprovided or missing values
    def _prepare_features(self, user_input: Dict) -> pd.DataFrame:
        features = pd.DataFrame(columns=self.rf_model.feature_names_in_, index=[0])
        defaults = self._get_brand_defaults(user_input['Brand'])

        for col, value in defaults.items():
            if col in features.columns:
                features[col] = value

        for col in features.columns:
            if col in self.encoders:
                if col in user_input:
                    try:
                        features[col] = self.encoders[col].transform([user_input[col]])
                    except ValueError:
                        features[col] = self.encoders[col].transform([self.encoders[col].classes_[0]])
                else:
                    features[col] = self.encoders[col].transform([self.encoders[col].classes_[0]])

        return features.fillna(0)
    #Why do you think the model required me to enter all the car values to return the "model,"
    #while the entire project was designed to only require 5 categorical inputs to return the "model"?
    #training vs predictions mismatching
    #no proper data preprocessing


    def get_similar_vehicles(self, model_name: str, n_similar: int = 5) -> List[Tuple[str, float, Dict]]:
        model_idx = self.df[self.df['Model'] == model_name].index[0]
        model_features = self.feature_matrix[model_idx].reshape(1, -1)
        #a matrix extrated from model_features that contains the numerical feature representations

        distances, indices = self.similarity_model.kneighbors(model_features)
        #gets the closest distance and indecies values between the "Recommended model" and other models
        similar_vehicles = []

        for idx, dist in zip(indices[0][1:], distances[0][1:]):
            similar_model = self.df.iloc[idx]
            if similar_model['Model'] != model_name: # <------
                similarity_score = 1 / (1 + dist)
                similar_vehicles.append((
                    similar_model['Model'],
                    similarity_score,
                    similar_model.to_dict()
                ))

        return similar_vehicles[:n_similar]
```

```python
import joblib
import pickle
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from typing import Dict, List, Tuple
from functools import lru_cache
from sklearn.neighbors import NearestNeighbors

class VehicleRecommender:
    #loading pre trianed best rf model
    def __init__(self, model_path: str, encoders_path: str, scaler_path: str, data_path: str):
        self.rf_model = joblib.load(model_path)
        with open(encoders_path, 'rb') as f:
            self.encoders = pickle.load(f)
        with open(scaler_path, 'rb') as f:
            self.scaler = pickle.load(f)
        self.df = pd.read_csv(data_path)

        #preped a dictionary linking brands to their respective models
        self.brand_models = self._create_brand_model_mapping()
        #similarity model - builds a knn model for finding similar car models (within the same range) but w/ differnt brands
        self._setup_similarity_model()




    def _setup_similarity_model(self):
        # Prepare feature matrix for similarity comparison
        feature_df = pd.DataFrame()

        # Add numeric features
        if 'Seating_Capacity' in self.df.columns:
            feature_df['Seating_Capacity'] = self.df['Seating_Capacity']
        # Add encoded categorical features
        cat_features = ['Body_Type', 'Fuel_Type', 'Transmission', 'Price_Range']
        for feat in cat_features:
            if feat in self.df.columns:
                feature_df[feat] = self.encoders[feat].transform(self.df[feat])
        # Normalize features
        self.feature_matrix = StandardScaler().fit_transform(feature_df)
        self.similarity_model = NearestNeighbors(n_neighbors=min(10, len(self.df)), metric='euclidean')
        self.similarity_model.fit(self.feature_matrix)
```

.**References**

[1] Kavinkumar.V, Rachamalla Rahul Reddy, Rohit Balasubramanian, Sridhar.M, Sridharan.K Dr. D. Venkataraman, “A Hybrid Approach for Recommendation System with Added Feedback Component”, IEEE, 2015**.**

[2] Doychin Doychev, Aonghus Lawlor, Rachael Rafter, and Barry Smyth, “An Analysis of Recommender Algorithms for Online News”, Conference and Labs of the Evaluation Forum, 2014.

[3] P. Boteju and L. Munasinghe, "Vehicle Recommendation System using Hybrid Recommender Algorithm and Natural Language Processing Approach," 2020 2nd International Conference on Advancements in Computing (ICAC), Malabe, Sri Lanka, 2020

[4] P. H. Putra, A. Azanuddin, B. Purba, and Y. A. Dalimunthe, “Random forest and decision tree algorithms for car price prediction”, JUMPA, vol. 4, no. 1, pp. 81–89, Sep. 2023

[5] J. Vyas, D. Das and S. K. Das, "Vehicular Edge Computing Based Driver Recommendation System Using Federated Learning," 2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), Delhi, India, 2020

[6] Gadhvi, T, & Shankar, P. "Autonomous Vehicle Guidance Using Neural Network and Random Forest Model." Proceedings of the ASME 2023 International Mechanical Engineering Congress and Exposition. Volume 6: Dynamics, Vibration, and Control. New Orleans, Louisiana, USA. October 29–November 2, 2023.