Milestone 3 Report: Car Recommendation Engine

Rohith Kumar Ballem

UFID: 30969136

1. Objective of the project and recap of the progress:

1.1 Objective and overview:

The goal of this project is to develop a car recommendation engine that assists users in selecting a vehicle based on the customer's preferences and needs. The system will analyze various car features and attributes, the sales and purchase behaviors of customers, and their ratings to provide recommendations that are personalized to people who are planning to purchase a new car. The interactive UI will allow the user to enter the preferences and the best car choices would be recommended.

This recommendation engine will be beneficial for:

- **Car buyers**: people who want to get a recommendation for a car with specific requirements, budget and on the basis of reviews and ratings.
- **Online marketplaces:** aiming to enhance user experience by suggesting the best vehicle options to their customers.

1.2 Progress and Recap:

In **Milestone 1**, three datasets are chosen from Kaggle representing **1. Car specification 2. Car Sales and 3. Car Ratings**. The data preprocessing is done where the missing data imputed (median and mode based on boxplot analysis), outliers are handled. The data is normalized, and exploratory data analysis is drawn on the data. As a continuation in **Milestone 2** the project is mainly concentrated on **Feature engineering, feature enhancement and data modeling**. In **Milestone 3** the focus is on deep evaluation and evaluation of model performance. The focus is also on deriving actionable insights and explaining predictions. Additionally, an interactive UI has been developed using **streamlit** which can take raw user input and recommend cars. The key findings are also included in the UI as visualizations.

Note: For **milestone3**, the final merged notebook of 1, 2 and 3 is named as "**IDS_Milestone3.ipynb**" that contains entire code. Additionally, for streamlit I exported the models, encoder, scaler and developed an UI for real time recommendations by loading the trained models in file named **"main.py"**

1.3 Dataset used in milestone 3:

The link for the dataset that contains data and clustering labels is "**clusteringanddata.csv**"

2. Tools used and Project timeline:

2.1 Tools and Libraries used:

- **Language**: Python
- **Notebook**: Jupyter Notebook
- **Libraries used**:
    1. **pandas**: Data manipulation and reading/writing CSV files.
    2. **numpy**: Numerical computations (linear algebra, array operations).
    3. **matplotlib** and **matplotlib.pyplot**: Data visualization (used for creating plots).
    4. **seaborn**: Statistical data visualization (works with matplotlib).
    5. **scipy.stats**:
        - **chi2_contingency**: Chi-squared test for statistical independence.
    6. **sklearn.preprocessing**:
        - **LabelEncoder**: For encoding categorical features.
        - **MinMaxScaler**: For feature scaling (normalization).
        - **StandardScaler**: For standardizing features.
        - **label_binarize**: For binarizing labels for multi-class classification.
    7. **sklearn.decomposition**:
        - **PCA**: For Principal Component Analysis (dimensionality reduction).
    8. **sklearn.model_selection**:
        - **train_test_split**: For splitting data into training and testing sets.
    9. **sklearn.metrics**:
        - **accuracy_score, precision_score, recall_score, f1_score**: Metrics for evaluating classification performance.
        - **roc_curve, auc**: ROC curve and AUC (Area Under the Curve) for evaluating classification.
        - **silhouette_score, davies_bouldin_score, adjusted_rand_score**: Metrics for clustering evaluation.
    10. **sklearn.linear_model**:
        - **LogisticRegression**: Logistic Regression classifier.
    11. **sklearn.ensemble**:
        - **RandomForestClassifier**: Random Forest classifier.
    12. **sklearn.neighbors**:
        - **KNeighborsClassifier**: K-Nearest Neighbors classifier.
    13. **sklearn.cluster**:
        - **KMeans**: K-Means clustering.
        - **AgglomerativeClustering**: Agglomerative Hierarchical Clustering.
    14. **sklearn.mixture**:
        - **GaussianMixture**: Gaussian Mixture Model clustering.
    15. **collections.Counter**: For counting elements in an iteration.
    16. **Joblib - dump / load** – For efficiently saving and loading large scikit-learn models or NumPy arrays.
    17. **Pickle - dump / load** – For serializing and deserializing Python objects, including models and data.

2.2 Project Timeline:

S.No	Task	Timeline
1	Data collection and Preprocessing	15rd February 2025
2	Exploratory Data Analysis	23rd February 2025
3	Feature Engineering and Feature Selection	4th April 2025
4	Data Modeling	7th April 2025
5	Evaluation and Testing	16th April 2025
6	Interpreting and Visualizing Results	23rd April 2025

3. Milestone2 all Model Metrics and Evaluations:

In milestone 2, the test size is taken as 0.3 and train 0.7. Additionally, in milestone 3, I have seen an issue in label encoding while applying the encoder on the raw user input. Corrected the issue and the code is rerun. The below is the corrected label encoding version.

```
# Label encoding
encoders = {}  # Dictionary to store encoders
for col in categorical_cols:
    le = LabelEncoder()
    car_full_analysis[col] = le.fit_transform(car_full_analysis[col].astype(str))
    encoders[col] = le
print("Encoded Categorical Columns:", categorical_cols)
```

Figure 1 fixed label encoder code

Here are the final model training and evaluation metrics of both supervised (Logistic regression, Random Forest Classifier and K nearest neighbor) and unsupervised(Kmeans clustering, GMM Clustering and Agglomerative clustering).

3.1 Supervised Models:

The Training and testing sets code is as below:

```
[130]: from sklearn.model_selection import train_test_split

       # Separate features and target variables
       X = car_full_analysis.drop(columns=['company_model'])  # Features
       y = car_full_analysis['company_model']  # Target (Company_model)

       # Split the data into training and testing sets

       X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

       print(f"Training set size: {X_train.shape[0]}")
       print(f"Test set size: {X_test.shape[0]}")

       Training set size: 12859
       Test set size: 5511
```

Figure 2 test train split code

a. Logistic Regression :

```
[133]: # Logistic regression
lrmodel = LogisticRegression(max_iter=1000, random_state=42)

# Train model
lrmodel.fit(X_train, y_train)

# Predict output
lrpreds = lrmodel.predict(X_test)
```

Figure 3 logistic regression code

b. Random Forest Classifier :

```
[134]: # Random Forest Classifier
rfmodel = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)

#Train model
rfmodel.fit(X_train, y_train)

# Predict output
rfpreds = rfmodel.predict(X_test)
```

Figure 4 Random forest code

c. K-nearest neighbor :

```
[135]: #KNN classifier
from sklearn.neighbors import KNeighborsClassifier
knnmodel = KNeighborsClassifier(n_neighbors=8)

# Train the model
knnmodel.fit(X_train, y_train)

# Make predictions
knnpreds = knnmodel.predict(X_test)
```

Figure 5 KNN code

The model performance metrics are :

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.9782	0.9770	0.9782	0.9764
Random Forest	0.9301	0.9009	0.9301	0.9103
K-Nearest Neighbors	0.9759	0.9774	0.9759	0.9754

The performance evaluation of three supervised machine learning models **Logistic Regression, Random Forest, and K-Nearest Neighbors (KNN)** was done using key metrics: **Accuracy, Precision, Recall, and F1-Score**. Logistic Regression achieved strong and consistent results, with an accuracy of 0.9782, indicating it is reliable and balanced predictive capability. **KNN** closely followed, demonstrating robust performance with an accuracy of 0.9759 and slightly higher precision at 0.9774, which is minimizing false positives. The **Random Forest** model showed comparatively lower performance, with an accuracy of 0.9301 and a precision of 0.9009, suggesting a higher tendency for false positives despite maintaining a good recall of 0.9301. Overall, Logistic Regression proved to be better model in **Milestone2.**

3.2 Unsupervised Models:

a. K- means Clustering:

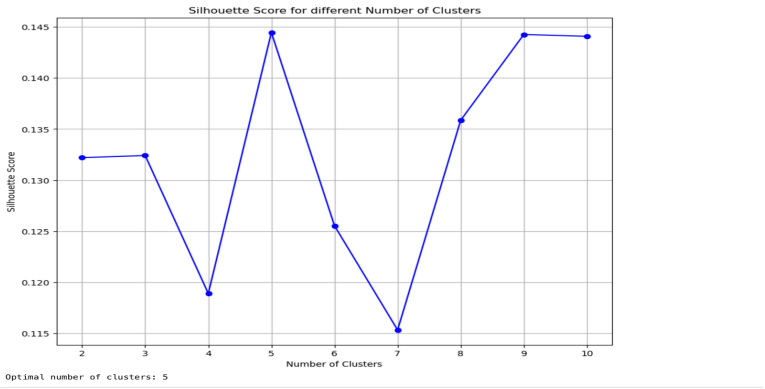


Figure 6 optimal clusters for kmeans

The optimal clusters for kmeans is analyzed using the silhouette score analysis. The optimal clusters for data is "**5**".

```
[145]: # K means

[146]: from sklearn.linear_model import LogisticRegression
       from sklearn.model_selection import train_test_split
       from sklearn.preprocessing import LabelEncoder, StandardScaler
       from sklearn.cluster import KMeans
       import pandas as pd
       import numpy as np

[147]: # Apply KMeans to all data
       kmeans = KMeans(n_clusters=5, random_state=42, n_init='auto')
       car_full_analysis['Cluster'] = kmeans.fit_predict(X)

[148]: def get_cluster_for_predicted_car(predicted_label, car_data):
           company_model = predicted_label
           predicted_car_row = car_data[car_data['company_model'] == company_model]
           predicted_car_cluster = predicted_car_row['Cluster'].values[0]
           return predicted_car_cluster, company_model

[149]: def recommend_similar_cars(predicted_label, car_data, top_n=5):
           predicted_cluster, company_model = get_cluster_for_predicted_car(predicted_label, car_data)
           similar_cars = car_data[car_data['Cluster'] == predicted_cluster]

           # similar cars based on the features
           similar_cars_scaled = similar_cars[['popularity_score', 'fuel_eff_score','Price ($)','Annual Income']]

           similar_cars = similar_cars.copy()
           similar_cars['score'] = similar_cars_scaled.sum(axis=1)

           recommended_cars = similar_cars.sort_values(by='score', ascending=False)
           return recommended_cars[['company_model','score']].drop_duplicates(subset='company_model').head(top_n), company_model

[150]: recommended_cars, company_model = recommend_similar_cars(lrpreds[4431], car_full_analysis)

[151]: print(f"Recommended cars based on predicted company_model: - {company_model} ")
       print("using Kmeans clustering")
       print(recommended_cars)
       print(f"Predicted car: {lrpreds[4431]}")
Recommended cars based on predicted company_model: - plymouth_voyager
using Kmeans clustering
              company_model     score
7420   chevrolet_monte carlo  3.535501
4075     toyota_land cruiser  3.330770
9435          dodge_durango  3.219554
17618      cadillac_escalade  3.213860
6065       pontiac_grand prix  3.208880
Predicted car: plymouth_voyager
```

Figure 7 Kmeans output

The kmeans clustering is performed and the sample prediction is made on {popularity score, fuel efficiency score, price and annual income} taking the lrpreds[4431}(output from logistic regression prediction).

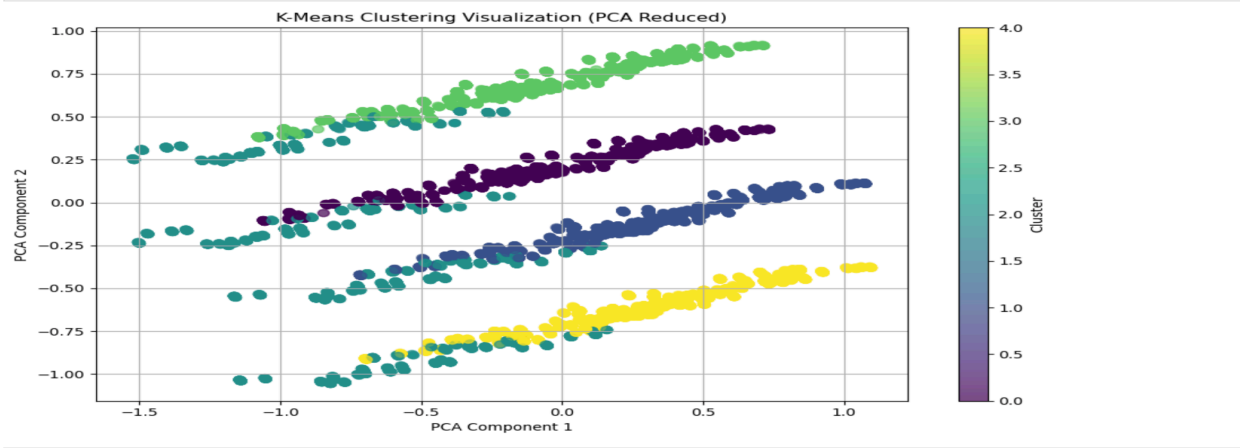


Figure 8 Kmeans cluster visualization

b. Agglomerative Clustering:

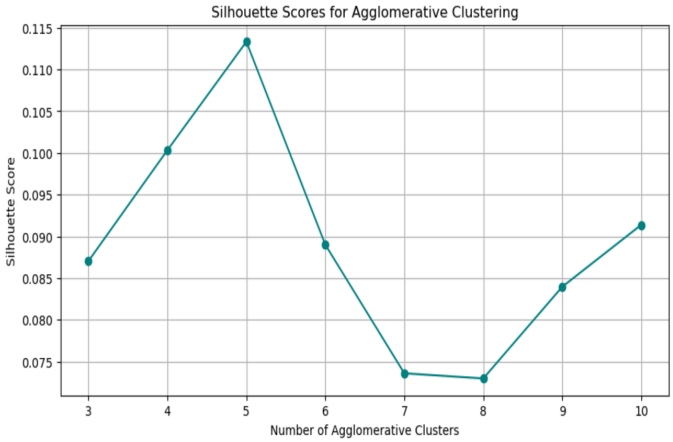


Figure 9 optimal cluster for agglomerative

The optimal clusters for Agglomerative is analyzed using the silhouette score analysis. The optimal clusters for data is "**5**".

```
[155]: from sklearn.cluster import AgglomerativeClustering

       # Apply Agglomerative Clustering with "5" from silhoutte method
       agglo = AgglomerativeClustering(n_clusters=5)
       car_full_analysis['Agglo_Cluster'] = agglo.fit_predict(X)

[156]: def get_cluster_for_predicted_car_agg(predicted_label, car_data):
           company_model = predicted_label
           predicted_car_row = car_data[car_data['company_model'] == company_model]
           predicted_car_cluster = predicted_car_row['Agglo_Cluster'].values[0]
           return predicted_car_cluster, company_model

[157]: def recommend_similar_cars_agg(predicted_label, car_data, top_n=5):
           predicted_cluster, company_model = get_cluster_for_predicted_car_agg(predicted_label, car_data)
           similar_cars = car_data[car_data['Agglo_Cluster'] == predicted_cluster]

           # similar cars based on the features
           similar_cars_scaled = similar_cars[['popularity_score', 'fuel_eff_score','Price ($)']]

           similar_cars = similar_cars.copy()
           similar_cars['score'] = similar_cars_scaled.sum(axis=1)

           recommended_cars_agg = similar_cars.sort_values(by='score', ascending=False)
           return recommended_cars_agg[['company_model','score']].drop_duplicates(subset='company_model').head(top_n), company_model

[531]: recommended_cars_agg, company_model = recommend_similar_cars_agg(lrpreds[567], car_full_analysis)

[533]: print(f"Recommended cars based on predicted company_model:  {company_model} ")
       print("using agglomerative clustering")
       print(recommended_cars_agg)
       print(f"Predicted car: {lrpreds[567]}")

       Recommended cars based on predicted company_model:  pontiac_sunfire
       using agglomerative clustering
              company_model      score
       17428      toyota_sienna  24.400864
       9324   volkswagen_passat  23.657392
       14054   chevrolet_impala  23.536020
       8964       ford_mustang  23.424841
       17293     subaru_outback  23.271211
       Predicted car: pontiac_sunfire
```

Figure 10 sample output for agglomerative

The agglomerative clustering is performed and the sample prediction is made on {popularity score, fuel efficiency score, price and annual income} taking the lrpreds[567}(output from logistic regression prediction).

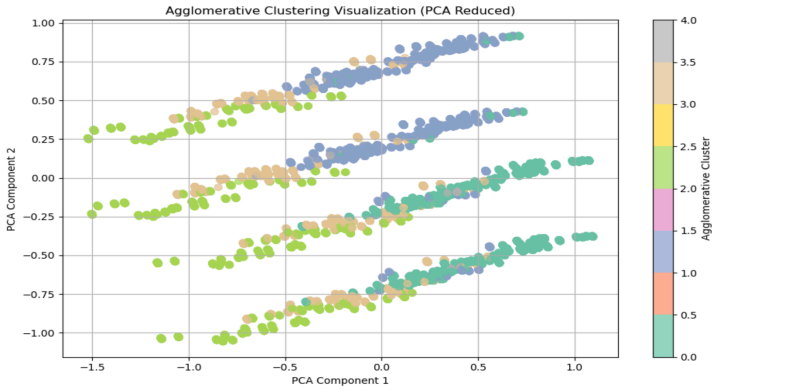


Figure 11 Agglomerative clustering image

c. GMM Clustering :

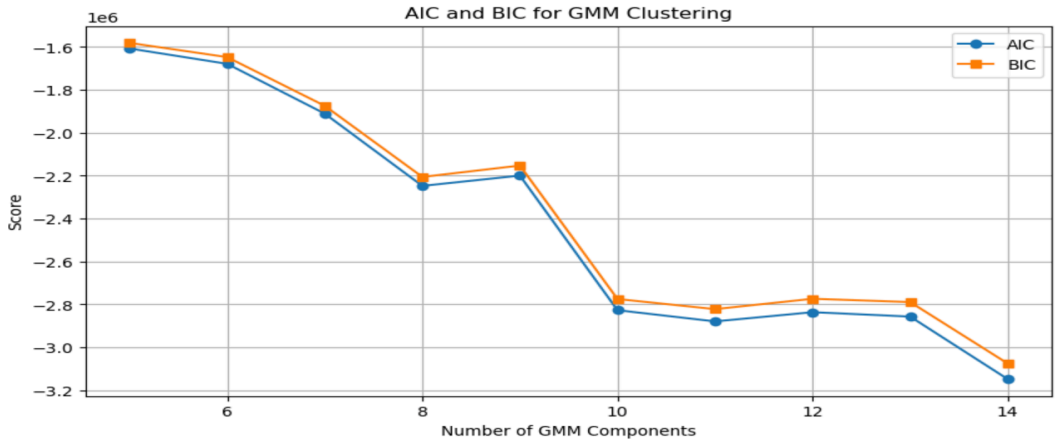


Figure 12 Optimal components for GMM

The number of components is decided as **14** as it has low AIC and BIC values.

```
[164]: from sklearn.mixture import GaussianMixture
       from sklearn.decomposition import PCA
       import matplotlib.pyplot as plt

       # Gaussian Mixture Model (GMM)
       gmm = GaussianMixture(n_components=14, random_state=42)
       car_full_analysis['GMM_Cluster'] = gmm.fit_predict(X)

[165]: def get_cluster_for_predicted_car_gmm(predicted_label, car_data):
           company_model = predicted_label
           predicted_car_row = car_data[car_data['company_model'] == company_model]
           predicted_car_cluster = predicted_car_row['GMM_Cluster'].values[0]
           return predicted_car_cluster, company_model

[166]: def recommend_similar_cars_gmm(predicted_label, car_data, top_n=5):
           predicted_cluster, company_model = get_cluster_for_predicted_car_gmm(predicted_label, car_data)
           similar_cars = car_data[car_data['GMM_Cluster'] == predicted_cluster]

           # similar cars based on the features
           similar_cars_scaled = similar_cars[['popularity_score', 'fuel_eff_score','Price ($)','Annual Income']]

           similar_cars = similar_cars.copy()
           similar_cars['score'] = similar_cars_scaled.sum(axis=1)

           recommended_cars_gmm = similar_cars.sort_values(by='score', ascending=False)
           return recommended_cars_gmm[['company_model','score']].drop_duplicates(subset='company_model').head(top_n), company_model

[167]: recommended_cars_gmm, company_model = recommend_similar_cars_gmm(lrpreds[4321], car_full_analysis)

[168]: print(f"Recommended cars based on company_model: {company_model}")
       print("using GMM clustering")
       print(recommended_cars_gmm)
       print(f"Predicted car: {lrpreds[4321]}")

Recommended cars based on company_model: chrysler_lhs
using GMM clustering
            company_model     score
7420  chevrolet_monte carlo  3.535501
3729           ford_taurus  3.239956
3425    lincoln_continental  3.099188
5552    ford_crown victoria  3.021670
7421      oldsmobile_aurora  3.004488
Predicted car: chrysler_lhs
```

Figure 13 sample output for GMM

The GMM clustering is performed and the sample prediction is made on {popularity score, fuel efficiency score, price and annual income} taking the lrpreds[4321}(output from logistic regression prediction) as an example.

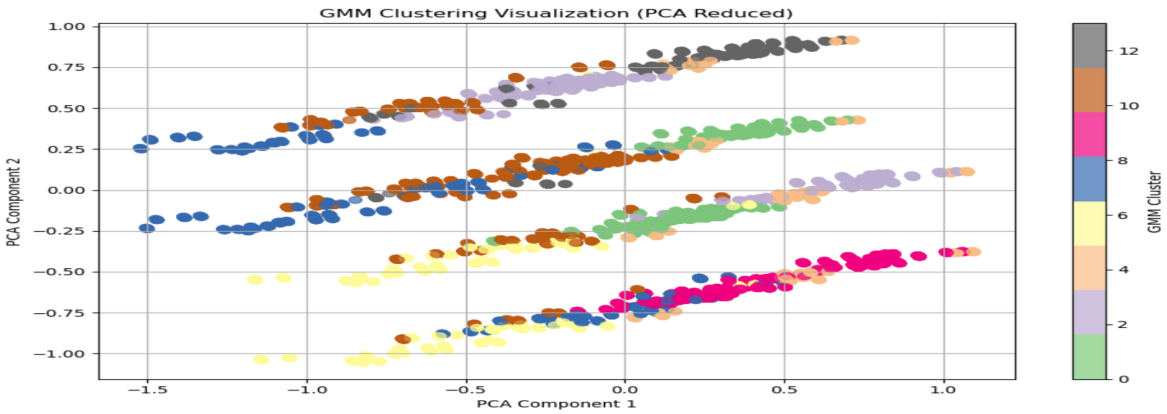


Figure 14 Clustering image for GMM

Clustering Models Evaluation Metrics

Clustering Model	Silhouette Score	Davies-Bouldin Index	Adjusted Rand Index (ARI)	Cluster Purity
KMeans	0.1444	2.1124	0.0247	0.0555
Agglomerative	0.1133	2.2867	0.0460	0.0907
Gaussian Mixture (GMM)	0.1241	2.0682	0.0950	0.1331

The evaluation of clustering models highlights shows performance characteristics across **KMeans, Agglomerative Clustering, and Gaussian Mixture Models (GMM). KMeans** achieved the highest **Silhouette Score** (0.1444), indicating relatively better-defined clusters, while GMM demonstrated the lowest **Davies-Bouldin Index** (2.0682), reflecting more compact and well separated clusters compared to others. **GMM** also outperformed in **Adjusted Rand Index (ARI)** at 0.0950 and **Cluster Purity** at 0.1331, suggesting great alignment. Agglomerative Clustering showed the **weakest** performance across most metrics, with the lowest Silhouette Score (0.1133) and highest Davies-Bouldin Index (2.2867. Overall, **GMM and Kmeans** demonstrates better clustering effectiveness in this context.

4. Model Optimization and New Performance Evaluation:

In **Milestone3,** the some models are optimized to enhance their performance and the evaluation is done again.

4.1 Supervised Models:

a. **Random Forest Classifier :**

```
[505]: # Random Forest Classifier Improvized changed "depth"
rfmodel = RandomForestClassifier(n_estimators=100, max_depth=12, random_state=42)

#Train model
rfmodel.fit(X_train, y_train)

# Predict output
rfpreds = rfmodel.predict(X_test)
```

Figure 15 Random forest enhanced model

The max depth is increased to predict and generalize better. This is done to improve the model's performance.

b. K-nearest neighbor:

```
[506]: #KNN improvised

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

knnmodel = KNeighborsClassifier()

# Define hyperparameters grid to tune the number of neighbors and the distance metric
param_grid = {
    'n_neighbors': [ 7, 8,9, 10],  # Testing different numbers of neighbors
    'metric': ['euclidean', 'manhattan', 'minkowski']  # Different distance metrics
}

# Initialize GridSearchCV
grid_search = GridSearchCV(knnmodel, param_grid, cv=5, n_jobs=-1, verbose=1, scoring='accuracy')

grid_search.fit(X_train, y_train)

# find the best
best_knn_model = grid_search.best_estimator_
print(f"Best Hyperparameters: {grid_search.best_params_}")

# Train the best model
best_knn_model.fit(X_train, y_train)

# Make predictions using the best model
knnpreds = best_knn_model.predict(X_test)

# Evaluate the model
print(f"Accuracy: {accuracy_score(y_test, knnpreds):.4f}")
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best Hyperparameters: {'metric': 'manhattan', 'n_neighbors': 7}
Accuracy: 0.9924
```

Figure 16 KNN enhanced model

The basic KNN model's performance heavily depends on choosing the right **number of neighbors (k)** and the **distance metric**. In this improvization, you applied **GridSearchCV** to systematically search for the best hyperparameters, ensuring that your KNN model is well-tuned for maximum accuracy.

```
[513]: # Evaluate each model
evaluate_model(y_test, lrpreds, lrmodel, "Logistic Regression")
evaluate_model(y_test, rfpreds, rfmodel, "Random Forest")
evaluate_model(y_test, knnpreds, best_knn_model, "KNN")
```

```
Logistic Regression - Accuracy: 0.9782, Precision: 0.9770, Recall: 0.9782, F1-score: 0.9764
Random Forest - Accuracy: 0.9530, Precision: 0.9336, Recall: 0.9530, F1-score: 0.9401
KNN - Accuracy: 0.9924, Precision: 0.9928, Recall: 0.9924, F1-score: 0.9922
```

s

Figure 17 New performance metrics

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.9782	0.9770	0.9782	0.9764
Random Forest	0.9530	0.9336	0.9530	0.9401
K-Nearest Neighbors	0.9924	0.9928	0.9924	0.9922

These results indicate that **K-Nearest Neighbors** achieved the highest overall performance after **tuning**, particularly excelling in precision and recall, making it highly reliable for consistent predictions. Logistic Regression remained **robust** and balanced metrics, while **Random Forest** showed notable improvement in precision due to **depth limitation**.

4.2 Unsupervised models :

In **Milestone 2**, the clustering algorithms (KMeans, GMM, Agglomerative) were initially built using a limited feature set consisting of only four attributes: **Popularity Score**, **Fuel Efficiency Score**, **Price**, and **Annual Income**. While these features provided a basic understanding of car groupings, they lacked depth in capturing the full characteristics of each vehicle.

To improve clustering performance and achieve more meaningful segmentation, the number of features was significantly expanded in **Milestone 3**. The enhanced feature set now includes

- **Engine attributes** (e.g., horsepower RPM, displacement per cylinder, power-to-weight ratio)
- **Physical dimensions** (e.g., wheelbase, full weight, volume)
- **Performance metrics** (e.g., max speed, torque RPM)
- **Design and structural elements** (e.g., body type, drive wheels, suspension types)
- **User-centric scores** (e.g., average rating, max trunk capacity)

a. K-means clustering:

```
[179]: def recommend_similar_cars(predicted_label, car_data, top_n=5):

    predicted_cluster, company_model = get_cluster_for_predicted_car(predicted_label, car_data)

    similar_cars = car_data[car_data['Cluster'] == predicted_cluster].copy()
    similar_cars = similar_cars[similar_cars['company_model'] != company_model]

    feature_cols = [
        'Annual Income', 'Transmission', 'Color', 'Price ($)', 'Sale_Year',
        'Body_type', 'number_of_seats', 'wheelbase_mm', 'full_weight_kg',
        'max_trunk_capacity_l', 'injection_type', 'engine_type',
        'valves_per_cylinder', 'stroke_cycle_mm', 'turnover_of_maximum_torque_rpm',
        'engine_hp_rpm', 'drive_wheels', 'turning_circle_m', 'transmission',
        'fuel_tank_capacity_l', 'max_speed_km_per_h', 'fuel_grade',
        'back_suspension', 'rear_brakes', 'front_brakes', 'front_suspension',
        'avg_rating', 'max_rating', 'min_rating', 'popularity_score',
        'fuel_eff_score', 'avg_track_mm', 'displacement_per_cylinder',
        'power_to_weight', 'volume_mm3'
    ]

    similar_cars_scaled = similar_cars[feature_cols]
    similar_cars['score'] = similar_cars_scaled.sum(axis=1)

    recommended_cars = similar_cars.sort_values(by='score', ascending=False)
    return recommended_cars[['company_model', 'score']].drop_duplicates('company_model').head(top_n), company_model
```

Figure 18 Kmeans on multiple features

The function for k-means is updated with “feature columns” which help in getting similar cars based on all the features in the cluster where the supervised model output is taken as input.

Sample output:

```
[182]: recommended_cars, company_model = recommend_similar_cars(lrpreds[2347], car_full_analysis)
       print(f"Recommended cars based on predicted company_model: - {company_model} ")
       print("using Kmeans clustering")
       print(recommended_cars)
       print(f"Predicted car from logistic is: {lrpreds[2347]}")
```

```
Recommended cars based on predicted company_model: - lincoln_navigator
using Kmeans clustering
      company_model      score
6754    toyota_4runner  23.572050
4509          audi_a6  23.098838
6715      nissan_quest  22.901953
1691  chevrolet_camaro  22.784191
5546    toyota_sienna  22.066463
Predicted car from logistic is: lincoln_navigator
```

Figure 19 Sample output

b. Agglomerative clustering:

```
[181]: def recommend_similar_cars_agg(predicted_label, car_data, top_n=5):

    predicted_cluster, company_model = get_cluster_for_predicted_car_agg(predicted_label, car_data)

    similar_cars = car_data[car_data['Agglo_Cluster'] == predicted_cluster].copy()
    similar_cars = similar_cars[similar_cars['company_model'] != company_model]

    feature_cols = [
        'Annual Income', 'Transmission', 'Color', 'Price ($)', 'Sale_Year',
        'Body_type', 'number_of_seats', 'wheelbase_mm', 'full_weight_kg',
        'max_trunk_capacity_l', 'injection_type', 'engine_type',
        'valves_per_cylinder', 'stroke_cycle_mm', 'turnover_of_maximum_torque_rpm',
        'engine_hp_rpm', 'drive_wheels', 'turning_circle_m','transmission','fuel_tank_capacity_l', 'max_speed_km_per_h', 'fuel_grade',
        'back_suspension', 'rear_brakes', 'front_brakes', 'front_suspension',
        'avg_rating', 'max_rating', 'min_rating', 'popularity_score',
        'fuel_eff_score', 'avg_track_mm', 'displacement_per_cylinder',
        'power_to_weight', 'volume_mm3'
    ]

    similar_cars_scaled = similar_cars[feature_cols]
    similar_cars['score'] = similar_cars_scaled.sum(axis=1)

    recommended_cars = similar_cars.sort_values(by='score', ascending=False)
    return recommended_cars[['company_model', 'score']].drop_duplicates('company_model').head(top_n), company_model
```

Figure 20 agglomerative on multiple features

The function for agglomerative is updated with “feature columns” which help in getting similar cars based on all the features in the cluster where the supervised model output is taken as input.

Sample output:

```
[183]: recommended_cars_agg, company_model = recommend_similar_cars_agg(lrpreds[678], car_full_analysis)
print(f"Recommended cars based on predicted company_model:  {company_model} ")
print("using agglomerative clustering")
print(recommended_cars_agg)
print(f"Predicted car from logistic is: {lrpreds[678]}")

Recommended cars based on predicted company_model:  nissan_pathfinder
using agglomerative clustering
          company_model      score
17567  chevrolet_camaro  23.826383
10357      toyota_celica  23.106614
17405      toyota_sienna  23.030215
10338         acura_tl  22.766297
9379   cadillac_deville  22.747500
Predicted car from logistic is: nissan_pathfinder
```

Figure 21 sample output for agglomerative

c. GMM clustering:

```
[180]: def recommend_similar_cars_gmm(predicted_label, car_data, top_n=5):

    predicted_cluster, company_model = get_cluster_for_predicted_car_gmm(predicted_label, car_data)

    similar_cars = car_data[car_data['GMM_Cluster'] == predicted_cluster].copy()
    similar_cars = similar_cars[similar_cars['company_model'] != company_model]

    feature_cols = [
        'Annual Income', 'Transmission', 'Color', 'Price ($)', 'Sale_Year',
        'Body_type', 'number_of_seats', 'wheelbase_mm', 'full_weight_kg',
        'max_trunk_capacity_l', 'injection_type', 'engine_type',
        'valves_per_cylinder', 'stroke_cycle_mm', 'turnover_of_maximum_torque_rpm',
        'engine_hp_rpm', 'drive_wheels', 'turning_circle_m', 'transmission',
        'fuel_tank_capacity_l', 'max_speed_km_per_h', 'fuel_grade',
        'back_suspension', 'rear_brakes', 'front_brakes', 'front_suspension',
        'avg_rating', 'max_rating', 'min_rating', 'popularity_score',
        'fuel_eff_score', 'avg_track_mm', 'displacement_per_cylinder',
        'power_to_weight', 'volume_mm3'
    ]

    similar_cars_scaled = similar_cars[feature_cols]
    similar_cars['score'] = similar_cars_scaled.sum(axis=1)

    recommended_cars = similar_cars.sort_values(by='score', ascending=False)
    return recommended_cars[['company_model', 'score']].drop_duplicates('company_model').head(top_n), company_model
```

Figure 22 GMM with additional features

The function for GMM is updated with “feature columns” which help in getting similar cars based on all the features in the cluster where the supervised model output is taken as input.

Sample output:

```
[184]: recommended_cars_gmm, company_model = recommend_similar_cars_gmm(lrpreds[2345], car_full_analysis)
print(f"Recommended cars based on company_model: {company_model}")
print("using GMM clustering")
print(recommended_cars_gmm)
print(f"Predicted car from logistic is: {lrpreds[2345]}")

Recommended cars based on company_model: cadillac_eldorado
using GMM clustering
          company_model      score
2287      toyota_sienna  23.346831
2341   chevrolet_impala  22.569960
5570          acura_tl  22.223226
3476  oldsmobile_aurora  22.213957
2039       buick_regal  22.209031
Predicted car from logistic is: cadillac_eldorado
```

Figure 23 sample output GMM

5. Models Output Interpretations, Bias and Limitations:

To interpret the model output for **supervised** models the **classification report** of the predictions is taken and the **feature importance**.

5.1 Logistic Regression: Interpretation, Bias and Limitations:

```
[200]: # Classification Report
print("\n Logistic regression Classification Report:")
print(classification_report(y_test, lrpreds))

# Confusion Matrix
cm = confusion_matrix(y_test, lrpreds)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Logistic Regression - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

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

          acura_integra       1.00      1.00      1.00        14
              acura_rl        0.64      0.90      0.75       112
              acura_tl        1.00      1.00      1.00        81
               audi_a4        1.00      1.00      1.00        31
               audi_a6        1.00      1.00      1.00        99
               audi_a8        1.00      0.45      0.62        11
         buick_century        1.00      1.00      1.00        13
         buick_lesabre        1.00      1.00      1.00        12
      buick_park avenue       1.00      1.00      1.00        61
          buick_regal         1.00      1.00      1.00        46
        cadillac_catera       1.00      1.00      1.00        27
       cadillac_deville       1.00      1.00      1.00        24
      cadillac_eldorado       1.00      1.00      1.00        70
      cadillac_escalade       1.00      1.00      1.00        49
       cadillac_seville       1.00      1.00      1.00        26
```

Figure 24 Logistic regression classification report

The code represents the classification report for logistic regression.

```
[535...
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

coefficients = pd.Series(np.abs(lrmodel.coef_).mean(axis=0), index=X_train.columns)

# Get Top 10 Important Features
top_features = coefficients.nlargest(10)

# Plot
plt.figure(figsize=(8,6))
top_features.plot(kind='barh', color='cornflowerblue')
plt.title("Top 10 Feature Importances - Logistic Regression")
plt.xlabel("Average Absolute Coefficient Value")
plt.gca().invert_yaxis()
plt.show()
```

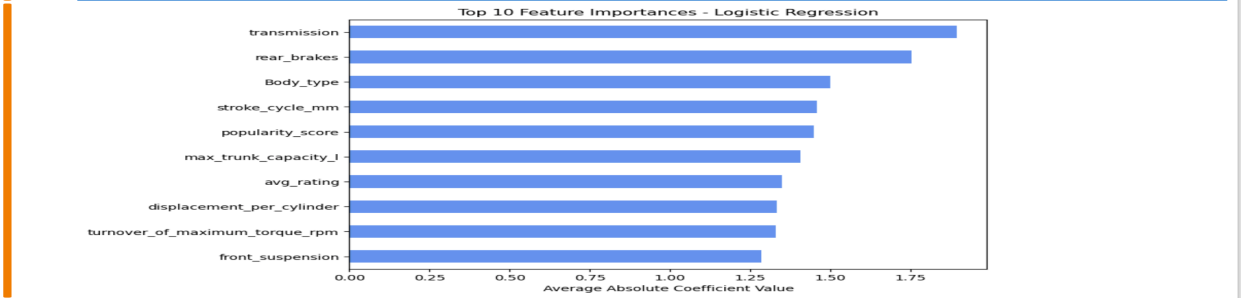


Figure 25 Feature importance logistic regression

The image represents the feature importance for logistic regression and output.

The Logistic Regression model achieved a strong overall accuracy of **98%**, with a weighted precision and recall also at **98%**, reflecting it is very good in classifying most of the car models. The classification report represents that the model performed exceptionally well on most classes. However, performance **was not good** for classes such as **Chrysler Town & Country, Saab 5-Sep, and Toyota Avalon**, where precision, recall, and F1-scores dropped to zero. This is a **limitation** in handling minority classes. The feature importance analysis shows that the model relies mostly on features like **Transmission**, **Rear Brakes**, **Body Type**, **Stroke Cycle (mm)** , **Popularity Score** and **Average Rating**. These are dominant categories which shows **bias** over other categories. Therefore, logistic regression model worked for most of the data, while the model's **limitation** is generalization capability for less common vehicle types.

Example with sample input:

I have also analyzed but just taking a sample test data and predicted using logistic regression:

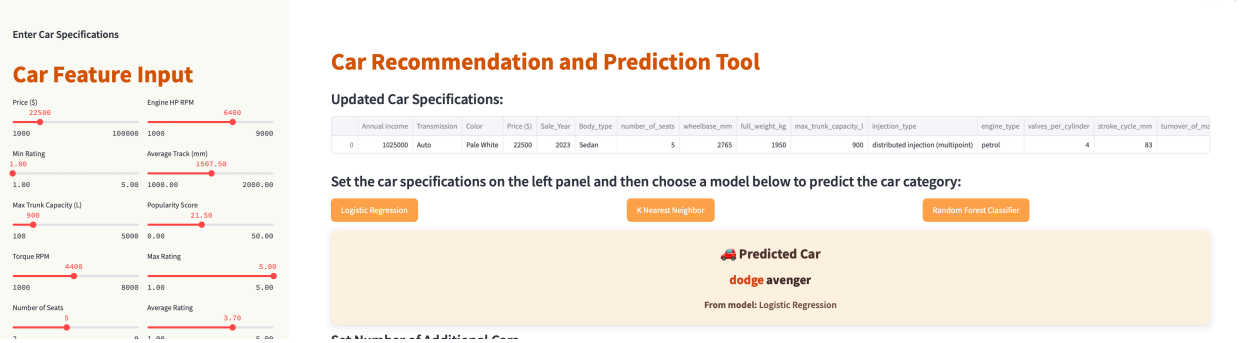
```
[50 rows x 35 columns]

[192]:
# Predict with Logistic Regression
logistic_preds_test = lrmodel.predict(sample_X)

# Predict with KNN
knn_preds_test = best_knn_model.predict(sample_X)

# Predict with Random Forest
rf_preds_test = rfmodel.predict(sample_X)

[437]:
# Predict with Logistic Regression
logistic_preds_test = lrmodel.predict(X_test.iloc[500:501])

# Predict with KNN
knn_preds_test = best_knn_model.predict(X_test.iloc[500:501])

# Predict with Random Forest
rf_preds_test = rfmodel.predict(X_test.iloc[500:501])

[439]:
import pandas as pd

# Combine predictions into a DataFrame
print(f" Result for row {11407} is : ")
results_df = pd.DataFrame({
    'Logistic Regression': logistic_preds_test,
    'KNN': knn_preds_test,
    'Random Forest': rf_preds_test
})

# Show the predictions
print(results_df)

 Result for row 11407 is :
  Logistic Regression            KNN  Random Forest
0       dodge_avenger  dodge_avenger  dodge_avenger
```

Figure 26 sample prediction for analysis

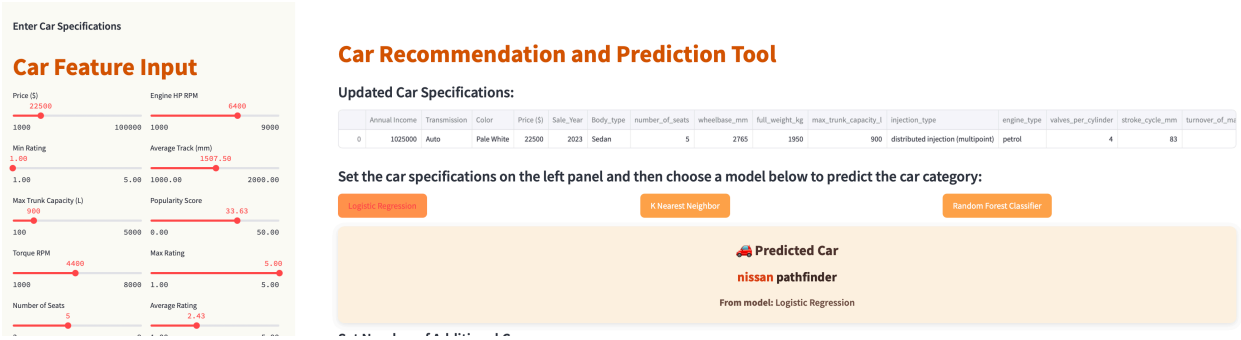
The code represents the taking of the prediction from X_test , the value is correctly predicted by logistic regression as “Dodge Avenger”. By changing the metrics “popularity score” and “average rating” the output changed from “dodge avenger” to “Nissan pathfinder”

Used my UI for this:

Before change :



After changing the features **average rating and popularity score** the prediction changed proving they are **high importance** features:



The **Nissan pathfinder** is justifiable as the new values are similar to the car.

5.2 Random Forest Classifier: Interpretation, Bias and Limitations:

```
[204]: # RANDOM FOREST — Classification Report
print("\nRandom Forest — Classification Report:")
print(classification_report(y_test, rfpreds, zero_division=0))

# RANDOM FOREST — Confusion Matrix
cm_rf = confusion_matrix(y_test, rfpreds)
plt.figure(figsize=(10,7))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Greens')
plt.title("Random Forest — Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Random Forest — Classification Report:
                          precision    recall  f1-score   support

             acura_integra       1.00      1.00      1.00        14
                acura_rl         0.97      1.00      0.98       112
                acura_tl         1.00      1.00      1.00        81
                 audi_a4         1.00      1.00      1.00        31
                 audi_a6         1.00      1.00      1.00        99
                 audi_a8         1.00      1.00      1.00        11
           buick_century         1.00      1.00      1.00        13
           buick_lesabre         0.00      0.00      0.00        12
         buick_park avenue       1.00      1.00      1.00        61
           buick_regal           1.00      1.00      1.00        46
         cadillac_catera         1.00      1.00      1.00        27
        cadillac_deville         0.00      0.00      0.00        24
      cadillac_eldorado          1.00      1.00      1.00        70
      cadillac_escalade          1.00      1.00      1.00        49
        cadillac_seville         1.00      1.00      1.00        26
        chevrolet_camaro         1.00      1.00      1.00        33
      chevrolet_cavalier         1.00      1.00      1.00        51
      chevrolet_corvette         1.00      1.00      1.00        74
        chevrolet_impala         1.00      1.00      1.00        24
        chevrolet_lumina         1.00      1.00      1.00        49
         chevrolet_malibu        0.30      1.00      0.47        71
         chevrolet_metro         1.00      1.00      1.00        77
    chevrolet_monte carlo        1.00      1.00      1.00        44
        chevrolet_prizm          1.00      1.00      1.00       123
        chrysler_300m            1.00      1.00      1.00        73
       chrysler_cirrus           1.00      1.00      1.00        11
     chrysler_concorde           1.00      1.00      1.00        71
         chrysler_lhs            1.00      1.00      1.00        99
chrysler_town & country          0.00      0.00      0.00        10
          dodge_avenger          1.00      1.00      1.00        34
          dodge_caravan          1.00      1.00      1.00        42
           dodge_dakota          0.00      0.00      0.00        20
         dodge_durango           1.00      1.00      1.00        79
        dodge_intrepid           1.00      1.00      1.00        23
           dodge_neon            1.00      1.00      1.00         7
         dodge_stratus           1.00      1.00      1.00        27
           dodge_viper           1.00      1.00      1.00        72
          ford_contour           1.00      1.00      1.00        34
    ford_crown victoria          1.00      1.00      1.00        45
           ford_escort           1.00      1.00      1.00        23
        ford_expedition          1.00      1.00      1.00        64
          ford_explorer          1.00      1.00      1.00        68
          ford_f-series          1.00      1.00      1.00        33
            ford_focus           1.00      1.00      1.00        44
          ford_mustang           1.00      1.00      1.00        52
           ford_ranger           1.00      1.00      1.00        35
           ford_taurus           1.00      1.00      1.00        50
         ford_windstar           1.00      1.00      1.00        36
          honda_accord           1.00      1.00      1.00        73
```

Figure 27 Random forest classification report

The code represents the classification report for logistic regression.

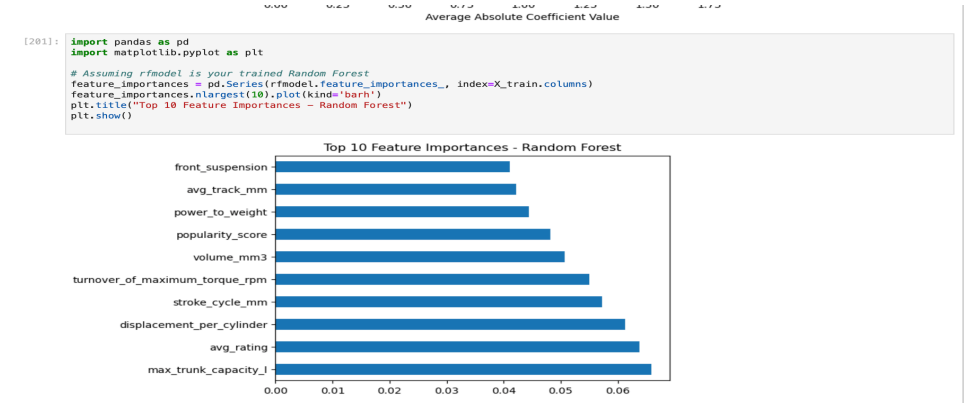


Figure 28 feature importance random forest

The image represents the feature importance for Random Forest classifier and output.

The Random Forest Classifier achieved an overall accuracy of **95%**, with a **precision of 93%**, **recall** of **95%**, and an F1-score of **94%**, indicating good performance , but less compared to other models. The feature importance analysis reveals that variables like **max_trunk_capacity_l**, **avg_rating**, and **displacement_per_cylinder** were the most important factors in predictions. Also, the classification report shows main **biases**, as several classes (e.g**., buick_lesabre, cadillac_deville, chrysler_town and dodge_dakota**) show **zero precision and recall**, indicating the model's inability to correctly get the outputs. This reflects a **limitation** in handling underrepresented classes. Overall, while Random Forest offers robust performance for well-represented categories, it exhibits **bias** towards dominant patterns in the data.

Example with sample input:

I have also analyzed but just taking a sample test data and predicted using random forest classifier:

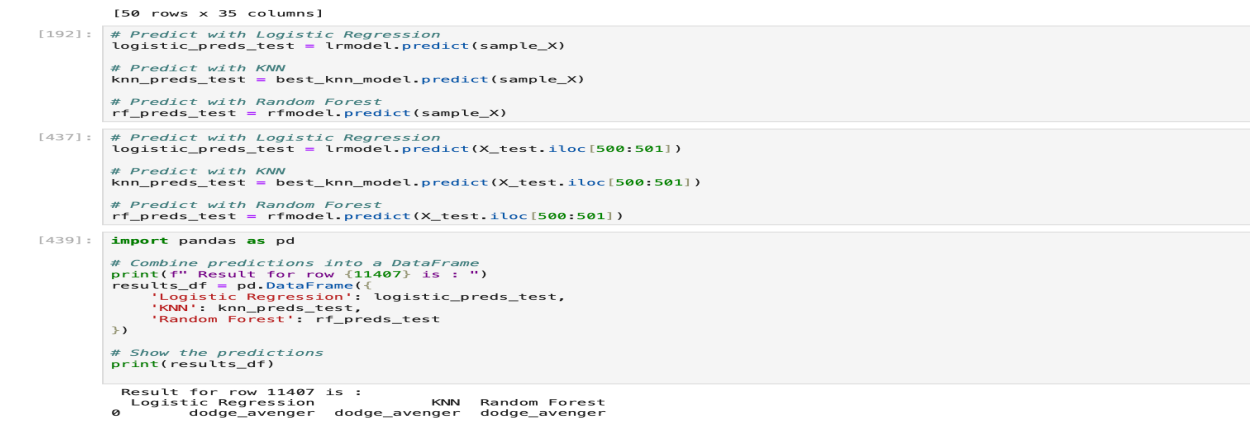
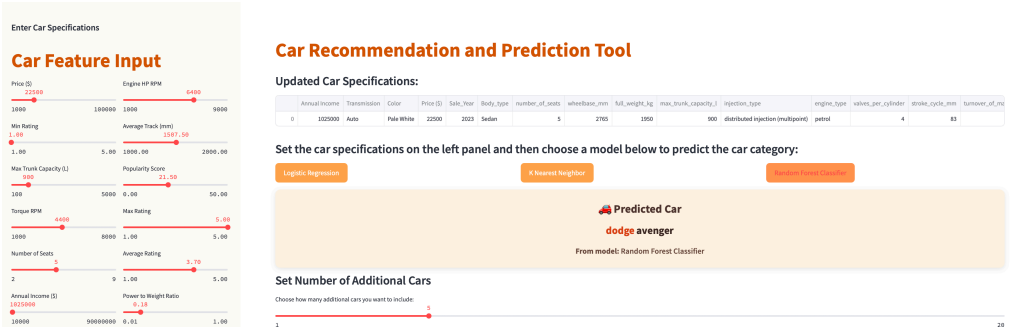


Figure 29 Sample prediction

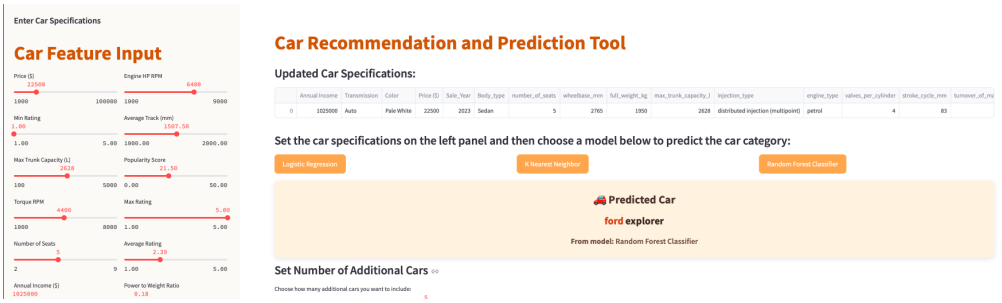
The code represents the taking of the prediction from X_test , the value is correctly predicted by Random forest as "**Dodge Avenger**".

By changing the metrics **max_trunk_capacity_l**, and **avg_rating** the output changed from “dodge avenger” to “ford explorer”

Before:



After changing the features **max_trunk_capacity_l**, and **avg_rating** the prediction changed proving they are **high importance** features:



5.3 K Nearest Neighbor:

```
[205]: # KNN – Classification Report
print("\nK-Nearest Neighbors – Classification Report:")
print(classification_report(y_test, knnpreds, zero_division=0))

# KNN – Confusion Matrix
cm_knn = confusion_matrix(y_test, knnpreds)
plt.figure(figsize=(10,7))
sns.heatmap(cm_knn, annot=True, fmt='d', cmap='Oranges')
plt.title("K-Nearest Neighbors – Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
K-Nearest Neighbors – Classification Report:
                        precision    recall  f1-score   support

          acura_integra       1.00      1.00      1.00        14
              acura_rl        0.86      0.99      0.92       112
              acura_tl        1.00      1.00      1.00        81
               audi_a4        1.00      1.00      1.00        31
               audi_a6        1.00      1.00      1.00        99
               audi_a8        1.00      0.64      0.78        11
         buick_century        1.00      1.00      1.00        13
         buick_lesabre        1.00      1.00      1.00        12
     buick_park avenue        1.00      1.00      1.00        61
          buick_regal         1.00      1.00      1.00        46
       cadillac_catera        1.00      1.00      1.00        27
      cadillac_deville        1.00      1.00      1.00        24
     cadillac_eldorado        1.00      1.00      1.00        70
     cadillac_escalade        1.00      1.00      1.00        49
      cadillac_seville        1.00      1.00      1.00        26
     chevrolet_camaro         1.00      1.00      1.00        33
    chevrolet_cavalier        1.00      1.00      1.00        51
    chevrolet_corvette        0.97      0.95      0.96        74
      chevrolet_impala        1.00      1.00      1.00        24
      chevrolet_lumina        1.00      1.00      1.00        49
      chevrolet_malibu        1.00      1.00      1.00        71
```

Figure 30 KNN classifiaction report

The code represents the classification report for KNN.

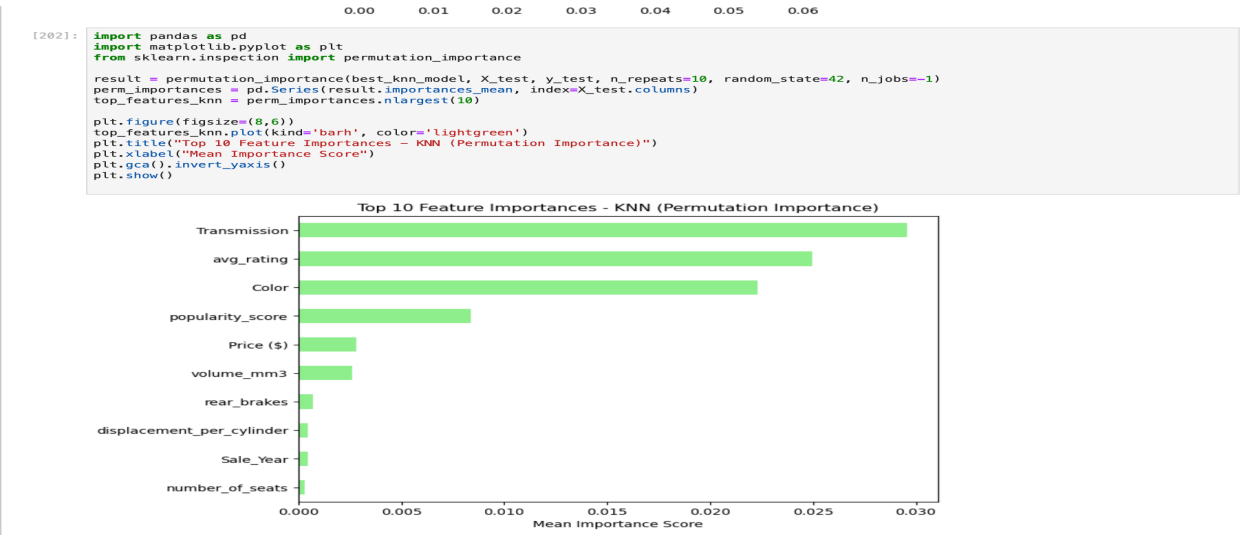


Figure 31 KNN feature importance

The above code generates the permutation importance for KNN.

The K-Nearest Neighbors (KNN) model has shown **excellent** performance with an overall **accuracy of 99%**, along with high **precision, recall, and F1-scores** of **99%** each. This indicates that the model is highly effective in classifying almost all car models accurately. The **permutation feature importance** analysis shows that **Transmission**, **Average Rating**, **Color**, and **Popularity Score** are the most influential factors which cause **bias** for the KNN predictions. While the **classification report** shows almost perfect scores across all of classes, there are slight **limitation** for certain categories such as **acura_rl** (precision **0.86**) and **saab_5-sep** (recall **0.58**), indicating **misclassifications** in models with limited feature distinctiveness. The key **limitation** of KNN lies as more features are added. Overall, KNN is excellent among all supervised models.

Example with sample input:

I have also analyzed but just taking a sample test data and predicted using KNN model:

```
[50 rows x 35 columns]

[192]:
# Predict with Logistic Regression
logistic_preds_test = lrmodel.predict(sample_X)

# Predict with KNN
knn_preds_test = best_knn_model.predict(sample_X)

# Predict with Random Forest
rf_preds_test = rfmodel.predict(sample_X)

[437]:
# Predict with Logistic Regression
logistic_preds_test = lrmodel.predict(X_test.iloc[500:501])

# Predict with KNN
knn_preds_test = best_knn_model.predict(X_test.iloc[500:501])

# Predict with Random Forest
rf_preds_test = rfmodel.predict(X_test.iloc[500:501])

[439]:
import pandas as pd

# Combine predictions into a DataFrame
print(f" Result for row {11407} is : ")
results_df = pd.DataFrame({
    'Logistic Regression': logistic_preds_test,
    'KNN': knn_preds_test,
    'Random Forest': rf_preds_test
})

# Show the predictions
print(results_df)

 Result for row 11407 is :
  Logistic Regression            KNN  Random Forest
0      dodge_avenger  dodge_avenger  dodge_avenger
```

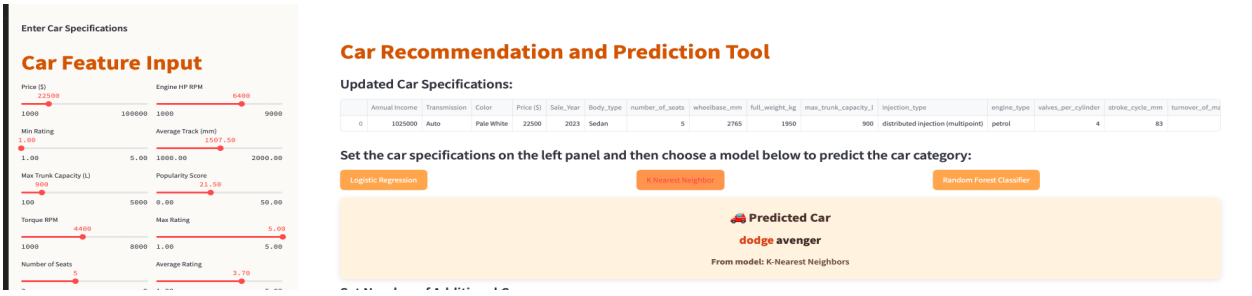
Figure 32 Sample prediction

The code represents the taking of the prediction from X_test , the value is correctly predicted by KNN as “Dodge Avenger”.

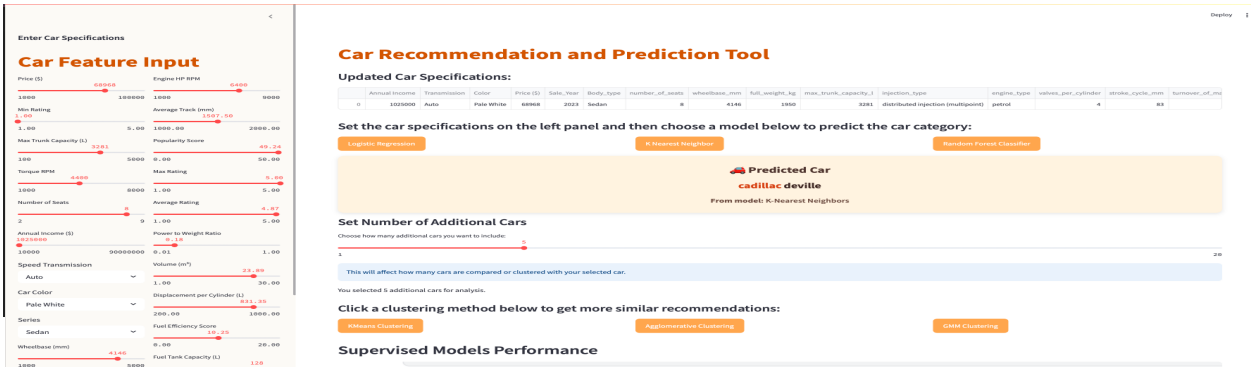
By changing the metrics **popularity score**, and **avg rating** the output changed from “dodge avenger” to “cardillac deville”

Used my UI for this:

Before:



After changing the features **popularity score**, **price, volume, number of seats** and **avg rating** the prediction changed proving they are **high importance** features:



5.4 K-Means clustering:

```
[207]: import pandas as pd
       import matplotlib.pyplot as plt

       kmeans_counts = car_full_analysis['Cluster'].value_counts().sort_index()
       print("Number of Cars in Each KMeans Cluster:")
       print(kmeans_counts)

       # Bar Plot
       plt.figure(figsize=(8,5))
       kmeans_counts.plot(kind='bar', color='skyblue', edgecolor='black')
       plt.title('Number of Cars in Each KMeans Cluster')
       plt.xlabel('Cluster Number')
       plt.ylabel('Number of Cars')
       plt.xticks(rotation=0)
       plt.grid(axis='y', linestyle='--', alpha=0.7)
       plt.show()

       # Pie Chart
       plt.figure(figsize=(7,7))
       kmeans_counts.plot(kind='pie', autopct='%1.1f%%', startangle=90)
       plt.title('Car Distribution Across KMeans Clusters')
       plt.ylabel('')
       plt.show()

       unique_clusters = sorted(car_full_analysis['Cluster'].unique())

       for cluster_num in unique_clusters:
           print(f"\n Cluster {cluster_num} - Top 5 Car Models:")
           top_models = car_full_analysis[car_full_analysis['Cluster'] == cluster_num]['company_model'].value_counts().head(5)
           print(top_models)

           # Visualize Top Models
           plt.figure(figsize=(8,4))
           top_models.plot(kind='barh', color='coral', edgecolor='black')
           plt.title(f'Top 5 Car Models in Cluster {cluster_num}')
           plt.xlabel('Count')
           plt.gca().invert_yaxis()  # Highest at top
           plt.grid(axis='x', linestyle='--', alpha=0.6)
           plt.show()
```

```
Number of Cars in Each KMeans Cluster:
Cluster
0    3247
1    4363
2    3201
3    4030
4    3529
Name: count, dtype: int64
```

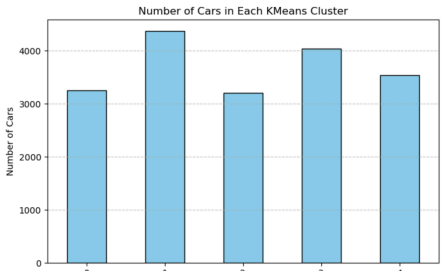


Figure 33 Kmeans cluster analysis

This code analyzes and visualizes the distribution of cars across K-Means clusters, highlighting the number of cars per cluster and the top 5 car models within each cluster.

For example:

Cluster 0 has acura_rl(146) and chevrolet_prizm(120) which cause inherent **bias towards over r epresented brands and categories.**

5.5 Agglomerative clustering:

```
[208]:
import pandas as pd
import matplotlib.pyplot as plt

agglo_counts = car_full_analysis['Agglo_Cluster'].value_counts().sort_index()
print("Number of Cars in Each Agglomerative Cluster:")
print(agglo_counts)

# Bar Plot
plt.figure(figsize=(8,5))
agglo_counts.plot(kind='bar', color='plum', edgecolor='black')
plt.title('Number of Cars in Each Agglomerative Cluster')
plt.xlabel('Agglomerative Cluster Number')
plt.ylabel('Number of Cars')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Pie Chart
plt.figure(figsize=(7,7))
agglo_counts.plot(kind='pie', autopct='%1.1f%%', startangle=90, colormap='Pastel1')
plt.title('Car Distribution Across Agglomerative Clusters')
plt.ylabel('')
plt.show()

unique_agglo_clusters = sorted(car_full_analysis['Agglo_Cluster'].unique())

for cluster_num in unique_agglo_clusters:
    print(f"\n Agglomerative Cluster {cluster_num} - Top 5 Car Models:")
    top_models = car_full_analysis[car_full_analysis['Agglo_Cluster'] == cluster_num]['company_model'].value_counts().head(5)
    print(top_models)

    # Visualize Top Models
    plt.figure(figsize=(8,4))
    top_models.plot(kind='barh', color='mediumorchid', edgecolor='black')
    plt.title(f'Top 5 Car Models in Agglomerative Cluster {cluster_num}')
    plt.xlabel('Count')
    plt.gca().invert_yaxis()  # Highest at top
    plt.grid(axis='x', linestyle='--', alpha=0.6)
    plt.show()
```

```
Number of Cars in Each Agglomerative Cluster:
Agglo_Cluster
0    6092
1    5696
2    3145
3    2979
4     458
Name: count, dtype: int64
```

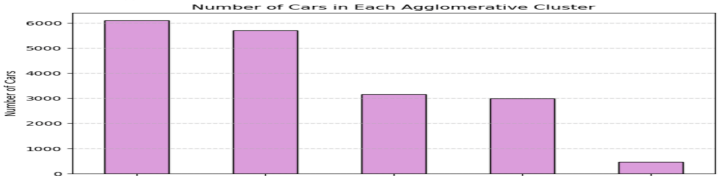


Figure 34 Agglomerative cluster analysis

This code analyzes and visualizes the distribution of cars across agglomerative clusters, highlighting the number of cars per cluster and the top 5 car models within each cluster.

For example: Cluster 0 has volkswagen_jetta(280) and volkswagen_passat (260) which cause inherent **bias towards over represented brands and categories.**

5.6 GMM Clustering:

```
[209]:
import pandas as pd
import matplotlib.pyplot as plt

gmm_counts = car_full_analysis['GMM_Cluster'].value_counts().sort_index()
print("Number of Cars in Each GMM Cluster:")
print(gmm_counts)

# Bar Plot
plt.figure(figsize=(8,5))
gmm_counts.plot(kind='bar', color='lightgreen', edgecolor='black')
plt.title('Number of Cars in Each GMM Cluster')
plt.xlabel('GMM Cluster Number')
plt.ylabel('Number of Cars')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Pie Chart
plt.figure(figsize=(7,7))
gmm_counts.plot(kind='pie', autopct='%1.1f%%', startangle=90, colormap='Set3')
plt.title('Car Distribution Across GMM Clusters')
plt.ylabel('')
plt.show()

unique_gmm_clusters = sorted(car_full_analysis['GMM_Cluster'].unique())

for cluster_num in unique_gmm_clusters:
    print(f"\n GMM Cluster {cluster_num} - Top 5 Car Models:")
    top_models = car_full_analysis[car_full_analysis['GMM_Cluster'] == cluster_num]['company_model'].value_counts().head(5)
    print(top_models)

    # Visualize Top Models
    plt.figure(figsize=(8,4))
    top_models.plot(kind='barh', color='orchid', edgecolor='black')
    plt.title(f'Top 5 Car Models in GMM Cluster {cluster_num}')
    plt.xlabel('Count')
    plt.gca().invert_yaxis()
    plt.grid(axis='x', linestyle='--', alpha=0.6)
    plt.show()
```

```
Number of Cars in Each GMM Cluster:
GMM_Cluster
0     1113
1     1800
2     1453
3     1641
4     1339
5      240
6     1510
7     1139
8      970
9     2336
10    1175
11    1838
12     417
13    1399
Name: count, dtype: int64
```

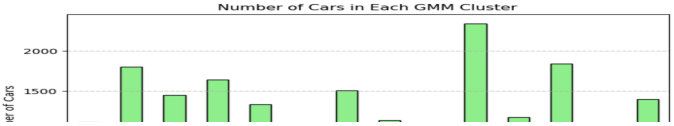


Figure 35 GMM cluster analysis

This code analyzes and visualizes the distribution of cars across GMM clusters, highlighting the number of cars per cluster and the top 5 car models within each cluster.

For example:

Cluster 0 has acura_rl(146) and volvo_s40 (70) which cause inherent **bias towards over represente d brands and categories.**

6. Interactive UI using streamlit for Car Recommendation engine:

6.1 Export of models, scaler and encoder:

```
[311]: # export models,encoder,scaler and cluster information

[313]: # export the models

       import joblib

       # Save the trained models to files
       joblib.dump(best_knn_model, 'knn_model.pkl')
       joblib.dump(rfmodel, 'rf_model.pkl')
       joblib.dump(lrmodel, 'logistic_regression_model.pkl')
       joblib.dump(kmeans, 'kmeans_model.pkl')
       joblib.dump(agglo,'agglo_model.pkl')
       joblib.dump(gmm,'gmm_model.pkl')

[313]: ['gmm_model.pkl']

[315]: # export the dataframe as it contains cluster information

       car_full_analysis.to_csv('clusteringanddata.csv')

[317]: # Save the scaler to a file
       joblib.dump(scaler_m, 'scaler.pkl')

       # dump encoders

       import pickle

       with open('label_encoders.pkl', 'wb') as f:
           pickle.dump(encoders, f)
```

Figure 36 Joblib export

The trained models have been exported for reuse in the CarRecommendation.py file, which conta ins the code for developing an interactive user interface. This UI enables real-time car recommen dations based on user input. "**clusteringanddata.csv**" which contains the data for clustering labe ls and **scaler** to scale the raw input, **encoders** from the user are also exported.

6.2 main.py file:

```
import numpy as np
import joblib
import pandas as pd
import pickle
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import hashlib
import json


lrmodel = joblib.load('models/logistic_regression_model.pkl')
knnmodel = joblib.load('models/knn_model.pkl')
rfcmodel = joblib.load('models/rf_model.pkl')
scaler =  joblib.load('models/scaler.pkl')

with open('models/label_encoders.pkl', 'rb') as f:
    label_encoders = pickle.load(f)

st.title("Car Recommendation Engine")


st.sidebar.markdown("# Enter Car Specifications")

def get_cluster_for_predicted_car(predicted_label, car_data):

    company_model = predicted_label
    predicted_car_row = car_data[car_data['company_model'] == company_model]

    if predicted_car_row.empty:
        raise ValueError(f"No car found in dataset matching '{predicted_label}'")

    predicted_car_cluster = predicted_car_row['Cluster'].values[0]
    return predicted_car_cluster, company_model


def recommend_similar_cars(predicted_label, car_data, top_n=5):

    predicted_cluster, company_model = get_cluster_for_predicted_car(predicted_label, car_data)

    similar_cars = car_data[car_data['Cluster'] == predicted_cluster].copy()

    feature_cols = [
        'Annual Income', 'Transmission', 'Color', 'Price ($)', 'Sale_Year',
        'Body_type', 'number_of_seats', 'wheelbase_mm', 'full_weight_kg',
        'max_trunk_capacity_l', 'injection_type', 'engine_type',
```

Figure 37 main.py cluster code

```python
def get_cluster_for_predicted_car(predicted_label, car_data):

    company_model = predicted_label
    predicted_car_row = car_data[car_data['company_model'] == company_model]

    if predicted_car_row.empty:
        raise ValueError(f"No car found in dataset matching '{predicted_label}'")

    predicted_car_cluster = predicted_car_row['Cluster'].values[0]
    return predicted_car_cluster, company_model


def recommend_similar_cars(predicted_label, car_data, top_n=5):

    predicted_cluster, company_model = get_cluster_for_predicted_car(predicted_label, car_data)

    similar_cars = car_data[car_data['Cluster'] == predicted_cluster].copy()

    feature_cols = [
        'Annual Income', 'Transmission', 'Color', 'Price ($)', 'Sale_Year',
        'Body_type', 'number_of_seats', 'wheelbase_mm', 'full_weight_kg',
        'max_trunk_capacity_l', 'injection_type', 'engine_type',
        'valves_per_cylinder', 'stroke_cycle_mm', 'turnover_of_maximum_torque_rpm',
        'engine_hp_rpm', 'drive_wheels', 'turning_circle_m', 'transmission',
        'fuel_tank_capacity_l', 'max_speed_km_per_h', 'fuel_grade',
        'back_suspension', 'rear_brakes', 'front_brakes', 'front_suspension',
        'avg_rating', 'max_rating', 'min_rating', 'popularity_score',
        'fuel_eff_score', 'avg_track_mm', 'displacement_per_cylinder',
        'power_to_weight', 'volume_mm3'
    ]

    similar_cars_scaled = similar_cars[feature_cols]
    similar_cars['score'] = similar_cars_scaled.sum(axis=1)

    recommended_cars = similar_cars.sort_values(by='score', ascending=False)
    return recommended_cars[['company_model', 'score']].drop_duplicates('company_model').head(top_n), company_model

def get_cluster_for_predicted_car_gmm(predicted_label, car_data):

    company_model = predicted_label
    predicted_car_row = car_data[car_data['company_model'] == company_model]

    if predicted_car_row.empty:
        raise ValueError(f"No car found in dataset matching '{predicted_label}'")
```

Figure 38 main.py clustering code

```python
col1, col2 = st.sidebar.columns(2)

with col1:
    price = st.slider('Price ($)', min_value=1000, max_value=100000, value=22500)
    min_rating = st.slider('Min Rating', min_value=1.0, max_value=5.0, value=1.0)
    max_trunk_capacity = st.slider('Max Trunk Capacity (L)', min_value=100, max_value=5000, value=900)
    max_torque_rpm = st.slider('Torque RPM', min_value=1000, max_value=8000, value=4400)

    num_seats = st.slider('Number of Seats', min_value=2, max_value=9, value=5)
    annual_income = st.slider('Annual Income ($)', min_value=10000, max_value=90000000, value=1025000)
    Transmission = st.selectbox('Speed Transmission', ['Auto', 'Manual'], index=0)
    Color = st.selectbox('Car Color', ['Black', 'Red', 'Pale White'], index=2)
    Series = st.selectbox('Series', ['SUV', 'Coupe', 'Hatchback 3 doors', 'Sedan', 'Axio Sedan 4-doors',
        'CN Sedan 4-doors', 'wagon', 'Crossover', 'J71 SUV 3-doors',
        'Avant wagon 5-doors', 'wagon 5 doors', 'Tourer wagon',
        'R Sedan 4-doors', 'Ute pickup', 'Cabriolet',
        'Access Cab pickup 2-doors', 'Minivan', 'Spyder cabriolet',
        'L Sedan 4-doors', 'Calais Coupe 2-doors', 'targa',
        'Pickup Single cabin Crew Cab', 'Hatchback',
        'Grand minivan 5-doors', 'Absolute minivan 5-doors',
        'wagon 5 doors SportCombi', 'GT hatchback',
        'King Cab pickup 2-doors', 'SUV 5 doors', 'Hatchback 5-doors',
        'Super Cab pickup 2-doors', 'SE Sedan', 'Grand minivan',
        'SUV 3-doors', 'GXP Sedan 4-doors', 'SUV 5-doors',
        'F-150 SVT Raptor SuperCrew pickup 4-doors', 'Sedan Long',
        'Sport Sedan', 'Sedan G4'], index=3)

    wheelbase = st.slider('Wheelbase (mm)', min_value=1000, max_value=5000, value=2765)
    full_weight = st.slider('Full Weight (kg)', min_value=800, max_value=4000, value=1950)

    injection_type = st.selectbox('Injection Type', [
        'direct injection', 'Multi-point fuel injection',
        'distributed injection (multipoint)', 'direct injection (direct)',
        'Injector', 'Common Rail'], index=2)
    engine_type = st.selectbox('Engine Type', ['Gasoline', 'petrol', 'Diesel'], index=1)
    valves_per_cylinder = st.slider('Valves per Cylinder', min_value=2, max_value=8, value=4)
    stroke_cycle = st.slider('Stroke Cycle (mm)', min_value=50, max_value=100, value=83)
    body_type = st.selectbox('Body Type', ['Crossover', 'Sedan', 'Wagon', 'Coupe', 'Pickup', 'Cabriolet',
        'Minivan', 'Hatchback', 'Targa'], index=1)
    drive_wheels = st.selectbox('Drive Wheels', [
        'Rear wheel drive', 'All wheel drive (AWD)', 'Front wheel drive',
        'full', 'Four wheel drive (4WD)'], index=2)
    sale_year = st.slider('Sale Year', min_value=2020, max_value=2025, value=2023)

with col2:
```

Figure 39 UI column declaration code

```python
st.title("Set the car specifications in left panel and choose a model below to predict the car :")
col1, col2, col3 = st.columns(3)


logistic_clicked = col1.button("Logistic Regression")
knn_clicked = col2.button("K Nearest Neighbor")
rfc_clicked = col3.button("Random Forest Classifier")

# Save prediction to session state
if logistic_clicked:
    prediction = lrmodel.predict(temp)
    st.session_state.last_model = 'logistic'
    st.session_state.last_prediction = prediction[0]

if knn_clicked:
    prediction = knnmodel.predict(temp)
    st.session_state.last_model = 'knn'
    st.session_state.last_prediction = prediction[0]

if rfc_clicked:
    prediction = rfcmodel.predict(temp)
    st.session_state.last_model = 'rfc'
    st.session_state.last_prediction = prediction[0]


if st.session_state.last_prediction is not None:

    try:
        company, model = st.session_state.last_prediction.split('_', 1)
    except ValueError:
        company, model = st.session_state.last_prediction, "Unknown"

    # Get model name formatted
    model_used = st.session_state.last_model.upper() if st.session_state.last_model else "Unknown Model"
    model_name_map = {
        'LOGISTIC': 'Logistic Regression',
        'KNN': 'K-Nearest Neighbors',
        'RFC': 'Random Forest Classifier'
    }
    model_pretty = model_name_map.get(model_used, model_used)
```

Figure 40 streamlit UI supervised code

```
#kmeans
with c1:
    if st.button("KMeans Clustering"):
        if st.session_state.last_prediction:
            model_used = st.session_state.last_model.upper()
            model_pretty = model_map.get(model_used, model_used)

            st.success(f"Running KMeans clustering based on previous model: {model_pretty}")
            try:
                recommended_top5, model_name = recommend_similar_cars(
                    st.session_state.last_prediction, car_full_analysis, top_n=num_additional_cars
                )

                try:
                    company_display, model_display = model_name.split('_', 1)
                except ValueError:
                    company_display, model_display = model_name, "Unknown"

                st.markdown(f"""
                    <div style='
                        background-color: #e3f2fd;
                        padding: 14px 24px;
                        margin-bottom: 24px;
                        border-left: 6px solid #2196f3;
                        border-radius: 10px;
                        font-size: 22px;
                        font-weight: 700;
                        color: #0d47a1;
                        text-transform: capitalize;
                    '>
                        {company_display} <span style='color:#0d47a1;'>{model_display}</span>
                    </div>
                """, unsafe_allow_html=True)

                for _, row in recommended_top5.iterrows():
                    try:
                        company, model = row['company_model'].split('_', 1)
                    except ValueError:
                        company, model = row['company_model'], "Unknown"
                    score = round(row['score'], 2)
                    st.markdown(f"""
                        <div style='background-color: #fff8e1; padding: 16px; border-radius: 10px;
                                    margin-bottom: 12px; box-shadow: 0 2px 8px rgba(0,0,0,0.05); text-transform: capitalize;'>
                            <div style='font-size: 22px; font-weight: bold; color: #4e342e;'>{company} <span style='color:#bf360c;'>{model}</span></div>
```

Figure 41 UI clustering code

The Car Recommendation Engine was developed using **Streamlit** to provide an interactive user interface that allows users to input car specifications, predict suitable car models using supervised machine learning models, and receive additional recommendations through clustering techniques.

The application workflow is structured as follows:

1. **User Input Panel**:
   The sidebar collects detailed car specifications through sliders and dropdowns, covering features like price, engine details, body type, fuel efficiency, and more. These inputs are dynamically displayed in a data frame for user verification.
2. **Data Preprocessing**:

   2.1 Categorical features are encoded using pre-trained **LabelEncoders**.

   2.2 Numerical features undergo **log transformation** and scaling using a pre-fitted **MinMaxScaler** to ensure consistency with model training conditions.

3. **Supervised Model Predictions**:
   Users can select from three models — **Logistic Regression**, **K-Nearest Neighbors (KNN)**, or **Random Forest Classifier** — to predict the most suitable car model based on the provided specifications. The predicted car is displayed with styled formatting for better user experience.
4. **Clustering Based Recommendations**:
   After prediction, users can select a clustering method (**KMeans**, **Agglomerative**, or **GMM**) to receive additional similar car recommendations. The recommendations are ranked based on a calculated similarity score derived from key features.
5. **Model Evaluation Visualizations**:

5.1 The application displays pre-generated images comparing **supervised model performance** (Accuracy, Precision, Recall, F1-score).
5.2 It also presents **clustering evaluation metrics** such as Silhouette Score, Davies-Bouldin Index, ARI, and Cluster Purity.

6. **Feature Importance Display**:
   Users can interactively view the top 10 feature importances for each supervised model, helping to interpret how different features influence predictions.
7. **Clustering Results Visualization**:
   The app allows users to toggle between clustering methods and visualize how cars are grouped, aiding in understanding the recommendation patterns.

6.3 User Interface explanation using streamlit:

To enhance user accessibility and deliver a seamless experience, an interactive user interface (UI) was developed using **Streamlit** for the Car Recommendation Engine. This UI allows users to in put their **preferences** through intuitive sliders, dropdowns, and selection buttons. This enables re al time car predictions and personalized recommendations. By using both **supervised** and **unsup ervised** machine learning models the interface helps in recommending cars based on the **count** o f cars needed as well. The UI also shows in key findings and models metrics for both supervised and unsupervised models.

The above images depict the **sidebar** in UI where in the user need to enter in specifications that u ser need recommendations for. There are multiple parameters like price, rating, number of seats, popularity score, max speed and weight of the car.

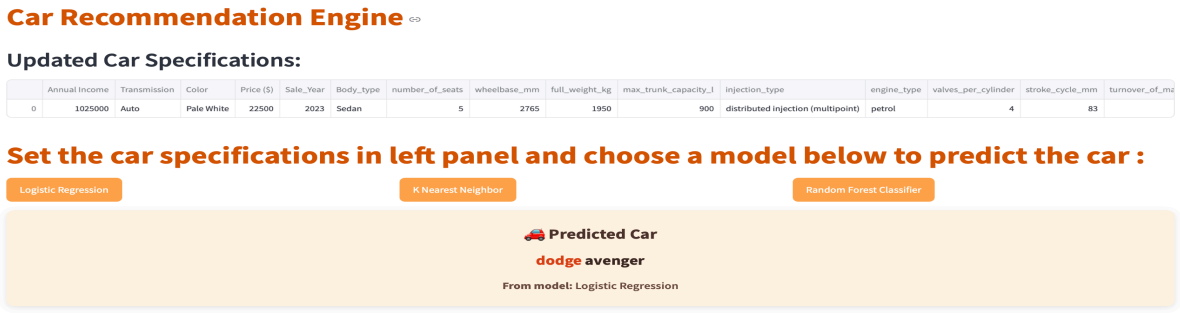


Figure 42 Supervised model prediction UI

Next, the user can see the updated selections as a table. The user now have an option to get the ca r using either (**Logistic Regression or KNN or random forest classifier**). The model predict(re commendation) will be given car company name and model. In this case Logistic regression pre dicted "**Dodge Avenger**".

Figure 43 Select number of cars UI

If the user wants more recommendations, the number of additional cars can be set. Then the user will have options to use 3 clustering methods to get the additional recommendations.

Figure 44 Unsupervised output UI

The clustering recommendations will be shown along with similarity score based on each clusteri ng method selected.

The Key findings in the models are added in the UI:

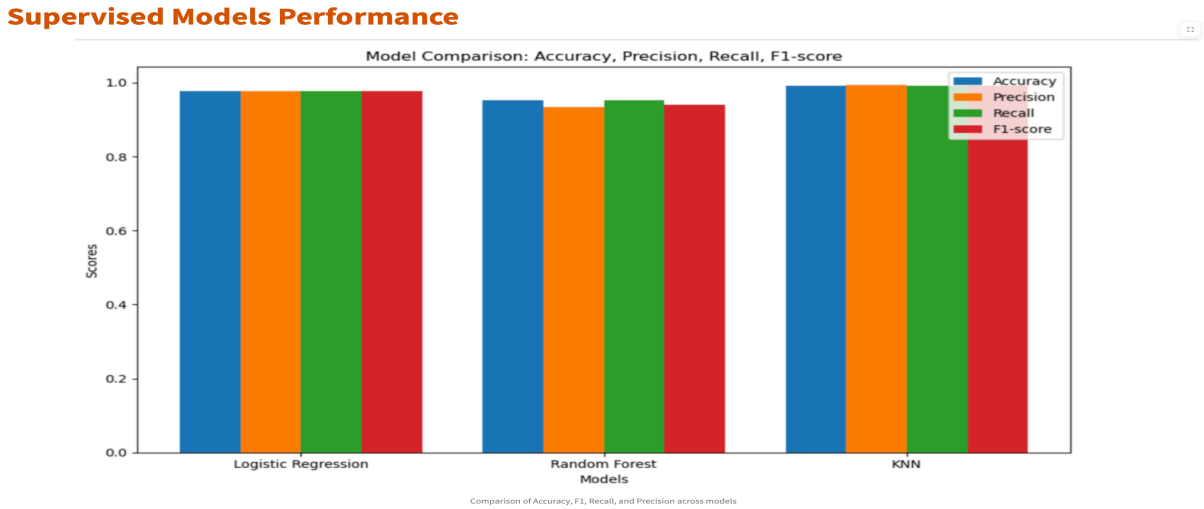


Figure 45 Supervised model metrics UI

The image displays supervised model performances.

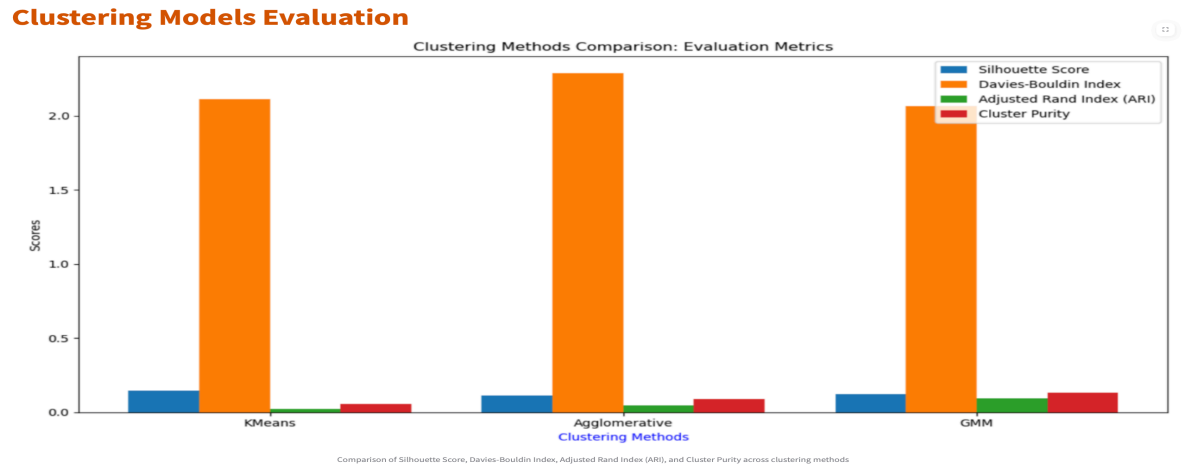


Figure 46 Unsupervised model metrics UI

The clustering models performance metrics are added as per the above image.

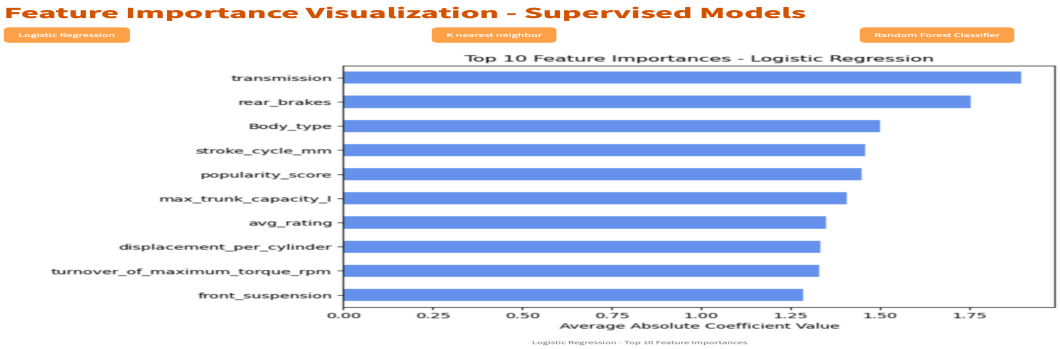


Figure 47 Feature importance supervised on UI

The feature importance for each supervised learning method is show cased. For example the imag e above shows the feature importance for logistic regression.

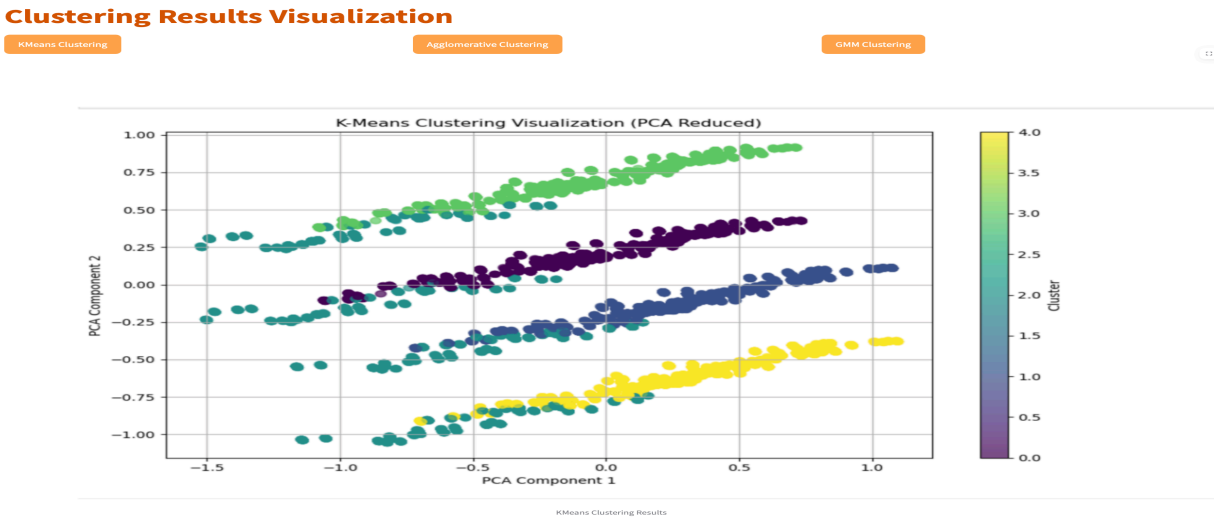


Figure 48 Clustering metrics on UI

The UI also allows the feature to visualize the clustering distribution for each cluster.

7. Conclusion:

From milestone 2, the models are enhanced by tuning. Based on the analysis of various paramete rs as discussed after tuning **K-Nearest Neighbor** performed the best and for the additional recom mendation clustering is implemented (“ K-means Clustering, Agglomerative clustering and Gaus sian mixture model”) is implemented and **GMM clustering and Kmeans clustering** turned out t o be better than other clustering methods. Overall, hybrid analysis (supervised and unsupervised models) is used to develop **“Car Recommendation engine**” .Then using the predicts from the m odels are analyzed, bias and limitations are identified using **feature importance** and **classificatio n report** for supervised and **cluster analysis** for unsupervised. These methods have drawn good f indings on the **predictions** and **User interface** has been implemented using **streamlit** wherein th e user can get real time car predictions and model metrics , feature importance and clustering ima ges are visualized.