Milestone 2 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**.

Some key insights from Milestone 1: (EDA RECAP)

1.2.1. Car_ specs Dataframe: (Represents the specifications of various cars):

a. The image below shows the descriptive statistics:

```
[283]: print(car_specs.describe())
```

```
            id_trim     Year_from       Year_to    payload_kg  \
count  70823.000000  70586.00000  70189.000000  23799.000000
mean   35477.818788   1997.06524   1913.441978    605.546199
std    20494.213522     14.99201    415.392957    320.441908
min        1.000000   1904.00000      0.000000    145.000000
25%    17724.500000   1988.00000   1994.000000    465.000000
50%    35453.000000   2000.00000   2005.000000    530.000000
75%    53240.500000   2008.00000   2013.000000    615.000000
max    70987.000000   2020.00000   2020.000000   3334.000000

       back_track_width_mm  front_track_width_mm  full_weight_kg  \
count         11198.000000          11204.000000    39682.000000
mean           1477.303179           1482.623527     2067.129127
std              96.061918             92.701493      619.628212
min            1050.000000           1105.000000      690.000000
25%            1425.000000           1430.000000     1680.000000
50%            1475.000000           1481.000000     1950.000000
75%            1542.000000           1542.000000     2270.000000
max            1869.000000           1869.000000     5352.000000

       minimum_trunk_capacity_l  number_of_cylinders  valves_per_cylinder  \
count              45953.000000         59544.000000         59334.000000
mean                 473.017170             4.972827             3.254138
std                  316.859243             1.585268             0.975289
min                   11.000000             1.000000             1.000000
25%                  338.000000             4.000000             2.000000
50%                  436.000000             4.000000             4.000000
75%                  515.000000             6.000000             4.000000
max                 4440.000000            16.000000             6.000000

       ...     engine_hp  number_of_gears  turning_circle_m  \
count  ...  59877.000000     58292.000000     40708.000000
mean   ...    166.659853         4.999074        11.255127
std    ...     93.269952         1.220068         1.306523
min    ...      5.000000         1.000000         5.100000
25%    ...    105.000000         4.000000        10.500000
50%    ...    141.000000         5.000000        11.000000
75%    ...    203.000000         6.000000        11.800000
max    ...   1914.000000        10.000000        71.000000
```

Figure 1car specs descriptive features

b. The image below shows the unique models for each car brand which is **used in Milestone 2** feature selection which will be represented further in the report:

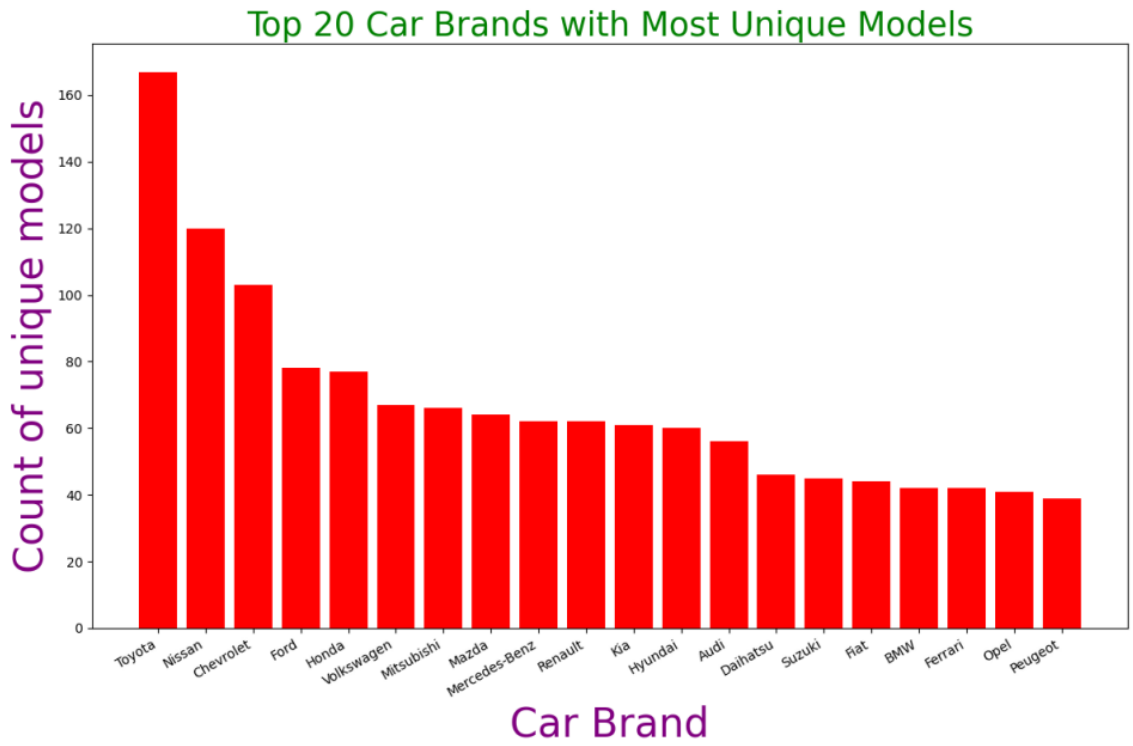


Figure 2 Bar chart for car count vs brand

c. Correlation matrix and analysis: the below insight from milestone1 is used for feature engineering and feature selection which is discussed in detail further in the report.

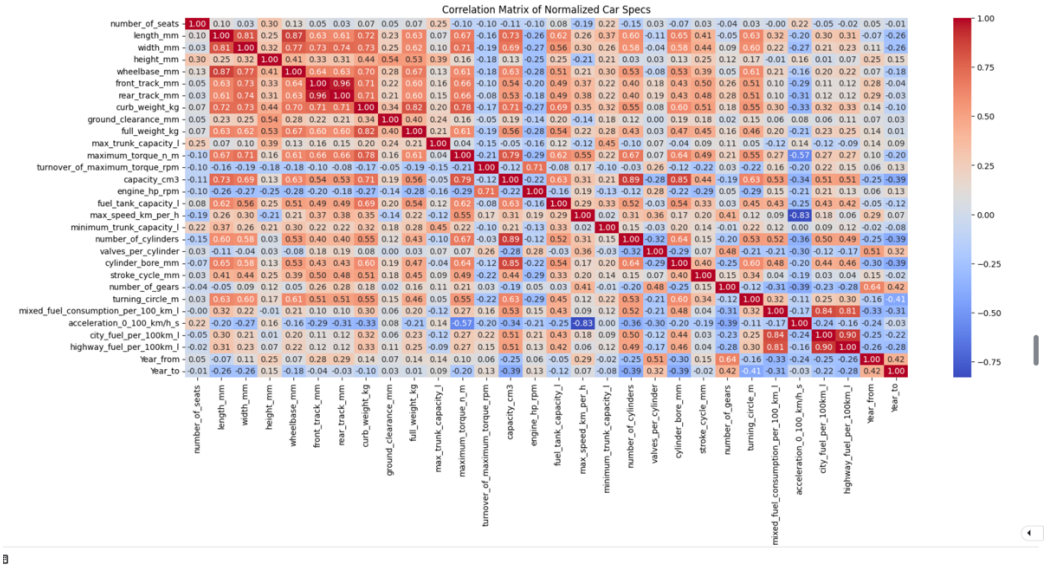


Figure 3 Correlation matrix for car_specs dataframe

Conclusion from the heatmap:

High Positive Correlations:

1. length_mm and width_mm **(corr = 0.81)**
2. length_mm and wheelbase_mm **(corr = 0.87)**
3. width_mm and length_mm **(corr = 0.81)**
4. wheelbase_mm and length_mm **(corr = 0.87)**
5. front_track_mm and rear_track_mm **(corr = 0.96)**
6. rear_track_mm and front_track_mm **(corr = 0.96)**
7. curb_weight_kg and full_weight_kg **(corr = 0.82)**
8. full_weight_kg and curb_weight_kg **(corr = 0.82)**
9. capacity_cm3 and number_of_cylinders **(corr = 0.89)**
10. capacity_cm3 and cylinder_bore_mm **(corr = 0.85)**
11. mixed_fuel_consumption_per_100_km_l and city_fuel_per_100km_l **(corr = 0.84)**
12. mixed_fuel_consumption_per_100_km_l and highway_fuel_per_100km_l **(corr = 0.81)**
13. city_fuel_per_100km_l and mixed_fuel_consumption_per_100_km_l **(corr = 0.84)**
14. city_fuel_per_100km_l and highway_fuel_per_100km_l **(corr = 0.90)**
15. highway_fuel_per_100km_l and mixed_fuel_consumption_per_100_km_l **(corr = 0.81)**
16. highway_fuel_per_100km_l and city_fuel_per_100km_l **(corr = 0.90)**

High Negative Correlations:

1. max_speed_km_per_h and acceleration_0_100_km/h_s **(corr = -0.83)**
2. acceleration_0_100_km/h_s and max_speed_km_per_h **(corr = -0.83)**

1.2.2 Car_ sales Dataframe: (Represents the sales data of cars):

a. The image below shows the descriptive statistics:

```
[310]: print(car_sales.describe())
```

	Annual Income	Price ($)	Phone
count	2.390600e+04	23906.000000	2.390600e+04
mean	8.308403e+05	28090.247846	7.497741e+06
std	7.200064e+05	14788.687608	8.674920e+05
min	1.008000e+04	1200.000000	6.000101e+06
25%	3.860000e+05	18001.000000	6.746495e+06
50%	7.350000e+05	23000.000000	7.496198e+06
75%	1.175750e+06	34000.000000	8.248146e+06
max	1.120000e+07	85800.000000	8.999579e+06

Figure 4 Car sales dataframe descriptive analysis

b. The below pic shows the bar graph for number of sales for each car company:

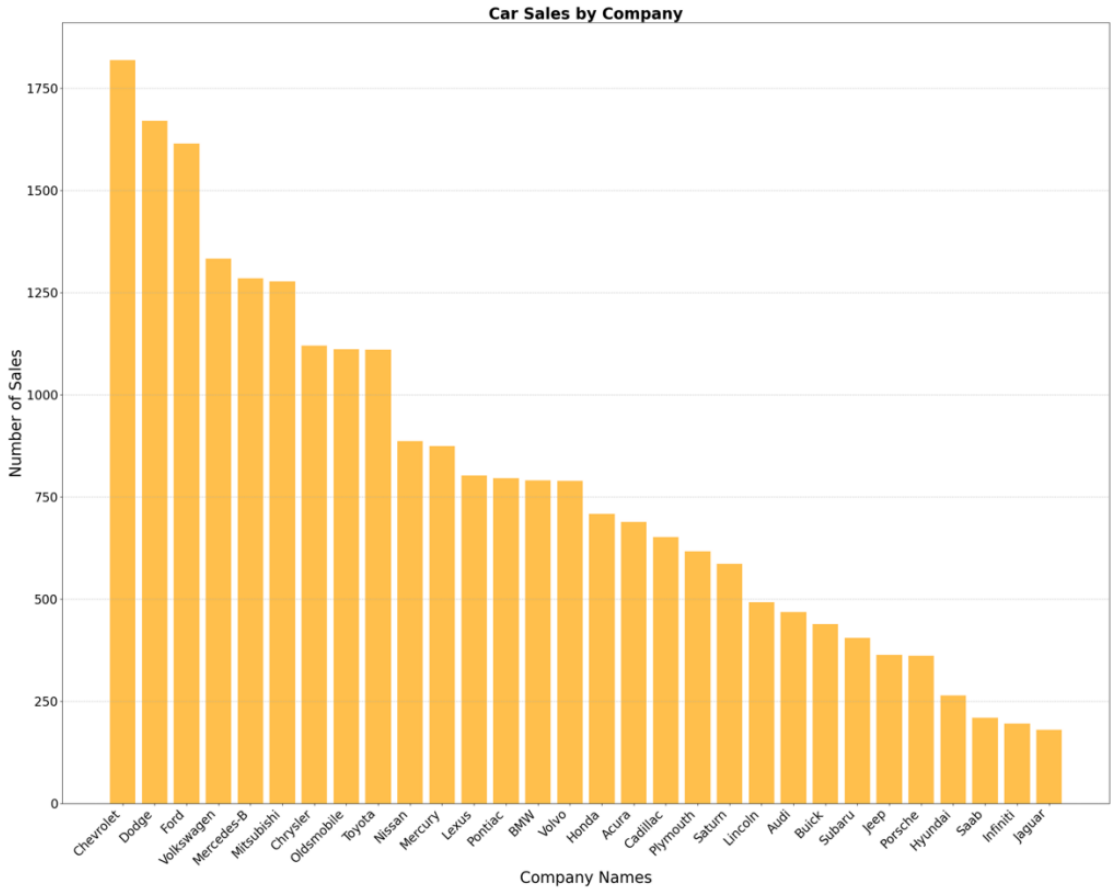
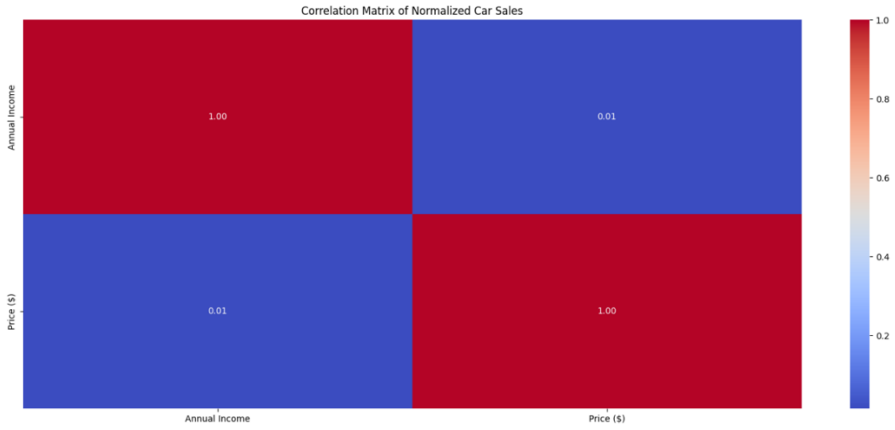


Figure 5 Bar graph of no of sales vs company

c. the below is correlation matrix of price and annual income of people who purchased cars. This insight is used in **feature engineering and selection in milestone 2.**



The Price and Annual income has a very weak correlation between them.

Figure 6 Correlation matrix for car_sales dataframe

1.2.3 Car_ ratings Dataframe: (Represents the ratings data of cars):

a. The below images shows descriptive statistics of car_ratings daraframe:

```
[333]: print(car_ratings.describe())
```

	Year	Rating
count	299045.000000	299045.000000
mean	2007.492247	3.980886
std	5.330847	0.993001
min	2000.000000	0.000000
25%	2003.000000	4.000000
50%	2006.000000	4.000000
75%	2011.000000	5.000000
max	2020.000000	5.000000

Figure 7 Car ratings descriptive analysis

b. The below graph indicates average rating of each car company:

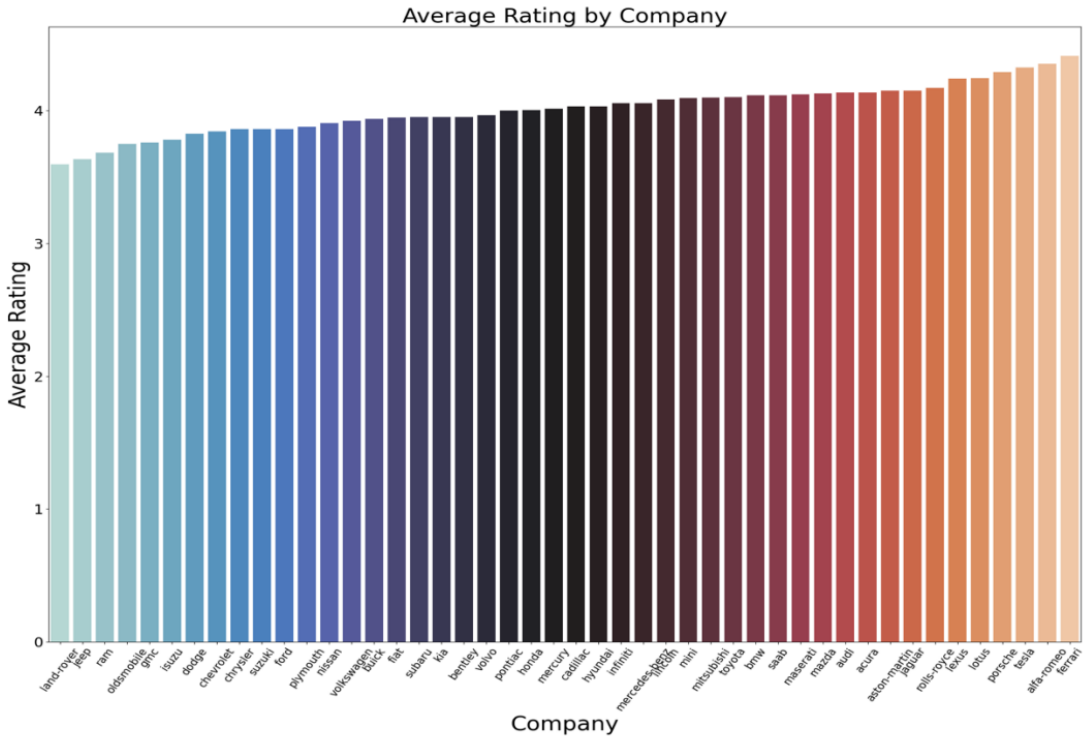
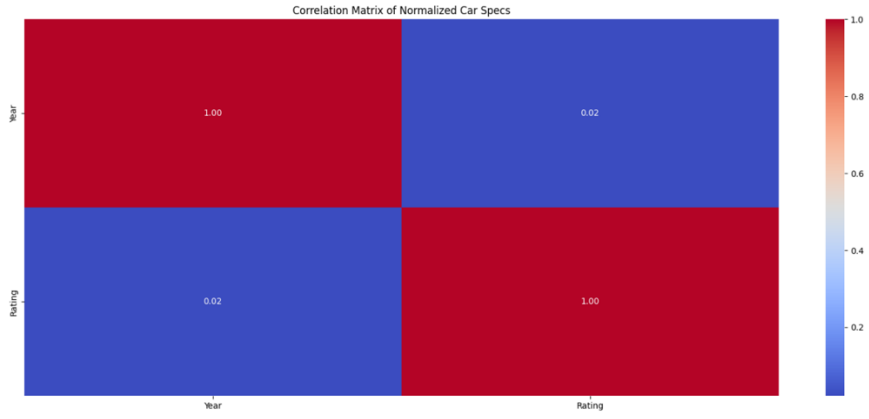


Figure 8 Average rating vs company bar graph

c. the below is correlation matrix of car ratings comparing year vs rating. This insight is used in **feature engineering and selection in milestone 2.**



The Year vs Rating is a very weak correlation between them based on the heatmap.

Figure 9 Correlation matrix for car ratings dataframe

As the report progresses the above insights from the EDA of milestone 1 will be discussed in detail as in how it is used in milestone 2.

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.

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. Exporting Cleaned DataFrames and Merging for Analysis:

The dataframes (car_specs, car_sales and car_ratings) that were cleaned and analyzed for EDA are exported and used for milestone 2.

The links of exported dataframes in the form of csv are :

a. car_specs
b. car_sales
c. car_ratings

3.1 export dataframes and import for the milestone:

```
[3]: # Milestone 2

[5]: import pandas as pd
     import matplotlib
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns

[7]: #Importing csvs saved from milestone 1

[9]: car_specs = pd.read_csv('car_specs.csv')

[11]: car_sales = pd.read_csv('car_sales.csv')

[13]: car_ratings = pd.read_csv('car_ratings.csv')
```

Figure 10 Code to import csvs from milestone 1

The data from the **three** dataframes is merged based on logic.

3.2 Merge car_specs and car_sales dataframes:

First car_specs and car_sales dataframes are merged:

Logic :

```
[15]: # standardize the columns in car_specs and car_sales for merge
      for i in [car_specs,car_sales]:
          i["Model"]=i["Model"].str.lower().str.strip()
          i["Company"] = i["Company"].str.lower().str.strip()
```

Figure 11 standardize columns for car_specs and car_sales

```
# New column added to get year
car_sales["Sale_Year"] = car_sales["Date"].dt.year
```

Figure 12 New feature sale year added for merge

First the columns are standardized and then the "sale_year" feature is added to get the year as car_specs have year_from and year_to features to compare. Then the following method is used for merging the two dataframes.

```
5]: # merge car_sales and car_specs

rowsmerged = []

for idx, sale in car_sales.iterrows():
    sale_company = sale["Company"]
    sale_model = sale["Model"]
    sale_year = sale["Sale_Year"]

    if pd.isna(sale_year):
        continue

    # Get specs for this company and model
    specsgroup = car_specs[
        (car_specs["Company"] == sale_company) &
        (car_specs["Model"] == sale_model)
    ]

    if specsgroup.empty:
        continue

    # Check for active generation
    activespecs = specsgroup[
        (specsgroup["Year_from"] <= sale_year) &
        (specsgroup["Year_to"] >= sale_year)
    ]

    if not activespecs.empty:
        selectedspec = active_specs.iloc[0]  # take the first active one
    else:
        # Fallback to closest Year_to
        specsgroup = specsgroup.copy()
        specsgroup["Year_to_diff"] = (specsgroup["Year_to"] - sale_year).abs()
        selectedspec = specsgroup.sort_values("Year_to_diff").iloc[0]

    # Drop any duplicate columns
    sale_cleaned = sale.drop(labels=selectedspec.index.intersection(sale.index), errors='ignore')
    merged_row = pd.concat([sale_cleaned, selectedspec])
    rowsmerged.append(merged_row)

# Build the merged DataFrame
merged_df = pd.DataFrame(rowsmerged)

print("Merged shape:", merged_df.shape)
print(merged_df[["Company", "Model", "Date", "Sale_Year", "Year_from", "Year_to"]].head())
```

Figure 13 Code to merge car_specs and car_sales

The car_specs and car_sales direct merge was not possible due to generation being involved in car_specs.

Merge Strategy:

1. If Company and Model features are matching then based on the sale_year of the car is checked if it falls in the range of year_from and year_to. If yes, the record is chosen for merge.

2. If the sale_year doesn't fall in the range, then the nearest year_to is chosen as a fallback.
3. The cleaned sales record and the selected specification were combined row-wise and stored in a list.

3.3 Merge resultant dataframe and car_ratings:

```
[18950 rows x 65 columns]
```

```
[20]: # standardize car_ratings dataframe
car_ratings["Company"] = car_ratings["Company"].str.lower().str.strip()
car_ratings["Model"] = car_ratings["Model"].str.lower().str.strip()
```

Figure 14 Standardize columns for merge

First Company and Model columns are standardized for merge as above.

```
[45]: #Create new aggregate features initially by grouping model and company on car_ratings dataframe
ratings_agg = car_ratings.groupby(["Company", "Model"]).agg(
    avg_rating=("Rating", "mean"),
    max_rating=("Rating", "max"),
    min_rating=("Rating", "min"),
    num_reviews=("Rating", "count")
).reset_index()

# also have a set for company alone
ratings_agg_company = car_ratings.groupby("Company").agg(
    comp_avg_rating=("Rating", "mean"),
    comp_max_rating=("Rating", "max"),
    comp_min_rating=("Rating", "min"),
    comp_num_reviews=("Rating", "count")
).reset_index()
```

```
[47]: # Now merging the 3rd dataframe car_ratings

# First do merge on company and model
merged_full = pd.merge(
    merged_df,
    ratings_agg,
    on=["Company", "Model"],
    how="left"
)

# Also maintain data for company level as well
merged_full = pd.merge(
    merged_full,
    ratings_agg_company,
    on="Company",
    how="left"
)
```

```
[49]: # Incase the data is not present for company and model combo impute by company ratings

merged_full["avg_rating"] = merged_full["avg_rating"].fillna(merged_full["comp_avg_rating"])
merged_full["max_rating"] = merged_full["max_rating"].fillna(merged_full["comp_max_rating"])
merged_full["min_rating"] = merged_full["min_rating"].fillna(merged_full["comp_min_rating"])
merged_full["num_reviews"] = merged_full["num_reviews"].fillna(merged_full["comp_num_reviews"])

# drop the company-level columns if no longer needed
merged_full.drop(columns=["comp_avg_rating", "comp_max_rating", "comp_min_rating", "comp_num_reviews"], inplace=True)
```

Figure 15 New features are created on ratings(aggregations) and merge code

The new aggregate features are created **"avg_rating", "max_rating", "min_rating" and "num_reviews"** by grouping (Company, Model). Similarly comp(company level aggregate columns are also created in case of fallback.

Then the merge is done and in case of any null matches for (Company , Model) the comp aggregates are used .

The merged dataframe is checked for any null values and the data is cleaned.

```
[18950 rows x 70 columns]

[29]: print(car_full.isnull().sum())

Car_id              0
Date                0
Customer Name       0
Gender              0
Annual Income       0
                   ...
avg_rating        586
max_rating        586
min_rating        586
num_reviews       586
has_reviews         0
Length: 70, dtype: int64

[30]: car_full = car_full[car_full['has_reviews'] != 0].reset_index(drop=True)
```

Figure 16 Null values handled after merge

Finally the fully merged dataframe is ready. The link to the dataset fully merged is [Fully_Merged_Dataset]

4. Feature Engineering and Encoding:

4.1 Correlation analysis from Milestone 1:

As part of the project new features are introduced. These features can be classified as “Interaction terms”, “aggregations” and “time- based features”.
Looking at the correlation analysis for “car_specs”, “car_sales” and “car_ratings” from “**IDS Milestone-1” EDA analysis (EDA RECAP)**

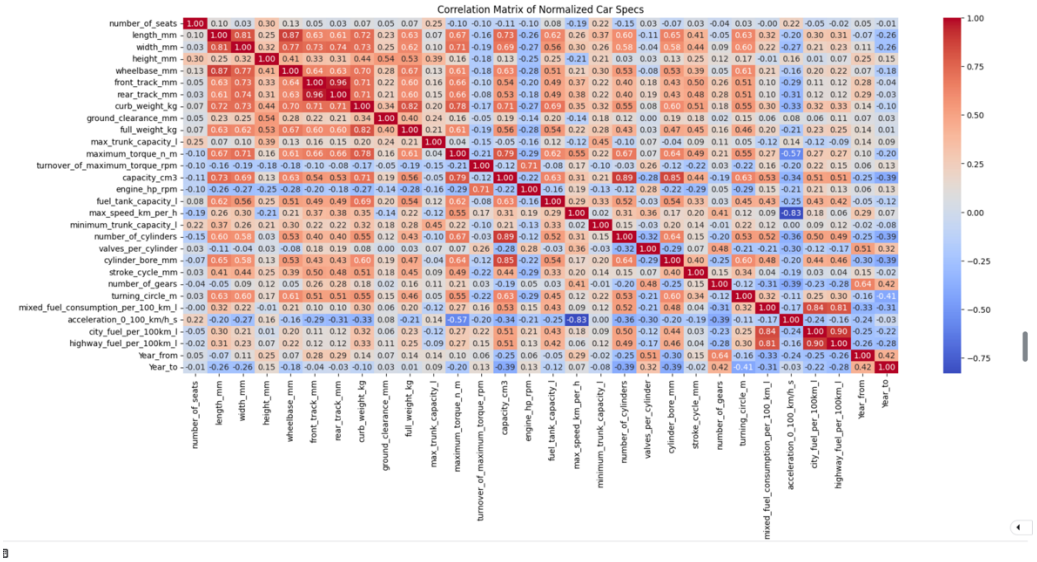


Figure 17 Correlation matrix from milestone1 car_specs

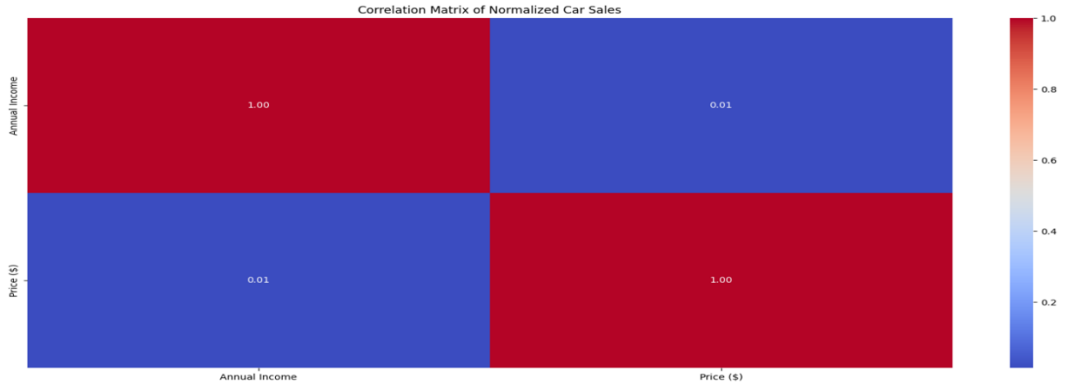
Conclusion from the heatmap:

High Positive Correlations:

1. length_mm and width_mm **(corr = 0.81)**
2. length_mm and wheelbase_mm **(corr = 0.87)**
3. width_mm and length_mm **(corr = 0.81)**
4. wheelbase_mm and length_mm **(corr = 0.87)**
5. front_track_mm and rear_track_mm **(corr = 0.96)**
6. rear_track_mm and front_track_mm **(corr = 0.96)**
7. curb_weight_kg and full_weight_kg **(corr = 0.82)**
8. full_weight_kg and curb_weight_kg **(corr = 0.82)**
9. capacity_cm3 and number_of_cylinders **(corr = 0.89)**
10. capacity_cm3 and cylinder_bore_mm **(corr = 0.85)**
11. mixed_fuel_consumption_per_100_km_l and city_fuel_per_100km_l **(corr = 0.84)**
12. mixed_fuel_consumption_per_100_km_l and highway_fuel_per_100km_l **(corr = 0.81)**
13. city_fuel_per_100km_l and mixed_fuel_consumption_per_100_km_l **(corr = 0.84)**
14. city_fuel_per_100km_l and highway_fuel_per_100km_l **(corr = 0.90)**
15. highway_fuel_per_100km_l and mixed_fuel_consumption_per_100_km_l **(corr = 0.81)**
16. highway_fuel_per_100km_l and city_fuel_per_100km_l **(corr = 0.90)**

High Negative Correlations:

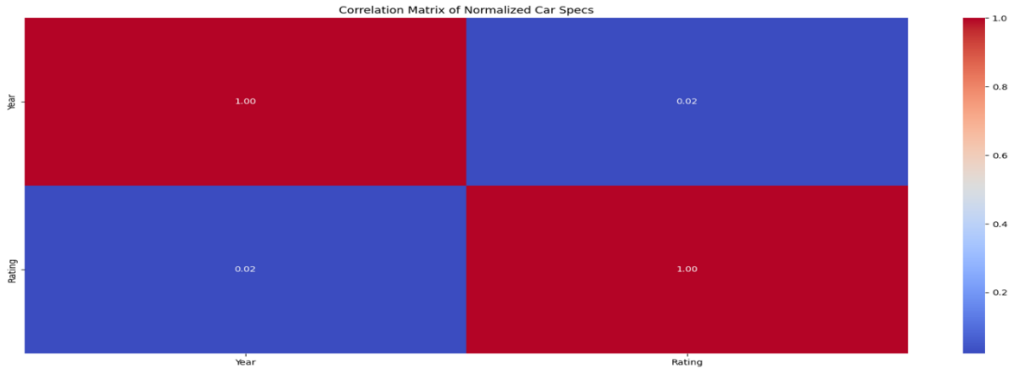
1. max_speed_km_per_h and acceleration_0_100_km/h_s **(corr = -0.83)**
2. acceleration_0_100_km/h_s and max_speed_km_per_h **(corr = -0.83)**



The Price and Annual income has a very weak correlation between them.

Figure 18 Correlation matrix from milestone 1 for car_sales

The above image shows the correlation for car_sales features.



The Year vs Rating is a very weak correlation between them based on the heatmap.

Figure 19 Correlation matrix from milestone1 for car ratings

The above image shows correlation for car_ratings features.

4.2 Correlation analysis from Milestone 2:

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

# Drop identifier columns
df_corr = car_full_analysis.drop(columns=exclude_cols)
corr_matrix = df_corr.corr().fillna(0)

# Plot
plt.figure(figsize=(14, 12))
sns.heatmap(corr_matrix, cmap="coolwarm", annot=False, fmt=".2f")
plt.title("Correlation Matrix (Log Transformed + MinMax Scaled Data)")
plt.show()
```

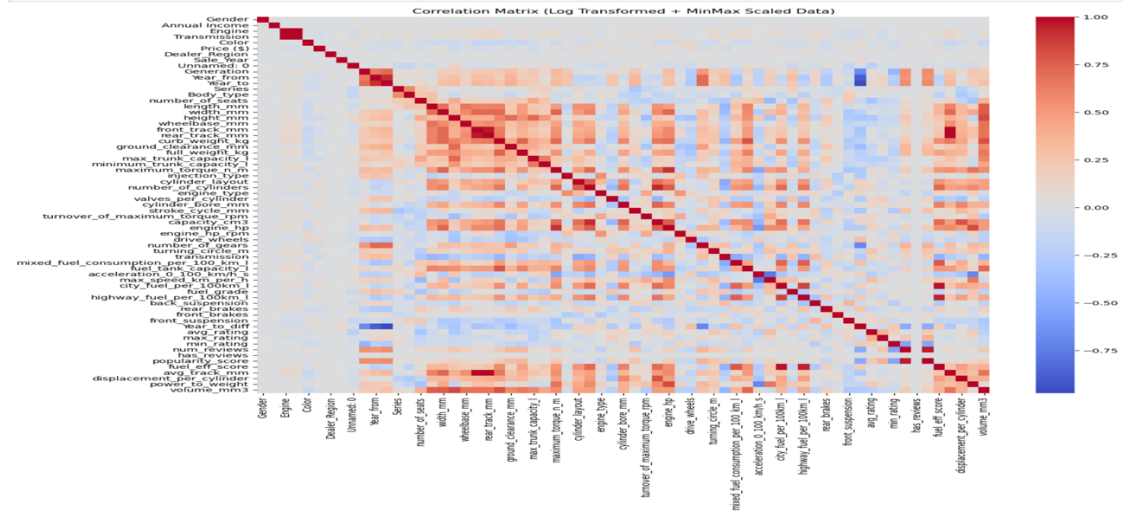


Figure 20 Correlation matrix of fully merged data

Correlation matrix analysis :

```
[96]: # Set correlation threshold
threshold = 0.8

# Get the correlation matrix
corr_matrix = car_full_analysis.corr(numeric_only=True)

# Find highly correlated pairs
high_corr = []

for i in range(len(corr_matrix.columns)):
    for j in range(i + 1, len(corr_matrix.columns)):
        corr_value = corr_matrix.iloc[i, j]
        if abs(corr_value) > threshold:
            high_corr.append({
                "Feature 1": corr_matrix.columns[i],
                "Correlation": corr_value,
                "Feature 2": corr_matrix.columns[j]

            })

# Convert to DataFrame
high_corr_df = pd.DataFrame(high_corr).sort_values(by="Correlation", ascending=False)

# Results
print(f"\nHighly Correlated Pairs (|correlation| > {threshold}):")
print(high_corr_df)
```

```
Highly Correlated Pairs (|correlation| > 0.8):
                  Feature 1  Correlation               Feature 2
0                    Engine     1.000000            Transmission
7             rear_track_mm     0.991648            avg_track_mm
6             front_track_mm     0.990118            avg_track_mm
13    city_fuel_per_100km_l     0.976215           fuel_eff_score
5             front_track_mm     0.963764            rear_track_mm
15             num_reviews     0.956753         popularity_score
14  highway_fuel_per_100km_l     0.944854           fuel_eff_score
10      number_of_cylinders     0.926164            capacity_cm3
1                 Year_from     0.915151                 Year_to
9         maximum_torque_n_m     0.900108               engine_hp
12    city_fuel_per_100km_l     0.851691  highway_fuel_per_100km_l
4                 height_mm     0.824555             volume_mm3
11                engine_hp     0.823648         power_to_weight
8           curb_weight_kg     0.801643             volume_mm3
2                 Year_from    -0.881953            Year_to_diff
3                   Year_to    -0.973877            Year_to_diff
```

Based on these parameters and additional required parameters the “Feature Engineering” is done.

4.3 Feature Engineering:

To enhance the performance and interpretability of the machine learning models, new features were engineered based on domain knowledge and correlation analysis. Below is a description of the newly introduced features:

(FROM Milestone 1)

Conclusion from the heatmap:

High Positive Correlations:

1. length_mm and width_mm **(corr = 0.81)**
2. length_mm and wheelbase_mm **(corr = 0.87)**
3. width_mm and length_mm **(corr = 0.81)**
4. wheelbase_mm and length_mm **(corr = 0.87)**
5. front_track_mm and rear_track_mm **(corr = 0.96)**
6. rear_track_mm and front_track_mm **(corr = 0.96)**
7. curb_weight_kg and full_weight_kg **(corr = 0.82)**
8. full_weight_kg and curb_weight_kg **(corr = 0.82)**
9. capacity_cm3 and number_of_cylinders **(corr = 0.89)**
10. capacity_cm3 and cylinder_bore_mm **(corr = 0.85)**
11. mixed_fuel_consumption_per_100_km_l and city_fuel_per_100km_l **(corr = 0.84)**
12. mixed_fuel_consumption_per_100_km_l and highway_fuel_per_100km_l **(corr = 0.81)**
13. city_fuel_per_100km_l and mixed_fuel_consumption_per_100_km_l **(corr = 0.84)**
14. city_fuel_per_100km_l and highway_fuel_per_100km_l **(corr = 0.90)**
15. highway_fuel_per_100km_l and mixed_fuel_consumption_per_100_km_l **(corr = 0.81)**
16. highway_fuel_per_100km_l and city_fuel_per_100km_l **(corr = 0.90)**

High Negative Correlations:

1. max_speed_km_per_h and acceleration_0_100_km/h_s **(corr = -0.83)**
2. acceleration_0_100_km/h_s and max_speed_km_per_h **(corr = -0.83)**

New Features added are :

1. **Volume (**volume_mm3): This feature is calculated using the product of length_mm, width_mm and height_mm. These three features are highly **correlated((Milestone1** car_specs), and combining them into a single volumetric measure helps reduce multicollinearity while preserving the physical dimensions of the vehicle.

```
car_full["volume_mm3"] = (
    car_full["length_mm"] *
    car_full["width_mm"] *
    car_full["height_mm"]
)
```

Figure 21 Code to create volume feature

2. **Fuel Efficiency Score** (fuel_eff_score) : (("city_fuel_per_100km_l")+ (highway_fuel_per_100km_l"))/2 has been taken to
   - It serves as a comprehensive indicator of overall fuel efficiency
   - These two original features are highly correlated, and this transformation provides a more effective representation for modeling.

```
car_full["fuel_eff_score"] = (car_full["city_fuel_per_100km_l"].astype(float) +
                              car_full["highway_fuel_per_100km_l"].astype(float)) / 2
```

Figure 22 Code to create fuel efficiecy score

3. **Average track mm** (avg_track_mm) : (("front_track_mm") + ("rear_track_mm"))/2 :This feature is calculated using the average of front_track_mm, rear_track_mm. These two features are highly **correlated ((Milestone1** car_specs), and combining them into a single measure helps reduce multicollinearity while preserving the physical dimensions of the vehicle.

```
car_full["avg_track_mm"] = (car_full["front_track_mm"] + car_full["rear_track_mm"]) / 2
```

Figure 23 Code to create average track mm

4. **Displacement per cylinder** (displacement_per_cylinder)= ("capacity_cm3" / "number_of_cylinders") is added and this feature tells how much engine displacement (volume) is available **per cylinder**. These two features are highly **correlated((Milestone1** car_specs), and combining them into a single measure helps reduce multicollinearity while preserving the physical dimensions of the vehicle.

```
car_full["displacement_per_cylinder"] = (car_full["capacity_cm3"])/car_full["number_of_cylinders"]
```

Figure 24 Code to create displacement per cylinder

5. **Popularity score (popularity_score) :** avg.rating * np.log1p("num_reviews") is taken to get a best feature based on ratings and number of reviews. This helps in having better parameter to leverage the ratings and reviews.

```
car_full["popularity_score"] = car_full["avg_rating"] * np.log1p(car_full["num_reviews"])
```

Figure 25 Code to create popularity score

6. **Sale Year** (Sale_year) : This feature is created to help in the merging of dataframes car_specs and car_sales logically. This is very important to match the specs of the sold car based on the **year**. The raw data is in the form of date.

```
# New column added to get year
car_sales["Sale_Year"] = car_sales["Date"].dt.year
```

Figure 26 Code to create sale year

7. **avg_rating, max_rating, min_rating and num_reviews** : These features were created for the merge of car_ratings dataframe with the other two. This helps in getting the important columns from the car_ratings dataframe and the data is **preserved.**

```
# Incase the data is not present for company and model combo impute by company ratings

merged_full["avg_rating"] = merged_full["avg_rating"].fillna(merged_full["comp_avg_rating"])
merged_full["max_rating"] = merged_full["max_rating"].fillna(merged_full["comp_max_rating"])
merged_full["min_rating"] = merged_full["min_rating"].fillna(merged_full["comp_min_rating"])
merged_full["num_reviews"] = merged_full["num_reviews"].fillna(merged_full["comp_num_reviews"])

# drop the company-level columns if no longer needed
merged_full.drop(columns=["comp_avg_rating", "comp_max_rating", "comp_min_rating", "comp_num_reviews"], inplace=True)
```

Figure 27 Code to create aggregate features

8. **Company and model (company_model) :** This column is created to merge the company and model and form as feature which will be used as a **target** variable.

```
# Create a joint target variable
car_full["company_model"] = car_full["Company"] + "_" + car_full["Model"]
```

Figure 28 Code to create company_model target variable

9. Power to weight (power_to_weight) : This parameter is taken on the domain knowledge that individually the power(engine_hp) and weight(curb_weight_kg) donot have meaningful contricbution. Hence these features are added to make model better.

```
car_full["power_to_weight"] = car_full["engine_hp"] / car_full["curb_weight_kg"]
```

Figure 29 Code to create power to weight feature

4.4 Encoding for categorical variables:

The unique values for the categorical data is analyzed first to decide on the encoding.

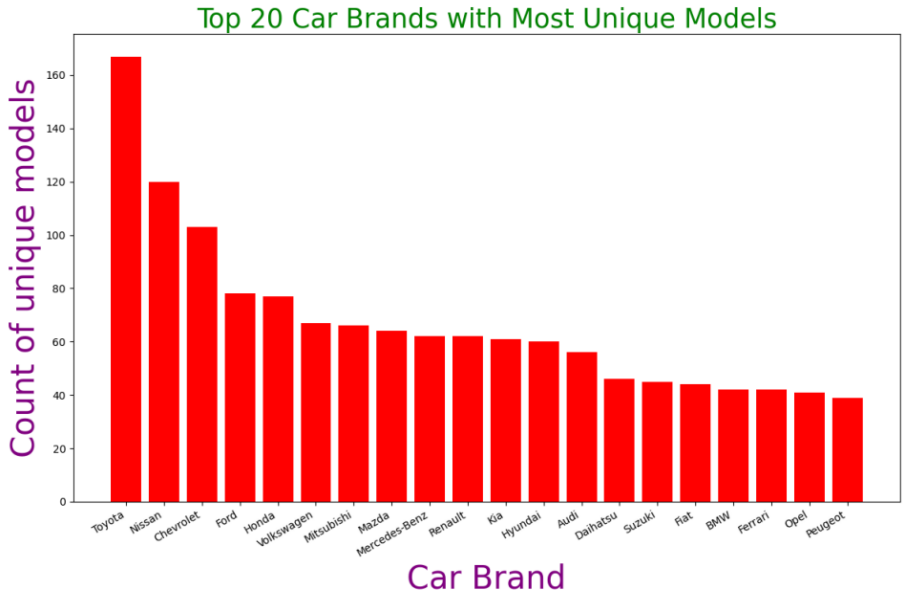


Figure 30 Bar graph from milestone 1 as reference

The above image is from **Milestone 1** EDA

```
# Analyzing unique values to decide the encoding type for categorical data
categorical_cols = car_full_analysis.select_dtypes(include=['object']).columns.tolist()
for col in categorical_cols:
    unique_count = car_full_analysis[col].nunique()
    print(f"{col}: {unique_count} unique values")
```

```
Gender: 2 unique values
Engine: 2 unique values
Transmission: 2 unique values
Color: 3 unique values
Dealer_Region: 7 unique values
Generation: 41 unique values
Series: 40 unique values
Body_type: 9 unique values
injection_type: 6 unique values
cylinder_layout: 4 unique values
engine_type: 3 unique values
drive_wheels: 5 unique values
transmission: 4 unique values
back_suspension: 43 unique values
rear_brakes: 3 unique values
front_brakes: 3 unique values
front_suspension: 29 unique values
company_model: 120 unique values
```

Figure 31 Code to analyze unique vaues for categorical features

In the above analysis as the unique values in some columns(**company_model, Generation and back_suspension**) are very high "**Label_Encoding**" has been chosen to encode the data.

```
from sklearn.preprocessing import LabelEncoder

# Take categorical columns and exclude company_model target columns
categorical_cols = car_full_analysis.select_dtypes(include=['object']).columns.tolist()
exclude_cols = ["company_model"]
categorical_cols = [col for col in categorical_cols if col not in exclude_cols]

# Label encoding
le = LabelEncoder()
for col in categorical_cols:
    car_full_analysis[col] = le.fit_transform(car_full_analysis[col].astype(str))

print("Encoded Categorical Columns:", categorical_cols)
```

```
Encoded Categorical Columns: ['Gender', 'Engine', 'Transmission', 'Color', 'Dealer_Region', 'Generation', 'Series', 'Body_type', 'injection_t
ype', 'cylinder_layout', 'engine_type', 'drive_wheels', 'transmission', 'back_suspension', 'rear_brakes', 'front_brakes', 'front_suspension']
```

Figure 32 Code to do label encoding

In the above image,the label encoding logic is implemented and the object type columns are encoded.

5. Feature Selection:

After the new features have been added to the fully merged and encoded dataframe, the methods are implemented to decide whether the features can be dropped based on the similarity and correlation. This helps in improving model more robust.

Methods used :

1. Correlation anlysis
2. Chi Square test
3. PCA Analysis

5.1 Correlation Analysis :

The correlation has been drawn with a threshold=0.8 and the pairs with highest correlation are drawn

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

# Drop identifier columns
df_corr = car_full_analysis.drop(columns=exclude_cols)
corr_matrix = df_corr.corr().fillna(0)

# Plot
plt.figure(figsize=(14, 12))
sns.heatmap(corr_matrix, cmap="coolwarm", annot=False, fmt=".2f")
plt.title("Correlation Matrix (Log Transformed + MinMax Scaled Data)")
plt.show()
```

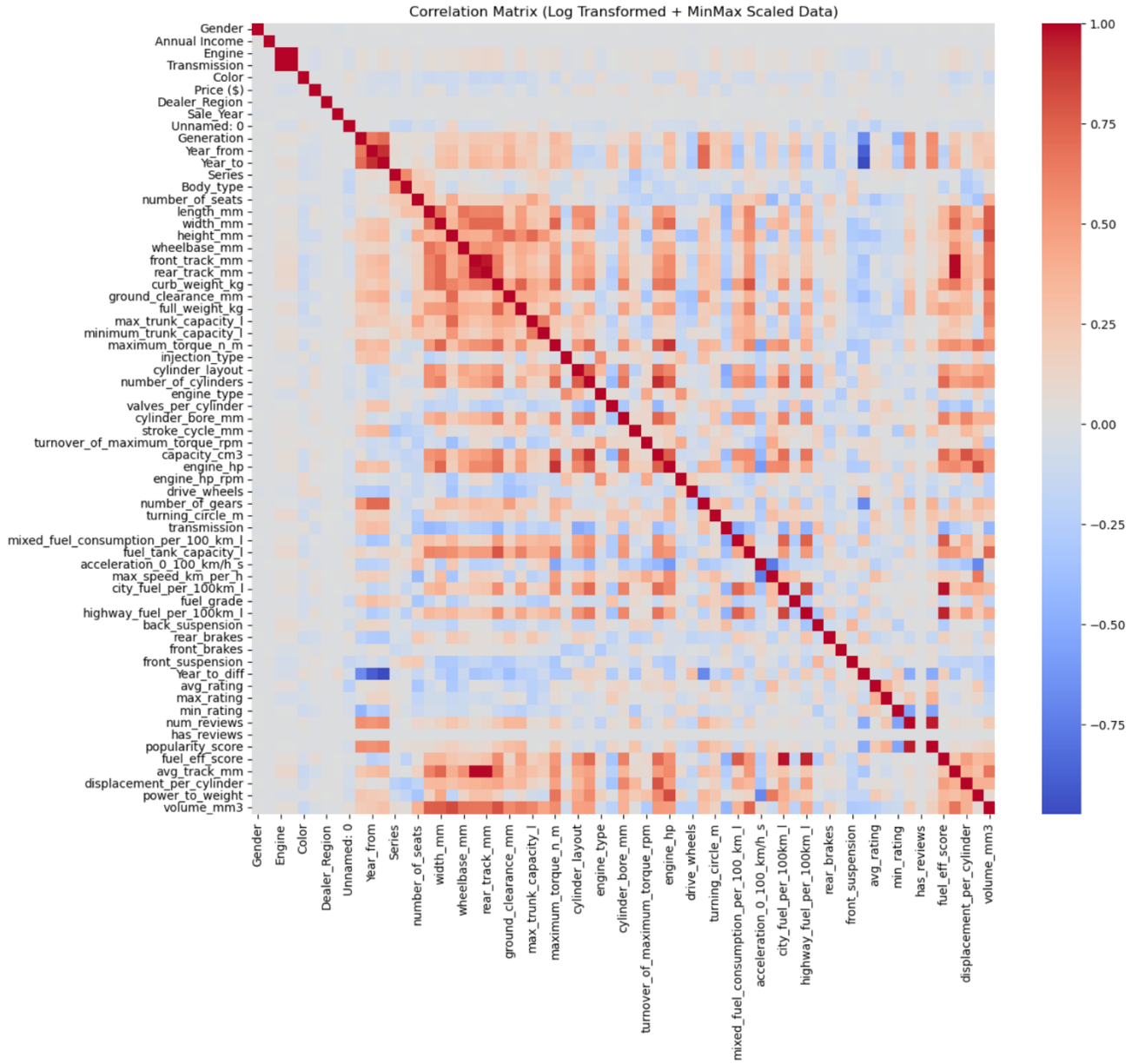


Figure 33 Correaltion matrix for fully merged dataframe

[96]:
```
# Set correlation threshold
threshold = 0.8

# Get the correlation matrix
corr_matrix = car_full_analysis.corr(numeric_only=True)

# Find highly correlated pairs
high_corr = []

for i in range(len(corr_matrix.columns)):
    for j in range(i + 1, len(corr_matrix.columns)):
        corr_value = corr_matrix.iloc[i, j]
        if abs(corr_value) > threshold:
            high_corr.append({
                "Feature 1": corr_matrix.columns[i],
                "Correlation": corr_value,
                "Feature 2": corr_matrix.columns[j]

            })

# Convert to DataFrame
high_corr_df = pd.DataFrame(high_corr).sort_values(by="Correlation", ascending=False)

# Results
print(f"\nHighly Correlated Pairs (|correlation| > {threshold}):")
print(high_corr_df)
```

```
Highly Correlated Pairs (|correlation| > 0.8):
                    Feature 1  Correlation                Feature 2
0                      Engine     1.000000             Transmission
7               rear_track_mm     0.991648             avg_track_mm
6               front_track_mm     0.990118             avg_track_mm
13     city_fuel_per_100km_l     0.976215          fuel_eff_score
5               front_track_mm     0.963764            rear_track_mm
15                 num_reviews     0.956753          popularity_score
14  highway_fuel_per_100km_l     0.944854          fuel_eff_score
10        number_of_cylinders     0.926164            capacity_cm3
1                   Year_from     0.915151                 Year_to
9          maximum_torque_n_m     0.900108               engine_hp
12     city_fuel_per_100km_l     0.851691  highway_fuel_per_100km_l
4                   height_mm     0.824555              volume_mm3
11                  engine_hp     0.823648         power_to_weight
8             curb_weight_kg     0.801643              volume_mm3
2                   Year_from    -0.881953            Year_to_diff
3                     Year_to    -0.973877            Year_to_diff
```

[“Engine”,”maximum_torque_n_m”,”engine_hp”,”rear_track_mm”,”front_track_mm”,”num_reviews”,”curb_weight_kg”,”Year_from”,”year_to”,”year_to_diff”,”capacity_cm3”] are set to be deleted.

5.2 Chi Square Test :

The Chi Sqaure test is done on the categorical data and the analysis is drawn as to which columns have p< 0.05)

[99]:
```
# Chi-Square Test

from scipy.stats import chi2_contingency
import pandas as pd

print("\nChi-Square Test Results (Categorical Features vs company_model):")

chi_square_results = []

for col in categorical_cols:
    contingency_table = pd.crosstab(car_full_analysis[col], car_full_analysis["company_model"])
    chi2, p, dof, expected = chi2_contingency(contingency_table)
    chi_square_results.append((col, chi2, p))

# Create results DataFrame
chi_square_df = pd.DataFrame(chi_square_results, columns=["Feature", "Chi2 Statistic", "P-Value"])
chi_square_df.sort_values(by="P-Value", inplace=True)

print("\nAll Chi-Square Test Results:")
print(chi_square_df)
```

```
Chi-Square Test Results (Categorical Features vs company_model):

All Chi-Square Test Results:
            Feature  Chi2 Statistic   P-Value
8     injection_type    91850.000000  0.000000
14       rear_brakes    36740.000000  0.000000
13   back_suspension   771540.000000  0.000000
12      transmission    55110.000000  0.000000
11      drive_wheels    73480.000000  0.000000
10       engine_type    36740.000000  0.000000
9    cylinder_layout    55110.000000  0.000000
15      front_brakes    36740.000000  0.000000
16  front_suspension   514360.000000  0.000000
6             Series   716430.000000  0.000000
5         Generation   734800.000000  0.000000
3              Color     7102.810845  0.000000
2       Transmission     6058.326584  0.000000
1             Engine     6058.326584  0.000000
7          Body_type   146960.000000  0.000000
4      Dealer_Region      857.671106  0.000165
0             Gender      125.771603  0.317783
```

[101]:
```
print("\nSignificant Categorical Features (p < 0.05):")
significant_features = chi_square_df[chi_square_df["P-Value"] < 0.05]
print(significant_features)
```

```
Significant Categorical Features (p < 0.05):
            Feature  Chi2 Statistic   P-Value
8     injection_type    91850.000000  0.000000
14       rear_brakes    36740.000000  0.000000
13   back_suspension   771540.000000  0.000000
12      transmission    55110.000000  0.000000
11      drive_wheels    73480.000000  0.000000
10       engine_type    36740.000000  0.000000
9    cylinder_layout    55110.000000  0.000000
15      front_brakes    36740.000000  0.000000
16  front_suspension   514360.000000  0.000000
6             Series   716430.000000  0.000000
5         Generation   734800.000000  0.000000
3              Color     7102.810845  0.000000
2       Transmission     6058.326584  0.000000
1             Engine     6058.326584  0.000000
7          Body_type   146960.000000  0.000000
4      Dealer_Region      857.671106  0.000165
```

Figure 34 Code to implement chisquare test and results

Conclusion: The Chi square test has been performed on the features and "**Gender**" having chi-square value 0.317 is set to be dropped.

5.3 PCA Analysis :

The variance plot is taken and the least and highest contributing columns are analyzed . Some more columns are dropped as per the PCA analysis least contributing factors.

First the number of components needed to retain 90% information in the data.

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Define target & feature columns
targetcol = 'company_model'
featurecols = [
    'Annual Income', 'Transmission', 'Color', 'Price ($)', 'Sale_Year', 'Unnamed: 0', 'Generation', 'Series',
    'Body_type', 'number_of_seats', 'wheelbase_mm', 'ground_clearance_mm', 'full_weight_kg',
    'max_trunk_capacity_l', 'minimum_trunk_capacity_l', 'injection_type', 'cylinder_layout',
    'engine_type', 'valves_per_cylinder', 'cylinder_bore_mm', 'stroke_cycle_mm', 'turnover_of_maximum_torque_rpm',
    'engine_hp_rpm', 'drive_wheels', 'number_of_gears', 'turning_circle_m', 'transmission',
    'mixed_fuel_consumption_per_100_km_l', 'fuel_tank_capacity_l', 'acceleration_0_100_km/h_s',
    'max_speed_km_per_h', 'fuel_grade', 'back_suspension', 'rear_brakes', 'front_brakes',
    'front_suspension', 'avg_rating', 'max_rating', 'min_rating', 'has_reviews', 'power_to_weight',
    'popularity_score', 'fuel_eff_score', 'avg_track_mm', 'displacement_per_cylinder', 'volume_mm3'
]

# Standardize features
X = car_full_analysis[featurecols].dropna()
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Explained variance plot
plt.figure(figsize=(10, 6))
plt.plot(np.cumsum(pca.explained_variance_ratio_), marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance by PCA Components')
plt.grid(True)
plt.axhline(y=0.90, color='g', linestyle='--', label='90% Threshold')
plt.legend()
plt.tight_layout()
plt.show()
```

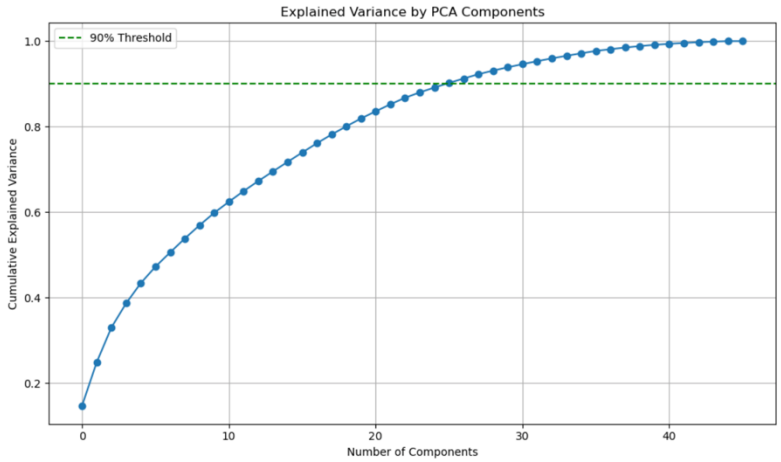


Figure 35 code and output of calculated variance

Based on this the number of components are chosen as : 25

The top contributing and least contributing factors are listed based on the above analysis.

[109]:
```
# Get feature contributions to each component
loadings = pd.DataFrame(pca.components_.T, columns=[f'PC{i+1}' for i in range(len(featurecols))], index=featurecols)

# Sum of absolute contributions across top N components
top_n = 25
featureimportance = loadings.iloc[:, :top_n].abs().sum(axis=1).sort_values(ascending=False)

# Show top and bottom contributing features
print("\n Top contributing features:")
print(featureimportance.head(10))

print("\n Least contributing features:")
print(featureimportance.tail(15))
```

```
 Top contributing features:
turning_circle_m    3.835363
Color               3.761606
Transmission        3.472540
Unnamed: 0          3.463166
max_rating          3.431688
Price ($)           3.422576
front_brakes        3.331918
fuel_grade          3.281059
back_suspension     3.276594
rear_brakes         3.241994
dtype: float64

 Least contributing features:
mixed_fuel_consumption_per_100_km_l    2.479308e+00
acceleration_0_100_km/h_s              2.466689e+00
number_of_gears                        2.434648e+00
cylinder_layout                        2.372210e+00
max_speed_km_per_h                     2.357670e+00
Annual Income                          2.353878e+00
ground_clearance_mm                    2.330247e+00
Sale_Year                              2.280913e+00
engine_type                            2.243004e+00
power_to_weight                        2.239770e+00
fuel_eff_score                         2.221893e+00
cylinder_bore_mm                       2.144911e+00
fuel_tank_capacity_l                   1.954255e+00
volume_mm3                             1.590881e+00
has_reviews                            3.123285e-17
dtype: float64
```

[111]:
```
# Removing few more features after PCA
columns_to_drop = ["acceleration_0_100_km/h_s","number_of_gears","has_reviews","ground_clearance_mm","cylinder_bore_mm","mixed_fuel_consumptio
car_full_analysis = car_full_analysis.drop(columns=columns_to_drop)
```

Figure 36 PCA top and least contributing factors

Based on the above “least contributing factors” the below columns are set to drop:

["acceleration_0_100_km/h_s","number_of_gears","has_reviews","ground_clearance_mm","cylinder_bore_mm","mixed_fuel_consumption_per_100_km_l","cylinder_layout","minimum_trunk_capacity_l"]

5.4 Obvious columns and unimportant:

The obviously unimportant columns that don’t aid the car recommendation process has been dropped.
They include :
["CustomerName","Car_id","Phone", "Dealer_Name", "id_trim","Trim","Dealer_No ","Body Style","Date","Company","Model"]

6. Data Modeling:

6.1 Concept Overview

The objective is to develop a two-stage recommendation system using both supervised and unsupervised machine learning techniques(hybrid):

1. **Supervised Learning Phase:**

   In the first stage, the input features are passed through three different supervised machine learning models to predict the most suitable car (i.e., the target company_model). These models are:

   a. Logistic Regression
   b. Random Forest Classifier
   c. K-Nearest Neighbors (KNN) Classifier

   Based on the trained models, **one** final car is predicted for a given set of input parameters.

2. **Clustering and Recommendation Phase:**

   In the second stage, the predicted car is used as a reference point in a clustering algorithm. The goal is to identify the top 5 most similar cars based on selected key features:

   a. popularity_score
   b. fuel_eff_score
   c. Price ($)
   d. Annual Income

   By identifying the cluster that the predicted car belongs to, the system then recommends other cars from the same cluster that are closest in feature space — offering similar options to the user based on their preferences.

6.2 Supervised learning phase:

In this phase the models are trained and compared based on various parameters.
The data is first split into training and testing sets. Below, shows the split and the code for the same.

```
[117]: 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 37 The data is split as testing and training sets

The data is split in the ratio 0.7:0.3 for train and test.
Then, the data is trained by three supervised machine learning algorithms:

a. Logistic Regression :

```
[125]: # 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 38 Code to implement logistic regression

The max_iter is set to 1000 to ensure convergence during training and random_state to 42 and the model is trained and predictions are drawn.

b. Random Forest Classifier :

```
]: # 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 39 Code to implement random forest classifier

The classifier is configured with n_estimators 100 and maxdepth 10.
The model is trained and predictions were drawn.

c. K nearest Neighbour Classifier :

```
[127]: #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 40 Code to implement KNN classifier

The KNN classifier is implemented with neighbor count 8 and the training is done. The predictions are drawn.

6.3 Supervised Models Evaluation and Comparison:

```
[131]: # Model evaluation function to compare supervised learnings
def evaluate_model(y_true, y_pred, model, modelname):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted', zero_division=0)
    recall = recall_score(y_true, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_true, y_pred, average='weighted', zero_division=0)
    print(f"{modelname} - Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall: {recall:.4f}, F1-score: {f1:.4f}")
```

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

```
Logistic Regression - Accuracy: 0.9924, Precision: 0.9897, Recall: 0.9924, F1-score: 0.9904
Random Forest - Accuracy: 0.9151, Precision: 0.8760, Recall: 0.9151, F1-score: 0.8895
KNN - Accuracy: 0.9886, Precision: 0.9894, Recall: 0.9886, F1-score: 0.9885
```

Figure 41 Code to evaluate all 3 ML models

The evaluate_model() function is used to draw the following comparisons on the models which included:

a. Accuracy
b. Precision
c. Recall
d. F1-score

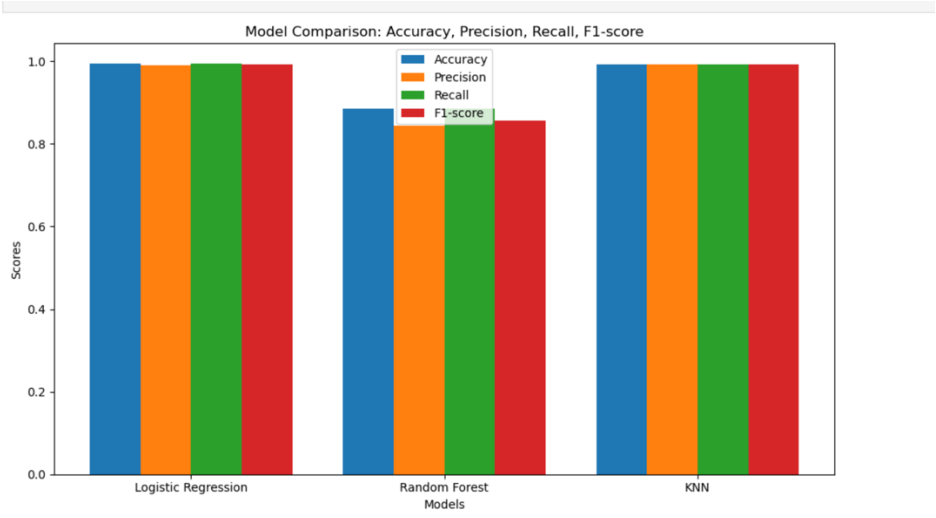


Figure 42 Comparing model parameters for 3 algorithms

Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.9924	0.9897	0.9924	0.9904
Random Forest	0.9151	0.8760	0.9151	0.8895
K-Nearest Neighbors	0.9886	0.9894	0.9886	0.9885

ROC analysis of the three models:

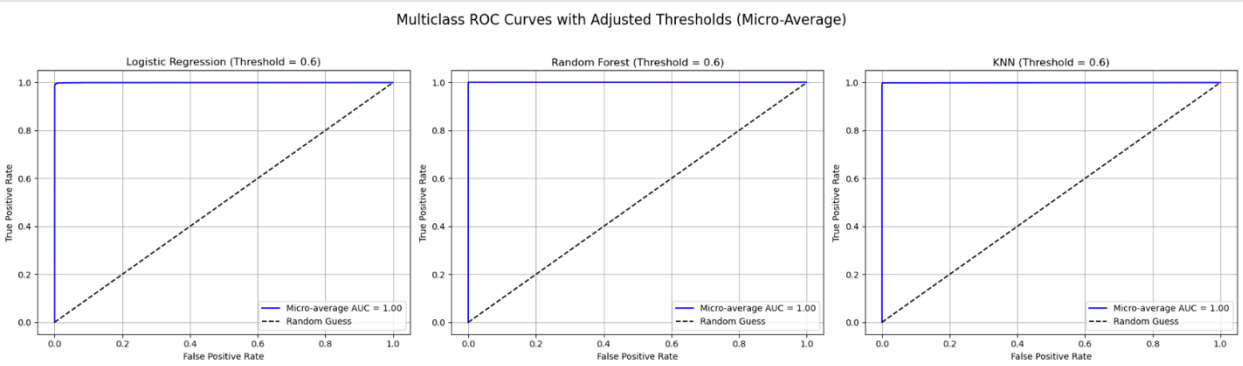


Figure 43 ROC curves for all 3 ML algorithms

➔ **Conclusion on supervised learning models:**

As per the metrics,

Logistic Regression has shown great accuracy with 99.24% and then Random Forest with 91.51 % and KNN with 98.86 %.

As Logistic regression has given more accuracy, precison, recall and F1 score. The “Logistic regr ession” will be used to train the model for the further recommendations using clustering. Also RO C curve is also perfect for Logistic Regression.

6.4 Clustering and Recommendation Phase:

In this phase three clustering algorithms are implemented to recommend cars that are similar to the one predicted by supervised model(Logistic regression). The objective is to recommend more cars based on the similar characteristics.

Method used :

For all three clustering methods (**1. K means Clustering 2. Agglomerative clustering and 3. Gausian mixture model**) :

a. The predicted car from logistic regression is located within dataset and assigned to a cluster.
b. The **similarity score** is calculated using the key features ['popularity_score', 'fuel_eff_score', 'Price ($)', 'Annual Income']
c. Then the top 5 best cars are recommended based on the similarity score.

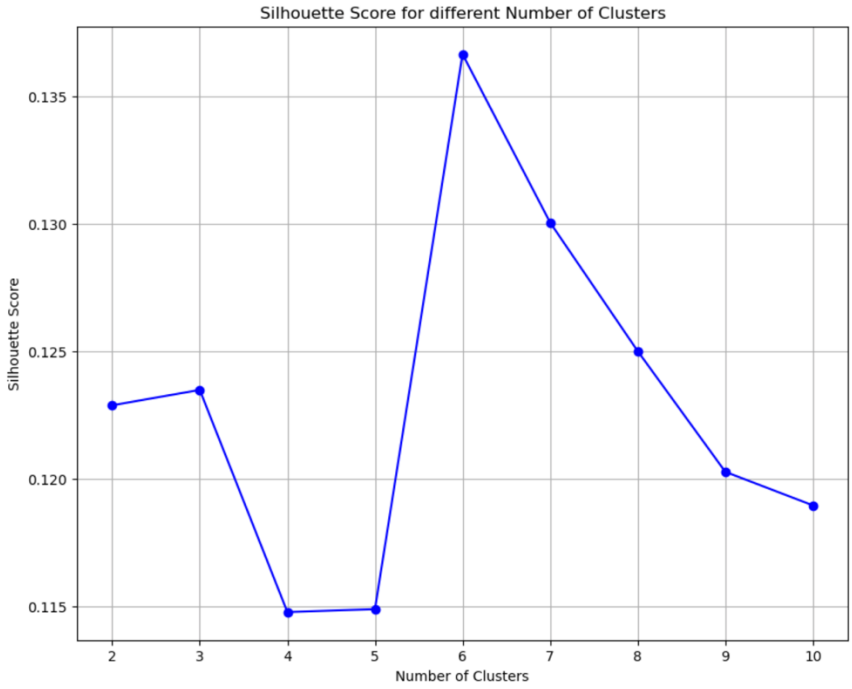
a. K-means Clustering:

```
147]: from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# The range of clusters to test
k_range = range(2, 11)

# scores array
silhouette_scores = []

# Loop through the number of clusters and fit KMeans
for k in k_range:
    # Initialize KMeans with k clusters
    kmeans = KMeans(n_clusters=k, random_state=42,n_init='auto')
    kmeans.fit(X)

    # Calculate the silhouette score
    score = silhouette_score(X, kmeans.labels_)
    silhouette_scores.append(score)

# Plot the silhouette scores to visualize the optimal number of clusters
plt.figure(figsize=(10, 8))
plt.plot(k_range, silhouette_scores, marker='o', linestyle='-', color='b')
plt.title('Silhouette Score for different Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.grid(True)
plt.show()

# Get the optimal number of clusters (k)
optimal_k = k_range[silhouette_scores.index(max(silhouette_scores))]
print(f"Optimal number of clusters: {optimal_k}")
```

Figure 44 silhoutte analysis to get optimal clusters

The number of clusters is determined using **Silhouette Score Analysis**(using silhouette_score from sklearn.metrics), which assesses how well each point fits within its cluster.

Output:




```
Optimal number of clusters: 6
```

Figure 45 Optimal clusters are 6

As per the analsyis the optimal number of clusters to be used is taken as 6 .

K-Means is then applied to segment the dataset into distinct, non-overlapping clusters based on feature similarity.

Code:

```
[240]: # K means

[242]: 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

[244]: # 'company_model' as the target column
       X = car_full_analysis.drop(columns=['company_model'])  # Features
       y = car_full_analysis['company_model']  # Target

[246]: # Encode target variable
       le = LabelEncoder()
       y_encoded = le.fit_transform(y)

[248]: # Split data into training and testing sets
       X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.4, random_state=42)

[250]: # Train the model
       lrmodel = LogisticRegression(max_iter=2000, random_state=42)
       lrmodel.fit(X_train, y_train)

[250]: LogisticRegression
       LogisticRegression(max_iter=2000, random_state=42)

[251]: # Apply KMeans to all data
       kmeans = KMeans(n_clusters=6, random_state=42, n_init='auto')
       car_full_analysis['Cluster'] = kmeans.fit_predict(X)

[252]: # Predict using Logistic Regression
       lrpreds = lrmodel.predict(X_test)

[253]: def get_cluster_for_predicted_car(predicted_label, car_data):
           company_model = le.inverse_transform([predicted_label])[0]
           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

[254]: 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

[255]: recommended_cars, company_model = recommend_similar_cars(lrpreds[0], car_full_analysis)
```

Figure 46 Code to implement Kmeans

First, the encoding is done for the target variable and then the data is split into testing and training sets.

Then logistic regression is performed . The model is trained and predictions are done.

The Kmeans clustering is implemented with (clusters count =6 as per the silhouette anlysis).

Two methods are implemented which finds out the cluster of the input car and recommends best car based on the best score with the parameters.

As an example, lrpreds[0] is chosen as input and the recommendations and output is as below:

```
[255]: recommended_cars, company_model = recommend_similar_cars(lrpreds[0], car_full_analysis)

[262]: print(f"Recommended cars based on predicted company_model: - {company_model} ")
       print("using Kmeans clustering")
       print(recommended_cars)
       print(f"Predicted car: {le.inverse_transform([lrpreds[0]])[0]}")

Recommended cars based on predicted company_model: - mercury_grand marquis
using Kmeans clustering
             company_model     score
2136    buick_park avenue  3.164693
964   ford_crown victoria  3.146330
3425  lincoln_continental  3.099188
1888  mercury_mountaineer  3.046633
4547     lincoln_town car  3.020719
Predicted car: mercury_grand marquis
```

Figure 47 Sample output of kmeans clustering

Here for the input -mercury_grand marquis the best recommedations are predicted as above with the similarity score.

Plot to understand the clustering for Kmeans:

```
[264]: import matplotlib.pyplot as plt
       from sklearn.decomposition import PCA

       # Reduce dimensions to 2D for visualization
       pca = PCA(n_components=2)
       X_pca = pca.fit_transform(X)

       # Add PCA components to the dataframe
       car_full_analysis['PCA1'] = X_pca[:, 0]
       car_full_analysis['PCA2'] = X_pca[:, 1]

       # Scatter plot of the clusters
       plt.figure(figsize=(10, 6))
       scatter = plt.scatter(
           car_full_analysis['PCA1'],
           car_full_analysis['PCA2'],
           c=car_full_analysis['Cluster'],
           cmap='viridis',
           s=50,
           alpha=0.7
       )

       plt.colorbar(scatter, label='Cluster')
       plt.title("K-Means Clustering Visualization (PCA Reduced)")
       plt.xlabel("PCA Component 1")
       plt.ylabel("PCA Component 2")
       plt.grid(True)
       plt.tight_layout()
       plt.show()
```

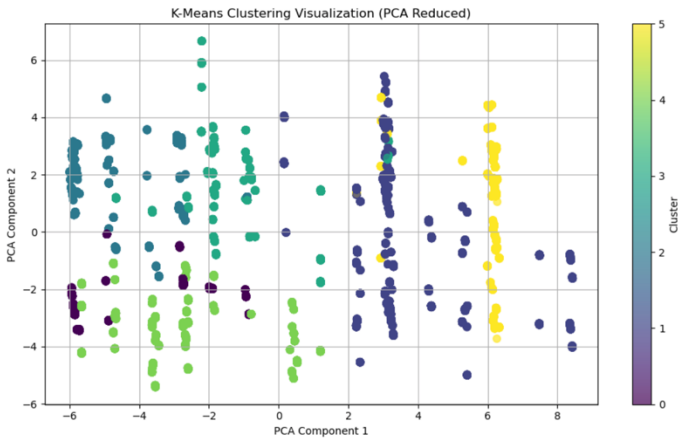


Figure 48 Kmeans cluster visualization

The clustering is visualized by using PCA for k-means clustering.

b. Agglomerative clustering :

code:

```
[300]: #  Agglomerative clustering

[302]: # Predict labels using the trained logistic regression model
lrpreds = lrmodel.predict(X_test)

# using X
X_for_clustering = X.copy()

[304]: from sklearn.cluster import AgglomerativeClustering

# Apply Agglomerative Clustering with "6" from silhoutte method
agglo = AgglomerativeClustering(n_clusters=6)
car_full_analysis['Agglo_Cluster'] = agglo.fit_predict(X_for_clustering)

[305]: def get_cluster_for_predicted_car_agg(predicted_label, car_data):
    company_model = le.inverse_transform([predicted_label])[0]
    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

[306]: 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
```

Figure 49 Code to implement agglomerative clustering

Here, as the logistic regression model is already trained, the predicts are **reused.** Then agglomerative clustering is implemented and the methods are implemented to identify the cluster after agglomerative clustering and calculating the similarity score.

Output:

```
[307]: recommended_cars_agg, company_model = recommend_similar_cars_agg(lrpreds[2347], car_full_analysis)

[308]: print(f"Recommended cars based on predicted company_model:  {company_model} ")
print("using agglomerative clustering")
print(recommended_cars_agg)
print(f"Predicted car: {le.inverse_transform([lrpreds[2347]])[0]}")

Recommended cars based on predicted company_model:  volvo_c70
using agglomerative clustering
              company_model     score
2970   chevrolet_monte carlo  2.793575
1762            dodge_viper  2.432922
16270      chevrolet_camaro  2.340590
3776             volvo_c70  2.229040
16918          honda_odyssey  2.219272
Predicted car: volvo_c70
```

Figure 50 Sample output for agglomerative clustering

lrpreds[2347] taken for testing the value is “volvo-c70” which is taken as input and the nearest best cars are predicted along with the scores.

Plot:

```
[309]: import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Step 1: Scale your features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_for_clustering)

# Step 2: PCA for 2D visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Step 3: Store PCA values for plotting
car_full_analysis['PCA1'] = X_pca[:, 0]
car_full_analysis['PCA2'] = X_pca[:, 1]

# Step 4: Plot the clusters
plt.figure(figsize=(10, 6))
scatter = plt.scatter(
    car_full_analysis['PCA1'],
    car_full_analysis['PCA2'],
    c=car_full_analysis['Agglo_Cluster'],
    cmap='Set2',
    s=50,
    alpha=0.7
)

plt.colorbar(scatter, label='Agglomerative Cluster')
plt.title("Agglomerative Clustering Visualization (PCA Reduced)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.grid(True)
plt.tight_layout()
plt.show()
```

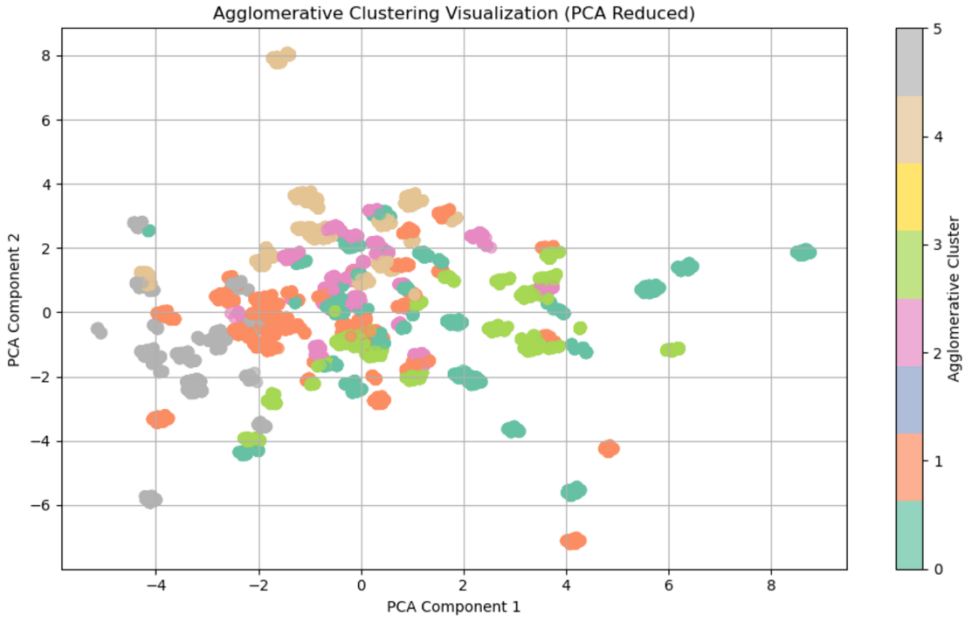


Figure 51 Plot to visualize agglomerative cluster

The clustering is visualized by using PCA for Agglomerative clustering as above.

c. Gaussian Mixture Model (GMM) clustering:

GMM is implemented as it's a probabilistic clustering technique does soft clustering meaning eac h car belongs to multiple clusters offering flexibility for similarity score.

Code and implementation:

[335]:
```
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Scale the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_for_clustering)

# Gaussian Mixture Model (GMM)
gmm = GaussianMixture(n_components=15, random_state=42)
car_full_analysis['GMM_Cluster'] = gmm.fit_predict(X_scaled)
```

[336]:
```
def get_cluster_for_predicted_car_gmm(predicted_label, car_data):
    company_model = le.inverse_transform([predicted_label])[0]
    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
```

[337]:
```
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
```

Figure 52 Code to implement GMM clustering

In a similar way the GMM clustering is implemented(using Guassian Mixture form sklearn.mixtu re). Here for GMM scaling is done and the functions are implemented.

Output:

[338]:
```
recommended_cars_gmm, company_model = recommend_similar_cars_gmm(lrpreds[4321], car_full_analysis)
```

[343]:
```
print(f"Recommended cars based on company_model: {company_model}")
print("using GMM clustering")
print(recommended_cars_gmm)
print(f"Predicted car: {le.inverse_transform([lrpreds[4321]])[0]}")
```

```
Recommended cars based on company_model: toyota_camry
using GMM clustering
            company_model     score
3729          ford_taurus  3.239956
6065   pontiac_grand prix  3.208880
1681     cadillac_deville  3.066401
10186    chevrolet_impala  3.020340
10729       nissan_maxima  3.019625
Predicted car: toyota_camry
```

Figure 53 Sample output of GMM clustering

For testing purpose the lrpreds[4321] (Toyota_camry) is taken as input for recommendation and t he rcommendations of 5 cars is given as per the score.

6.5 Unsupervised Models(clustering) Evaluation and Comparison:

Clustering Algorithm	Silhouette Score	Davies-Bouldin Index	Adjusted Rand Index (ARI)	Cluster Purity
K-Means Clustering	0.1183	2.2919	0.1383	0.1519
Agglomerative Clustering	0.1173	2.0439	0.1056	0.1369
Gaussian Mixture Model	0.1183	2.2919	0.1383	0.1519

a. Silhouette Score: measures how similar a point is to its own cluster compared to other clusters

Kmeans and guassian show high silhouette score represents that they formed better clusters than a gglomerative clustering.

b. Davies-Bouldin Index (DBI) evaluates intra-cluster similarity and inter-cluster differences.

Agglomerative clustering with low DBI(2.0439) has shown better compactness and well separate d.

c. Adjusted Rand Index(ARI): measures how closely the predicted clusters match the true label s. A higher ARI means better alignment with actual class labels.

In this aspect kmeans and GMM outshined agglomerative clustering.

d. **Cluster purity :** Cluster Purity measures the extent to which clusters contain a single class. A higher purity indicates more homogenous clusters.

Kmeans and GMM clustering have better cluster purity.

Hence **Kmeans** and **GMM** are better recommended clusters than agglomerative clustering.

Conclusion:

On analyzing various clusters and parameters the **Kmeans clustering** is chosen as the best along with **logistic regression** as the best for supervised learning based on previous analysis.

7. Future scope:

As part of milestone 2 the aspects of **Feature engineering, Feature selection and Model training** are implemented completely. In the upcoming milestone the aim is to conduct more robust evaluation of trained models and verify the performance. The next step would be to identify potential biases or limitations in the model. The later steps would be to build UI to visualize model accuracy, feature contributions and **allow users** to enter preferences and receive car recommendations **dynamically** and implement automatic pdf reporting tool to summarize findings.

8. Conclusion:

The preprocessed dataframes from milestone 1 are taken and the three dataframes are merged. Then based on the **EDA from milestone1** and as per the requirement new features are created . Based on **Correlation analysis, Chi-square test and PCA analysis** the feature selection is done**." Label encoding"** is performed on the categorical data Then, the supervised machine learning models (**Logistic regression, Random forest classifier and K nearest neighbor models**) are trained and prediction is performed. Based on the analysis of various parameters as discussed "**Logistic Regression**" is chosen as the best and then the output is passed on to Clustering models to predict best cars based on the cluster to which the output belong to and **similarity score.** The three clustering methods are implemented (" K-means Clustering, Agglomerative clustering and Gaussian mixture model") is implemented and **Kmeans Clustering** turned out to be better than other clustering methods. Overall, hybrid analysis (supervised and unsupervised models) are used to perform **"Car Recommendation**". The testing is also performed for each clustering method and it predicted **5 additional cars with score similarity** with the initial prediction from logistic regression.