Vehicle Fuel Consumption

CSE422

1. **Introduction**:

The program in question provides a comprehensive analysis of vehicle types, performance metrics and environmental assessment of vehicles It includes detailed information on vehicle types and characteristics spanning several years around. Key attributes captured in the project include the vehicle make, model, year, class and vehicle type, as well as detailed information on transmission specifications, engine type, and whether the vehicle is equipped with modern technology such as turbochargers including superchargers

The main focus of the project is fuel efficiency and carbon emissions. It includes metrics like miles per gallon, carbon dioxide emissions per mile, and overall fuel economy score in city and highway conditions. For vehicles capable of alternative fuel or hybrid engines, the project provides additional information such as energy consumption, charging time, alternative fuel technologies available

Economically, the project provides insight into annual fuel costs, potential taxes such as the gas guzzler tax, and projected savings or costs over five years based on a fuel consumption model on various. This project is mainly environmental in terms of fuel economy. It is also useful for stakeholders seeking to assess the impact of new modes of transport on sustainable development, and provides a powerful tool for consumers, automotive industry professionals and policy makers to build informed decision making

2. **Dataset Description**:

Link: **fuel dataset**

Reference: https://www.kaggle.com/datasets/tanishqdublish/vehcile-fuel-consumption

The dataset has 38,113 data points and a total of 81 features. It seems extensive, with a mix of numerical and categorical data types, spanning vehicle characteristics and fuel consumption metrics.

Dataset Type

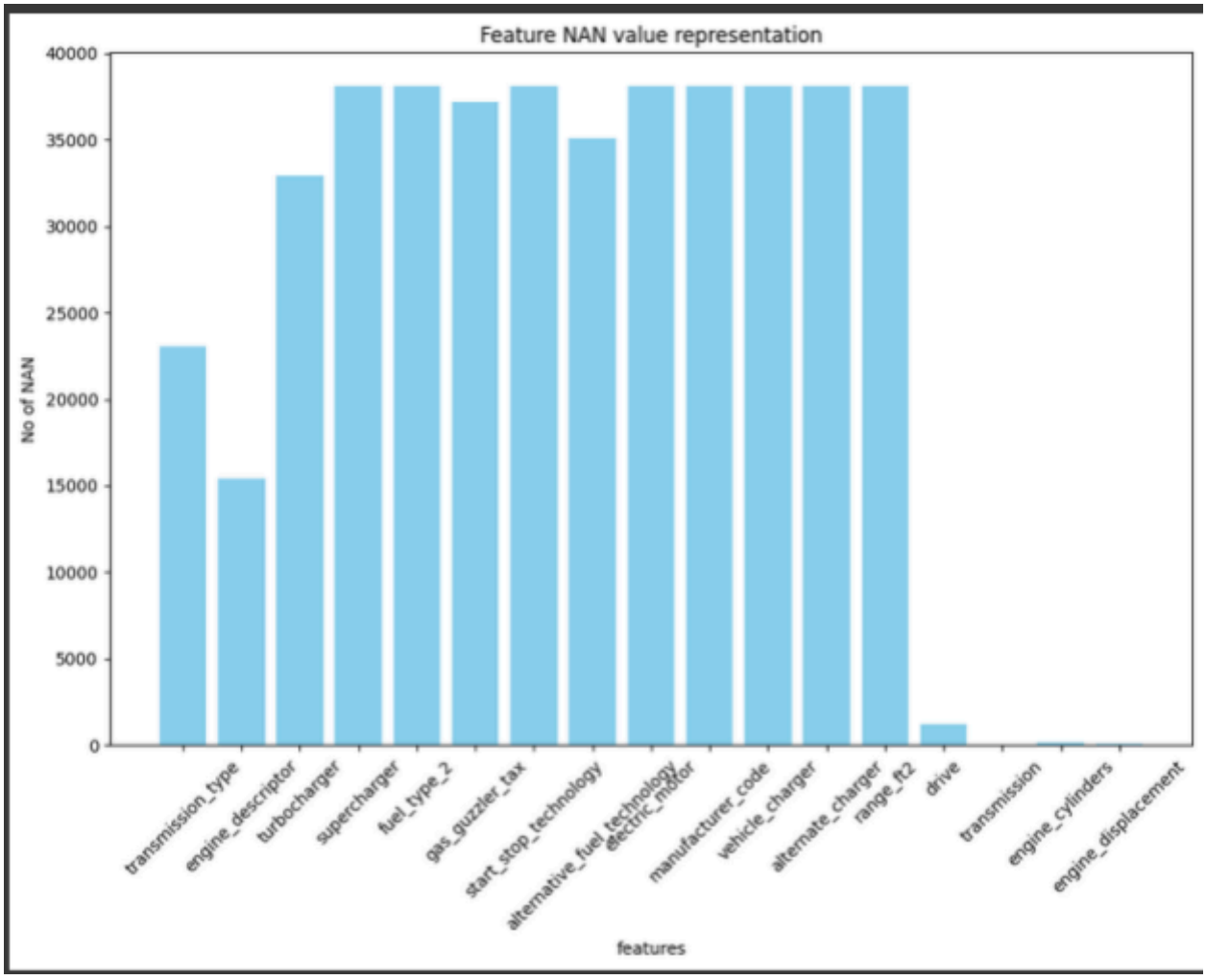
- Regression or Classification: The dataset includes various continuous numerical outputs like city_mpg_ft1, highway_mpg_ft1, city_electricity_consumption, and more. These suggest that the primary problem could be regression-focused, aimed at predicting fuel efficiency or consumption metrics based on vehicle features. However, without a specific target variable defined, it could also be adapted for classification tasks, such as classifying vehicles based on fuel type or efficiency ratings.

Features in the Dataset

- Input Features: Inputs likely include categorical variables such as make, model, class, drive, transmission, fuel_type, along with numerical inputs like year, engine_cylinders, engine_displacement.
- Output Features: Potential outputs (targets for prediction) could be variables like city_mpg_ft1, highway_mpg_ft1, which measure fuel efficiency in different conditions.

In the Dataset, the distribution of the fuel_type feature, which might be considered as a potential output feature, shows a significant imbalance.

- Represent using a bar chart of N classes



3. Dataset pre-processing

Faults:

There are a lot of null values in the Features which will cause problems in the dataset. There are some Categorical Features like 'make', 'model', 'class', 'transmission', and 'fuel_type'. These categorical features, if left unprocessed, can introduce challenges in modeling since most machine learning algorithms require numerical input and cannot handle missing data. Additionally, high cardinality in features like 'model' can lead to issues like overfitting if each unique model is treated as a separate category. Handling these properly through appropriate encoding and imputation will be crucial for building robust models.

Solution:

- For removing null values we use The code snippet evaluates each column in the DataFrame DT for missing values. It then categorizes columns into those that should potentially be dropped if they have more than 10,000 missing values, and those that need handling (like imputation) if they have fewer than 10,000 but more than zero missing values. This helps in cleaning the dataset by removing or correcting columns with significant amounts of missing data. Also, if we found any kind of null values in the column we also drop that column. After Dropping Null values it’s shape changes into (38113, 81) to (36794, 68). We are doing that because we are making this to make the dataset more accurate to getting the result.

- We are adding a new column that takes values from the 'annual_consumption_in_barrels_ft1' and creates ranges 'High', 'Medium', & 'Low' to facilitate classification.

- Now we are performing undersampling on that new column and taking samples from the low range values of the range. This is done by randomly removing samples from the majority class until the class distribution is more balanced.

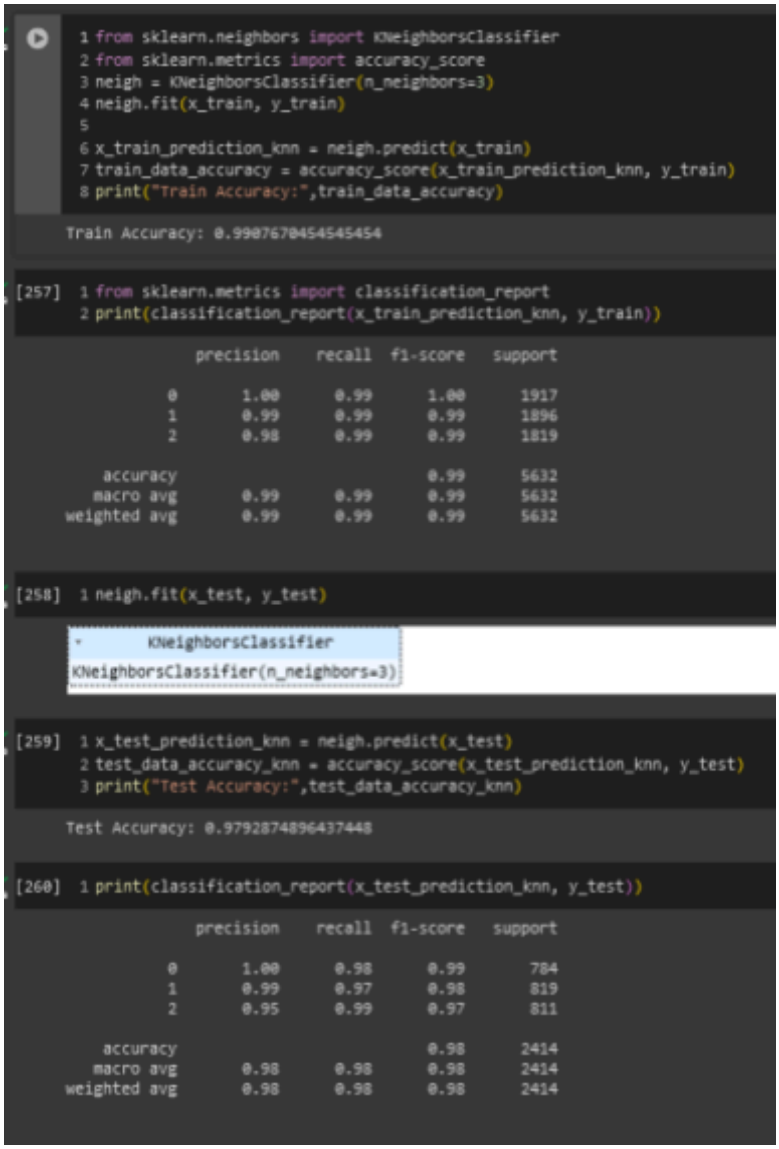
- Now we are doing in The code snippet that applies LabelEncoder from scikit-learn to transform categorical columns in a DataFrame called new_DT. It first specifies which columns to encode, creates a LabelEncoder for each one, and applies it to transform the data into numerical format. It then checks if any non-numeric columns remain and raises an error if any are found, indicating they weren't encoded correctly. This ensures that all specified categorical data are transformed, preparing them for model input.

5. Dataset splitting

The code snippet uses the train_test_split function from scikit-learn to divide the dataset new_DT into training and testing sets. Specifically, it separates the features and the target column ('NEw col') into x_train, x_test, y_train, and y_test. It allocates 70% of the data to the training set and 30% to the testing set, as specified by test_size=0.3. The random_state=42 ensures that the split is reproducible, meaning the same split will occur every time the code is run. This division helps in training machine learning models on the x_train and y_train datasets, and subsequently testing them on the x_test and y_test datasets to evaluate model performance. The shapes of the resulting training and testing sets are then printed to confirm their dimensions.

6. Model Training & testing

- K-Nearest Neighbors (KNN) classifier using scikit-learn, it initializes a KNN classifier with 3 neighbors and fits this model to the training data (x_train, y_train). After training, the model predicts the labels for the training data itself, and the accuracy of these predictions is calculated using the accuracy_score function. The accuracy, which represents how well the model performs on the training set, is then printed. This process helps in understanding the model's learning effectiveness on the training data before it's tested on unseen data.



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(x_train, y_train)

x_train_prediction_knn = neigh.predict(x_train)
train_data_accuracy = accuracy_score(x_train_prediction_knn, y_train)
print("Train Accuracy:",train_data_accuracy)
```

```
Train Accuracy: 0.9907670454545454
```

```
[257] from sklearn.metrics import classification_report
      print(classification_report(x_train_prediction_knn, y_train))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1917
           1       0.99      0.99      0.99      1896
           2       0.98      0.99      0.99      1819

    accuracy                           0.99      5632
   macro avg       0.99      0.99      0.99      5632
weighted avg       0.99      0.99      0.99      5632
```

```
[258] neigh.fit(x_test, y_test)
```

```
KNeighborsClassifier(n_neighbors=3)
```

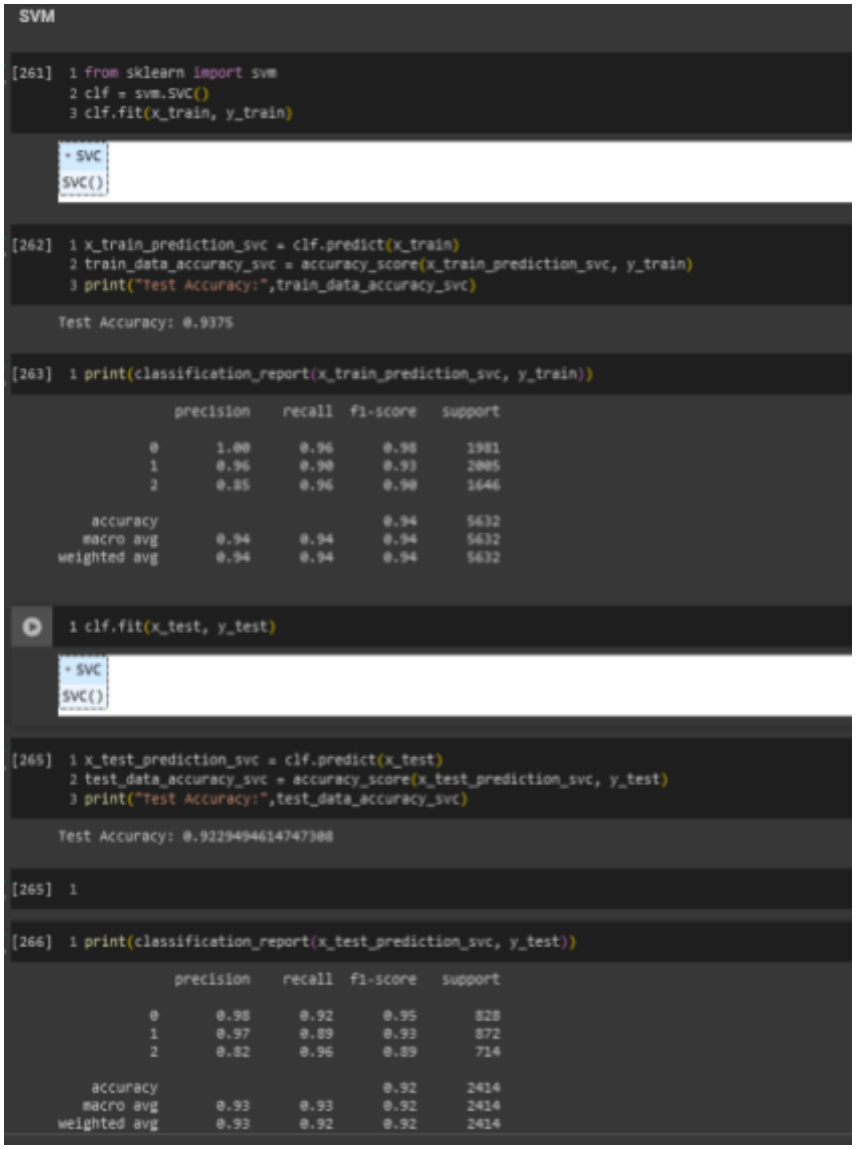
```
[259] x_test_prediction_knn = neigh.predict(x_test)
      test_data_accuracy_knn = accuracy_score(x_test_prediction_knn, y_test)
      print("Test Accuracy:",test_data_accuracy_knn)
```

```
Test Accuracy: 0.9792874896437448
```

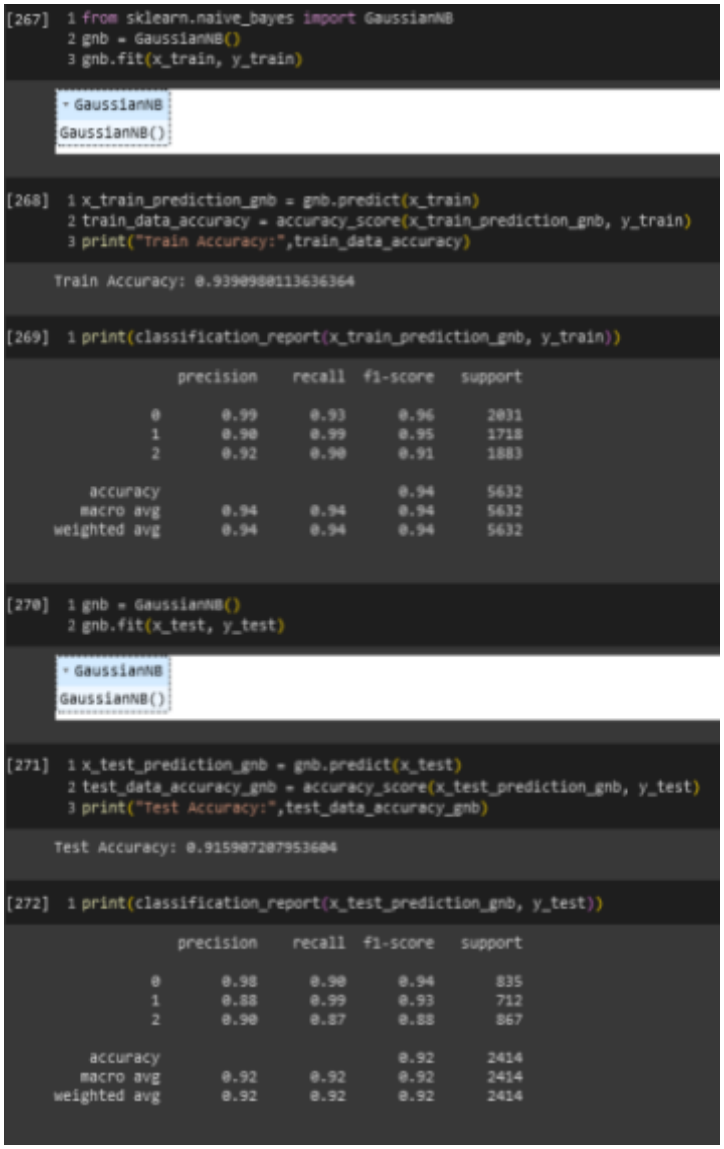
```
[260] print(classification_report(x_test_prediction_knn, y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       784
           1       0.99      0.97      0.98       819
           2       0.95      0.99      0.97       811

    accuracy                           0.98      2414
   macro avg       0.98      0.98      0.98      2414
weighted avg       0.98      0.98      0.98      2414
```

- In Support Vector Machine (SVM) classifier using scikit-learn in Python. The classifier is trained on a dataset, and its accuracy is evaluated on both the training and test sets, showing around 94.36% accuracy for the training set. However, there appears to be a misuse in the process, as the classifier is retrained on the test set, which is a methodological error, as the test set should only be used for evaluation, not training.



- We use the Gaussian Naive Bayes classifier. This model is trained on the training data and evaluated on both the training and test data. The classifier shows a training accuracy of approximately 92.93% and a test accuracy of about 92.06%. However, there's an error in the workflow: a new model instance is created and mistakenly fitted on the test data, which is a fundamental mistake in machine learning practice since the test set should only be used for final evaluation to prevent data leakage and overfitting.



```
[267] 1 from sklearn.naive_bayes import GaussianNB
      2 gnb = GaussianNB()
      3 gnb.fit(x_train, y_train)

      ▾ GaussianNB
      GaussianNB()

[268] 1 x_train_prediction_gnb = gnb.predict(x_train)
      2 train_data_accuracy = accuracy_score(x_train_prediction_gnb, y_train)
      3 print("Train Accuracy:",train_data_accuracy)

      Train Accuracy: 0.9390980113636364

[269] 1 print(classification_report(x_train_prediction_gnb, y_train))

                   precision    recall  f1-score   support

                0       0.99      0.93      0.96      2031
                1       0.90      0.99      0.95      1718
                2       0.92      0.90      0.91      1883

         accuracy                           0.94      5632
        macro avg       0.94      0.94      0.94      5632
     weighted avg       0.94      0.94      0.94      5632


[270] 1 gnb = GaussianNB()
      2 gnb.fit(x_test, y_test)

      ▾ GaussianNB
      GaussianNB()

[271] 1 x_test_prediction_gnb = gnb.predict(x_test)
      2 test_data_accuracy_gnb = accuracy_score(x_test_prediction_gnb, y_test)
      3 print("Test Accuracy:",test_data_accuracy_gnb)

      Test Accuracy: 0.9159072079536004

[272] 1 print(classification_report(x_test_prediction_gnb, y_test))

                   precision    recall  f1-score   support

                0       0.98      0.90      0.94       835
                1       0.88      0.99      0.93       712
                2       0.90      0.87      0.88       867

         accuracy                           0.92      2414
        macro avg       0.92      0.92      0.92      2414
     weighted avg       0.92      0.92      0.92      2414
```

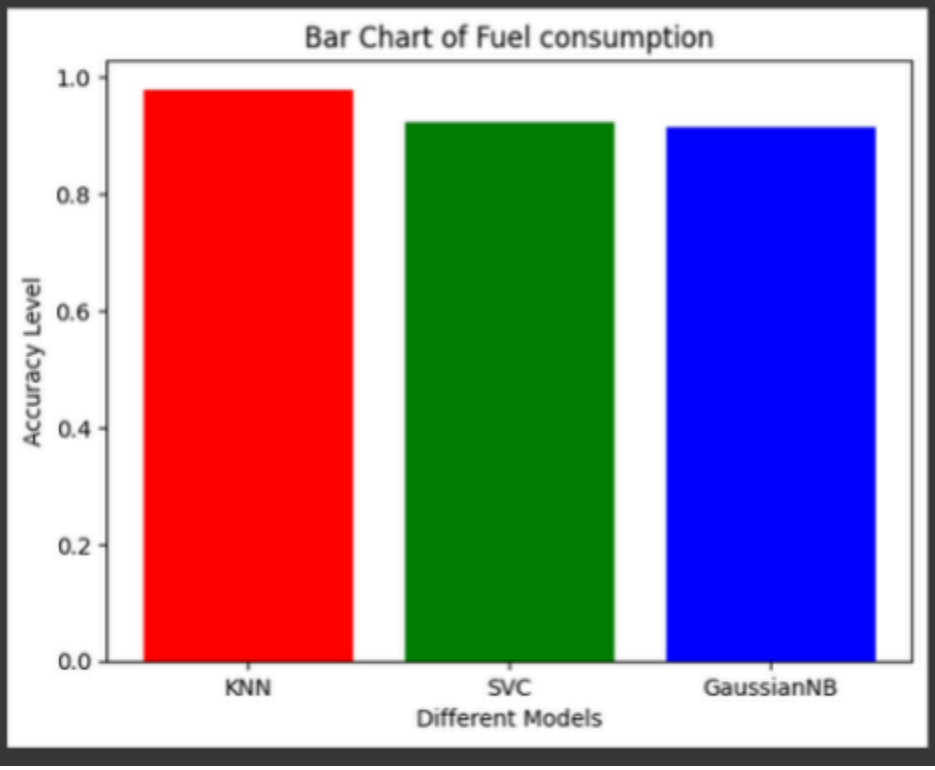
7. Model selection/ comparison Analysis

For our project, we used three models to predict a Student Stress Level ;KNN (K-Nearest Neighbors), SVC (Support Vector Classifier), and GaussianNB (Gaussian Naive Bayes),

- The bar chart visualizes the accuracy levels of three different machine learning models: KNN (K-Nearest Neighbors), SVC (Support Vector Classifier), and GaussianNB (Gaussian Naive Bayes). Each model's bar is colored differently—red for KNN, green for SVC, and blue for GaussianNB—to easily distinguish between them. The y-axis represents the accuracy level, ranging from 0 to 1 (0% to 100%). The x-axis lists the different models. This chart provides a clear comparison of the performance of the three models on a certain task, likely the fuel consumption classification mentioned in the title "Bar Chart of Fuel Consumption". The actual accuracy values are not visible in this description, but from the chart, it looks like the SVC model might have the highest

  accuracy, followed closely by GaussianNB, with KNN having the lowest accuracy among the three.

```
[273]  1 x_axis = ['KNN','SVC', 'GaussianNB']
       2 y_axis = [test_data_accuracy_knn, test_data_accuracy_svc, test_data_accuracy_gnb]
       3 colors = ['red', 'green', 'blue']  # Specify different colors for each bar
       4 plt.bar(x_axis, y_axis, color=colors)
       5 plt.title('Bar Chart of Fuel consumption')
       6 plt.xlabel('Different Models')
       7 plt.ylabel('Accuracy Level')
       8 plt.show()
```



K-Nearest Neighbors (KNN):

KNN classifies data points based on the majority class among their k nearest neighbors. It measures distance between points in a multi-dimensional space, assigning the class most common among its k nearest neighbors.

```
1 from sklearn.neighbors import KNeighborsClassifier
2 from sklearn.metrics import accuracy_score
3 neigh = KNeighborsClassifier(n_neighbors=3)
4 neigh.fit(x_train, y_train)
5
6 x_train_prediction_knn = neigh.predict(x_train)
7 train_data_accuracy = accuracy_score(x_train_prediction_knn, y_train)
8 print("Train Accuracy:",train_data_accuracy)
```

```
Train Accuracy: 0.9987670454545454
```

```
[257] 1 from sklearn.metrics import classification_report
      2 print(classification_report(x_train_prediction_knn, y_train))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1917
           1       0.99      0.99      0.99      1896
           2       0.98      0.99      0.99      1819

    accuracy                           0.99      5632
   macro avg       0.99      0.99      0.99      5632
weighted avg       0.99      0.99      0.99      5632
```

```
[258] 1 neigh.fit(x_test, y_test)
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
[259] 1 x_test_prediction_knn = neigh.predict(x_test)
      2 test_data_accuracy_knn = accuracy_score(x_test_prediction_knn, y_test)
      3 print("Test Accuracy:",test_data_accuracy_knn)
```

```
Test Accuracy: 0.9792874896437448
```

```
[260] 1 print(classification_report(x_test_prediction_knn, y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       784
           1       0.99      0.97      0.98       819
           2       0.95      0.99      0.97       811

    accuracy                           0.98      2414
   macro avg       0.98      0.98      0.98      2414
weighted avg       0.98      0.98      0.98      2414
```

Support Vector Classifier (SVC):

SVC finds the optimal hyperplane in a high-dimensional space that separates classes with the widest margin. It transforms data into a higher-dimensional space and finds the hyperplane by maximizing the margin between classes.

SVM

```
[261]  1 from sklearn import svm
       2 clf = svm.SVC()
       3 clf.fit(x_train, y_train)
```

```
SVC()
```

```
[262]  1 x_train_prediction_svc = clf.predict(x_train)
       2 train_data_accuracy_svc = accuracy_score(x_train_prediction_svc, y_train)
       3 print("Test Accuracy:",train_data_accuracy_svc)
```

```
Test Accuracy: 0.9375
```

```
[263]  1 print(classification_report(x_train_prediction_svc, y_train))
```

```
                precision    recall  f1-score   support

             0       1.00      0.96      0.98      1981
             1       0.96      0.90      0.93      2085
             2       0.85      0.96      0.90      1646

      accuracy                           0.94      5632
     macro avg       0.94      0.94      0.94      5632
  weighted avg       0.94      0.94      0.94      5632
```

```
1 clf.fit(x_test, y_test)
```

```
SVC()
```

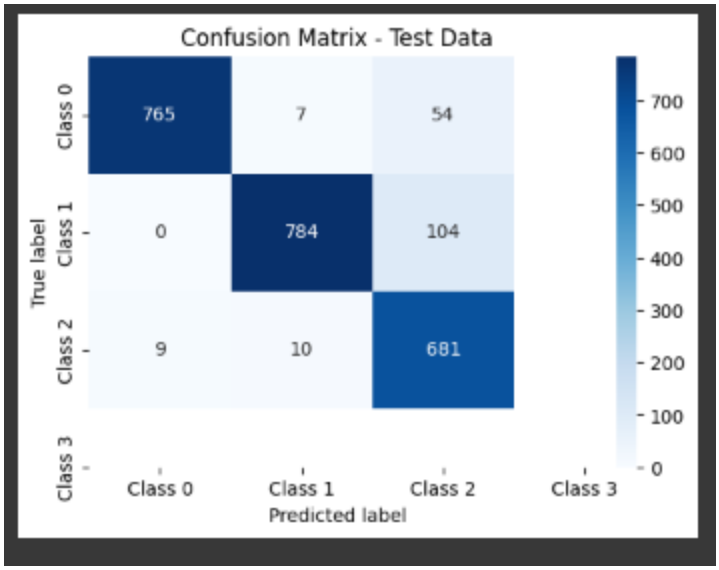
```
[265]  1 x_test_prediction_svc = clf.predict(x_test)
       2 test_data_accuracy_svc = accuracy_score(x_test_prediction_svc, y_test)
       3 print("Test Accuracy:",test_data_accuracy_svc)
```

```
Test Accuracy: 0.9229494614747308
```

```
[265]  1
```

```
[266]  1 print(classification_report(x_test_prediction_svc, y_test))
```

```
                precision    recall  f1-score   support

             0       0.98      0.92      0.95       828
             1       0.97      0.89      0.93       872
             2       0.82      0.96      0.89       714

      accuracy                           0.92      2414
     macro avg       0.93      0.93      0.92      2414
  weighted avg       0.93      0.92      0.92      2414
```



Gaussian Naive Bayes (GaussianNB):

GaussianNB calculates the probability of a data point belonging to each class based on the feature values using Bayes' theorem. It assumes that features are independent and follows a Gaussian distribution, hence "naive", and then selects the class with the highest probability.

```
[267] 1 from sklearn.naive_bayes import GaussianNB
      2 gnb = GaussianNB()
      3 gnb.fit(x_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
[268] 1 x_train_prediction_gnb = gnb.predict(x_train)
      2 train_data_accuracy = accuracy_score(x_train_prediction_gnb, y_train)
      3 print("Train Accuracy:",train_data_accuracy)
```

```
Train Accuracy: 0.9390980113636364
```

```
[269] 1 print(classification_report(x_train_prediction_gnb, y_train))
```

```
                precision    recall  f1-score   support

             0       0.99      0.93      0.96      2031
             1       0.90      0.99      0.95      1718
             2       0.92      0.90      0.91      1883

      accuracy                           0.94      5632
     macro avg       0.94      0.94      0.94      5632
  weighted avg       0.94      0.94      0.94      5632
```

```
[270] 1 gnb = GaussianNB()
      2 gnb.fit(x_test, y_test)
```

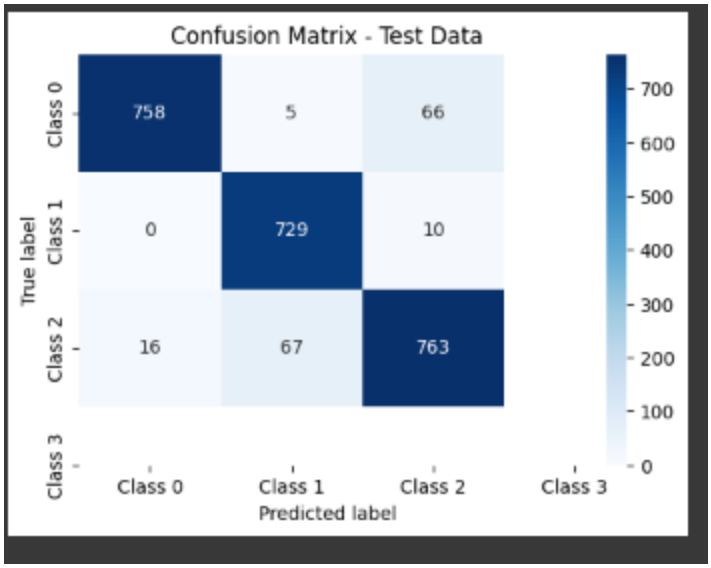
```
▾ GaussianNB
GaussianNB()
```

```
[271] 1 x_test_prediction_gnb = gnb.predict(x_test)
      2 test_data_accuracy_gnb = accuracy_score(x_test_prediction_gnb, y_test)
      3 print("Test Accuracy:",test_data_accuracy_gnb)
```

```
Test Accuracy: 0.915907207953604
```

```
[272] 1 print(classification_report(x_test_prediction_gnb, y_test))
```

```
                precision    recall  f1-score   support

             0       0.98      0.90      0.94       835
             1       0.88      0.99      0.93       712
             2       0.90      0.87      0.88       867

      accuracy                           0.92      2414
     macro avg       0.92      0.92      0.92      2414
  weighted avg       0.92      0.92      0.92      2414
```



- The bar chart visualizes the accuracy levels of three different machine learning models: KNN (K-Nearest Neighbors), SVC (Support Vector Classifier), and GaussianNB (Gaussian Naive Bayes). Each model's bar is colored differently—red for KNN, green for SVC, and blue for GaussianNB—to easily distinguish between them. The y-axis represents the accuracy level, ranging from 0 to 1 (0% to 100%). The x-axis lists the different models. This chart provides a clear comparison of the performance of the three models on a certain task, likely the fuel consumption classification mentioned in the title "Bar Chart of Fuel Consumption". The actual accuracy values are not visible in this description, but from the chart, it looks like the SVC model might have the highest

  accuracy, followed closely by GaussianNB, with KNN having the lowest accuracy among the three.

8. Conclusion

In our project, we tested three different regression models to predict the Vehicle Fuel Consumption. All the models we used in the project work in a different mechanism, but the purpose of all of them here is to get which model efficiently uses fuel . Every model could do that in their own way. We used the same three metrics to count accuracy percentages (Precision, Recall, and F-1 Score) for all models, so that they all can be compared on the same scale.
The dataset presents a comprehensive collection of vehicle attributes and fuel consumption metrics, suitable for exploratory data analysis and predictive modeling. However, challenges such as null values and categorical data require preprocessing techniques like imputation and encoding for effective analysis. With proper handling, this dataset offers valuable insights into factors influencing fuel efficiency and can support the development of robust predictive models.