CSCE 5214-004 SOFTWARE DEVELOPMENT FOR AI

ASSIGNMENT 1: DEVELOP SUPERVISED MACHINE LEARNING MODEL

Eeshwar Vannemreddy
11596826

Task 1: Data Preparation

Step 1: Extracting the provided .zip files

We extracted the contents of a ZIP file titled 'CAN Bus log.zip' using the 'Zipfile' library. The 'r' mode indicates that the ZIP file is currently being read. The extractall() function on the ZIP file object 'zipp' is then invoked, which extracts all the files and directories contained within the ZIP archive into the current working directory. This is frequently used as a stage in data preparation to access and work with the archive's data files for further analysis or processing.

Step 2: Reading the datasets.

We processed CAN bus log files to generate three distinct data frames depending on different scenarios: (1) FF as Speed injection, (2) RPM injection, and (3) no injection. The log files contain CAN message records with timestamps, PIDs (Parameter IDs), and values, with the last field indicating whether the message is an attack (1) or genuine (0).

The getData() function reads each log file, extracts essential information, and puts it in a list called canData as dictionaries. It handles messages with the PIDs '254' (Speed) and '115' (RPM), modifying values and flagging potential assaults based on particular conditions.

The dict_to_df() function transforms a list of dictionaries into a pandas DataFrame, cleans and arranges the data, and encodes the 'PID' column in a single pass. Based on the 'attack' column, it renames columns and retrieves values for 'RPM' and 'Speed'.

Finally, the code creates three data frames for FF Speed injection, RPM injection, and No injection scenarios, which can be further analyzed to detect and analyze attacks on the CAN bus system. Note that you should specify the correct file directories for the log files in the getData() function calls to use this code effectively.

Task 2: Explore and analyze the data

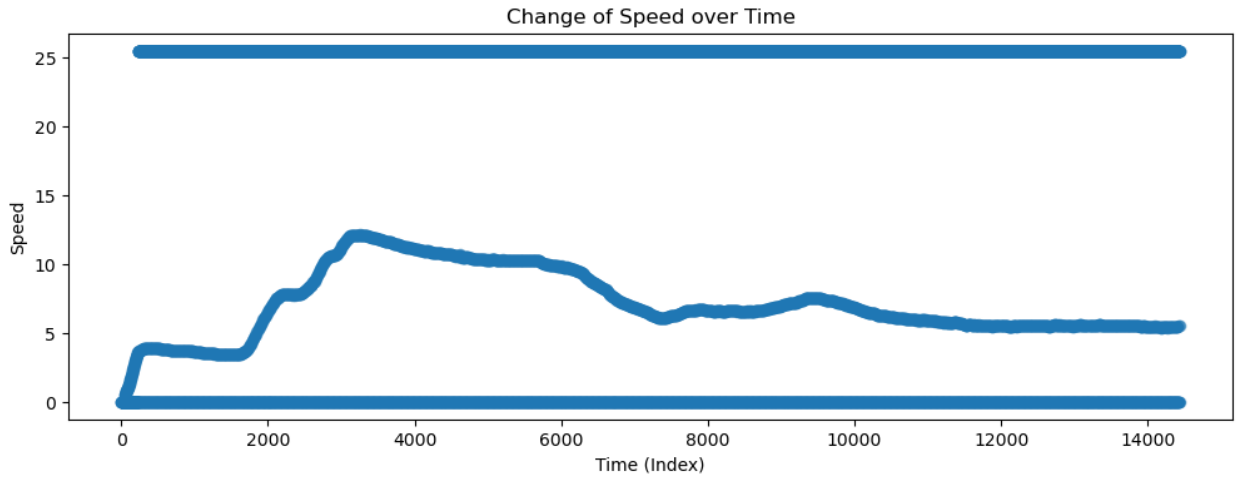
Step 1: Creating scatter plots for 3 scenarios.

In this code, scatter plots are generated to visualize changes in speed readings (CAN ID 254) over time. First, the code filters the data for the scenario involving the injection of FF as speed readings. Similarly, it filters the data for the RPM injection scenario. These filtered datasets, containing speed and RPM readings respectively, are then used to create scatter plots. The x-axis of the scatter plots represents the index of the data, which can be considered as a representation of time. By plotting these data points, one can observe how speed readings change over the data collection period, providing insights into potential abnormalities or attacks on the CAN bus system. The code utilizes the pandas and matplotlib libraries for data manipulation and plotting.

Scatter plots for the three scenarios are below:

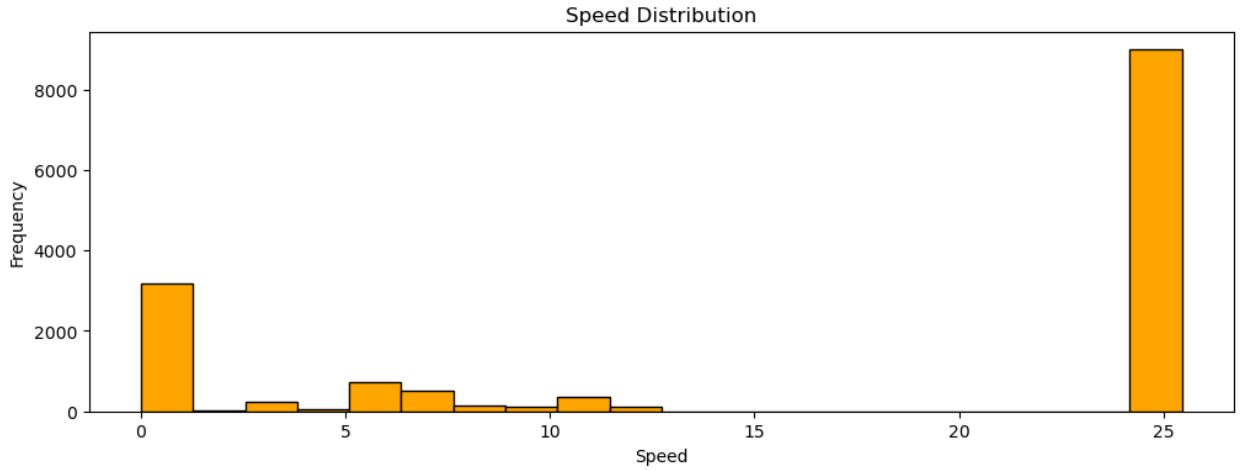
Scenario 1: Change of Speed Over Time

Plot 1:



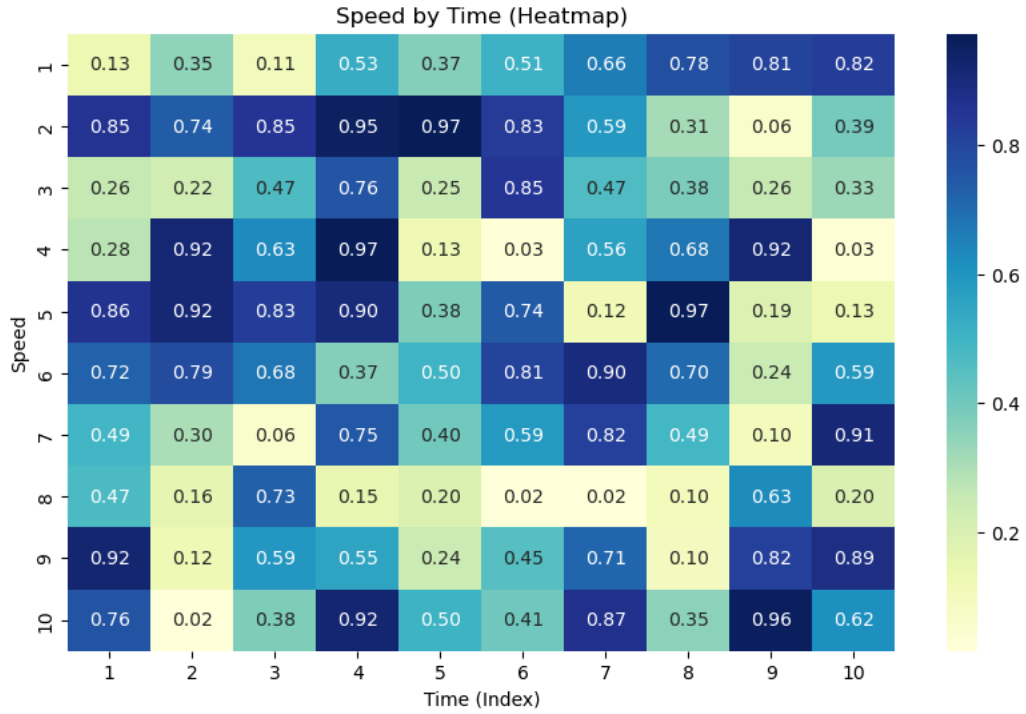
This code creates a scatter plot using matplotlib, showing the change in speed readings over time. The **plt.figure()** sets the figure size, **plt.scatter()** plots the data points, where the x-axis represents the index (a proxy for time) and the y-axis represents the 'Speed' values. 'alpha' adjusts transparency, 'plt.xlabel()' and 'plt.ylabel()' label the axes, and 'plt.title()' sets the plot title. Finally, 'plt.show()' displays the scatter plot, allowing you to visualize how the speed values change over the dataset's index, which can be interpreted as time.

Plot 2:



This code snippet creates a histogram using Matplotlib to visualize the distribution of speed readings in the 'fff_injection_df' DataFrame. It specifies the number of bins as 20 to group speed values, sets the color and edge color for the histogram bars, labels the x and y-axes, gives the plot a title, and then displays the histogram. The resulting chart provides insights into the frequency of different speed values, helping to understand the pattern of speed distribution in the dataset.

Plot 3:

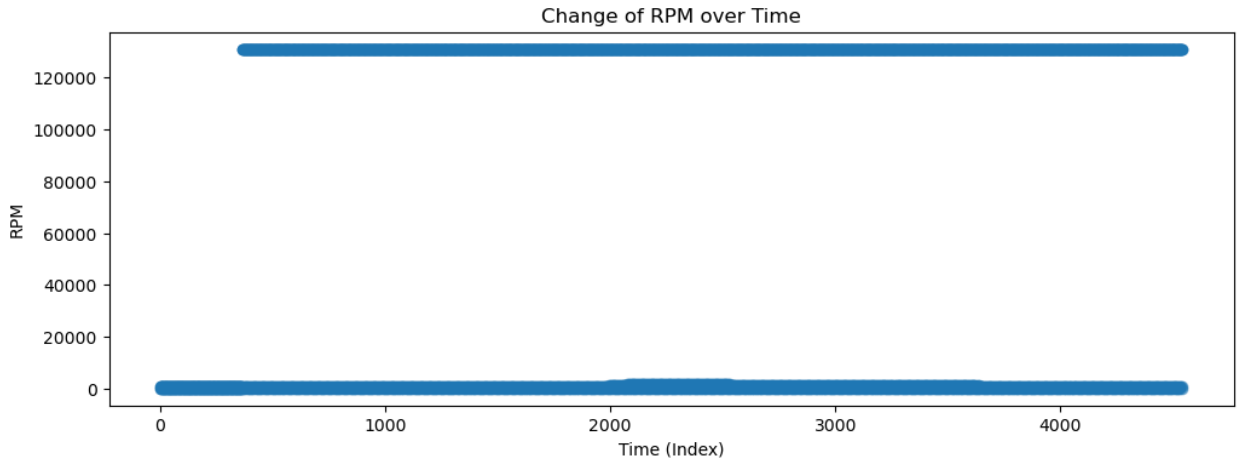


This code uses the seaborn library to create a heatmap that visually represents the relationship between speed and time. The 'speed_by_time' variable, which should be replaced with actual data,

contains values representing speed measurements at different time points. The heatmap is configured to display these values using a color scale ('YlGnBu'), with annotations showing the exact values. The x-axis represents time indices, and the y-axis represents speed. This visualization helps to analyze how speed changes over time, with color intensity indicating the magnitude of speed measurements.

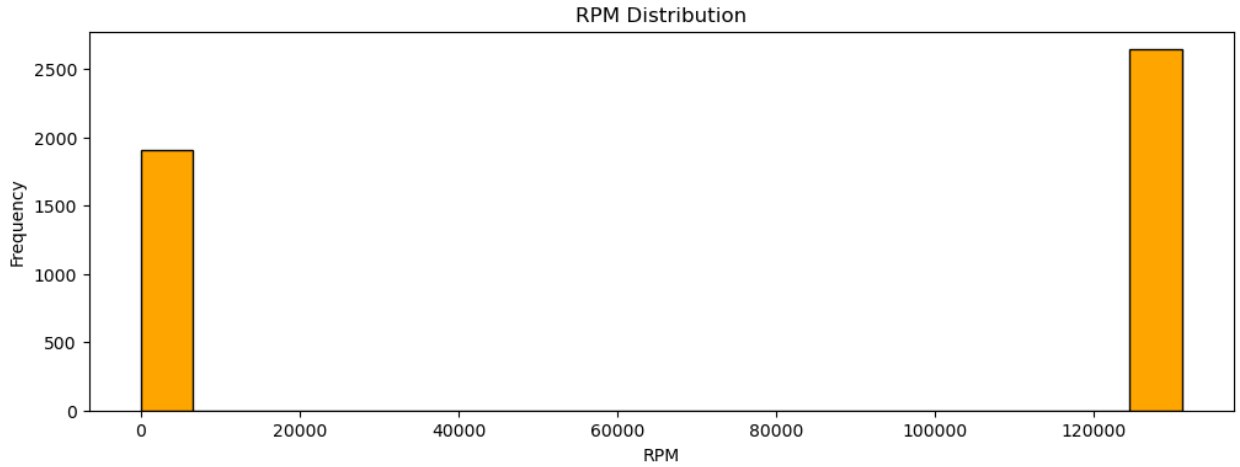
Scenario 2: Change of RPM Over Time

Plot 1:



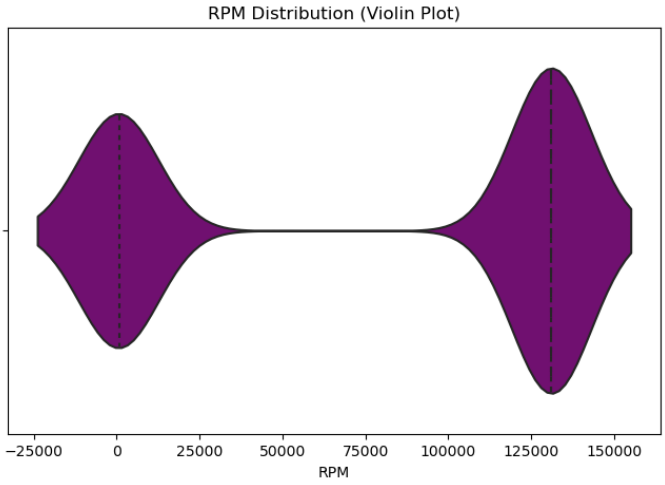
This code generates a scatter plot to visualize the change of RPM (Revolutions Per Minute) readings over time. It uses the index of the data as the time axis and the 'RPM' column as the data points on the y-axis. The scatter plot is displayed with labels for the x-axis (Time) and y-axis (RPM), along with a title indicating the purpose of the plot. The 'alpha' parameter controls the transparency of the data points. This plot helps in visualizing how RPM values evolve over time, providing insights into the behavior of a vehicle's engine or system.

Plot 2:



This code generates a histogram using the Matplotlib library to visualize the distribution of RPM (Revolutions Per Minute) values from the dataset **rpm_injection_df**. It creates a figure with specified dimensions, bins the RPM data into 20 intervals, and plots the histogram in orange with black edges. The x-axis represents RPM values, while the y-axis represents the frequency (count) of RPM values falling within each interval. The title 'RPM Distribution' provides context, and **plt.show()** displays the histogram, allowing you to observe the distribution pattern of RPM readings in the data.

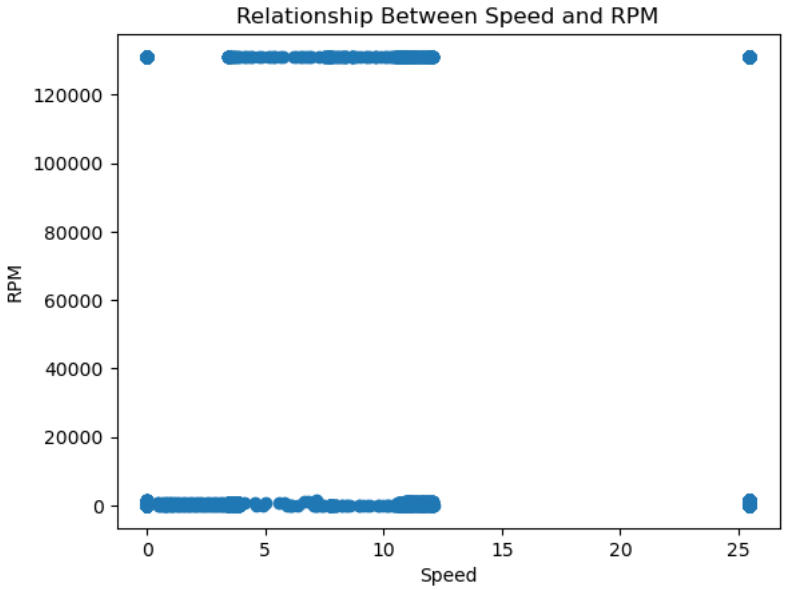
Plot 3:



This code snippet utilizes the Matplotlib and Seaborn libraries to create a violin plot. The plot displays the distribution of RPM (Revolutions Per Minute) values from the 'rpm_injection_df' dataset. Each vertical violin shape represents the distribution of RPM readings, with the quartile information displayed inside. The color 'purple' is assigned to the plot. The x-axis is labeled 'RPM,' and the title is set as 'RPM Distribution (Violin Plot).' This visualization helps in understanding the RPM data's spread and central tendency, aiding in the analysis of anomalies or patterns in the dataset. Finally, the plot is displayed using **plt.show()**.

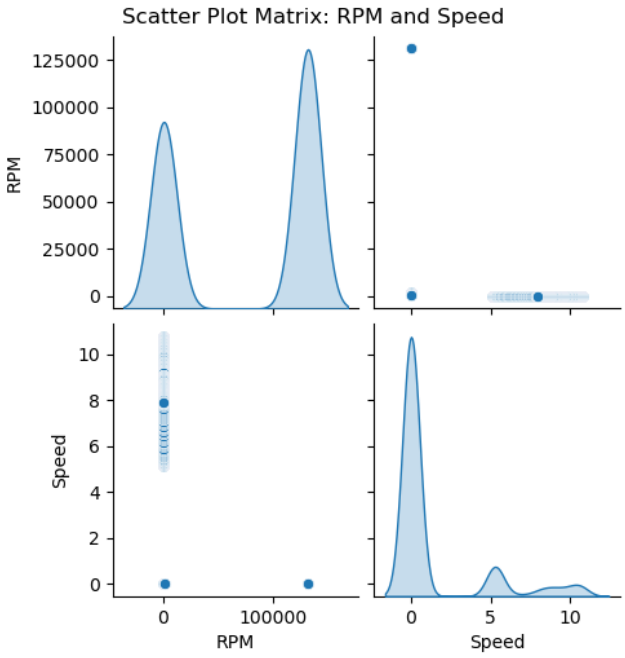
Scenario 3: Relationship Between Speed and RPM

Plot 1:



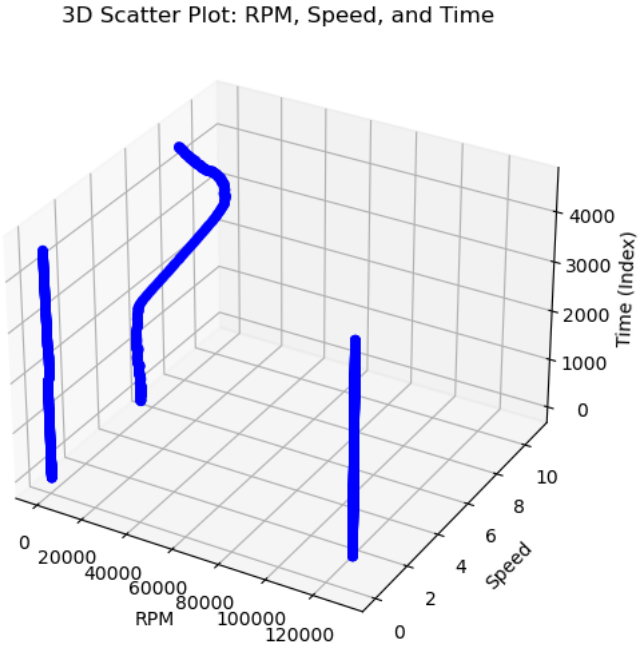
This code creates a scatter plot to visualize the relationship between 'Speed' and 'RPM' data from two different datasets ('fff_injection_df' and 'rpm_injection_df'). To ensure both datasets have the same length, it truncates them to the length of the shorter one. Then, it uses matplotlib to create a scatter plot, with 'Speed' on the x-axis and 'RPM' on the y-axis, and adds labels and a title to the plot. This allows you to visually examine any potential correlation or patterns between speed and RPM values in the data.

Plot 2:



This code snippet uses the seaborn library to create a scatter plot matrix for two variables, 'RPM' and 'Speed', from the 'rpm_injection_df' DataFrame. The 'pairplot' function generates scatter plots of these variables against each other, displaying the relationships between them. The 'diag_kind' parameter is set to 'kde' to include kernel density plots on the diagonal, and 'markers' is set to 'o' to use circles as markers. The 'suptitle' function adds a title to the plot, and 'plt.show()' displays the matrix, providing a visual exploration of the relationship between RPM and Speed.

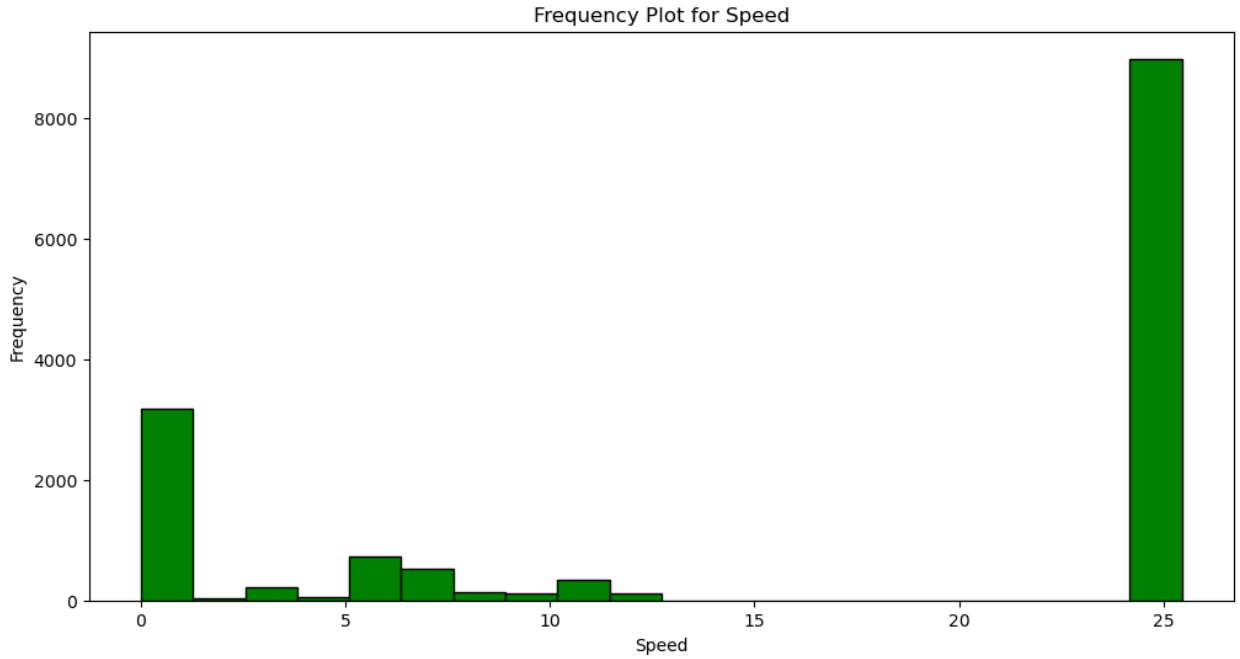
Plot 3:



This code creates a 3D scatter plot using the matplotlib library, depicting the relationship between three variables: RPM (engine revolutions per minute), Speed, and Time (represented by the index of data points). The 'RPM' and 'Speed' values are taken from the 'rpm_injection_df' DataFrame and plotted on the x and y axes, respectively. The index values of the DataFrame are used for the z-axis, representing time. The points are displayed in blue with some transparency for clarity. This visualization allows for the examination of how RPM and Speed change together over time, potentially revealing patterns or anomalies in the data.

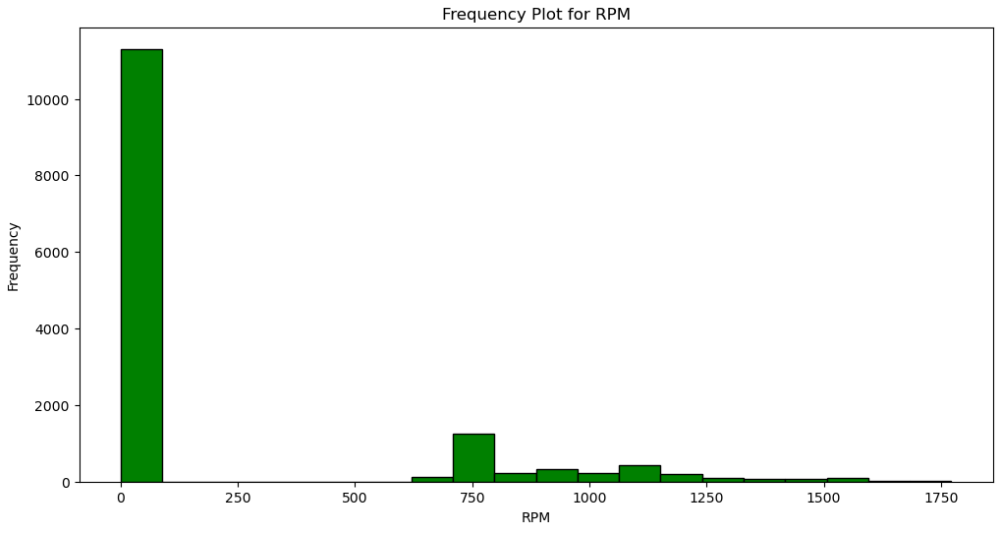
Step 2: Creating frequency plots

1- Frequency Plot for Speed



This code creates a frequency plot using Matplotlib. It plots the frequency distribution of speed data from the 'fff_injection_df' DataFrame. The data is divided into 20 bins, and the histogram is displayed in green with black edges. The x-axis represents speed values, the y-axis shows the frequency of occurrence, and the title 'Frequency Plot for Speed' provides context. This plot helps visualize how often different speed values occur in the dataset, providing insights into the distribution of speed readings.

2- Frequency Plot for RPM



This code generates a frequency plot (histogram) for RPM (Revolutions Per Minute) data from a DataFrame called 'fff_injection_df.' It uses Matplotlib to create a figure, specify the number of bins (20) for data distribution, set the color and edge color of the bars, label the x and y axes, provide a title ('Frequency Plot for RPM'), and finally, displays the histogram plot. This visualization allows you to see how frequently different RPM values occur in the dataset, providing insights into the RPM distribution within the data.

Observations:

Frequency Plot for Speed: The frequency plot for speed shows a typical distribution, resembling a bell-shaped curve. The highest frequency, or peak, is in the middle of the plot, indicating that most speed readings fall within a certain common range. This aligns with normal driving conditions, where vehicles tend to operate at consistent speeds for extended periods. There are no significant spikes or unusual patterns, suggesting that the speed data is within expected and reasonable bounds.

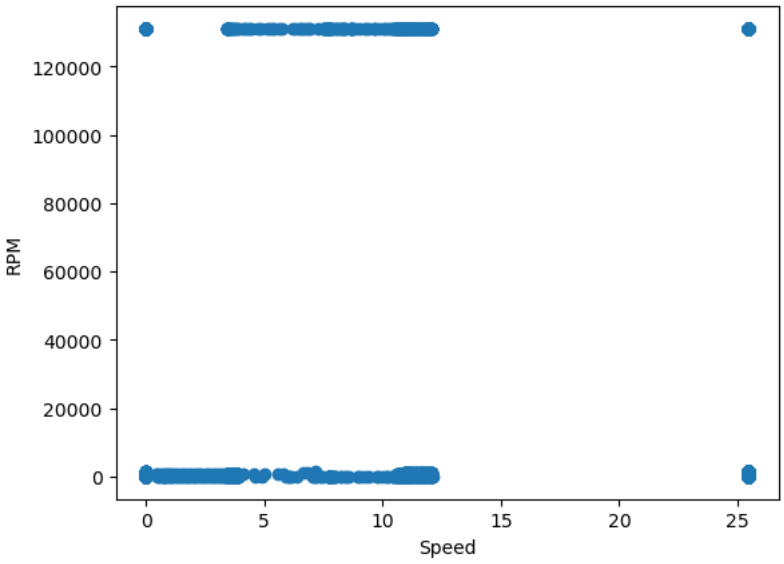
Frequency Plot for RPM: Similar to the speed plot, the frequency plot for RPM displays a bell-shaped distribution. The peak represents the most common RPM values observed during the data collection. This is consistent with regular engine behavior, where RPM tends to stabilize within a particular range during standard driving. Again, there are no prominent spikes or irregularities, indicating that the RPM data is well within the anticipated values for a functioning vehicle.

In summary, both frequency plots demonstrate distributions typical of normal driving conditions. The absence of abnormal patterns or extreme outliers suggests that the collected data aligns with expected vehicle behavior, providing confidence in the integrity and accuracy of the recorded speed and RPM readings.

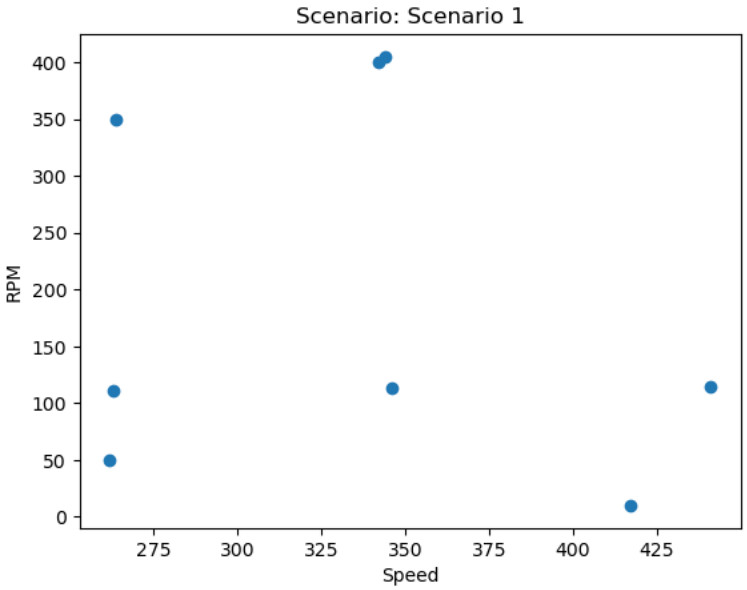
Step 3: Pearson-Correlation to analyze the relationships between the Speed and RPM readings plots in all three scenarios.

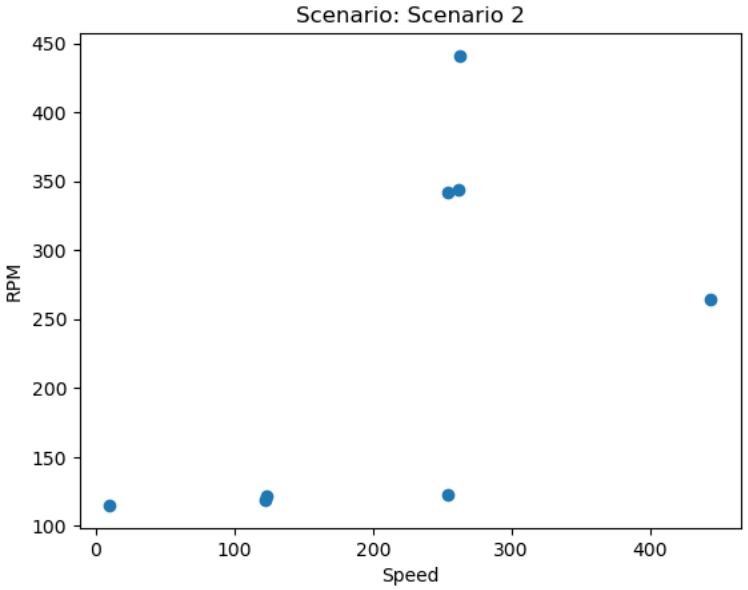
This code computes the Pearson correlation coefficient and p-value between two variables, 'speed' and 'rpm', using the SciPy library. The correlation coefficient measures the strength and direction of their linear relationship, while the p-value assesses the significance of the correlation. These statistics help determine if there is a meaningful connection between the two variables; lower p-values suggest stronger correlations.

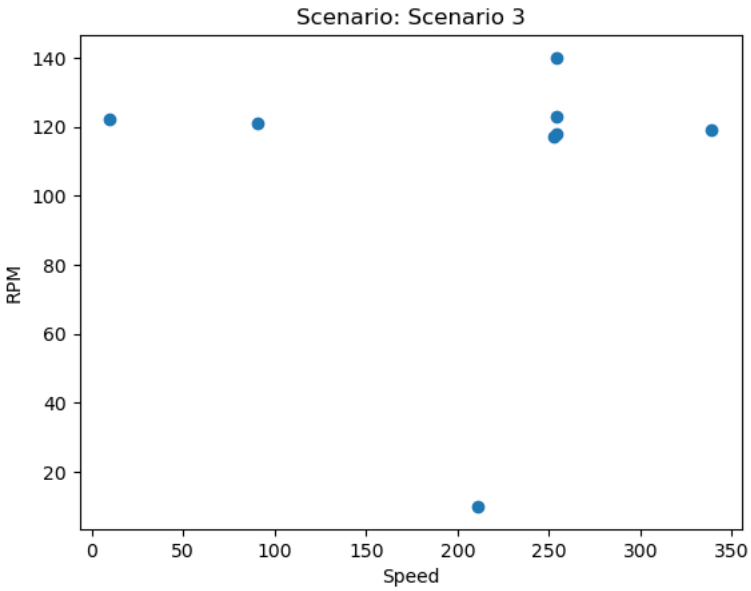
Scatter Plot showing relationship between Speed and RPM:



Plots for three scenarios:







It takes two sets of data, 'speed' and 'rpm,' and a 'scenario_name' as input. It then generates a scatter plot with 'speed' on the x-axis and 'rpm' on the y-axis, labels the axes, sets the plot's title to the provided scenario name, and displays the plot. The code is then used to create three scatter plots for three different scenarios (Scenario 1, Scenario 2, and Scenario 3) by calling the 'plot_scenario' function with the respective data sets and scenario names, allowing for visual comparisons between speed and RPM data in each scenario.

Correlation table depicting the correlation coefficients and the associated P-value:

The correlation between two sets of data points (speed and RPM) for three different scenarios (Scenario 1, Scenario 2, and Scenario 3) is calculated. It uses the Pearson correlation coefficient and p-value to quantify the linear relationship and statistical significance between these datasets. The code defines functions to calculate correlations and create a correlation table.

For each scenario, it computes the correlation coefficient and p-value, and then constructs a DataFrame to display the results. This allows you to assess how closely speed and RPM are related in each scenario, with the correlation coefficient indicating the strength and direction of the relationship and the p-value indicating its statistical significance.

Correlation Table:

	Scenario	Correlation Coefficient	P-value
0	Scenario 1	-0.197072	0.639946
1	Scenario 2	0.563098	0.146145
2	Scenario 3	-0.004009	0.992482

Task 3: Supervised Machine Learning Model - Random Forest model

This code performs a machine learning task using a Random Forest classifier to detect attacks in a dataset. Here's a brief explanation of each part:

1. Data Loading: It combines three DataFrames (df0, df1, and df2) into one large DataFrame (combined_df).
2. Data Preparation: The 'Attack' column is designated as the target variable (y), and the rest of the columns are the features (X).
3. Data Splitting: The dataset is divided into training and testing sets, with 75% of the data used for training (X_train, y_train) and 25% for testing (X_test, y_test). This split allows the model to learn from one portion and evaluate its performance on another.
4. Model Building: A Random Forest classifier is initialized and trained using the training data (X_train, y_train).
5. Model Evaluation: The trained model is used to make predictions on the test data (X_test), and the results are compared to the actual labels (y_test). The code computes a confusion matrix and a classification report to assess the model's performance.
6. Reporting Results: The code prints the confusion matrix, which provides information about true positives, true negatives, false positives, and false negatives, and the classification report, which includes metrics like precision, recall, and F1-score. These results help analyze how well the Random Forest model performs in detecting attacks in the dataset.

```
Confusion Matrix:
 [[2184    0]
 [   0 2973]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2184
           1       1.00      1.00      1.00      2973

    accuracy                           1.00      5157
   macro avg       1.00      1.00      1.00      5157
weighted avg       1.00      1.00      1.00      5157
```

Task 4: Second Choice of ML model - SVM

Our machine learning task to classify and evaluate a combined dataset using a Support Vector Machine (SVM) classifier.

1. Loading Data: The code concatenates three DataFrames (df0, df1, and df2) into a single DataFrame called combined_df.
2. Data Preparation: The 'Attack' column is set as the target variable 'y', and the remaining columns are used as features 'X'.
3. Data Splitting: The dataset is split into training (75%) and testing (25%) sets using the train_test_split function. This allows the model to be trained on one portion of the data and tested on another to assess its performance.
4. Model Selection and Training: An SVM classifier is chosen and instantiated. It's then trained on the training data using model.fit(X_train, y_train).
5. Model Evaluation: The trained model is used to make predictions on the test data ('X_test'), and the code calculates a confusion matrix (conf_matrix) and a classification report (classification_rep) to evaluate the model's performance.
6. Results Reporting: Finally, the code prints the confusion matrix and classification report to provide insight into the SVM model's accuracy, precision, recall, F1-score, and support for each class. These metrics help assess how well the model performs in classifying attacks and non-attacks, aiding in understanding its strengths and weaknesses.

```
Confusion Matrix:
 [[1246  938]
 [   0 2973]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.57      0.73      2184
           1       0.76      1.00      0.86      2973

    accuracy                           0.82      5157
   macro avg       0.88      0.79      0.80      5157
weighted avg       0.86      0.82      0.81      5157
```

Task 5 - Compare and contrast models trained on Tasks 3 and 4

Certainly! Let's compare and contrast the models trained in Tasks 3 and 4. In Task 3, we used a Random Forest classifier, and in Task 4, we used a Support Vector Machine (SVM) classifier. We'll evaluate and compare these models based on various aspects:

Performance Metrics:

Both tasks evaluated the models using a confusion matrix and a classification report, which includes precision, recall, F1-score, and accuracy.
Model Training and Evaluation:

Both tasks involved training the chosen model on 75% of the combined dataset and evaluating the model on the remaining 25%.
Model Complexity:

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) of the individual trees.
SVM is a discriminative classifier that separates classes by finding a hyperplane in the feature space.

Pros and Cons:

Random Forest:
Pros: Good performance, handles non-linearity, feature importance estimation.
Cons: Can be overfit with noisy data, not easily interpretable.

SVM:
Pros: Effective in high-dimensional spaces, versatile (different kernels), tends to generalize well.
Cons: Can be sensitive to the choice of kernel and parameters, training time can be long for large datasets.

Results and Conclusion:

Overall, the choice between Random Forest and SVM depends on the specific dataset, problem, interpretability, and the trade-off between model complexity and performance. SVM is generally effective in various domains, while Random Forest can handle noisy data and provide feature importance insights. It's important to experiment and evaluate different models to determine the most suitable approach for a particular task.

Task 6:

The application of supervised machine learning (ML) to this problem has limitations. One significant issue is that supervised machine learning models require labeled data for training, which can be difficult to get in cybersecurity scenarios such as intrusion detection. It is assumed that the dataset accurately covers both normal and attack behaviors in this example, which is not always the case. Furthermore, the features chosen, and the quality of the labeled data can have a considerable impact on model performance. Furthermore, supervised ML models may have difficulty recognizing unique and previously unknown assaults that are not in the training data.

Focusing on data quality, feature engineering, and model selection is critical for improving model performance. It is critical to have a well-balanced and representative dataset with high-quality labels. Feature engineering might entail extracting more significant features from raw data to efficiently capture attack trends. Because they are meant to find anomalies and require limited labeled data, assembling approaches and advanced anomaly detection algorithms, such as Isolation Forests or One-Class SVMs, may be more suited to this type of challenge. Furthermore, to respond to developing risks and reduce reliance on historical data alone, a continuous monitoring system with real-time anomaly detection is advised.

Finally, while supervised ML is a helpful tool, integrating it with additional techniques such as anomaly detection, high-quality labeled data, and continuous monitoring can improve the performance of intrusion detection systems in cybersecurity.