Data Preprocessing and Visualization

Vidita Dhavane

In [1]:
```
#import pandas for data manipulation
import pandas as pd
```

In [2]:
```
#load the dataset into a dataframe
dataFrame=pd.read_excel("Data Science - Intern - Data Set.xlsx")
dataFrame
```

Out[2]:

	uniqueid	ts	lat	lng	external_bat_voltage	internal_bat_voltage	engineload	tanklevel	throttle	coolant	...	live_location	to_b
0	it_220403501	1709340731	19.069340	78.355148	NaN	NaN	28	NaN	NaN	33	...	NaN	
1	it_220403501	1709340794	19.069347	78.355141	NaN	NaN	22	NaN	NaN	41	...	NaN	
2	it_220403501	1709340857	19.069414	78.355179	NaN	NaN	25	NaN	NaN	48	...	NaN	
3	it_220403501	1709340920	19.069258	78.355225	NaN	NaN	35	NaN	NaN	52	...	NaN	
4	it_220403501	1709340983	19.069094	78.355339	NaN	NaN	0	NaN	NaN	57	...	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	
18429	it_220403501	1713635865	18.869371	79.020393	NaN	NaN	50	NaN	NaN	89	...	NaN	
18430	it_220403501	1713635929	18.867130	79.018272	NaN	NaN	0	NaN	NaN	92	...	NaN	
18431	it_220403501	1713635993	18.862076	79.013924	NaN	NaN	10	NaN	NaN	87	...	NaN	
18432	it_220403501	1713636056	18.859493	79.011673	NaN	NaN	52	NaN	NaN	87	...	NaN	
18433	it_220403501	1713636120	18.858324	79.009033	NaN	NaN	72	NaN	NaN	90	...	NaN	

18434 rows × 141 columns

In [3]:
```
#overview of the dataset
dataFrame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18434 entries, 0 to 18433
Columns: 141 entries, uniqueid to adblue_level
dtypes: float64(122), int64(13), object(6)
memory usage: 19.8+ MB
```

In [4]:
```
#see the datatypes for each column
dataFrame.dtypes
```

Out[4]:
```
uniqueid                          object
ts                                 int64
lat                              float64
lng                              float64
external_bat_voltage             float64
                                  ...
boost_pressure                   float64
intake_manifold_1_temperature    float64
air_inlet_pressure               float64
exhaust_gas_temperature          float64
adblue_level                     float64
Length: 141, dtype: object
```

In [5]:
```
#check the number of rows and columns
print("Number of rows in dataset are :" ,dataFrame.shape[0])
print("Number of columns in dataset are :",dataFrame.shape[1])
```

```
Number of rows in dataset are : 18434
Number of columns in dataset are : 141
```

In [6]:
```
#check for null values
dataFrame.isnull().sum()
```

Out[6]:
```
uniqueid                             0
ts                                   0
lat                                  0
lng                                  0
external_bat_voltage             18434
                                 ...
boost_pressure                   18434
intake_manifold_1_temperature    18434
air_inlet_pressure               18434
exhaust_gas_temperature          18434
adblue_level                      3664
Length: 141, dtype: int64
```

Data Cleaning

In [7]:
```
#drop the entire column if it has null values or is empty
# Get the total number of rows
total_rows = len(dataFrame)

# Identify columns with all values as null
columns_to_drop = dataFrame.columns[dataFrame.isnull().sum() == total_rows]

# Drop those columns
dataFrame_cleaned = dataFrame.drop(columns=columns_to_drop)

print("Dropped columns:", columns_to_drop)
```

```
Dropped columns: Index(['external_bat_voltage', 'internal_bat_voltage', 'tanklevel', 'throttle',
       'intakeairtemp', 'imap', 'warmups', 'maf', 'fuelrail', 'mil',
       ...
       'request_id', 'live_location', 'to_be_expired_at', 'status', 'event_id',
       'particulated_trap_inlet_pressure', 'boost_pressure',
       'intake_manifold_1_temperature', 'air_inlet_pressure',
       'exhaust_gas_temperature'],
      dtype='object', length=108)
```

In [8]:
```
#check if the columns have been dropped successfully
dataFrame_cleaned
```

Out[8]:

	uniqueid	ts	lat	lng	engineload	coolant	engineoiltemp	vehiclespeed	rpm	obddistance	...	engine_throttle_valve1_pos
0	it_220403501	1709340731	19.069340	78.355148	28	33	1774.9688	0.00000	749.000	60780800	...	100.0
1	it_220403501	1709340794	19.069347	78.355141	22	41	1774.9688	0.00000	1208.375	60780800	...	99.6
2	it_220403501	1709340857	19.069414	78.355179	25	48	1774.9688	1.40625	752.625	60780800	...	99.6
3	it_220403501	1709340920	19.069258	78.355225	35	52	1774.9688	2.81250	996.375	60780820	...	99.6
4	it_220403501	1709340983	19.069094	78.355339	0	57	1774.9688	4.53125	1065.750	60780840	...	100.0
...	...	...	...	...	...	...	...	...	...	...	...	...
18429	it_220403501	1713635865	18.869371	79.020393	50	89	1774.9688	14.28125	945.125	71872976	...	100.0
18430	it_220403501	1713635929	18.867130	79.018272	0	92	1774.9688	38.28125	1130.750	71873344	...	100.0
18431	it_220403501	1713635993	18.862076	79.013924	10	87	1774.9688	32.34375	749.750	71874064	...	100.0
18432	it_220403501	1713636056	18.859493	79.011673	52	87	1774.9688	17.71875	820.625	71874464	...	100.0
18433	it_220403501	1713636120	18.858324	79.009033	72	90	1774.9688	16.21875	765.125	71874816	...	100.0

18434 rows × 33 columns

In [9]:
```
#check if the previous step has been executed successfully
dataFrame_cleaned.isnull().sum()
```

Out[9]:
```
uniqueid                                  0
ts                                        0
lat                                       0
lng                                       0
engineload                                0
coolant                                   0
engineoiltemp                             0
vehiclespeed                              0
rpm                                       0
obddistance                               0
runtime                                  14
engine_torque_percent                     0
selected_gear                             0
current_gear                              0
fuel_consumption                          0
fuel_level                                2
fl_level                                  0
fuel_rate                                 0
fuel_economy                              0
accelerator_pedal_pos                     0
pluscode                                  0
vibration_status                          0
can_raw_data                              0
engine_throttle_valve1_pos                0
engine_throttle_valve2_pos                0
enginefueltemp                            0
drivers_demand_engine_torque_percent      0
engine_torque_mode                        0
accelerator_pedal_pos_2                   0
brake_switch_status                       0
clutch_switch_status                      0
parking_switch_status                     0
adblue_level                           3664
dtype: int64
```

In [10]:
```
# Get the number of unique values in each column , if there is only 1 unique value in a column simply drop that as it w
on't provide any insights
unique_values_per_column = dataFrame_cleaned.nunique()
print(unique_values_per_column)
```

```
uniqueid                                    1
ts                                      18434
lat                                     15917
lng                                     15255
engineload                                101
coolant                                    68
engineoiltemp                               1
vehiclespeed                             3692
rpm                                      5600
obddistance                             16747
runtime                                  6295
engine_torque_percent                      81
selected_gear                               1
current_gear                               20
fuel_consumption                         6223
fuel_level                                250
fl_level                                  251
fuel_rate                                 573
fuel_economy                             6339
accelerator_pedal_pos                     251
pluscode                                 2639
vibration_status                            1
can_raw_data                            17874
engine_throttle_valve1_pos                112
engine_throttle_valve2_pos                  1
enginefueltemp                              1
drivers_demand_engine_torque_percent       83
engine_torque_mode                          5
accelerator_pedal_pos_2                     1
brake_switch_status                         2
clutch_switch_status                        2
parking_switch_status                       2
adblue_level                              131
dtype: int64
```

In [11]:
```
# Count the number of zeros in each column
zero_counts = (dataFrame_cleaned == 0).sum()
print(zero_counts)
```

```
uniqueid                                    0
ts                                          0
lat                                        30
lng                                        30
engineload                               2817
coolant                                     0
engineoiltemp                               0
vehiclespeed                             2237
rpm                                      1133
obddistance                                 0
runtime                                     0
engine_torque_percent                    2201
selected_gear                               0
current_gear                             5340
fuel_consumption                            0
fuel_level                                  0
fl_level                                    2
fuel_rate                                2733
fuel_economy                             1080
accelerator_pedal_pos                    8006
pluscode                                    0
vibration_status                        18434
can_raw_data                                0
engine_throttle_valve1_pos                  0
engine_throttle_valve2_pos                  0
enginefueltemp                              0
drivers_demand_engine_torque_percent     8212
engine_torque_mode                          0
accelerator_pedal_pos_2                     0
brake_switch_status                         0
clutch_switch_status                        0
parking_switch_status                       0
adblue_level                                0
dtype: int64
```

Statistical Analysis

In [12]:
```
import numpy as np
#replace zeros in lat and lgn column with mean

columns_to_fill = ['lat', 'lng']
mean_values = dataFrame_cleaned[columns_to_fill].replace(0, np.nan).mean()
dataFrame_cleaned[columns_to_fill] = dataFrame_cleaned[columns_to_fill].replace(0, mean_values)
```

In [13]:
```
# Calculate the median of non-zero values and fill colimn current_gear
median_gear = dataFrame_cleaned[dataFrame_cleaned['current_gear'] != 0]['current_gear'].median()
dataFrame_cleaned['current_gear'] = dataFrame_cleaned['current_gear'].replace(0, median_gear)
```

In [14]:
```
# Calculate the mean of non-zero values
mean_engineload = dataFrame_cleaned[dataFrame_cleaned['engineload'] != 0]['engineload'].mean()
dataFrame_cleaned['engineload'] = dataFrame_cleaned['engineload'].replace(0, mean_engineload)
```

In [15]:
```
# Find indices where 'fuel_economy'  0
indices_to_drop = dataFrame_cleaned[dataFrame_cleaned['fuel_economy'] == 0].index
dataFrame_cleaned = dataFrame_cleaned.drop(indices_to_drop)
```

In [16]:
```
#drop all the column which have only 1 unique value
dataFrame_cleaned=dataFrame_cleaned.drop(['uniqueid','engineoiltemp','selected_gear','engine_throttle_valve2_pos','engi
nefueltemp','accelerator_pedal_pos_2','vibration_status'],axis='columns')
```

In [17]:
```
print("Number of rows in dataset after cleaning are :" ,dataFrame_cleaned.shape[0])
print("Number of columns in dataset after cleaning are :",dataFrame_cleaned.shape[1])
```

```
Number of rows in dataset after cleaning are : 17354
Number of columns in dataset after cleaning are : 26
```

In [18]:
```
#check for duplicates
duplicates = dataFrame_cleaned.duplicated()
print("Number of duplicate rows:", duplicates.sum())
```

```
Number of duplicate rows: 0
```

Overview of the statistical properties

In [19]:
```
#check the overall statistics of the data
dataFrame_cleaned.describe()
```

Out[19]:

	ts	lat	lng	engineload	coolant	vehiclespeed	rpm	obddistance	runtime	engine_torque_percent
count	1.735400e+04	17354.000000	17354.000000	17354.000000	17354.000000	17354.000000	17354.000000	1.735400e+04	17342.000000	17354.000000
mean	1.711640e+09	18.751160	78.992325	46.392505	85.033652	35.433739	948.176357	6.649797e+07	2132.330588	28.792440
std	1.278968e+06	0.304981	0.385512	25.183278	7.043764	16.376169	333.256477	3.263905e+06	92.670827	22.968699
min	1.709341e+09	17.317802	78.118828	1.000000	36.000000	0.000000	0.000000	6.078080e+07	1970.850000	0.000000
25%	1.710902e+09	18.712359	78.761806	23.000000	84.000000	26.250000	767.125000	6.373202e+07	2053.162500	7.000000
50%	1.711691e+09	18.848518	79.076946	45.021131	85.000000	39.500000	979.875000	6.652453e+07	2132.950000	25.500000
75%	1.712681e+09	18.912683	79.351242	66.000000	89.000000	47.484375	1169.843750	6.940579e+07	2213.787500	49.000000
max	1.713636e+09	19.069500	79.481766	100.000000	98.000000	81.703125	1975.250000	7.187482e+07	2287.100000	80.000000

8 rows × 21 columns

Feature Engineering

In [20]:
```
#convert the ts column to date and time
from datetime import datetime

# Assuming you have a DataFrame named `dataFrame_cleaned` with a column `ts`
# Convert 'ts' from Unix timestamp to datetime
dataFrame_cleaned['ts_converted'] = pd.to_datetime(dataFrame_cleaned['ts'], unit='s')

# Print the first few rows to verify
print(dataFrame_cleaned[['ts', 'ts_converted']].head())
```

```
           ts        ts_converted
2  1709340857 2024-03-02 00:54:17
3  1709340920 2024-03-02 00:55:20
4  1709340983 2024-03-02 00:56:23
5  1709341725 2024-03-02 01:08:45
6  1709344633 2024-03-02 01:57:13
```

In [21]:
```
#extract the date only from date time format
#dataFrame_cleaned['ts_converted'] = dataFrame_cleaned['ts_converted'].dt.date
dataFrame_cleaned['ts_converted'] = dataFrame_cleaned['ts_converted'].dt.strftime('%m/%d/%Y')
```

In [22]:
```
dataFrame_cleaned
```

Out[22]:

	ts	lat	lng	engineload	coolant	vehiclespeed	rpm	obddistance	runtime	engine_torque_percent	...	pluscode	
2	1709340857	19.069414	78.355179	25.000000	48	1.40625	752.625	60780800	1970.85	18	...	7JFW3994+	0CF00300
3	1709340920	19.069258	78.355225	35.000000	52	2.81250	996.375	60780820	1970.90	27	...	7JFW3994+	0CF00300
4	1709340983	19.069094	78.355339	45.021131	57	4.53125	1065.750	60780840	1970.90	1	...	7JFW3994+	0CF00300
5	1709341725	19.069117	78.356079	22.000000	58	0.09375	750.750	60780920	1970.95	20	...	7JFW3994+	0CF00300
6	1709344633	19.069157	78.356049	45.021131	53	0.00000	0.000	60780920	1970.95	0	...	7JFW3994+	0CF00300
...	...	...	...	...	...	...	...	...	...	...	...	...	
18429	1713635865	18.869371	79.020393	50.000000	89	14.28125	945.125	71872976	2287.00	33	...	7JCXV29C+	0CF00300
18430	1713635929	18.867130	79.018272	45.021131	92	38.28125	1130.750	71873344	2287.05	0	...	7JCXV289+	0CF00300
18431	1713635993	18.862076	79.013924	10.000000	87	32.34375	749.750	71874064	2287.05	7	...	7JCXV267+	0CF00300
18432	1713636056	18.859493	79.011673	52.000000	87	17.71875	820.625	71874464	2287.05	39	...	7JCXV256+	0CF00300
18433	1713636120	18.858324	79.009033	72.000000	90	16.21875	765.125	71874816	2287.10	45	...	7JCXV255+	0CF00300

17354 rows × 27 columns

In [23]:
```
# Identify categorical columns
categorical_columns = dataFrame_cleaned.select_dtypes(include=['object', 'category']).columns

print("Categorical columns:\n", categorical_columns)
```

```
Categorical columns:
 Index(['pluscode', 'can_raw_data', 'brake_switch_status',
       'clutch_switch_status', 'parking_switch_status', 'ts_converted'],
      dtype='object')
```

In [24]:
```
#check how many unique values are present in the specific column
unique_value_counts = dataFrame_cleaned['brake_switch_status'].value_counts()
print("Unique value counts :\n", unique_value_counts)
```

```
Unique value counts :
 brake_switch_status
Released    16090
Pressed      1264
Name: count, dtype: int64
```

Converting categorical values to numeric

In [25]:
```
#converting categorical values to numeric where released=0 and pressed=1
# Define the mapping
mapping = {'Released': 0, 'Pressed': 1}

# List of columns to apply the mapping
columns_to_replace = ['brake_switch_status', 'clutch_switch_status', 'parking_switch_status']

# Replace values in the specified columns
dataFrame_cleaned[columns_to_replace] = dataFrame_cleaned[columns_to_replace].replace(mapping)

print("Updated columns:\n", dataFrame[columns_to_replace].head())
```

```
Updated columns:
   brake_switch_status clutch_switch_status parking_switch_status
0            Released             Released               Pressed
1            Released              Pressed               Pressed
2             Pressed              Pressed              Released
3            Released              Pressed              Released
4            Released              Pressed              Released
```

In [26]:
```
dataFrame_cleaned=dataFrame_cleaned.drop(columns=['ts'])
dataFrame_cleaned
```

Out[26]:

	lat	lng	engineload	coolant	vehiclespeed	rpm	obddistance	runtime	engine_torque_percent	current_gear	...	pluscode	
2	19.069414	78.355179	25.000000	48	1.40625	752.625	60780800	1970.85	18	129	...	7JFW3994+	0CF0030
3	19.069258	78.355225	35.000000	52	2.81250	996.375	60780820	1970.90	27	129	...	7JFW3994+	0CF0030
4	19.069094	78.355339	45.021131	57	4.53125	1065.750	60780840	1970.90	1	3	...	7JFW3994+	0CF0030
5	19.069117	78.356079	22.000000	58	0.09375	750.750	60780920	1970.95	20	129	...	7JFW3994+	0CF0030
6	19.069157	78.356049	45.021131	53	0.00000	0.000	60780920	1970.95	0	16	...	7JFW3994+	0CF0030
...	...	...	...	...	...	...	...	...	...	...	...	...	
18429	18.869371	79.020393	50.000000	89	14.28125	945.125	71872976	2287.00	33	11	...	7JCXV29C+	0CF0030
18430	18.867130	79.018272	45.021131	92	38.28125	1130.750	71873344	2287.05	0	16	...	7JCXV289+	0CF0030
18431	18.862076	79.013924	10.000000	87	32.34375	749.750	71874064	2287.05	7	16	...	7JCXV267+	0CF0030
18432	18.859493	79.011673	52.000000	87	17.71875	820.625	71874464	2287.05	39	13	...	7JCXV256+	0CF0030
18433	18.858324	79.009033	72.000000	90	16.21875	765.125	71874816	2287.10	45	13	...	7JCXV255+	0CF0030

17354 rows × 26 columns

In [27]:
```
# Calculate the mean of the 'adblue_level' column and substitute it in place of NaN
mean_value = dataFrame_cleaned['adblue_level'].mean()
dataFrame_cleaned.loc[:, 'adblue_level'] = dataFrame_cleaned['adblue_level'].fillna(mean_value)
print("NaN values in 'adblue_level' have been filled with the mean.")
```

```
NaN values in 'adblue_level' have been filled with the mean.
```

In [28]:
```
dataFrame_cleaned
```

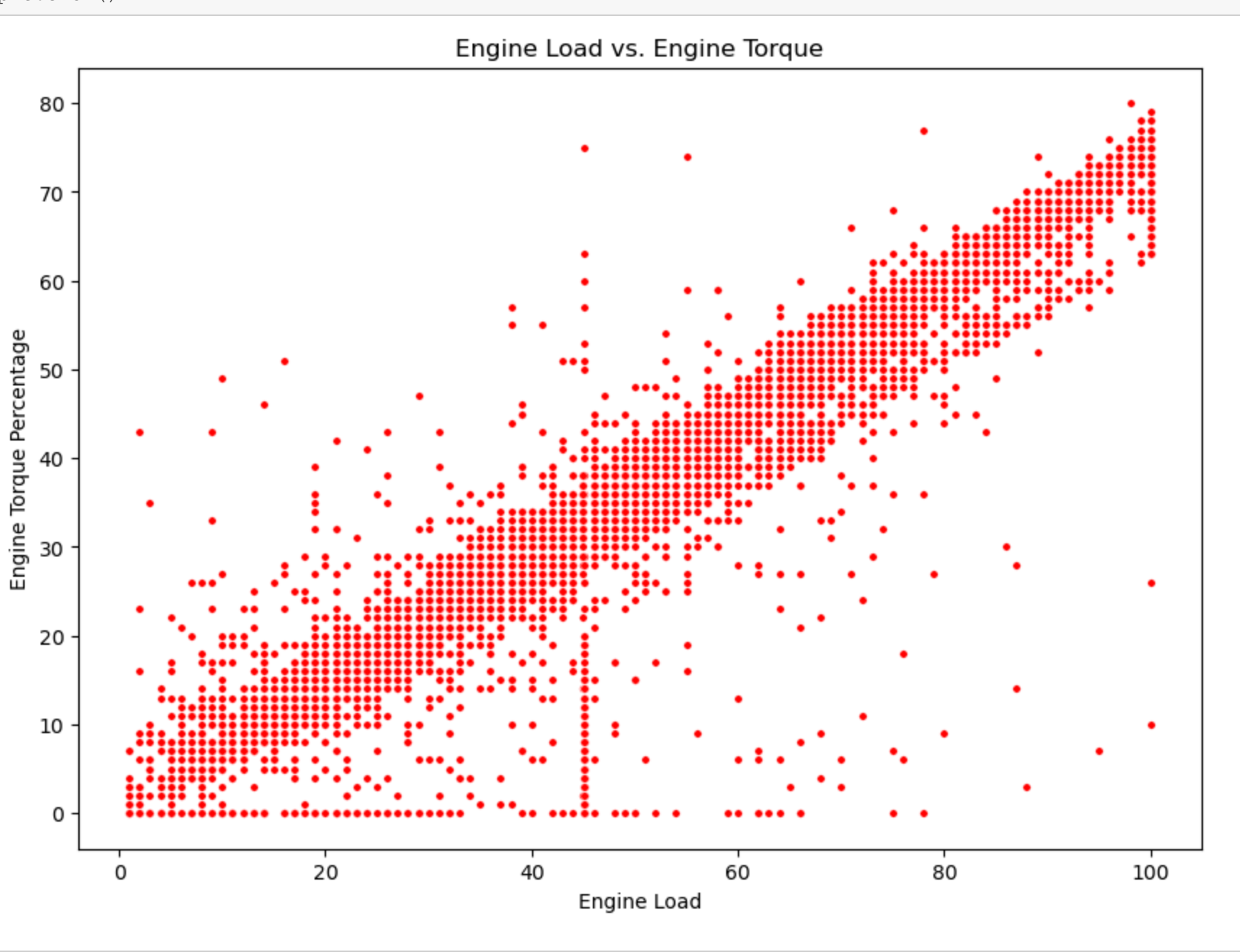
Out[28]:

	lat	lng	engineload	coolant	vehiclespeed	rpm	obddistance	runtime	engine_torque_percent	current_gear	...	pluscode	
2	19.069414	78.355179	25.000000	48	1.40625	752.625	60780800	1970.85	18	129	...	7JFW3994+	0CF0030
3	19.069258	78.355225	35.000000	52	2.81250	996.375	60780820	1970.90	27	129	...	7JFW3994+	0CF0030
4	19.069094	78.355339	45.021131	57	4.53125	1065.750	60780840	1970.90	1	3	...	7JFW3994+	0CF0030
5	19.069117	78.356079	22.000000	58	0.09375	750.750	60780920	1970.95	20	129	...	7JFW3994+	0CF0030
6	19.069157	78.356049	45.021131	53	0.00000	0.000	60780920	1970.95	0	16	...	7JFW3994+	0CF0030
...	...	...	...	...	...	...	...	...	...	...	...	...	
18429	18.869371	79.020393	50.000000	89	14.28125	945.125	71872976	2287.00	33	11	...	7JCXV29C+	0CF0030
18430	18.867130	79.018272	45.021131	92	38.28125	1130.750	71873344	2287.05	0	16	...	7JCXV289+	0CF0030
18431	18.862076	79.013924	10.000000	87	32.34375	749.750	71874064	2287.05	7	16	...	7JCXV267+	0CF0030
18432	18.859493	79.011673	52.000000	87	17.71875	820.625	71874464	2287.05	39	13	...	7JCXV256+	0CF0030
18433	18.858324	79.009033	72.000000	90	16.21875	765.125	71874816	2287.10	45	13	...	7JCXV255+	0CF0030

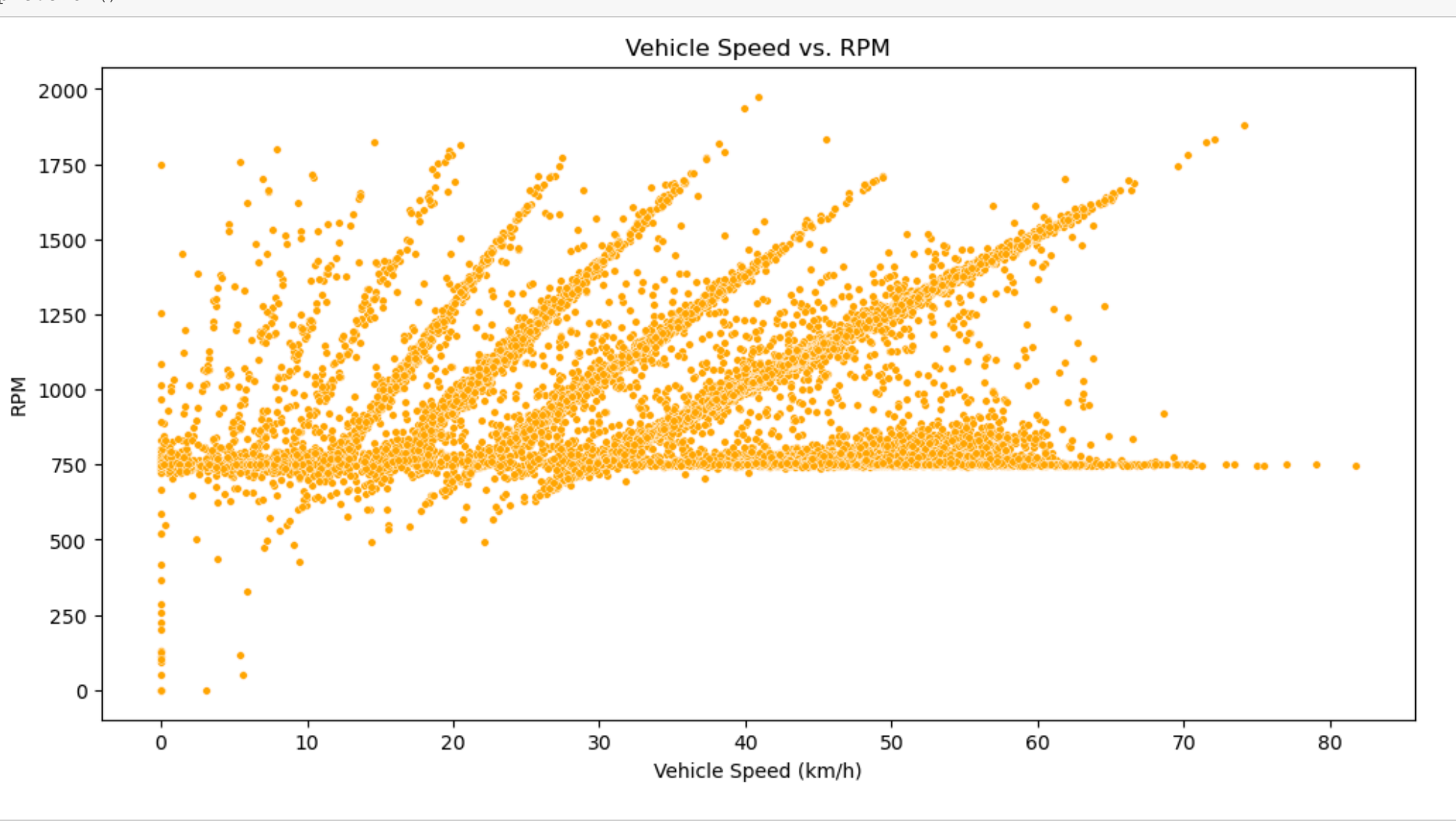
17354 rows × 26 columns

Exploratory Data Analysis

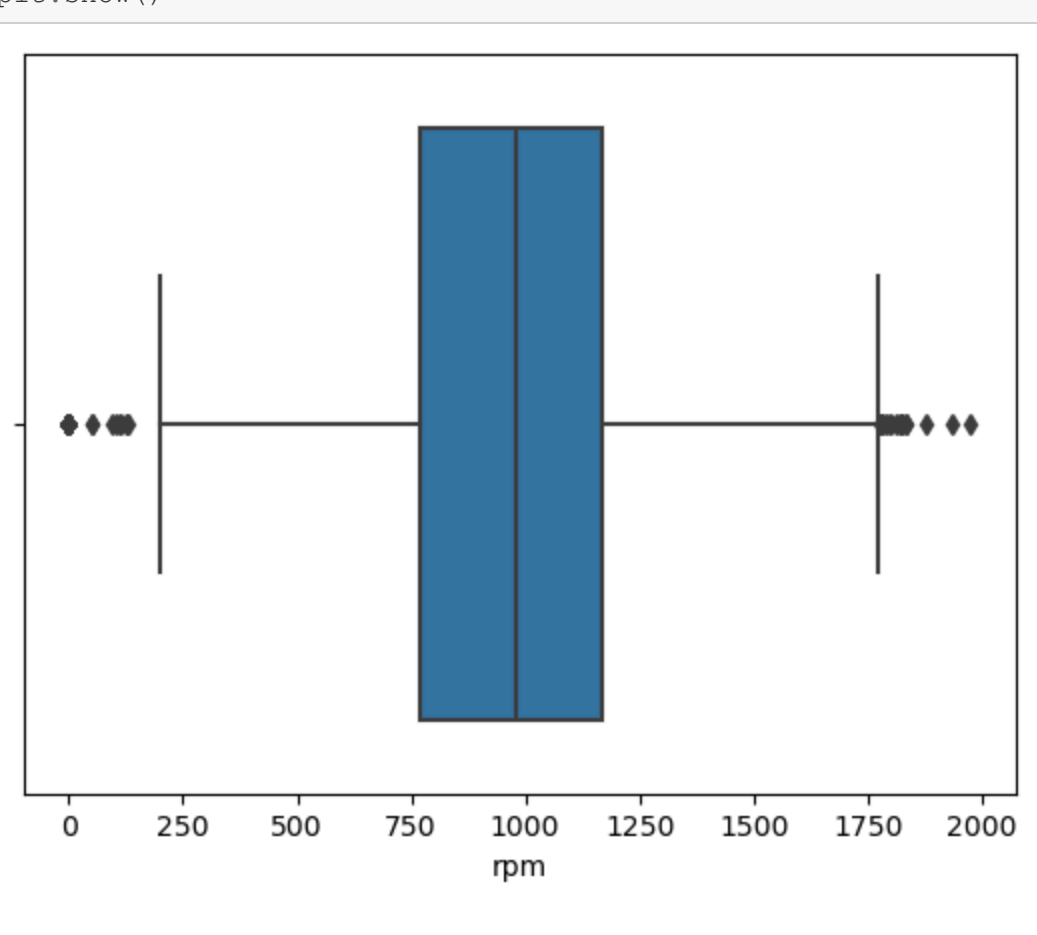
In [29]:
```
#Analyzing vehicle performance and efficiency by using the below two columns
#Engine Load and Torque
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10, 7))
sns.scatterplot(data=dataFrame_cleaned, x='engineload', y='engine_torque_percent',color='red',s=15)
plt.title('Engine Load vs. Engine Torque')
plt.xlabel('Engine Load')
plt.ylabel('Engine Torque Percentage')
plt.show()
```



In [30]:
```
# Vehicle Speed vs. RPM
plt.figure(figsize=(12, 6))
sns.scatterplot(data=dataFrame_cleaned, x='vehiclespeed', y='rpm',color='orange',s=16)
plt.title('Vehicle Speed vs. RPM')
plt.xlabel('Vehicle Speed (km/h)')
plt.ylabel('RPM')
plt.show()
```

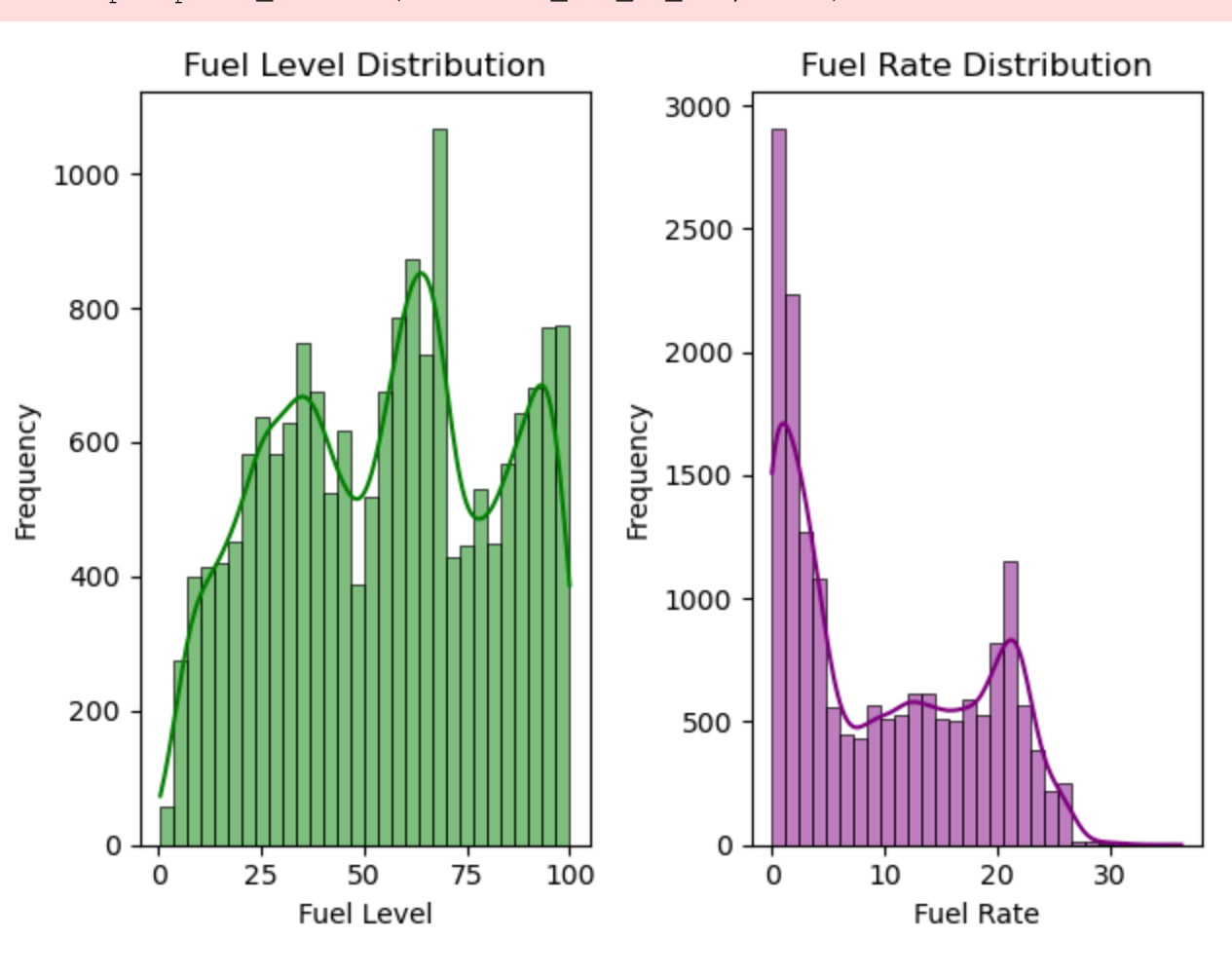


In [31]:
```
#Detecting outliers in Engine Data using boxplots which could indicate mechanical issues or unusual driving behavior.
sns.boxplot(x=dataFrame_cleaned['rpm'])
plt.show()
```

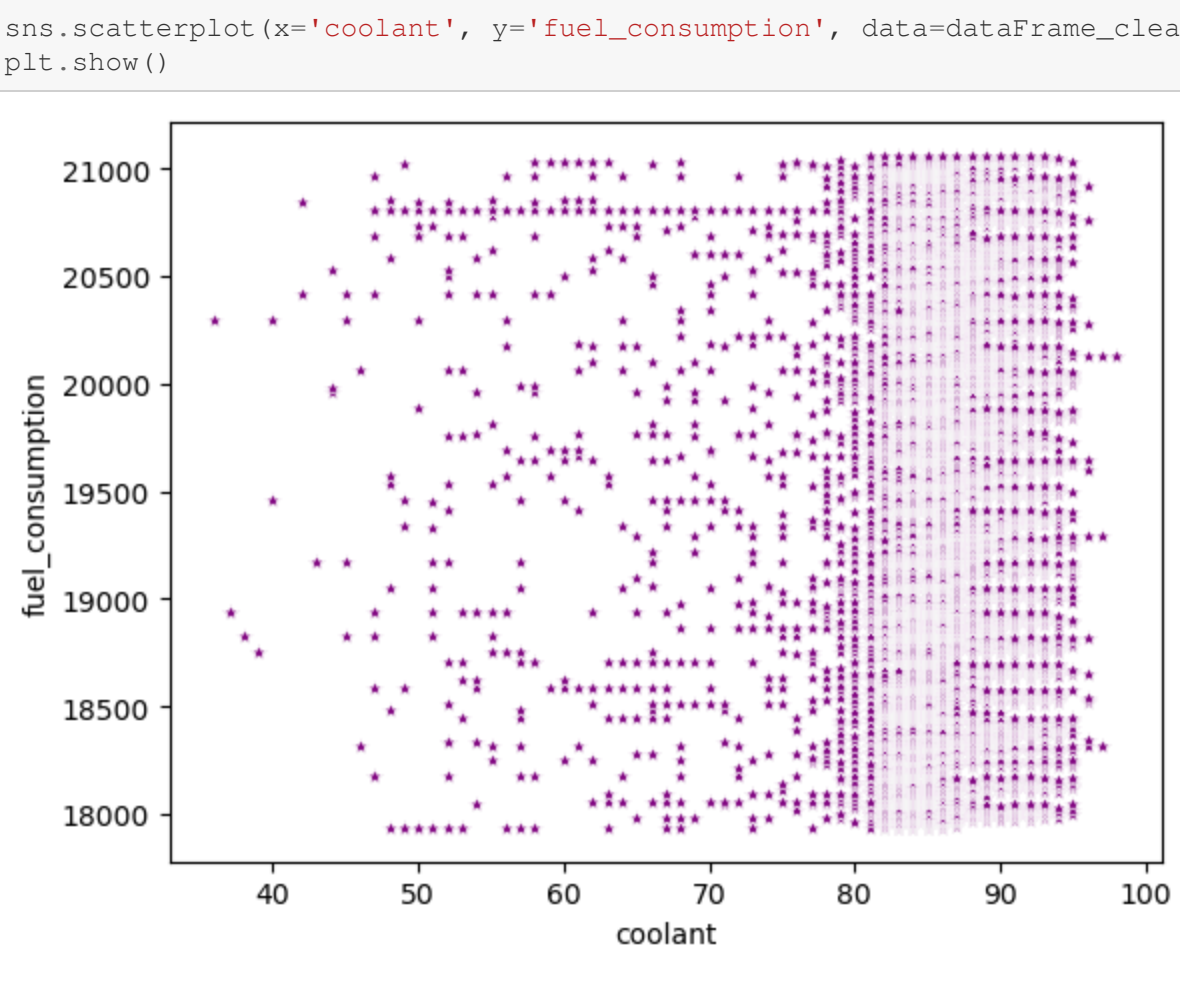


In [32]:
```
#distribution of fuel_level and fuel_rate across your dataset

# Fuel Level Histogram
plt.subplot(1, 2, 1)
sns.histplot(data=dataFrame_cleaned, x='fuel_level', bins=30, kde=True, color='green')
plt.title('Fuel Level Distribution')
plt.xlabel('Fuel Level')
plt.ylabel('Frequency')

# Fuel Rate Histogram
plt.subplot(1, 2, 2)
sns.histplot(data=dataFrame_cleaned, x='fuel_rate', bins=30, kde=True, color='purple')
plt.title('Fuel Rate Distribution')
plt.xlabel('Fuel Rate')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

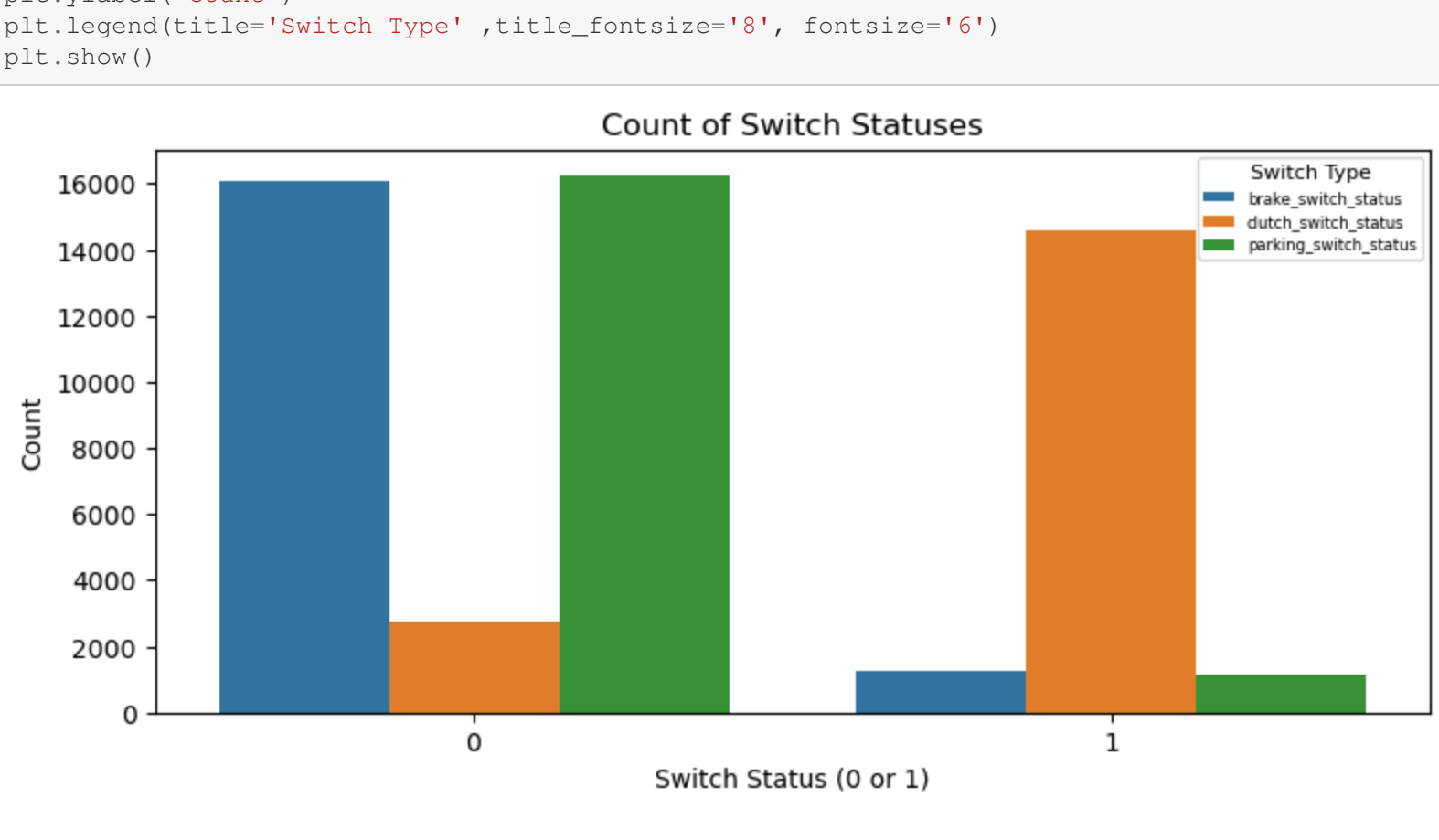
```
C:\Users\Vivek\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecate
d and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Vivek\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecate
d and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [33]:
```
#Impact of Environmental Factors
#Coolant Temperature and Engine Performance: Exploring the relationship between coolant temperature and fuel_consumptio
n

sns.scatterplot(x='coolant', y='fuel_consumption', data=dataFrame_cleaned, marker='*',color='purple')
plt.show()
```

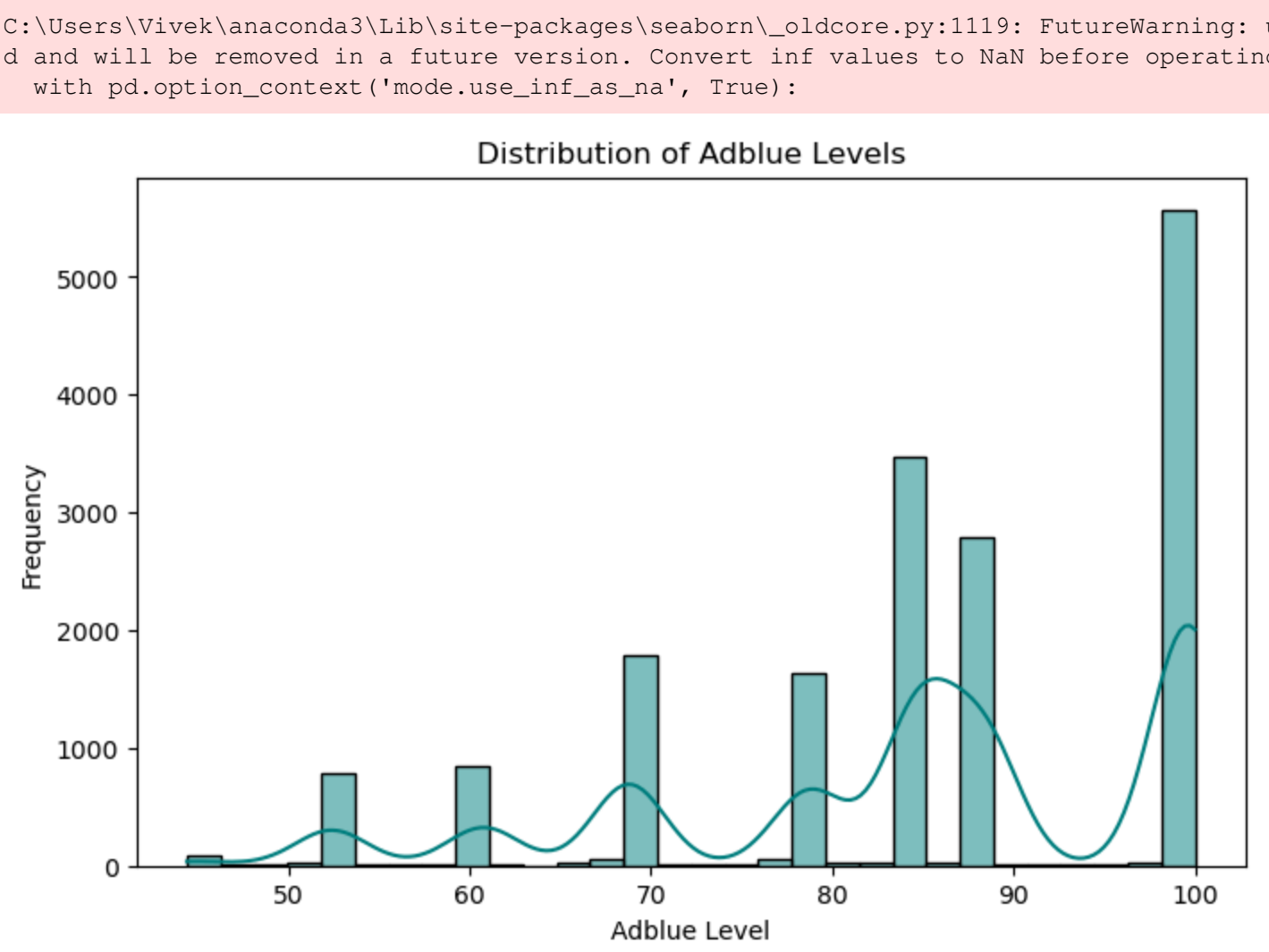


In [34]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Aggregate the counts of 0 and 1 for each column
status_counts = {
    'brake_switch_status': dataFrame_cleaned['brake_switch_status'].value_counts(),
    'clutch_switch_status': dataFrame_cleaned['clutch_switch_status'].value_counts(),
    'parking_switch_status': dataFrame_cleaned['parking_switch_status'].value_counts()
}

# Convert to DataFrame for easier plotting
status_df = pd.DataFrame(status_counts).fillna(0).astype(int)
status_df = status_df.reset_index().melt(id_vars='index', var_name='Status Type', value_name='Count')
status_df.columns = ['Switch Status', 'Status Type', 'Count']

# Plot
plt.figure(figsize=(9, 4))
sns.barplot(data=status_df, x='Switch Status', y='Count', hue='Status Type')
plt.title('Count of Switch Statuses')
plt.xlabel('Switch Status (0 or 1)')
plt.ylabel('Count')
plt.legend(title='Switch Type' ,title_fontsize='8', fontsize='6')
plt.show()
```



In [35]:
```
#The distribution, trends, and relationships of Adblue Level in the dataset.

plt.figure(figsize=(8, 5))
sns.histplot(dataFrame_cleaned['adblue_level'].dropna(), bins=30, kde=True,color='teal')
plt.title('Distribution of Adblue Levels')
plt.xlabel('Adblue Level')
plt.ylabel('Frequency')
plt.show()
```

```
C:\Users\Vivek\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecate
d and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [37]:
```
#saved the file to perform further analysis in PowerBI
dataFrame_cleaned.to_excel('C:/Users/Vivek/Desktop/Taabi mobolity/dataFrame_cleanedm.xlsx', index=False)
```

In [ ]:

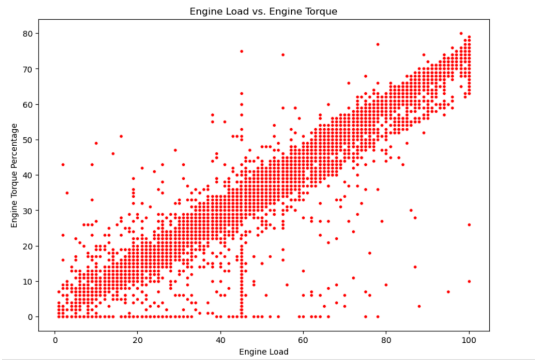
Report

1. **Data loading and preparation:**
    - Data loading and preparation done which included identification of variables , determining the size of dataset ,identifying missing values etc

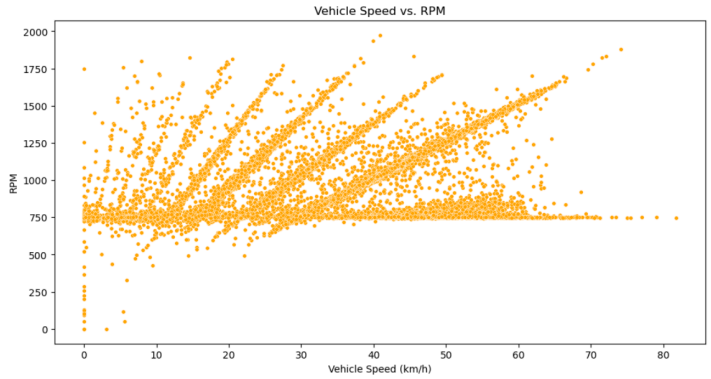
2. **Data cleaning and preprocessing:**
    - Handled missing values
    - Converted data type . Converted the columns brake_switch_status , clutch_switch_status and parking_switch_status values to numeric values 0 and 1 . The values were initially categorical i.e pressed and released .
    - Handled duplicate values

3. **Exploratory data analysis:**
    - Prepared visuals for data analysis like scatter plots, column charts etc

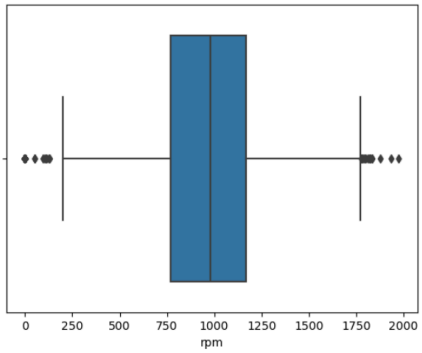
    1. **Analyzing vehicle performance and efficiency by using the below two columns**
       Engine Load and Torque



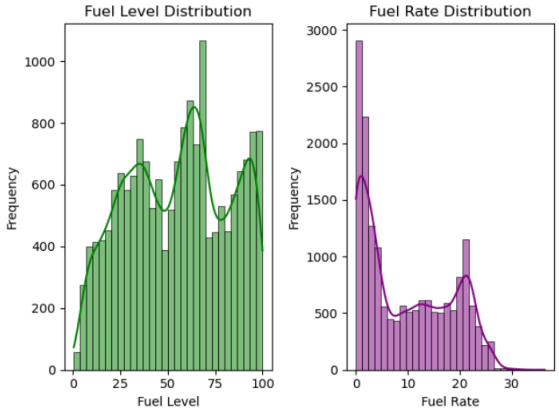
    2. **The scatter plot visualizes the relationship between vehicle speed and RPM. It helps identify any correlation or pattern between these two variables, such as whether higher speeds correspond to higher RPMs or if there's a consistent trend.**



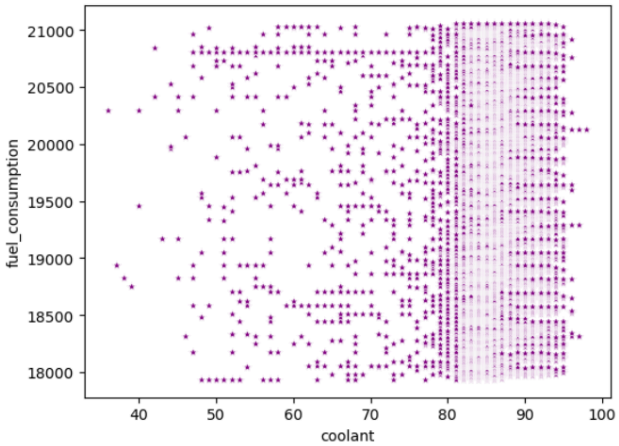
3. **Detecting outliers in Engine Data using boxplots which could indicate mechanical issues or unusual driving behavior. i.e outliers**



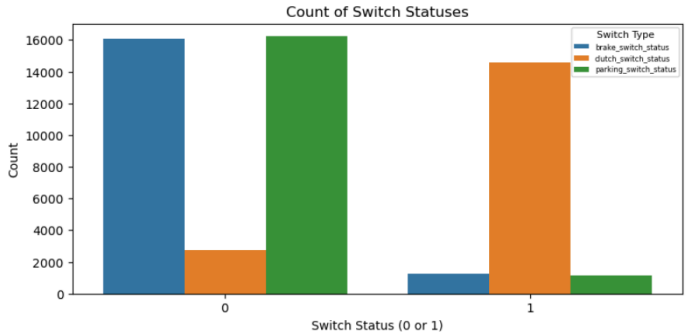
4. **Distribution of fuel_level and fuel_rate**



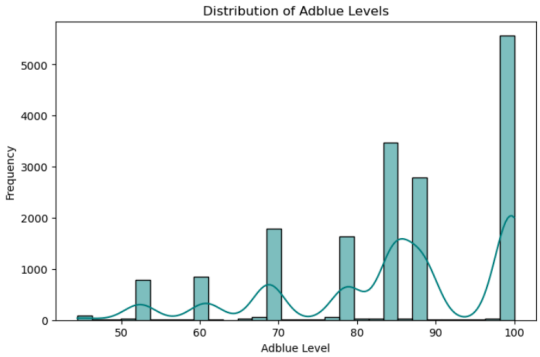
5. **Impact of Environmental Factors**
   Coolant Temperature and Engine Performance: Exploring the relationship between coolant temperature and fuel_consumption



6. **The bar chart shows the frequency of 'Pressed' (1) and 'Released' (0) statuses for the brake, clutch, and parking switches, revealing their activity patterns. It helps compare the usage or engagement levels of each switch type.**



7. **The distribution, trends, and relationships of Adblue Level in the dataset.**



4. **Feature Engineering:**
   - Created a new feature ts_converted where the timestamp column ts was converted to a proper date and time format .

5. Insights and Recommendations:

Insights from the data:

1. **Fuel Efficiency**:
    - **Fuel Consumption and Coolant**: A higher coolant temperature might indicate more engine strain, leading to increased fuel consumption.
    - **Optimal RPM and Speed**: Keeping the vehicle within a moderate RPM and speed range improves fuel economy.
2. **Vehicle Performance**:
    - **RPM and Torque Correlation**: High RPM results in increased torque, but excessive RPM can reduce fuel efficiency.
    - **Current Gear and Speed**: Higher gears at appropriate speeds optimize both fuel consumption and vehicle performance.
3. **Safety**:
    - **Brake Usage**: Frequent brake usage may indicate aggressive driving, which can reduce fuel efficiency and lead to quicker wear on brake components.
4. **Emissions**:
    - **Adblue Level**: Maintaining optimal levels can help in reducing NOx emissions, crucial for environmental regulations compliance.

Recommendations:

1. **Improving Fuel Efficiency**:
    - Encourage drivers to maintain a moderate RPM (e.g., between 2,000 and 3,000) for optimal fuel consumption.
    - Educate drivers about coasting and accelerating smoothly to minimize sudden brake usage, which can save fuel.
2. **Enhancing Vehicle Performance**:
    - Encourage timely gear shifting and avoid running the engine at too high an RPM for prolonged periods.
    - Regularly monitor coolant temperature and engine load to prevent engine overheating, which can strain performance.
3. **Boosting Safety**:
    - Introduce driver training on defensive driving techniques, such as minimizing aggressive braking and acceleration.
    - Implement automated warnings when clutch or brake switches are pressed excessively.
4. **Reducing Emissions**:
    - Regularly check and maintain AdBlue levels in vehicles to ensure compliance with emission standards.
    - Monitor and replace worn components that affect engine load and fuel consumption, reducing emissions further.

Below is the PowerBI dashboard for better visualization.