taabi
AI Powered Mobility Solutions

TAABI MOBILITY

SINCE 2022

VEHICLE DATASET REPORT

Submitted By : Subhajeet Khawas
Roll no. 23MB0063
23mb0063@iitism.ac.in

Objective:

Taabi Solutions, a SaaS company, leverages IoT data to monitor fleet operations in real-time. The aim of this analysis is to identify factors affecting fleet performance, reduce operational costs, and enhance uptime.

Data analysis:

The analysis consists of several key steps as outlined below:

1. Importing necessary libraries:

The first step in the coding process involved importing essential Python libraries that facilitate data manipulation, statistical analysis, and visualization. The primary libraries used include:

- Pandas: For data manipulation and analysis, particularly for handling DataFrames.
- NumPy: For numerical computing and array operations.
- Scikit-learn: For implementing machine learning algorithms and model evaluation.
- Matplotlib and Seaborn: For data visualization to illustrate key findings.
- Plotly Express and Graph Objects: For creating interactive visualizations and dashboards.
- SciPy: For statistical computations and advanced scientific analysis.
- Warnings (part of Python Standard Library): For suppressing unnecessary warnings to maintain clean outputs.

2. Loading the master excel file

The master excel file named “Data Science - Intern Finals - Data Set.xlsx” was loaded into python for carrying out analysis

3. Descriptive statistics

The shape of the dataset was obtained (18434, 141)

The dataset contains 18434 rows and 141 columns

Datatypes of all the variables was noted as follows:

```
[194]: df.dtypes

[194]: ts                                          datetime64[ns]
       lat                                                float64
       lng                                                float64
       engineload                                         float64
       coolant                                            float64
       vehiclespeed                                       float64
       rpm                                                float64
       obddistance                                        float64
       runtime                                            float64
       engine_torque_percent                              float64
       current_gear                                       float64
       fuel_consumption                                   float64
       fuel_level                                         float64
       fl_level                                           float64
       fuel_rate                                          float64
       fuel_economy                                       float64
       accelerator_pedal_pos                              float64
       pluscode                                            object
       can_raw_data                                        object
       engine_throttle_valve1_pos                         float64
       drivers_demand_engine_torque_percent               float64
       engine_torque_mode                                   int64
       brake_switch_status                                 object
       clutch_switch_status                                object
       parking_switch_status                               object
       adblue_level                                       float64
       speed_to_rpm_ratio                                 float64
       fuel_efficiency_score                              float64
       throttle_efficiency                                float64
       runtime_hours                                      float64
       is_accelerating                                      int64
       is_stationary                                        int64
       region_code                                         object
       low_adblue_alert                                     int64
       efficiency_score                                   float64
       aggressive_acceleration                              int64
       stop_frequency                                       int64
       distance_covered                                   float64
       rpm_to_load_ratio                                  float64
       fuel_consumption_per_hour                          float64
       fuel_performance_ratio                             float64
       hour                                                 int32
       min                                                  int32
       sec                                                  int32
       month                                                int32
       day                                                  int32
       dayofweek                                            int32
       speed_rolling_avg                                  float64
       stop_cluster                                         int32
       dtype: object
```

A summary of statistical metrics for the numeric columns of the dataframe was generated as follows:

```
df.describe()
```

	ts	lat	lng	external_bat_voltage	internal_bat_voltage	engineload	tanklevel	throttle	coolant	intakeairtemp	..
count	1.843400e+04	18434.000000	18434.000000	0.0	0.0	18434.000000	0.0	0.0	18434.000000	0.0	..
mean	1.711641e+09	18.724254	78.865351	NaN	NaN	38.141206	NaN	NaN	84.571336	NaN	..
std	1.275937e+06	0.816184	3.208032	NaN	NaN	29.889358	NaN	NaN	7.718346	NaN	..
min	1.709341e+09	0.000000	0.000000	NaN	NaN	0.000000	NaN	NaN	30.000000	NaN	..
25%	1.710903e+09	18.701308	78.758909	NaN	NaN	10.000000	NaN	NaN	83.000000	NaN	..
50%	1.711692e+09	18.853032	79.092408	NaN	NaN	31.000000	NaN	NaN	85.000000	NaN	..
75%	1.712682e+09	18.921385	79.352821	NaN	NaN	64.000000	NaN	NaN	89.000000	NaN	..
max	1.713636e+09	19.069500	79.481804	NaN	NaN	100.000000	NaN	NaN	98.000000	NaN	..

8 rows x 135 columns

4. Handling null columns (Columns containing only null values)

Calculate the proportion (or percentage) of missing values in each column of the dataframe and many columns had 100% null values within them.
These columns are removed because such columns do not provide any useful information for analysis or modelling.

Also these columns have zero variance, they will take up storage and memory, thereby increasing computation power so these are removed

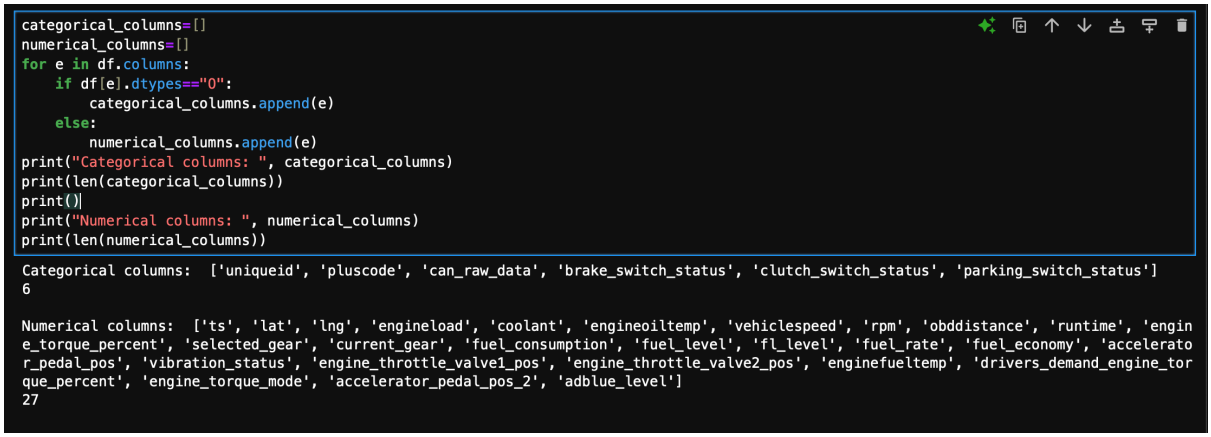
As a result 108 columns are removed from the dataframe
The resultant dataframe has a shape of (18434, 33)
The remaining columns are as follows:

```
Data columns (total 33 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   uniqueid                              18434 non-null  object
 1   ts                                    18434 non-null  int64
 2   lat                                   18434 non-null  float64
 3   lng                                   18434 non-null  float64
 4   engineload                            18434 non-null  int64
 5   coolant                               18434 non-null  int64
 6   engineoiltemp                         18434 non-null  float64
 7   vehiclespeed                          18434 non-null  float64
 8   rpm                                   18434 non-null  float64
 9   obddistance                           18434 non-null  int64
 10  runtime                               18420 non-null  float64
 11  engine_torque_percent                 18434 non-null  int64
 12  selected_gear                         18434 non-null  int64
 13  current_gear                          18434 non-null  int64
 14  fuel_consumption                      18434 non-null  float64
 15  fuel_level                            18432 non-null  float64
 16  fl_level                              18434 non-null  float64
 17  fuel_rate                             18434 non-null  float64
 18  fuel_economy                          18434 non-null  float64
 19  accelerator_pedal_pos                 18434 non-null  float64
 20  pluscode                              18434 non-null  object
 21  vibration_status                      18434 non-null  int64
 22  can_raw_data                          18434 non-null  object
 23  engine_throttle_valve1_pos            18434 non-null  float64
 24  engine_throttle_valve2_pos            18434 non-null  int64
 25  enginefueltemp                        18434 non-null  int64
 26  drivers_demand_engine_torque_percent  18434 non-null  int64
 27  engine_torque_mode                    18434 non-null  int64
 28  accelerator_pedal_pos_2               18434 non-null  int64
 29  brake_switch_status                   18434 non-null  object
 30  clutch_switch_status                  18434 non-null  object
 31  parking_switch_status                 18434 non-null  object
 32  adblue_level                          14770 non-null  float64
dtypes: float64(14), int64(13), object(6)
```

5. Categorical and Numerical columns identification

With the following code, categorical and numerical columns are segregated from the dataframe.

6 categorical columns and 27 numerical columns were obtained



```
categorical_columns=[]
numerical_columns=[]
for e in df.columns:
    if df[e].dtypes=="O":
        categorical_columns.append(e)
    else:
        numerical_columns.append(e)
print("Categorical columns: ", categorical_columns)
print(len(categorical_columns))
print()
print("Numerical columns: ", numerical_columns)
print(len(numerical_columns))
```

```
Categorical columns:  ['uniqueid', 'pluscode', 'can_raw_data', 'brake_switch_status', 'clutch_switch_status', 'parking_switch_status']
6

Numerical columns:  ['ts', 'lat', 'lng', 'engineload', 'coolant', 'engineoiltemp', 'vehiclespeed', 'rpm', 'obddistance', 'runtime', 'engin
e_torque_percent', 'selected_gear', 'current_gear', 'fuel_consumption', 'fuel_level', 'fl_level', 'fuel_rate', 'fuel_economy', 'accelerato
r_pedal_pos', 'vibration_status', 'engine_throttle_valve1_pos', 'engine_throttle_valve2_pos', 'enginefueltemp', 'drivers_demand_engine_tor
que_percent', 'engine_torque_mode', 'accelerator_pedal_pos_2', 'adblue_level']
27
```

6. Identifying and removing columns containing only 1 unique value

These columns are needed to be removed as these columns have no predictive power or variance and are redundant also.

Hence these columns are also removed.

As a result 108 columns are removed from the dataframe

The resultant dataframe has a shape of (18434, 26)

```
]: # Identify columns with only one unique value
columns_with_one_unique_value = [col for col in df.columns if df[col].nunique() == 1]

# Print the columns with only one unique value
print("Columns with only one unique value:")
print(columns_with_one_unique_value)

print(f"Original dataframe shape: {df.shape}")
```

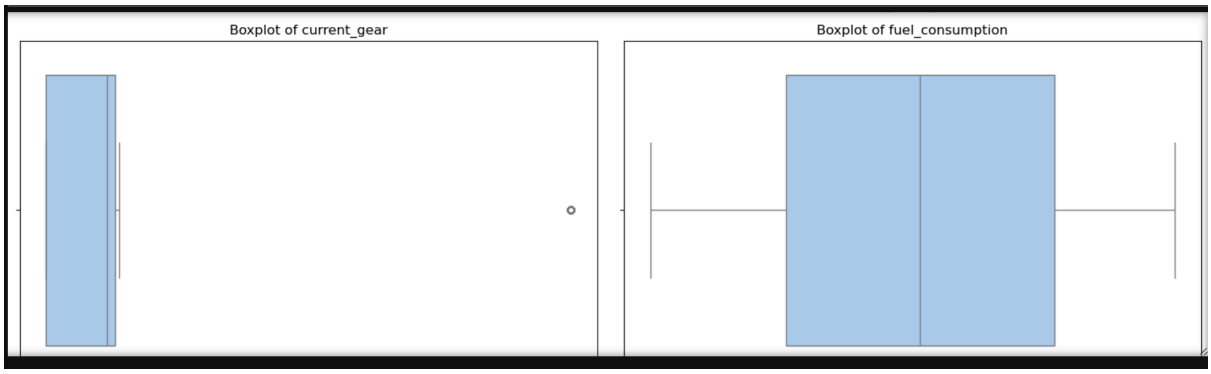
```
Columns with only one unique value:
['uniqueid', 'engineoiltemp', 'selected_gear', 'vibration_status', 'engine_throttle_valve2_pos', 'enginefueltemp', 'accelerator_pedal_pos_
2']
Original dataframe shape: (18434, 33)
```

```
]: # Remove these columns from the dataframe
df = df.drop(columns=columns_with_one_unique_value)

# Verify the removal
print(f"Updated dataframe shape: {df.shape}")
```

```
Updated dataframe shape: (18434, 26)
```

7. Outlier detection and removal

First outliers are checked using boxplots



To confirm the presence of outliers IQR method is used:

```
: # Detect outliers using IQR
for col in numerical_columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    outliers = df[(df[col] < (Q1 - 1.5 * IQR)) | (df[col] > (Q3 + 1.5 * IQR))]
    print(f"Outliers in {col}: {len(outliers)}")
```

```
Outliers in ts: 0
Outliers in engineload: 0
Outliers in coolant: 1084
Outliers in vehiclespeed: 0
Outliers in rpm: 1161
Outliers in obddistance: 0
Outliers in runtime: 0
Outliers in engine_torque_percent: 0
Outliers in current_gear: 474
Outliers in fuel_consumption: 0
Outliers in fuel_level: 0
Outliers in fl_level: 0
Outliers in fuel_rate: 0
Outliers in fuel_economy: 2837
Outliers in accelerator_pedal_pos: 0
Outliers in engine_throttle_valve1_pos: 1485
Outliers in drivers_demand_engine_torque_percent: 0
Outliers in adblue_level: 0
```

The columns containing outliers are then identified and then handled via IQR method and Winsorization (values greater than Upper Limit are replaced by Upper Limit and values less than Lower Limit are replaced by Lower Limit)

```
# Remove outliers using IQR and Winsorization
for e in outlier_columns:
    q1=np.quantile(df[e],0.25)
    q3=np.quantile(df[e],0.75)
    iqr=q3-q1
    lb=q1-1.5*iqr
    ub=q3+1.5*iqr
    #df[e]=np.where(((df[e]>ub) | (df[e]<lb)),np.median(df[e]),df[e])
    df[e]=np.where(df[e]<lb,lb,df[e])
    df[e]=np.where(df[e]>ub,ub,df[e])
```

```
# Detect outliers using IQR
for col in outlier_columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    outliers = df[(df[col] < (Q1 - 1.5 * IQR)) | (df[col] > (Q3 + 1.5 * IQR))]
    print(f"Outliers in {col}: {len(outliers)}")
```

```
Outliers in coolant: 0
Outliers in current_gear: 0
Outliers in fuel_economy: 0
```

8. Handling missing values

3 columns contain missing values

runtime contains 0.0759%, fuel_level contains 0.0108% and adblue_level contains 19.8763% missing values respectively.

```
df.isnull().mean()
```

```
ts                                          0.000000
lat                                         0.000000
lng                                         0.000000
engineload                                  0.000000
coolant                                     0.000000
vehiclespeed                                0.000000
rpm                                         0.000000
obddistance                                 0.000000
runtime                                     0.000759
engine_torque_percent                       0.000000
current_gear                                0.000000
fuel_consumption                            0.000000
fuel_level                                  0.000108
fl_level                                    0.000000
fuel_rate                                   0.000000
fuel_economy                                0.000000
accelerator_pedal_pos                       0.000000
pluscode                                    0.000000
can_raw_data                                0.000000
engine_throttle_valve1_pos                  0.000000
drivers_demand_engine_torque_percent        0.000000
engine_torque_mode                          0.000000
brake_switch_status                         0.000000
clutch_switch_status                        0.000000
parking_switch_status                       0.000000
adblue_level                                0.198763
dtype: float64
```

These missing values are imputed using the KNN Imputation method.

KNN imputation for missing values

```
from sklearn.impute import KNNImputer

# Step 1: Select only the numerical columns
numerical_data = df[numerical_columns[1:]]

# Step 2: Initialize the KNN Imputer
knn_imputer = KNNImputer(n_neighbors=5)  # You can adjust n_neighbors as needed

# Step 3: Perform KNN imputation
imputed_data = knn_imputer.fit_transform(numerical_data)

# Step 4: Replace the original numerical columns with imputed data
df[numerical_columns[1:]] = imputed_data

# Display a preview of the dataframe
df.head()
```

9. New features construction

New Feature	Reason for creation
speed_to_rpm_ratio	Analyses how effectively the engine's revolutions (RPM) translate into vehicle speed, helping to detect inefficiencies during high RPM and low-speed conditions.
fuel_efficiency_score	Measures fuel efficiency by combining fuel rate and fuel economy. Helps identify inefficient driving conditions like aggressive acceleration or high-speed operation.
throttle_efficiency	Quantifies the relationship between engine load and throttle input. Provides insights into driver behaviour and engine stress during acceleration.
runtime_hours	Converts total runtime from seconds to hours for better readability and easier interpretation in time-based analyses like fuel consumption or wear-and-tear over long trips.
is_accelerating	Identifies periods of acceleration based on changes in accelerator pedal position. Helps in studying aggressive driving behaviour and its impact on fuel efficiency and vehicle health.

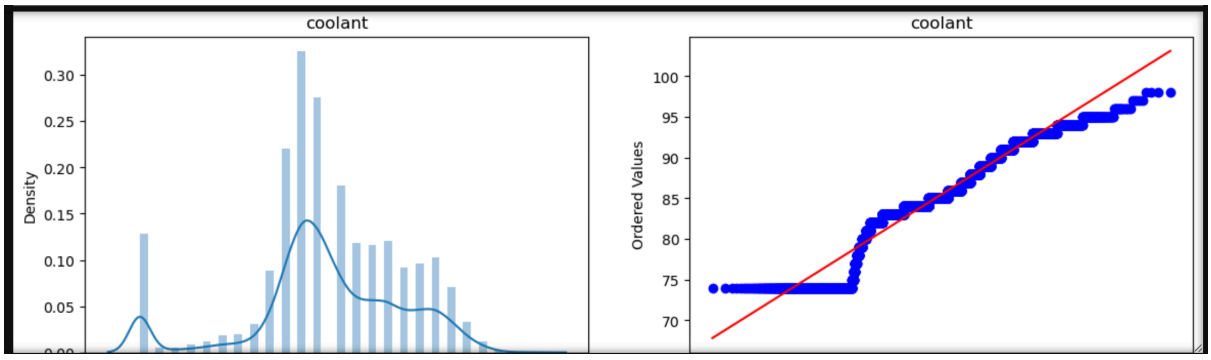
region_code	Detects stationary periods using latitude and longitude data. Useful for analyzing idle times, identifying inefficiencies in routes, and optimizing vehicle operations.
is_stationary	Extracts the geographical region from the pluscode. Enables clustering and analyzing performance metrics region-wise for targeted improvements.
low_adblue_alert	Flags low AdBlue levels for emission compliance and operational health. Ensures timely refills to avoid performance degradation or compliance violations.
efficiency_score	Combines speed, fuel economy, and engine load into a single metric for evaluating overall driving efficiency. Higher scores indicate efficient driving patterns.
aggressive_acceleration	Flags significant changes in acceleration, indicating aggressive driving behaviour. Helps identify risky driving practices that could lead to higher fuel consumption and maintenance costs.
stop_frequency	Tracks the number of stops made by the vehicle. Useful for analyzing stop-and-go patterns, route inefficiencies, or operational delays.
distance_covered	Calculates the distance covered between consecutive data points. Helps in trip analysis and monitoring vehicle performance over specific intervals.
rpm_to_load_ratio	Evaluates the stress on the engine by comparing RPM to engine load. Higher ratios may indicate overworked engines or inefficient operation, especially under heavy loads.
fuel_consumption_per_hour	Quantifies fuel usage over time by combining fuel rate and runtime. Helps correlate fuel efficiency with operational duration and trip planning.
fuel_performance_ratio	Assesses fuel economy relative to speed and engine load, helping to determine optimal driving conditions for better fuel performance.
hour	Extracts the hour from the timestamp to analyse time-of-day patterns. Helps identify operational peaks or inefficiencies during specific hours, such as high idle times during rush hours.

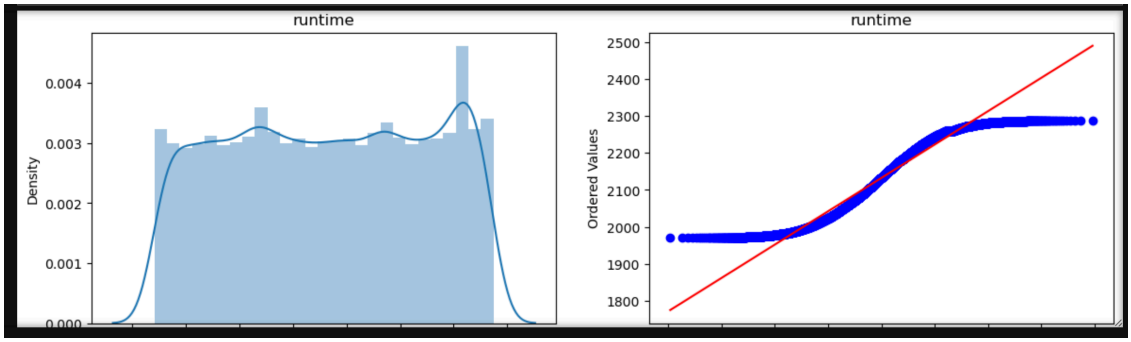
min	Extracts the minute from the timestamp for granular time-based analysis. Useful for detecting trends in short time intervals, such as sudden accelerations or stops.
sec	Extracts the second from the timestamp to capture high-resolution time events. Enables precise identification of rapid changes in vehicle behaviour, such as aggressive acceleration.
month	Identifies the month of operation. Helps in seasonal trend analysis, such as variations in fuel efficiency or vehicle usage across different months.
day	Extracts the day of the month for detailed time analysis. Useful for studying daily operational patterns and detecting anomalies on specific days.
dayofweek	Determines the day of the week. Enables analysis of weekly trends, such as higher activity on weekdays compared to weekends, or identifying maintenance scheduling patterns.

10. Exploratory Data Analysis (EDA)

10.1 Normality check

To start off the EDA, we first check if the data is normally distributed or not, we check via plotting the pdf of the variables as well as QQ plots, below graphs show 2-4 variables (graphs plotted for all variables in the attached ipynb file)

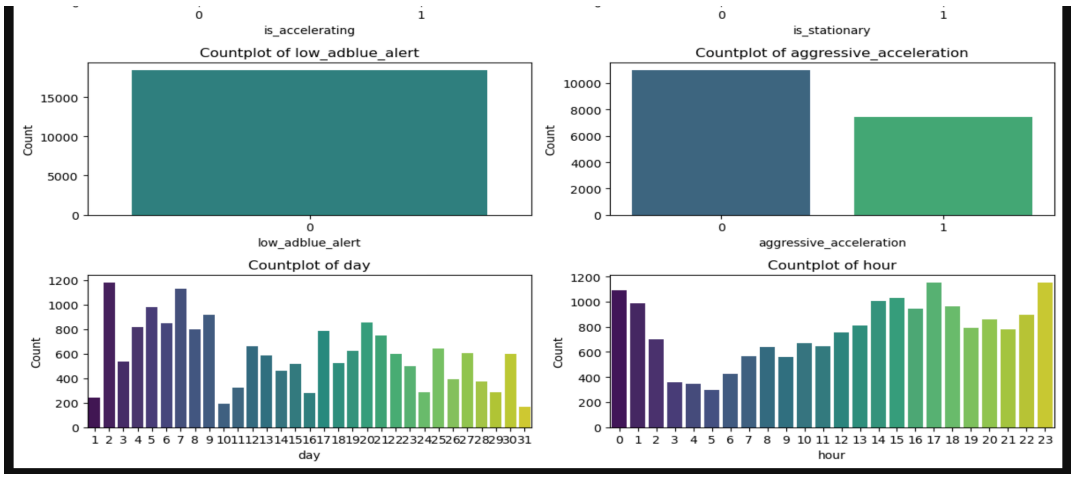




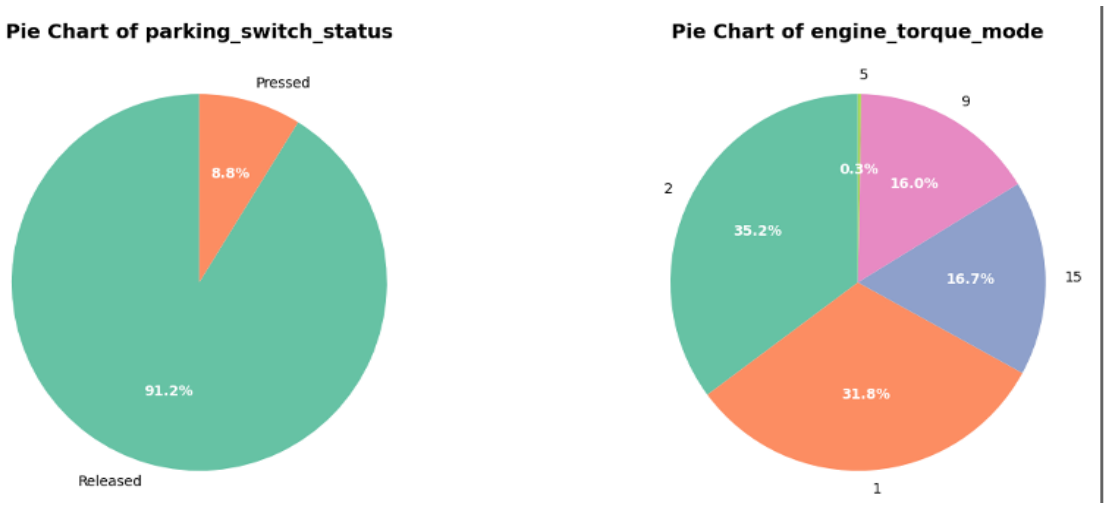
Most of the variables are not normally distributed. Hence many statistical tests and assumptions fail on these variables (like Z score method to handle outliers)

10.2 Univariate Analysis

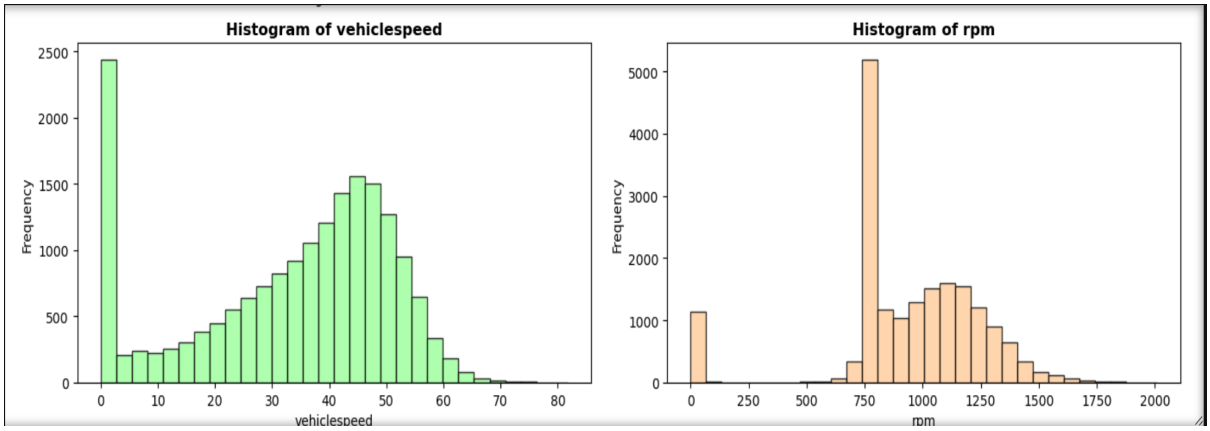
For the categorical variables, countplots are plotted to show the unique values (and their number) of the categories present in these variables



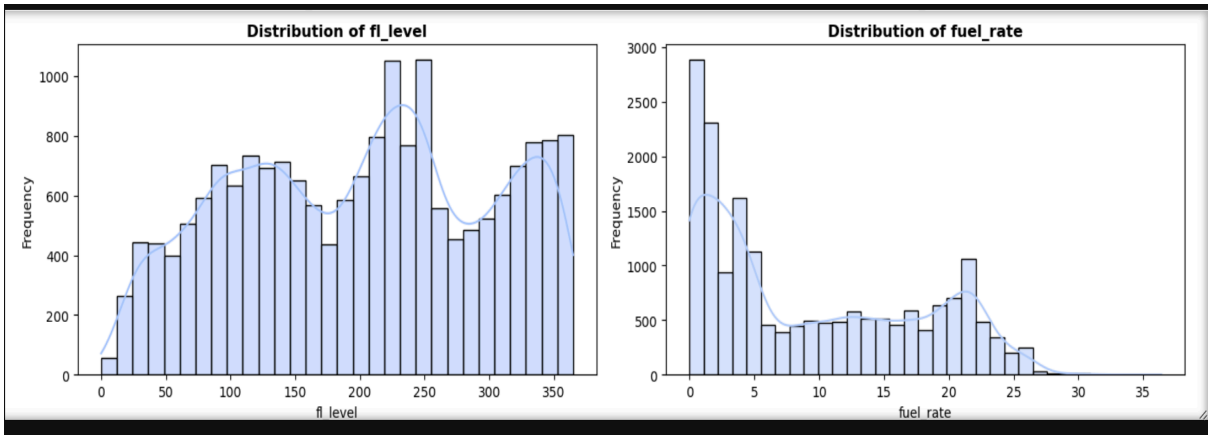
Pie charts are also plotted to get the above numbers in terms of percentages for a better visualisation



For the numerical columns, histograms are plotted for every variable



Distplots were also plotted for the numerical columns



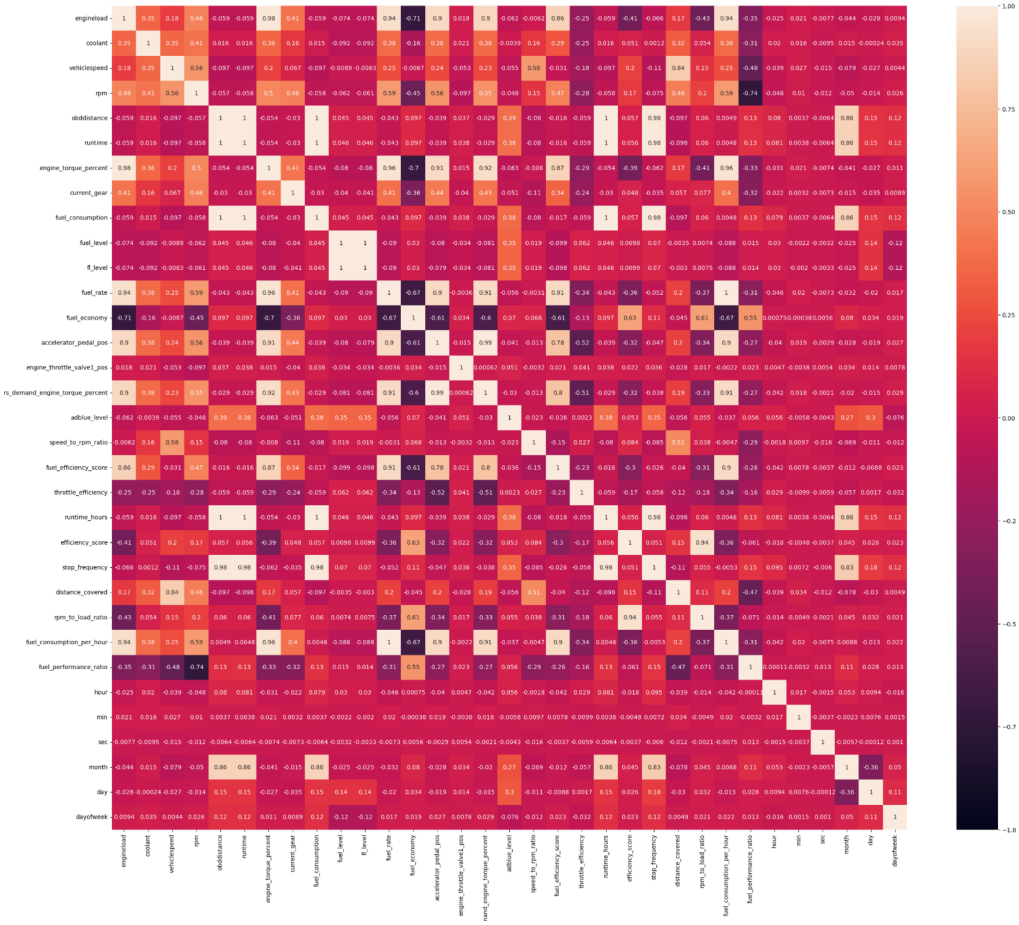
10.3 Multivariate Analysis

Heatmap of the correlation matrix was plotted for the numerical columns of the dataframe

High (very strong) correlation found between variables are as follows (Correlation coefficient > 0.8):

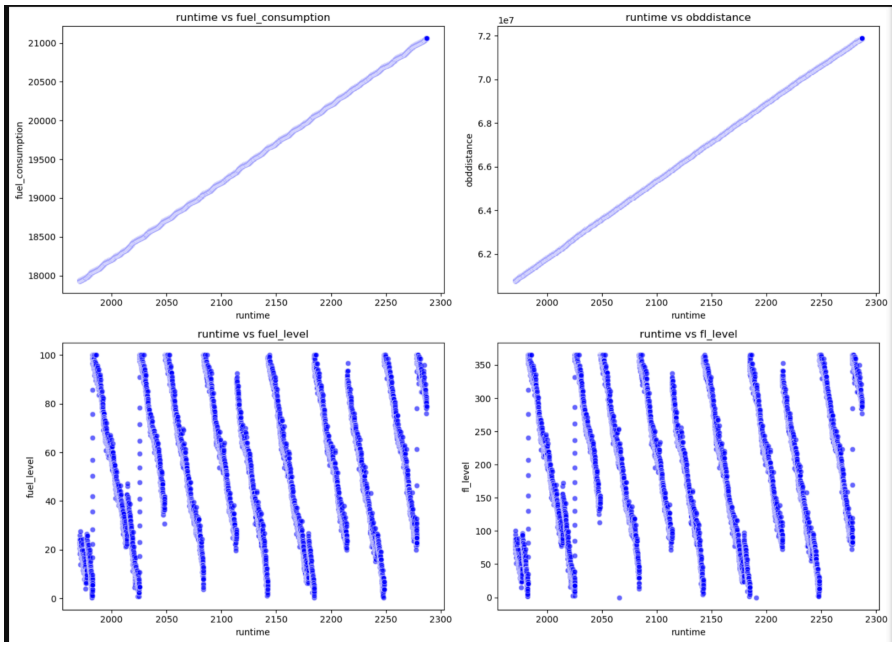
Column1	Column2	Correlation Coefficient
engineload	engine_torque_percent	0.982212198
engineload	fuel_rate	0.939174548
engineload	accelerator_pedal_pos	0.902727264
engineload	drivers_demand_engine_torque_percent	0.897413092
engineload	fuel_efficiency_score	0.859622868
engineload	fuel_consumption_per_hour	0.936209044
vehiclespeed	distance_covered	0.838592381
obddistance	runtime	0.999966697
obddistance	fuel_consumption	0.999959372
obddistance	runtime_hours	0.999966697
obddistance	stop_frequency	0.983811955
obddistance	month	0.862987728
runtime	fuel_consumption	0.999970385
runtime	stop_frequency	0.98487781
runtime	month	0.863256483
engine_torque_percent	fuel_rate	0.959264935
engine_torque_percent	accelerator_pedal_pos	0.910516449
engine_torque_percent	drivers_demand_engine_torque_percent	0.923055821
engine_torque_percent	fuel_efficiency_score	0.874899768
engine_torque_percent	fuel_consumption_per_hour	0.956358521
fuel_consumption	runtime_hours	0.999970385
fuel_consumption	stop_frequency	0.98444613
fuel_consumption	month	0.863571176
fuel_level	fl_level	0.999603972

fuel_rate	accelerator_pedal_pos	0.904892057
fuel_rate	drivers_demand_engine_torque_percent	0.912269648
fuel_rate	fuel_efficiency_score	0.905472674
fuel_rate	fuel_consumption_per_hour	0.997978345
accelerator_pedal_pos	drivers_demand_engine_torque_percent	0.985563635
accelerator_pedal_pos	fuel_consumption_per_hour	0.901607521
drivers_demand_engine_torque_percent	fuel_consumption_per_hour	0.909719167
fuel_efficiency_score	fuel_consumption_per_hour	0.904720612
runtime_hours	stop_frequency	0.98487781
runtime_hours	month	0.863256483
efficiency_score	rpm_to_load_ratio	0.939415073
stop_frequency	month	0.82992234



The top 5 features having the most correlation among themselves, are noted and scatter plots are plotted amongst themselves in order to verify the correlation coefficients and find the relationship among these variables.

The results are as follows:



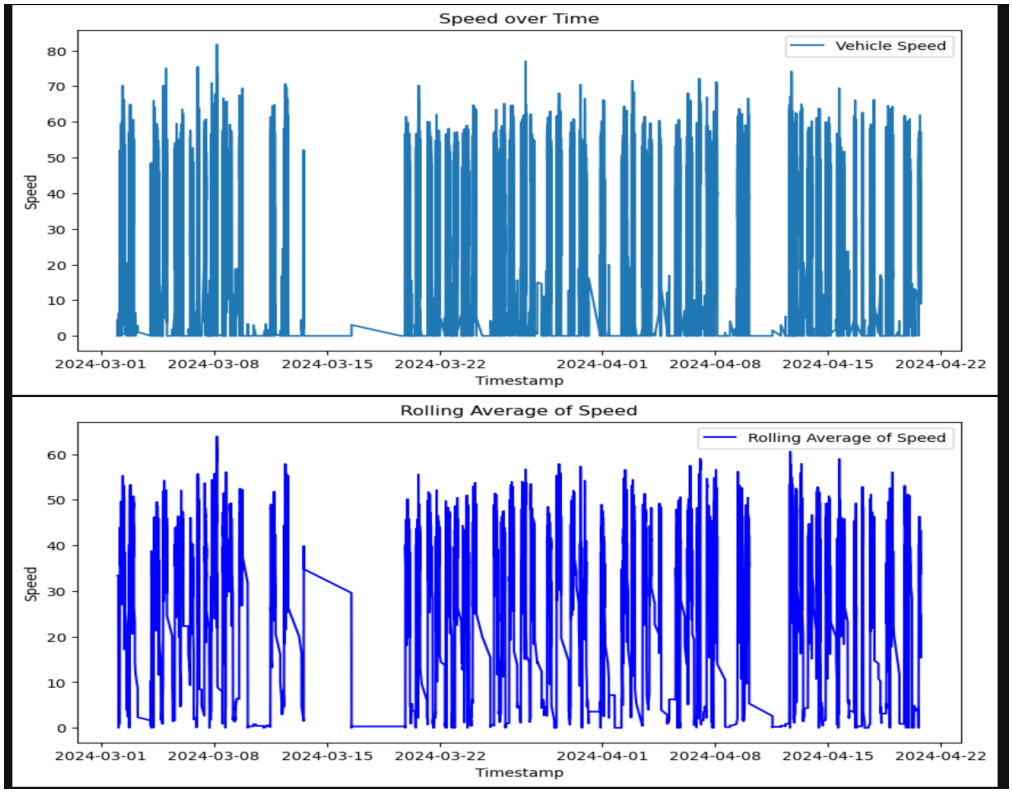
10.4 Time Series Analysis

Speed Over Time (Top Plot):

1. Shows erratic speed variations, highlighting inconsistent driving patterns.
2. Periods of inactivity are evident with flat or zero values.
3. Gaps suggest potential data loss or vehicle downtime.

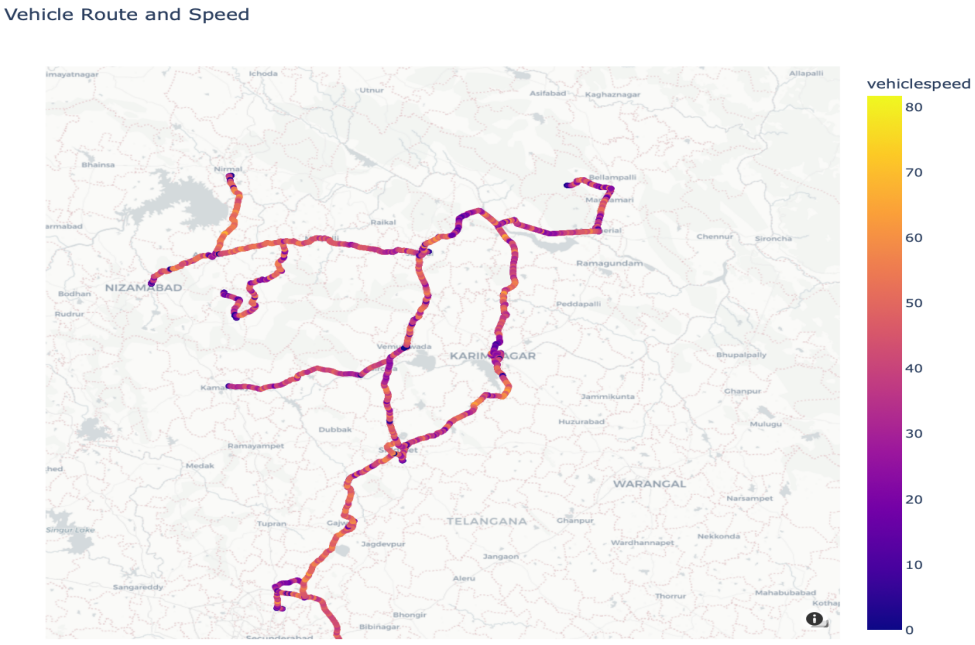
Rolling Average of Speed (Bottom Plot):

1. Smooths fluctuations, revealing broader trends in speed behaviour.
2. Highlights periods of consistent speed and abrupt changes.

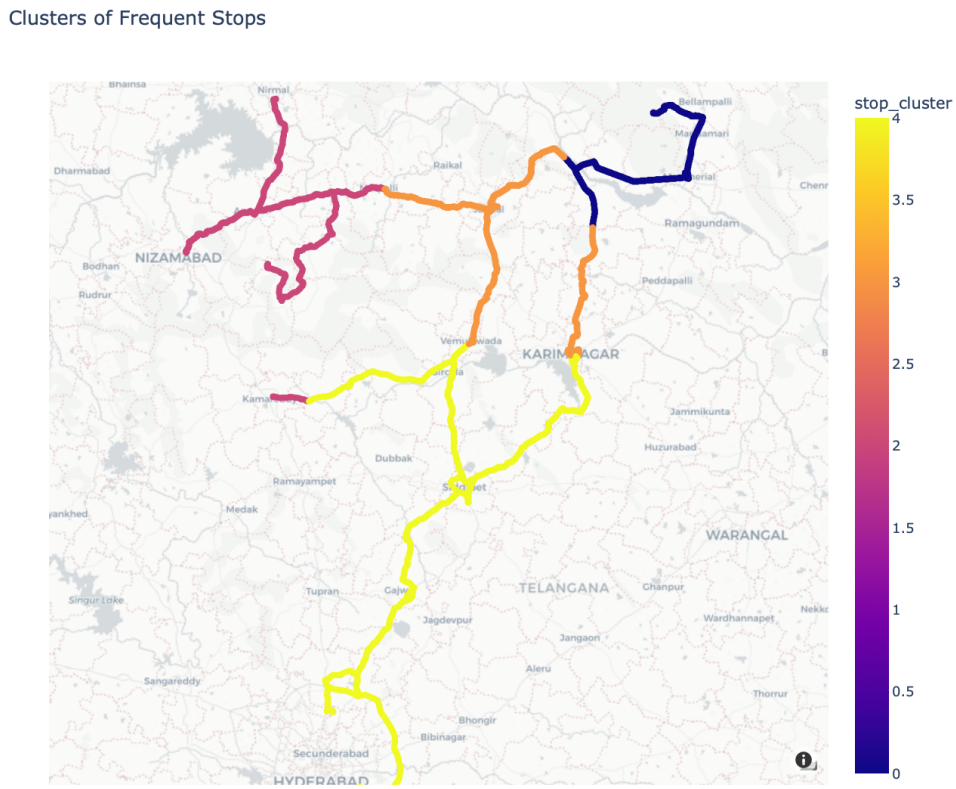


10.5 Geo Spatial Analysis

The plot maps vehicle routes with speed as a color gradient. High-speed segments are in yellow, while low-speed segments are in purple. This visualization identifies speed variations across routes, highlights traffic-prone areas, and helps analyze driving patterns geographically.



The below plot groups locations (latitude/longitude) into clusters (stop_cluster) based on how often stops occur there. The darker areas on the map represent a higher concentration of stops, while lighter areas have fewer stops. This helps identify frequently visited areas.



11 Encoding categorical variables

There are 3 categorical variables that contain string values which needs to be converted to numeric values before any analysis, as computers understand only numbers not strings

These 3 categorical columns contain values which do not have any inherent order or ranking present within them, hence these are nominal categorical variables

Hence OneHotEncoding (OHE) is perfomed to these variables and the first column after applying OHE was removed in order to avoid the dummy variable trap (multicollinearity)

```python
from sklearn.preprocessing import OneHotEncoder
import numpy as np

# Columns to apply one-hot encoding
ohe_columns = ['brake_switch_status', 'clutch_switch_status', 'parking_switch_status']

# Instantiate the OneHotEncoder
encoder = OneHotEncoder(sparse_output=False, drop="first")

# Fit and transform the specified columns
encoded_data = encoder.fit_transform(df[ohe_columns])

# Get the new column names based on categories
encoded_columns = encoder.get_feature_names_out(ohe_columns)

# Create a DataFrame for the encoded data
encoded_df = pd.DataFrame(encoded_data, columns=encoded_columns, index=df.index)

# Drop the original columns from the dataframe
df_remaining = df.drop(columns=ohe_columns)

# Concatenate the one-hot encoded dataframe with the remaining dataframe
df_final = pd.concat([df_remaining, encoded_df], axis=1)

# Display a preview of the final dataframe
df_final.head()
```

12. PCA

12.1 Need for PCA

PCA is done to reduce the number of features (dimensionality) in a dataset while retaining the most important information. It simplifies the data, making it easier to analyse and visualize, and helps improve computational efficiency for modelling.

12.2 Standardization

Standardization ensures that all features are on the same scale, preventing features with larger values from dominating the PCA. This allows PCA to fairly capture variance across all features and produce meaningful principal components.

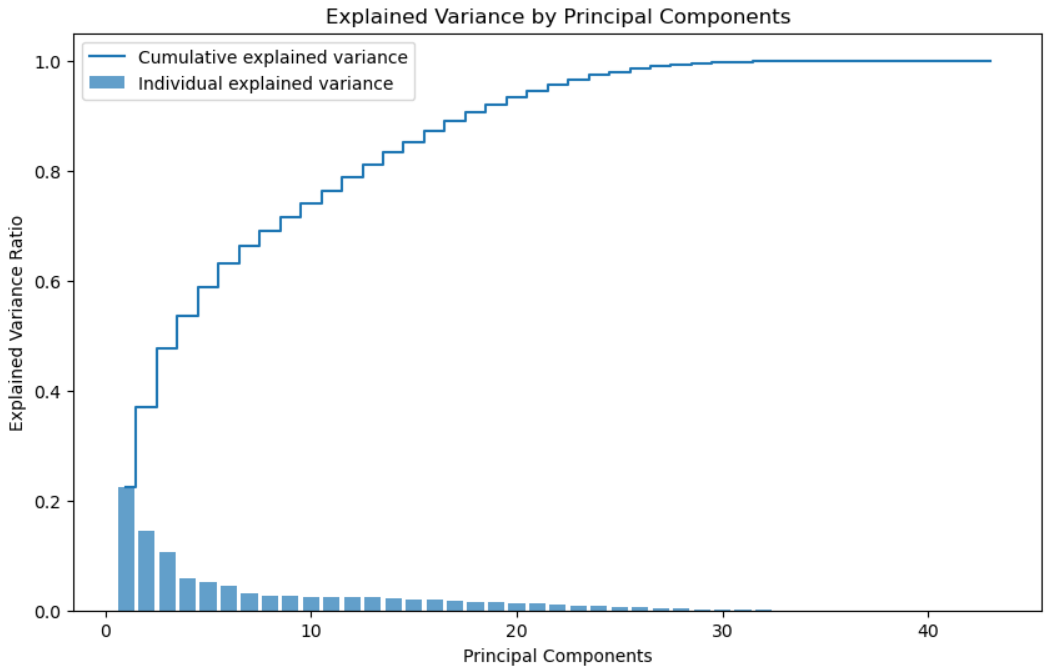
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler


# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_final[numerical_columns])
```

12.3 PCA on all variables

Initially PCA is applied on all variables, that is, 43 PC's (Principal Components) are formed

12.4 Finding the optimal no. of PC's

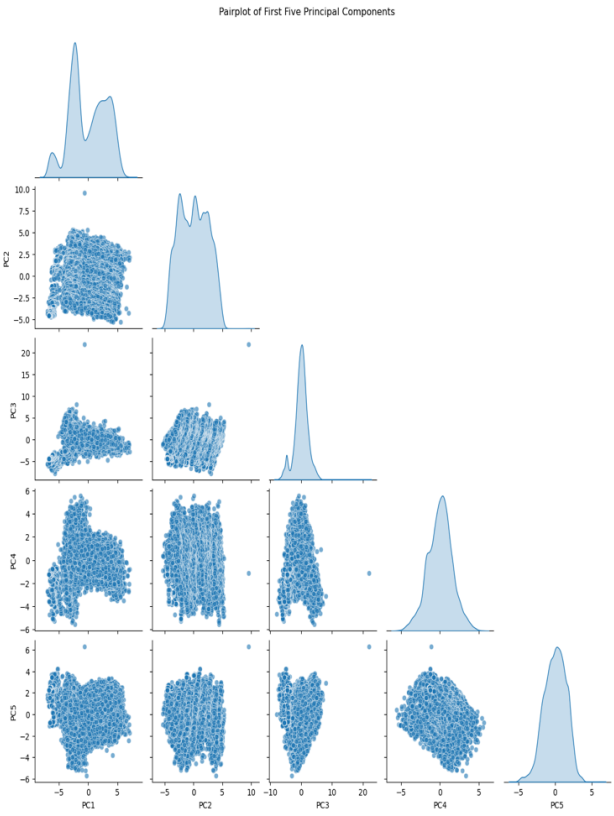
The ideal percentage of variance that needs to be captured by a model is 90%
Taking this into consideration, a graph was plotted with the cumulated explained variance and verified via code to reach to the optimal no of PC's, that was, 18.

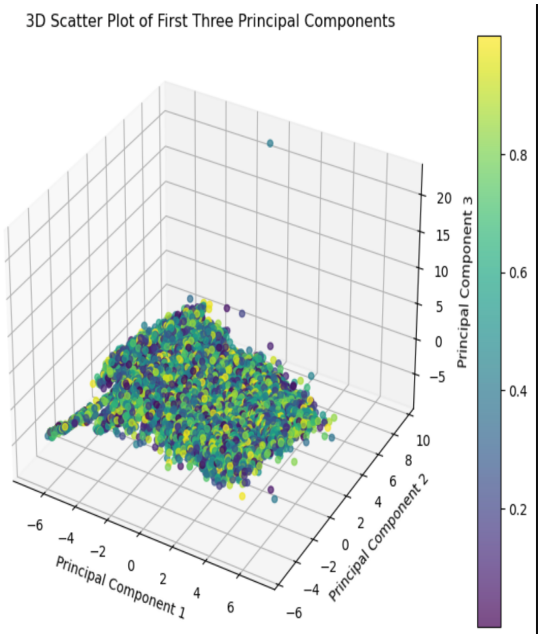


```
[83]: # Determine the number of components to retain (e.g., 90% variance)
      cumulative_variance = np.cumsum(explained_variance)
      components_to_retain = np.argmax(cumulative_variance >= 0.90) + 1
      print(f"Number of components to retain 90% variance: {components_to_retain}")

      Number of components to retain 90% variance: 18
```

12.5 Visualisations on the PC's formed

The top PC's are visualised via a 3d plot to obtain their interactions and a pairplot between the first 5 PC's was also created



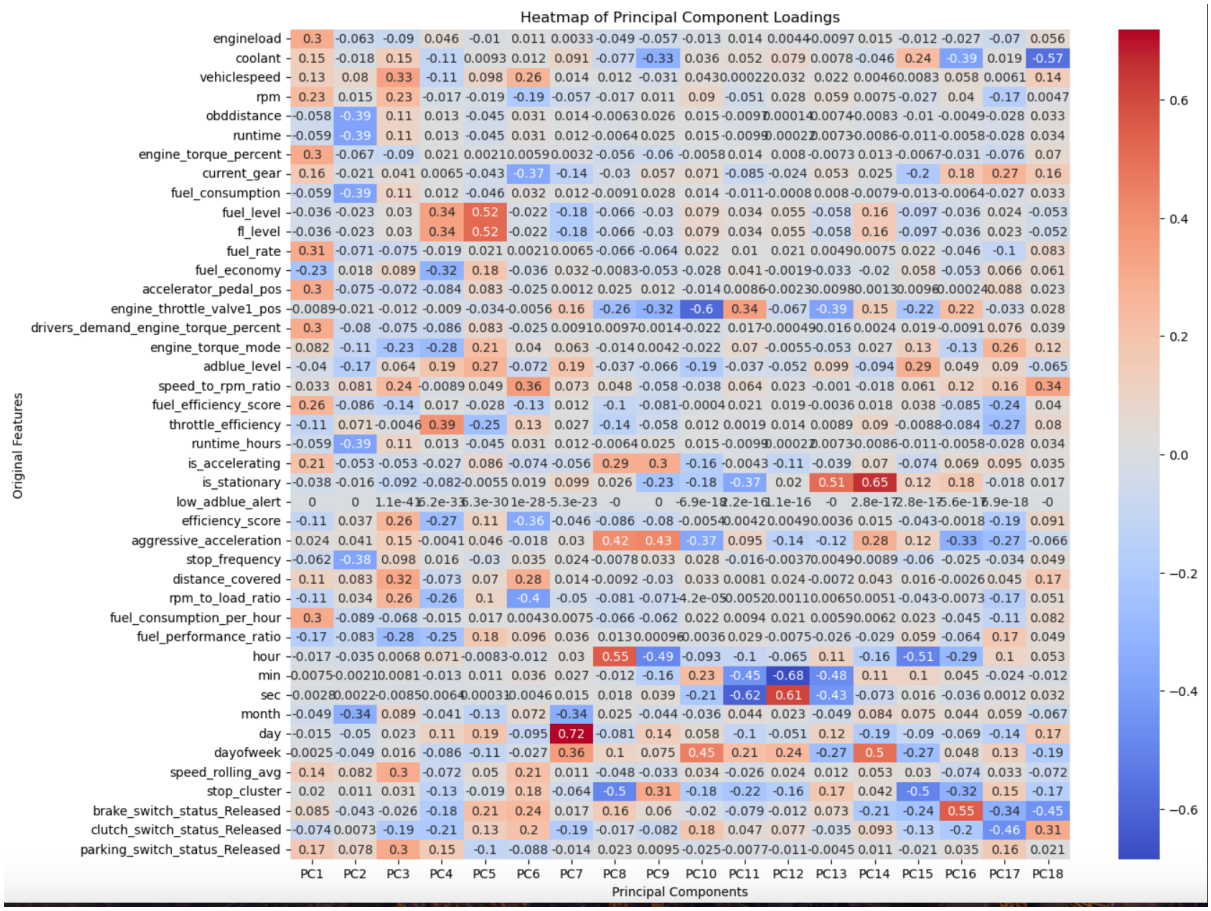


12.6 PC's dependence on variables

A heatmap was created to find out which PC depends on which variable present in the original dataset the most

This was done to visualise the entire dataset's variables with the formed PC's

In order to find deeper insights, a table was also created as mentioned below.



By creating the below table (which Principal Component depends the most on which variables from the dataset), we identify the most influential original features driving each principal component. This helps us understand the underlying structure of our data, aiding in feature selection, dimensionality reduction, and interpretation of latent patterns.

PC	Top Feature 1	Top Feature 2	Top Feature 3	Top Feature 4	Top Feature 5
PC1	fuel_rate	engine_torque_percent	fuel_consumption_per_hour	accelerator_pedal_pos	engineload
PC2	distance_covered	speed_rolling_avg	speed_to_rpm_ratio	vehiclespeed	parking_switch_status_Released

PC3	vehiclespeed	distance_covered	parking_switch_status_Released	speed_rolling_avg	efficiency_score
PC4	throttle_efficiency	fl_level	fuel_level	adblue_level	parking_switch_status_Released
PC5	fuel_level	fl_level	adblue_level	brake_switch_status_Released	engine_torque_mode
PC6	speed_to_rpm_ratio	distance_covered	vehiclespeed	brake_switch_status_Released	speed_rolling_avg
PC7	day	dayofweek	adblue_level	engine_throttle_valve1_pos	is_stationary
PC8	hour	aggressive_acceleration	is_accelerating	brake_switch_status_Released	dayofweek
PC9	aggressive_acceleration	stop_cluster	is_accelerating	day	dayofweek
PC10	dayofweek	min	clutch_switch_status_Released	rpm	fuel_level
PC11	engine_throttle_valve1_pos	dayofweek	aggressive_acceleration	engine_torque_mode	speed_to_rpm_ratio
PC12	sec	dayofweek	coolant	clutch_switch_status_Released	fuel_level
PC13	is_stationary	stop_cluster	day	hour	adblue_level
PC14	is_stationary	dayofweek	aggressive_acceleration	fl_level	fuel_level
PC15	adblue_level	coolant	engine_torque_mode	is_stationary	aggressive_acceleration
PC16	brake_switch_status_Released	engine_throttle_valve1_pos	current_gear	is_stationary	speed_to_rpm_ratio
PC17	current_gear	engine_torque_mode	fuel_performance_ratio	parking_switch_status_Released	speed_to_rpm_ratio
PC18	speed_to_rpm_ratio	clutch_switch_status_Released	day	distance_covered	current_gear

12.7 Insights after PCA implementation

1. Fuel Efficiency and Driver Behaviour:

PCs like PC1 and PC3 highlight the critical role of fuel-related metrics (fuel_rate, efficiency_score) and driving behaviour (accelerator_pedal_pos) in overall efficiency. High acceleration correlates with higher fuel consumption.

2. Route Patterns and Vehicle Speed:

PC2 and PC6 emphasize speed and route dynamics, showing how metrics like vehiclespeed, distance_covered, and speed_to_rpm_ratio influence movement efficiency.

3. Maintenance and Fluid Monitoring:

PCs such as PC4 and PC5 focus on fluid levels (fuel_level, adblue_level) and throttle efficiency, underscoring the importance of monitoring maintenance indicators.

Insights and Recommendations:

1. Engine Performance and Efficiency Profile

Insights:

- The average engine load is relatively low at 38.14%, suggesting either conservative driving patterns or frequent idle time
- Average RPM of 937.04 indicates significant idle time or low-speed operation
- The RPM-Torque correlation of 0.504 shows a moderate positive relationship, indicating that engine power delivery is generally efficient but scope of optimization persists

Recommendations:

- Implement an idle-time reduction strategy as excessive idling impacts fuel efficiency and engine wear

- Consider setting up automated alerts for prolonged idle times exceeding 5 minutes
- Develop a driver training program focusing on optimal RPM ranges for different operation conditions

2. Fuel Management System

Insights:

- Average fuel consumption is notably high at 19,533.07 units
- Fuel level variation of 99.6% indicates vehicles are being run from full to near-empty
- The weak negative correlation (-0.097) between speed and fuel consumption suggests inefficient driving patterns

Recommendations:

- Implement a fuel management system with real-time monitoring
- Set up refuelling guidelines to maintain fuel levels between 25-75% to optimize fuel system health
- Create driver scorecards based on fuel efficiency metrics to promote better driving habits

3. Temperature Management

Insights:

- Average coolant temperature of 84.57°C is within normal operating range
- Fuel temperature average of 215°C suggests potential sensor calibration issues

Recommendations:

- Investigate and recalibrate temperature sensors, particularly for engine oil and fuel
- Implement predictive maintenance models based on temperature pattern analysis
- Set up real-time temperature monitoring with automated alerts for anomalies

4. Driver Behaviour Analysis

Insights:

- 5.01% of all recorded instances show aggressive acceleration patterns
- Analysis of acceleration patterns reveal:
    - High correlation between accelerator position and engine throttle
    - Significant torque demand during aggressive acceleration events
- Gear usage analysis shows:
    - Major use of gears 16-17 (7,859 instances)
    - Potential underutilization of middle gears (3-10)
- Accelerator pedal position averages 34.95%, suggesting moderate acceleration patterns

Recommendations:

- Institute a driver scoring system based on acceleration patterns
- Implement real-time feedback systems for harsh braking and acceleration
- Review and optimize gear shift patterns through driver training
- Implement real-time feedback systems for optimal gear selection
- Consider gamification elements to encourage smoother driving patterns

5. Vehicle Operation Patterns

Insights:

- Maximum speed recorded is 81.70 km/h with an average of 33.36 km/h
- Significant variation in gear usage suggests potential transmission stress
- Engine torque averaging 28.04% indicates underutilization of vehicle capacity

Recommendations:

- Optimize route planning to maintain consistent speeds and reduce stop-start patterns

- Implement predictive maintenance models based on gear usage patterns
- Develop operation-specific guidelines for optimal vehicle utilization

6. Engine Performance and Fuel Efficiency

Insights:

- The analysis reveals an inverse relationship between engine load and fuel consumption:
    - Low load: 19,564.81 units
    - Medium load: 19,514.34 units
    - High load: 19,480.19 units
- Vehicle performance metrics show optimal efficiency in the medium load range:
    - Medium load achieves the highest average speed (37.15 km/h)
    - RPM readings are most balanced in medium load (1,111.82 RPM)

Recommendations:

- Implement a driver coaching program focused on maintaining medium engine load ranges for optimal fuel efficiency
- Develop real-time alerts and notifications when vehicles operate in low-efficiency zones for extended periods
- Consider route optimization to minimize low-load operation periods

7. Vehicle Health Monitoring

Insights:

- Engine temperature and load analysis indicates:
    - Correlation between high engine load and elevated coolant temperatures
    - Consistent engine oil temperature patterns across operating conditions
- Operational patterns suggest:
    - Regular periods of high-stress operation
    - Varying maintenance requirements based on usage patterns

Recommendations:

- Develop predictive maintenance schedules based on engine load patterns
- Implement automated alerts for unusual temperature-load relationships
- Create vehicle-specific maintenance protocols based on operational patterns

8. Operational Efficiency

Insights:

- Speed and fuel economy analysis reveals:
    - Optimal fuel efficiency achieved at moderate speeds
    - Significant variation in fuel consumption based on driving patterns
- Geographic analysis (based on plus codes) indicates:
    - Route-specific efficiency patterns
    - Location-based performance variations

Recommendations:

- Optimize route planning based on fuel efficiency patterns
- Implement zone-based performance metrics
- Develop location-specific driving guidelines

The recommendations suggested above comprises of suggestions which may take time to implement and some of them may be implemented in a shorter span of time. The following table classifies the above recommendations into different categories based on their estimated implementation time:

Immediate Implementations (0-3 months)	**Short-Term Implementations (3-6 months)**	**Long-Term Implementations (6-12 months)**
Deploy real-time driver feedback systems	Establish predictive maintenance ML and DL models	Implement full fleet analytics dashboard using visualisation tools like Tableau, Power BI
Implement aggressive acceleration alerts	Develop route optimization algorithms	Integrate Generative AI features (Ex: LLM based chatbots and analysers)
Begin driver scoring system development	Launch driver training programs (Ex: gamification)	Create comprehensive fleet health monitoring system