EV Battery Efficiency

1. About Dataset

The dataset contains comprehensive operational and condition information for a fleet of electric vehicles' battery systems. It includes electrical measurements such as voltage and current, along with thermal data like battery temperature. State of charge and cycle count provide insight into battery usage and wear. Health indicators such as battery health percentage and remaining capacity (in ampere-hours) reflect the current performance and degradation level of the battery. The dataset also includes an estimate of the remaining useful life (RUL) in days, which predicts how long the battery is expected to function effectively. Timestamped measurements enable time-series analysis, while the vehicle ID associates data with individual vehicles, allowing for tracking and comparison of battery performance across the fleet.

2. Machine Learning Type

This dataset is primarily suited for Supervised Machine Learning tasks, specifically:

- **Regression:**
  Predict continuous numeric outcomes such as the remaining capacity (Ah) or the estimated remaining useful life (RUL) in days based on features like voltage, current, temperature, state of charge, cycle count, and battery health percentage.
- **Classification:**
  Classify the battery health status into categories such as healthy, degraded, or critical based on the operational parameters and usage history.

3. Data Collection:

The dataset is sourced from electric vehicle battery management systems and operational logs, collected from a fleet of vehicles under real-world conditions. It contains structured, time-stamped measurements of electrical, thermal, and usage parameters essential for monitoring battery performance and health. The data is suitable for predictive maintenance, battery life estimation, and training supervised learning models aimed at forecasting battery degradation, remaining capacity, and remaining useful life (RUL). This dataset enables advanced analytics to optimize battery management and enhance vehicle reliability.

4. Data Preprocessing

- **Library Imports and Data Loading:**
- Import necessary libraries such as pandas, numpy, and datetime, and load the dataset into memory for exploratory analysis and preprocessing.
- **Handling Missing Values:**
  Identify and address missing entries in columns like current_A and temperature_C. Imputation techniques such as replacing missing values with the column mean, median, or using interpolation methods can be applied, depending on the data distribution.
- **Handling Duplicates:**
  Detect and remove any duplicate rows to prevent biased model training and inaccurate predictions.
- **Data Cleaning:**
  Ensure all columns have appropriate data types (e.g., convert timestamps to datetime objects). Correct inconsistent formats or erroneous entries, such as negative voltages or impossible state-of-charge percentages.
- **Normalization/Scaling:**
  Apply scaling techniques like Min-Max scaling or StandardScaler (Z-score normalization) to continuous features such as voltage, current, temperature, and state of charge to ensure balanced feature contribution during model training.
- **Outlier Detection:**
  Use statistical methods such as the Interquartile Range (IQR) or Z-score thresholds to identify and handle extreme outlier values in parameters like voltage or current that could distort model learning.
- **Encoding Categorical Data (if any):**
  If there are any categorical features (e.g., battery type, vehicle model) present or added later, convert them to numeric form using techniques like one-hot encoding or label encoding to make the data compatible with machine learning algorithms.

4.1 Library Imports and Data Loading:

The necessary libraries Pandas, NumPy, Matplotlib, Seaborn, and Scikit-learn were imported to perform data manipulation, visualization, and machine learning tasks. The dataset was loaded using pandas.read_csv() from the file battery_data.csv, and the first few records were displayed using df.head() to explore the structure, data types, and key features present in the dataset.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

import warnings

warnings.filterwarnings("ignore")

df = pd.read_csv(r"C:\Users\vjsha\OneDrive\Desktop\vvvvvvvvvv.csv")

df.head()
```

	vehicle_id	timestamp	voltage_V	current_A	temperature_C	state_of_charge_%	cycle_count	battery_health_%	remaining_capacity_Ah	estimated_rul_days
0	1051	01-01-2023 00:00	329.73	-9.56	29.9	36.4	1466	91.65	90.11	1244
1	1092	01-01-2023 00:30	342.72	-55.08	31.7	98.2	1793	78.15	17.59	1475
2	1014	01-01-2023 01:00	390.94	-9.51	36.6	64.2	1104	82.07	82.65	149
3	1071	01-01-2023 01:30	301.73	-71.83	21.9	62.4	530	73.08	87.52	249
4	1060	01-01-2023 02:00	313.93	NaN	NaN	77.3	826	75.67	83.55	82

4.2 Handling Missing Values:

All missing values in the dataset were identified by performing a null value check across all columns. This process revealed which features contained incomplete data. In this dataset, columns such as current_A and temperature_C were found to have missing values that required appropriate handling before analysis and modeling.

```
print("Missing values per column:\n", df.isnull().sum())

print("\nTotal missing values in dataset:", df.isnull().sum().sum())

print("\nRows with missing values:")

print(df[df.isnull().any(axis=1)])

df_mean_filled = df.fillna(df.mean(numeric_only=True))

df_zero_filled = df.fillna(0)

df_ffill = df.fillna(method='ffill')
```

df_bfill = df.fillna(method='bfill')

print("\nMissing values after filling with mean:\n", df_mean_filled.isnull().sum())

```
Missing values per column:
 vehicle_id              0
timestamp               0
voltage_V              20
current_A              20
temperature_C          20
state_of_charge_%      18
cycle_count             0
battery_health_%       20
remaining_capacity_Ah   0
estimated_rul_days      0
dtype: int64

Total missing values in dataset: 98

Rows with missing values:
     vehicle_id         timestamp  voltage_V  current_A  temperature_C  \
4          1060  01-01-2023 02:00     313.93        NaN            NaN
5          1020  01-01-2023 02:30     305.52      -0.33           44.2
7          1086  01-01-2023 03:30     402.66      82.97           21.0
10         1087  01-01-2023 05:00     311.74      26.45            NaN
20         1001  01-01-2023 10:00     375.10      19.08           16.2
..          ...               ...        ...        ...            ...
458        1004  10-01-2023 13:00     348.10     -29.87            NaN
460        1005  10-01-2023 14:00     413.67     -84.28            NaN
...
battery_health_%        0
remaining_capacity_Ah   0
estimated_rul_days      0
dtype: int64
```

4.3 Handling Duplicates and Blank Data:

A check was performed for duplicate rows, completely blank rows, and entirely blank columns to ensure data integrity. Duplicate entries were identified and removed to avoid redundancy and prevent skewing the analysis. Additionally, the dataset was scanned for any rows or columns that were entirely blank, and such rows or columns were dropped to maintain consistency and completeness for subsequent processing.

duplicates = df[df.duplicated()]

print("Number of duplicate rows:", duplicates.shape[0])

print("\nDuplicate rows (if any):")

print(duplicates)

df = df.drop_duplicates()

print("\nShape after removing duplicates:", df.shape)

df.replace(r'^\s*$', np.nan, regex=True, inplace=True)

print("\nMissing values after replacing blanks with NaN:\n", df.isnull().sum())

```
Number of duplicate rows: 0

Duplicate rows (if any):
Empty DataFrame
Columns: [vehicle_id, timestamp, voltage_V, current_A, temperature_C, state_of_charge_%, cycle_count, battery_health_%, remaining_capacity_Ah, estimated_rul_days]
Index: []

Shape after removing duplicates: (500, 10)

Missing values after replacing blanks with NaN:
 vehicle_id              0
timestamp               0
voltage_V              20
current_A              20
temperature_C          20
state_of_charge_%      18
cycle_count             0
battery_health_%       20
remaining_capacity_Ah   0
estimated_rul_days      0
dtype: int64
```

4.4 Removing Duplicates:

A check was performed to identify duplicate rows within the dataset. Duplicate entries can introduce redundancy and potentially distort analysis or predictive model results. All duplicate rows detected were removed to ensure that each record in the dataset is unique. This step helped maintain data quality and ensured accurate outcomes in subsequent data processing and analysis.

```
duplicate_rows = df[df.duplicated()]

print("Number of duplicate rows:", duplicate_rows.shape[0])

df = df.drop_duplicates()

print("Shape after removing duplicates:", df.shape)
```

```
Number of duplicate rows: 0
Shape after removing duplicates: (500, 10)
```

4.5 Z-Score Normalization:

Z-score normalization was applied to the numeric columns in the dataset to standardize the data. This technique transforms features by subtracting the mean and scaling to unit variance, ensuring that each feature contributes equally during model training and improving the convergence of machine learning algorithms.

```
from scipy.stats import zscore

numeric_df = df.select_dtypes(include=np.number)
```

```
z_scores = numeric_df.apply(zscore)

print("Z-Score Normalized Data:")

print(z_scores.head())
```

```
Z-Score Normalized Data:
   vehicle_id  voltage_V  current_A  temperature_C  cycle_count  \
0    0.071970        NaN        NaN            NaN     0.648119
1    1.459907        NaN        NaN            NaN     1.238779
2   -1.180559        NaN        NaN            NaN    -0.005762
3    0.749012        NaN        NaN            NaN    -1.042579
4    0.376639        NaN        NaN            NaN    -0.507914

   battery_health_%  remaining_capacity_Ah  estimated_rul_days
0               NaN               1.301286            0.406060
1               NaN              -1.557117            0.817991
2               NaN               1.007247           -1.546598
3               NaN               1.199200           -1.368273
4               NaN               1.042721           -1.666076
```

4.6 Min-Max Normalization:

Min-Max normalization was applied to the numeric columns in the dataset to rescale the feature values to a specific range, typically [0, 1]. This technique ensures that all numeric features contribute equally and proportionally to the model, preventing features with larger scales from dominating the learning process.

```
min_max_scaled = (numeric_df - numeric_df.min()) / (numeric_df.max() -
numeric_df.min())

print("Min-Max Normalized Data:")

print(min_max_scaled.head())
```

```
Min-Max Normalized Data:
   vehicle_id  voltage_V  current_A  temperature_C  cycle_count  \
0    0.515152   0.081221   0.447733       0.494983     0.721987
1    0.929293   0.117391   0.217416       0.555184     0.894820
2    0.141414   0.251655   0.447986       0.719064     0.530655
3    0.717172   0.003258   0.132665       0.227425     0.227273
4    0.606061   0.037228        NaN            NaN     0.383721

   battery_health_%  remaining_capacity_Ah  estimated_rul_days
0          0.721815               0.893153            0.616090
1          0.271514               0.081786            0.733707
2          0.402268               0.809689            0.058554
3          0.102402               0.864175            0.109470
4          0.188793               0.819758            0.024440
```

4.7 Outliers Detection Using IQR Method:

To identify anomalous values that could affect data analysis or modeling, outlier detection was performed using the Interquartile Range (IQR) method on all numeric columns.

```
numeric_cols = df.select_dtypes(include=np.number)

Q1 = numeric_cols.quantile(0.25)

Q3 = numeric_cols.quantile(0.75)

IQR = Q3 - Q1

outlier_condition = (numeric_cols < (Q1 - 1.5 * IQR)) | (numeric_cols > (Q3 + 1.5 * IQR))

outliers = numeric_cols[outlier_condition.any(axis=1)]

print("Number of outlier rows:", outliers.shape[0])

print("\nSample outlier rows:")

print(outliers.head())
```

```
Number of outlier rows: 10

Sample outlier rows:
     vehicle_id   voltage_V  current_A  temperature_C  cycle_count  \
33         1091  659.702667       4.88           33.7          301
60         1091  659.702667     -51.37           28.6          518
150        1026  659.702667      19.37           28.1         1557
159        1095  659.702667     -50.72           26.7         1895
202        1036  659.702667      19.25           33.3          404

     battery_health_%  remaining_capacity_Ah  estimated_rul_days
33              77.63                  80.81                1852
60              91.91                  63.90                 366
150             74.92                  95.06                 136
159             90.70                  71.10                 302
202             90.95                  13.86                1424
```

4.8 Correlation and Covariance Analysis

A statistical analysis was conducted on the numeric features of the dataset to understand the relationships and dependencies between variables. Correlation and covariance matrices were computed for features such as voltage, current, temperature, state of charge, and battery health to identify significant patterns and interactions useful for modeling.

numeric_df = df.select_dtypes(include='number')

print("Correlation Matrix:")

print(numeric_df.corr())

print("\nCovariance Matrix:")

print(numeric_df.cov())

```
Correlation Matrix:
                       vehicle_id  voltage_V  current_A  temperature_C  \
vehicle_id               1.000000  -0.065211  -0.058726      -0.036376
voltage_V               -0.065211   1.000000   0.010328       0.070147
current_A               -0.058726   0.010328   1.000000      -0.006158
temperature_C           -0.036376   0.070147  -0.006158       1.000000
cycle_count              0.016686  -0.045942  -0.048073       0.041236
battery_health_%         0.014600  -0.013089   0.051354       0.026945
remaining_capacity_Ah   -0.005563  -0.032542   0.044325      -0.081701
estimated_rul_days      -0.013904  -0.052857   0.015255       0.003267

                       cycle_count  battery_health_%  remaining_capacity_Ah  \
vehicle_id                0.016686          0.014600              -0.005563
voltage_V                -0.045942         -0.013089              -0.032542
current_A                -0.048073          0.051354               0.044325
temperature_C             0.041236          0.026945              -0.081701
cycle_count               1.000000         -0.069765               0.039319
battery_health_%         -0.069765          1.000000               0.009988
remaining_capacity_Ah     0.039319          0.009988               1.000000
estimated_rul_days        0.034397          0.042634               0.013760

                       estimated_rul_days
vehicle_id                      -0.013904
voltage_V                       -0.052857
current_A                        0.015255
...
cycle_count                  10700.078677
battery_health_%               205.927079
remaining_capacity_Ah          196.159506
estimated_rul_days          315097.525788
```

4.9 Descriptive Statistics:

To understand the distribution and center of the numeric data, three key statistical measures of central tendency were computed:

- **Mean:** The average value for each numeric column, such as voltage, current, and temperature, was calculated to represent the arithmetic center of the data.
- **Median:** The middle value in each numeric column was determined, providing a robust measure that is less sensitive to outliers.
- **Mode:** The most frequently occurring value in each numeric column was computed. In cases with multiple modes, only the first mode was considered.

```
numeric_df = df.select_dtypes(include='number')
print("Mean:\n", numeric_df.mean())
print("\nMedian:\n", numeric_df.median())
print("\nMode:\n", numeric_df.mode().iloc[0])
```

```
Mean:
 vehicle_id                1048.874000
voltage_V                  365.661347
current_A                   -1.813438
temperature_C               29.870833
cycle_count               1107.190000
battery_health_%            85.390146
remaining_capacity_Ah       57.095320
estimated_rul_days        1016.292000
dtype: float64

Median:
 vehicle_id                1050.000
voltage_V                  360.395
current_A                   -1.325
temperature_C               29.900
cycle_count               1151.500
battery_health_%            85.995
remaining_capacity_Ah       57.320
estimated_rul_days         979.500
dtype: float64

Mode:
 vehicle_id                1061.000000
voltage_V                  659.702667
...
battery_health_%            78.070000
remaining_capacity_Ah       11.420000
estimated_rul_days        1290.000000
Name: 0, dtype: float64
```

5. Data Visualization:

Data visualization techniques were applied to gain intuitive insights into the structure, distribution, and relationships within the dataset. Visualizations such as histograms, scatter plots, box plots, and heatmaps helped identify patterns, trends, and anomalies in battery parameters like voltage, current, temperature, state of charge, and battery health that might not be evident from statistical summaries alone.

5.1 Box Plot Visualization:

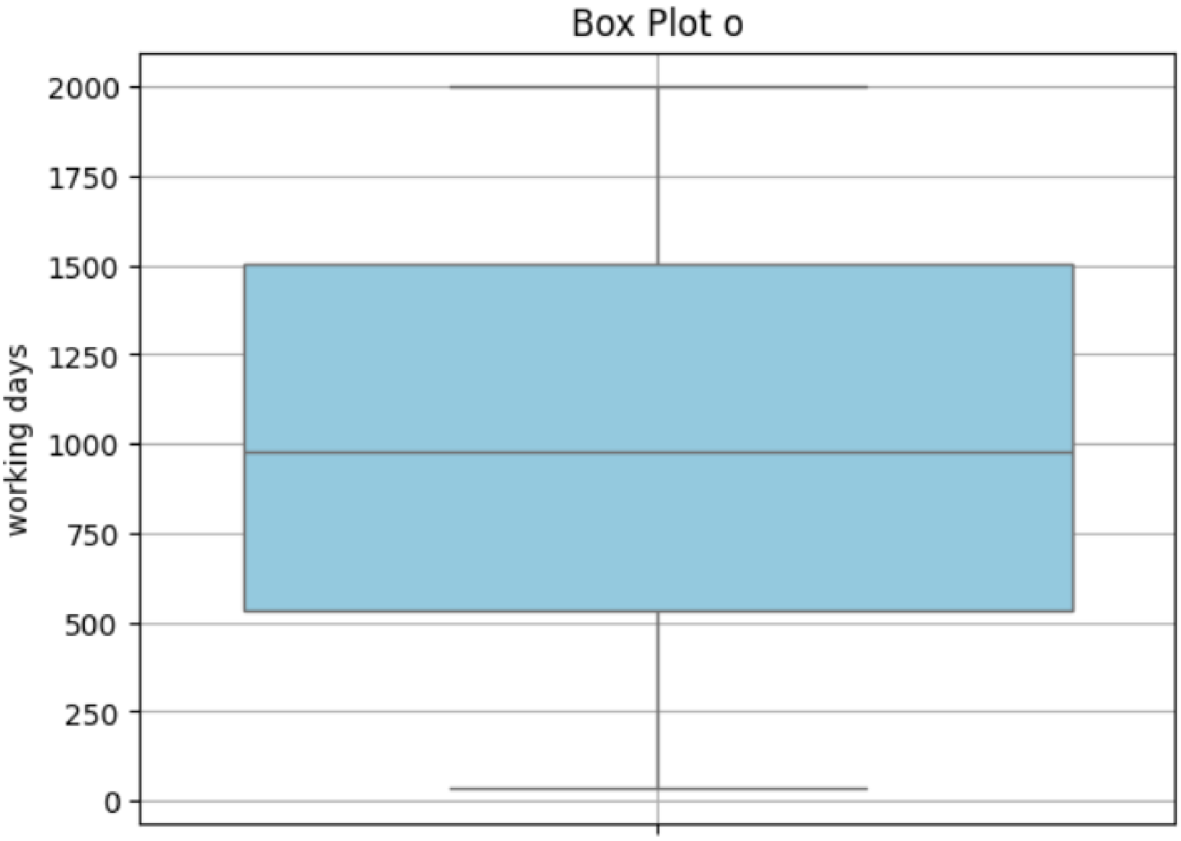
A box plot was created for the voltage_V column to visualize its distribution and detect any potential outliers in the data.

import seaborn as sns

import matplotlib.pyplot as plt

sns.boxplot(y=df['estimated_rul_days'].dropna(), color='skyblue')

```
plt.title('Box Plot o')

plt.ylabel('working days')

plt.grid(True)

plt.show()
```

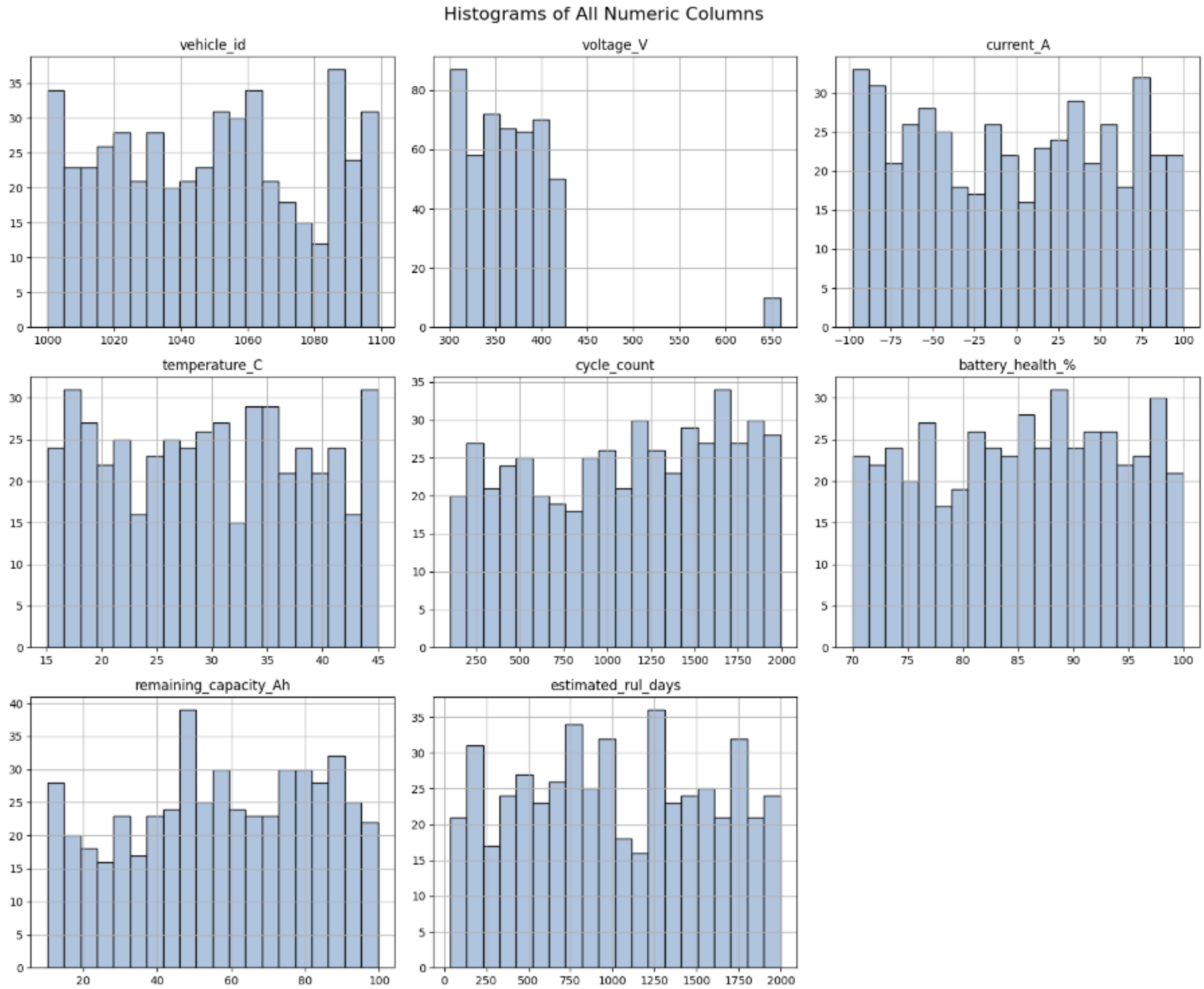


5.2 Histogram

Histograms were generated for all numeric columns in the dataset—such as voltage, current, temperature, state of charge, cycle count, and battery health—to visually examine their distributions. This step is essential in exploratory data analysis (EDA) for understanding how data points are spread across each feature and to identify skewness or irregularities.

```
import matplotlib.pyplot as plt

numeric_df = df.select_dtypes(include='number')

numeric_df.hist(figsize=(15, 12), bins=20, color='lightsteelblue', edgecolor='black')

plt.tight_layout()

plt.suptitle('Histograms of All Numeric Columns', fontsize=16, y=1.02)
```

```
plt.show()
```

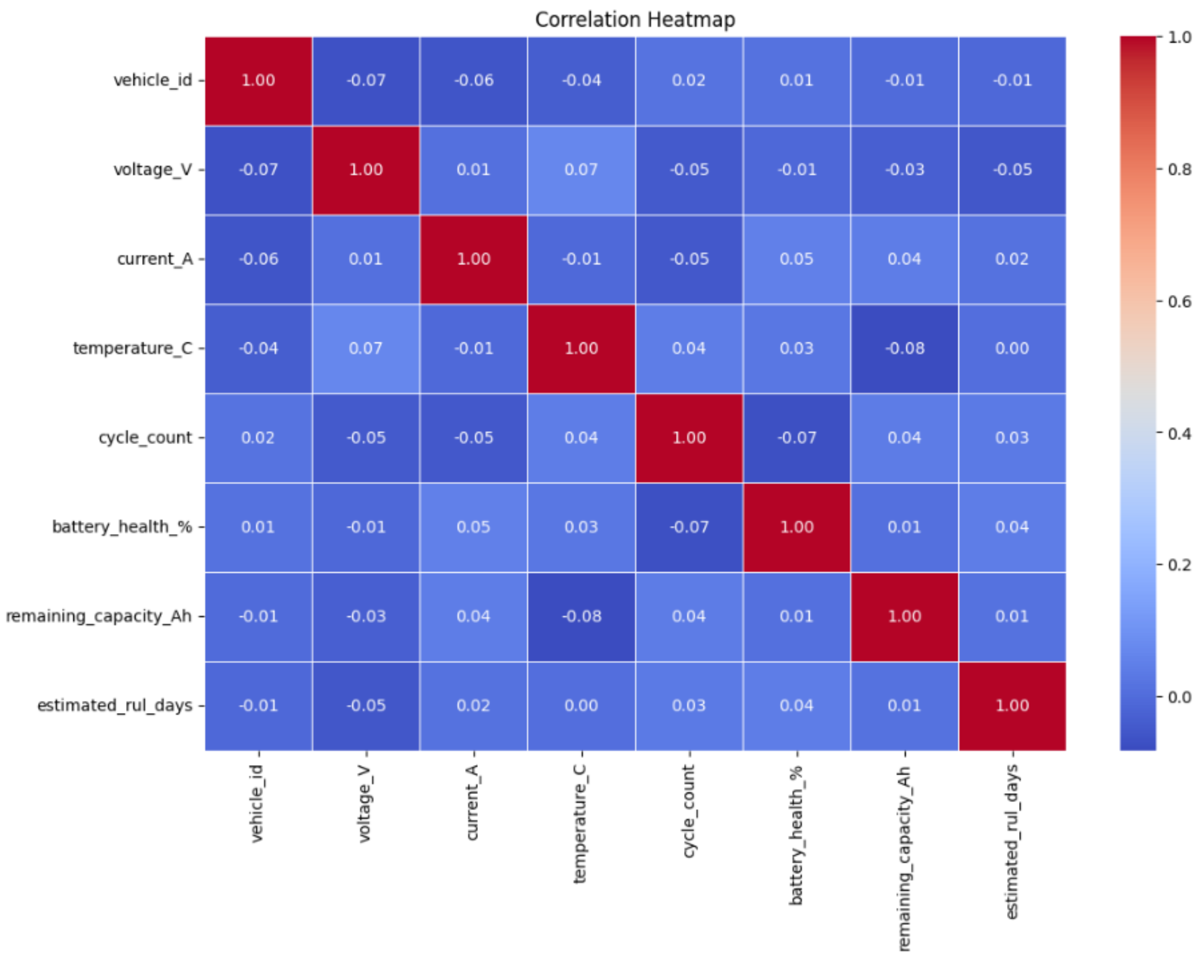


5.3 Correlation Heatmap:

A correlation heatmap was generated to visualize the linear relationships between all numeric features in the dataset, such as voltage, current, temperature, state of charge, battery health, and estimated RUL. This visualization helped identify strongly correlated variables, detect potential multicollinearity, and guided informed decisions for feature selection in model development.

```
import seaborn as sns

import matplotlib.pyplot as plt

numeric_df = df.select_dtypes(include='number')

corr_matrix = numeric_df.corr()

plt.figure(figsize=(12, 8))
```

```
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

plt.title("Correlation Heatmap")

plt.show()
```

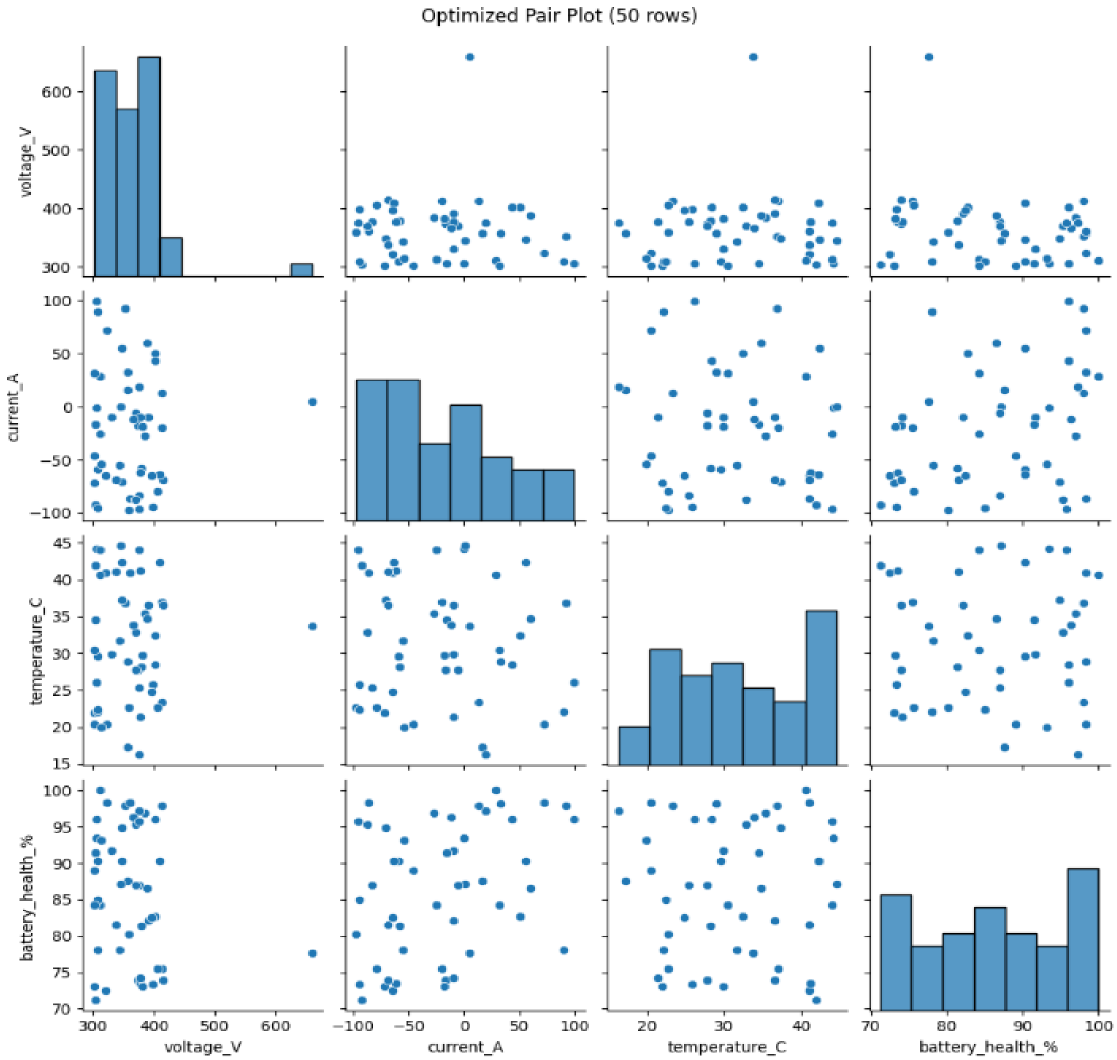


5.4 Pair Plot:

A pair plot was created using a subset of 50 rows from selected numeric columns such as voltage, current, temperature, state of charge, and battery health to visualize the pairwise relationships and distribution patterns between features. This helped efficiently identify trends, clusters, and correlations within the dataset.

```
import seaborn as sns

import matplotlib.pyplot as plt

selected_columns = ['voltage_V', 'current_A', 'temperature_C', 'battery_health_%']

subset_df = df[selected_columns].dropna().head(50)  # Limit to 50 rows

sns.pairplot(subset_df)
```

```
plt.suptitle("Optimized Pair Plot (50 rows)", y=1.02)

plt.show()
```



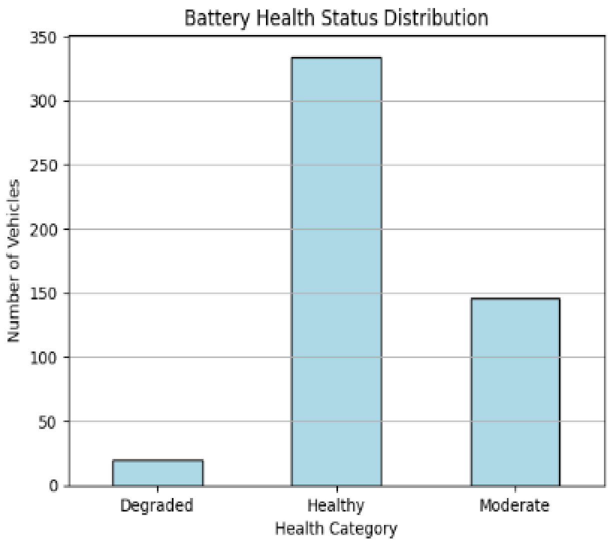
5.5 Bar and Pie Charts:

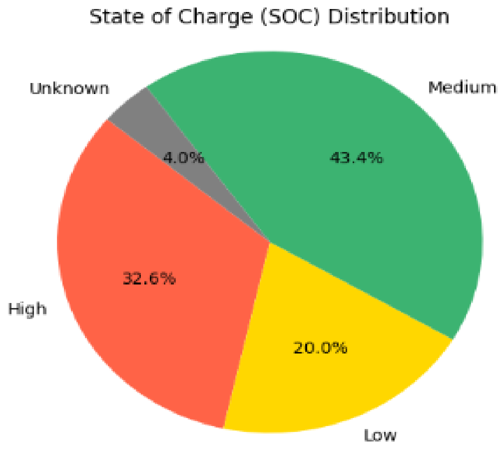
A bar chart was plotted to show the count of battery health status cases, distinguishing between different health levels such as healthy, moderate, and degraded (if categorized). Additionally, a pie chart was used to represent the distribution of state of charge levels (e.g., low, medium, high), providing a clear visual of the proportions of vehicles operating at various charge levels.

```
import matplotlib.pyplot as plt

import pandas as pd

def categorize_health(health):

    if health >= 80:
```

```
        return 'Healthy'
    elif health >= 60:
        return 'Moderate'
    else:
        return 'Degraded'
df['Battery_Health_Status'] = df['battery_health_%'].apply(categorize_health)
df['Battery_Health_Status'].value_counts().sort_index().plot(
    kind='bar', color='lightblue', edgecolor='black'
)
plt.title('Battery Health Status Distribution')
plt.xlabel('Health Category')
plt.ylabel('Number of Vehicles')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
df['state_of_charge_%'] = df['state_of_charge_%'].astype(str).str.replace('%', '', regex=False)
df['state_of_charge_%'] = pd.to_numeric(df['state_of_charge_%'], errors='coerce')
def categorize_soc(soc):
    if pd.isnull(soc):
        return 'Unknown'
    elif float(soc) < 30:
        return 'Low'
    elif float(soc) <= 70:
        return 'Medium'
```

```
    else:
        return 'High'
df['SOC_Level'] = df['state_of_charge_%'].apply(categorize_soc)
soc_sizes = df['SOC_Level'].value_counts().sort_index()
soc_labels = soc_sizes.index
plt.pie(soc_sizes, labels=soc_labels, autopct='%1.1f%%', startangle=140,
        colors=['tomato', 'gold', 'mediumseagreen', 'gray'])
plt.title('State of Charge (SOC) Distribution')
plt.axis('equal')
plt.show()
```



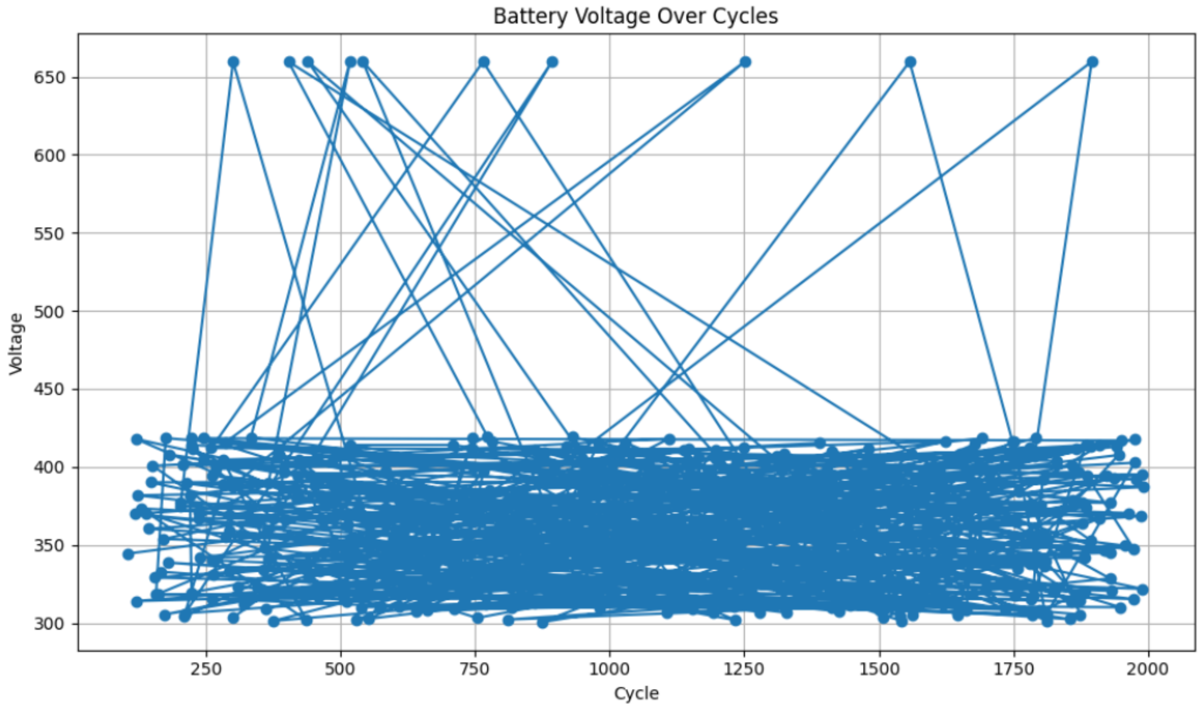


5.6 Line Chart:

A line chart was plotted to show the variation in battery voltage (Voltage_V) across vehicle entries, using the entry index as the x-axis. This visualization helped in identifying voltage fluctuations, stability trends, and any potential anomalies in voltage levels over time or across vehicles. By observing the line pattern, one can assess the consistency of voltage readings and detect any irregular spikes or drops that may require further investigation.

```
import pandas as pd
```

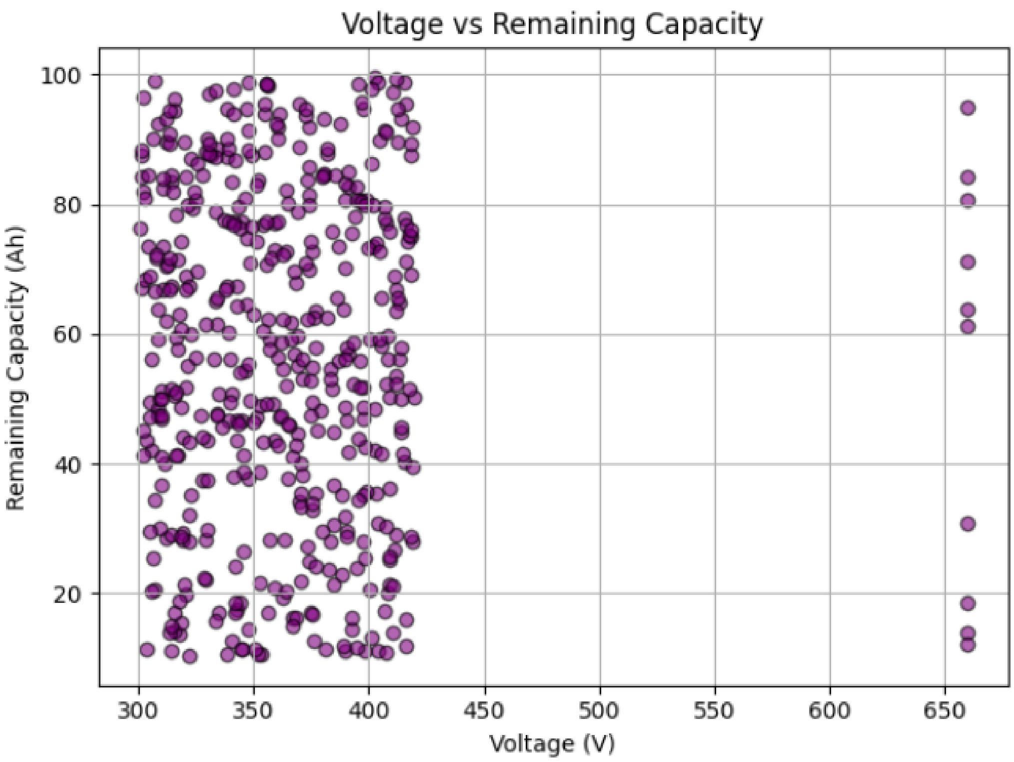
```
import matplotlib.pyplot as plt

df = pd.read_csv(r"C:\Users\vjsha\OneDrive\Desktop\vvvvvvvvvv.csv")

plt.figure(figsize=(10, 6))

plt.plot(df['cycle_count'], df['voltage_V'], marker='o')

plt.title("Battery Voltage Over Cycles")

plt.xlabel("Cycle")

plt.ylabel("Voltage")

plt.grid(True)

plt.tight_layout()

plt.show()
```



5.7 Scatter Plot:

A scatter plot was used to visualize the relationship between battery temperature (temperature_C) and state of charge (state_of_charge_%). This helped in identifying patterns, potential correlations, and outliers between battery thermal conditions and charge levels. The plot provided visual insight into how temperature variations might influence or reflect the battery's state of charge across different vehicles or operating conditions.

```python
import matplotlib.pyplot as plt

plt.scatter(
    df['voltage_V'],
    df['remaining_capacity_Ah'],
    alpha=0.6,
    color='purple',      # fill color
    edgecolors='black'   # edge color (note plural 'edgecolors')
)

plt.title('Voltage vs Remaining Capacity')

plt.xlabel('Voltage (V)')

plt.ylabel('Remaining Capacity (Ah)')

plt.grid(True)

plt.tight_layout()

plt.show()
```



6. Exploring Cleaned Dataset:

The finalized dataset, after completing all data cleaning and preprocessing steps—including handling missing values, duplicate removal, normalization, and type conversions—was saved as cleaned_data.csv. To ensure data integrity, the file was reloaded using pandas and checked for structure consistency, correct data types, and successful preservation of all cleaned entries. This step confirmed the dataset was ready for further analysis and model development.

df.to_csv('cleaned_data.csv', index=False)

print("Cleaned dataset saved as 'cleaned_data.csv'")

verified_df = pd.read_csv('cleaned_data.csv')

print("\nVerified - First 5 rows of saved dataset:")

print(verified_df.head())

```
Cleaned dataset saved as 'cleaned_data.csv'

Verified - First 5 rows of saved dataset:
   vehicle_id         timestamp  voltage_V  current_A  temperature_C  \
0        1051  01-01-2023 00:00     329.73      -9.56           29.9
1        1092  01-01-2023 00:30     342.72     -55.08           31.7
2        1014  01-01-2023 01:00     390.94      -9.51           36.6
3        1071  01-01-2023 01:30     301.73     -71.83           21.9
4        1060  01-01-2023 02:00     313.93        NaN            NaN

  state_of_charge_%  cycle_count  battery_health_%  remaining_capacity_Ah  \
0              36.4         1466             91.65                  90.11
1              98.2         1793             78.15                  17.59
2              64.2         1104             82.07                  82.65
3              62.4          530             73.08                  87.52
4              77.3          826             75.67                  83.55

   estimated_rul_days
0                1244
1                1475
2                 149
3                 249
4                  82
```