Battery Data Analysis Documentation

Introduction

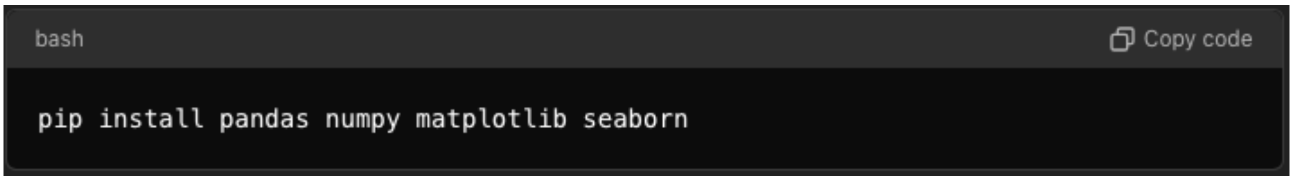
This documentation describes the Python script used to analyze battery performance data. The script reads data from CSV files, processes it, and generates various plots to visualize the performance characteristics of different batteries. The key libraries used are pandas, numpy, matplotlib, seaborn, and datetime.

Prerequisites

Ensure that you have the following libraries installed:

- pandas
- numpy
- matplotlib
- seaborn

You can install these libraries using pip:



```bash
pip install pandas numpy matplotlib seaborn
```

Overview

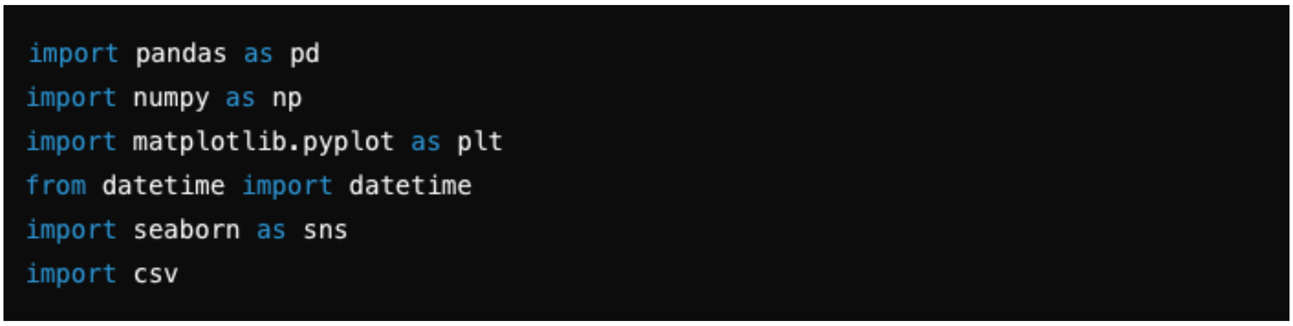
The script performs the following key functions:

1. **Reading and Preprocessing Data**: It loads metadata and impedance data from CSV files.
2. **Grouping Batteries**: It assigns batteries to groups based on their IDs.
3. **Visualizing Battery Cycles**: It generates various plots to visualize battery cycles, capacity, and other parameters during charge and discharge.

Detailed Explanation

1. Importing Libraries

The script begins by importing the necessary libraries:



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
import seaborn as sns
import csv
```

2. Reading Data

The script reads metadata and impedance data from CSV files into pandas DataFrames:

```python
metadata_df = pd.read_csv("metadata.csv")
file_data = pd.read_csv("impedance.csv")
file_data = file_data.dropna()  # Dropping missing values
```

3. Grouping Batteries

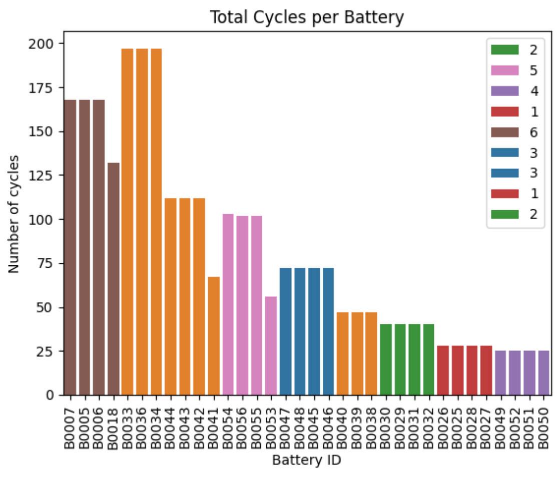
The script defines a dictionary to group batteries by their IDs. It then assigns each battery in the metadata DataFrame to its respective group:

```python
battery_group = {"1": ["B0025", "B0026", "B0027", "B0028"],
                ...
metadata_df["Group"] = ""
for key in list(battery_group.keys()):
    for cell in battery_group[key]:
        metadata_df.loc[metadata_df["battery_id"] == cell, "Group"] = key
```

4. Plotting Total Cycles per Battery

A bar plot is created using Seaborn to visualize the total number of cycles for each battery, grouped by their IDs:

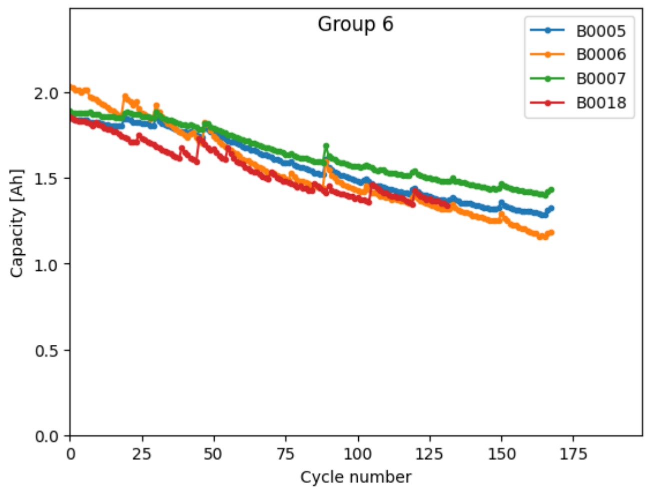
```python
plt.rcParams.update(plt.rcParamsDefault)
sns.countplot(data=metadata_df[metadata_df["type"]=="discharge"], x="battery_id",
plt.tick_params(axis='x', rotation=90)
plt.ylabel("Number of cycles")
plt.xlabel("Battery ID")
plt.title('Total Cycles per Battery')
plt.show()
```



5. Plotting Capacity vs. Cycle Number

The function plot_capacity_vs_cycle() is defined to plot the battery capacity as a function of the cycle number for a specified group of batteries:

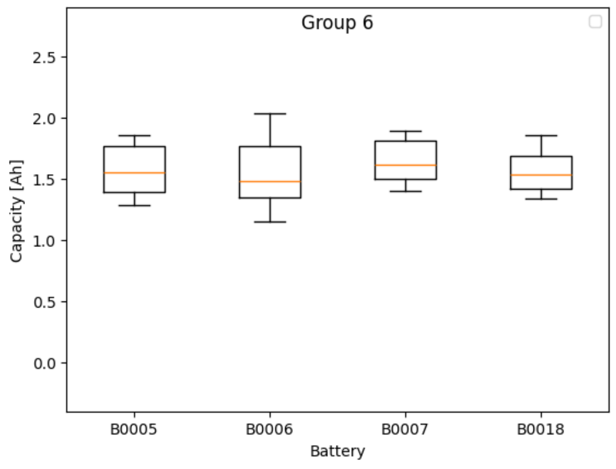
```
def plot_capacity_vs_cycle(cells: dict, group: str):
    ...
GROUP = "6"
plot_capacity_vs_cycle(battery_group, GROUP)
```



6. Creating Box Plots of Capacity

The function plot_capacity_boxplot() creates box plots to visualize the distribution of battery capacities within each group:

```
def plot_capacity_boxplot(cells: dict, group: str):
    ...
plot_capacity_boxplot(battery_group, GROUP)
```



7. Creating Histograms of Capacity and Cycle Number

The function plot_capacity_histogram() creates histograms for the capacity and cycle number:

```
def plot_capacity_histogram(cells: dict, group: str):
    ...
plot_capacity_histogram(battery_group, GROUP)
```

8. Visualizing Charge and Discharge Data

The script generates line plots to visualize voltage, temperature, and current over time for a specific battery during discharge and charge cycles:

```
BATTERY_ID = "B0005"
battery_data = pd.read_csv("discharge.csv")
ax.plot(battery_data["Time"], battery_data["Voltage_measured"], marker='.')
...
```

Battery Data Processing Script - Brief Documentation

Overview

This script processes battery data by reading metadata from a CSV file (metadata.csv), extracting specific battery data files (discharge, charge, and impedance), and consolidating them into three separate CSV files: discharge.csv, charge.csv, and impedance.csv. The processed data includes an additional column for battery_id to associate each entry with the corresponding battery.

Key Steps

1. **Read Metadata**: Load the metadata from metadata.csv using pandas.

```
metadata_df = pd.read_csv("metadata.csv")
```

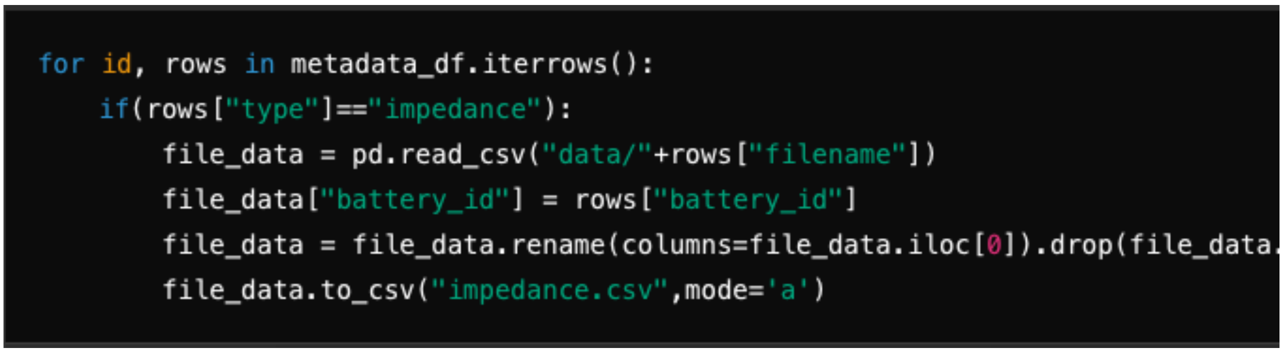
2. **Initialize CSV for Discharge Data**: Create and initialize discharge.csv by appending battery data from each relevant file.

```
for id, rows in metadata_df.iterrows():
    if(rows["type"]=="discharge"):
        file_data = pd.read_csv("data/"+rows["filename"])
        file_data["battery_id"] = rows["battery_id"]
        file_data = file_data.rename(columns=file_data.iloc[0]).drop(file_data.
        file_data.to_csv("discharge.csv",mode='a')
```

3. **Initialize CSV for Charge Data**: Similar to the discharge data, the script initializes charge.csv by appending data from charge files, adding the battery_id column.

```
for id, rows in metadata_df.iterrows():
    if(rows["type"]=="charge"):
        file_data = pd.read_csv("data/"+rows["filename"])
        file_data["battery_id"] = rows["battery_id"]
        file_data = file_data.rename(columns=file_data.iloc[0]).drop(file_data.
        file_data.to_csv("charge.csv",mode='a')
```

4. **Initialize CSV for Impedance Data**: The same process is applied to impedance data, creating and populating impedance.csv.



```
for id, rows in metadata_df.iterrows():
    if(rows["type"]=="impedance"):
        file_data = pd.read_csv("data/"+rows["filename"])
        file_data["battery_id"] = rows["battery_id"]
        file_data = file_data.rename(columns=file_data.iloc[0]).drop(file_data.
        file_data.to_csv("impedance.csv",mode='a')
```

Output

- **discharge.csv**: Consolidated data of all discharge files with an added battery_id column.
- **charge.csv**: Consolidated data of all charge files with an added battery_id column.
- **impedance.csv**: Consolidated data of all impedance files with an added battery_id column.

Summary

This script automates the process of aggregating battery data from multiple CSV files into three main datasets, allowing for easier analysis and visualization of battery performance.

Conclusion

This code performs a detailed analysis of battery performance data by grouping batteries, visualizing cycle data, and generating multiple plots for different battery attributes during charge and discharge cycles.

- **Data Processing**: The code begins by cleaning and organizing battery metadata and impedance data. It assigns each battery to a group and focuses on the discharge data for further analysis.

- **Visualization**:

    - **Group-wise Analysis**: Batteries are grouped, and the code produces visual representations of the total number of cycles per battery. This helps identify the distribution and frequency of cycles within each battery group.
    - **Capacity vs. Cycle Plots**: For a selected battery group, the capacity over cycles is plotted, highlighting trends and degradation patterns. A horizontal line indicates a critical capacity threshold.
    - **Box Plots**: The code creates box plots to show the distribution of capacities within each group, providing insights into variability and outliers.
    - **Histograms**: Histograms of cycle numbers and capacities further explore the distribution of these variables for the selected group.
- **Battery-Specific Analysis**:

    - **Discharge Data**: Detailed plots for a specific battery (e.g., B0005) are generated, showing how voltage, temperature, and current change over time during discharge cycles.
    - **Charge Data**: Similar plots are created for the charge cycles, allowing a direct comparison of the battery's behavior during both charging and discharging processes.

Insights and Utility

This comprehensive analysis allows for:

- **Identifying Degradation Patterns**: By visualizing capacity over cycles, the code helps detect when batteries start losing significant capacity, which is crucial for battery life prediction.
- **Group Comparisons**: Group-based plots enable comparisons between different sets of batteries, facilitating insights into group-specific performance.
- **Detailed Battery Behavior**: Plots specific to individual batteries, like B0005, give an in-depth look at how various factors (voltage, temperature, current) evolve, offering essential data for performance monitoring and troubleshooting.

Overall, this analysis is a powerful tool for understanding battery performance, predicting battery life, and ensuring effective battery management in applications.