**Steps to Prepare Before Running the Code:**

1. **Create a project directory** with three subfolders: 'model\_measurements', 'estimation\_measurements', and 'outputs'.

* The 'model\_measurements' folder will contain CSV files with data used for creating the battery model.
* The 'estimation\_measurements' folder will contain CSV files with data used for SOC estimation.
* The 'outputs' folder will be empty initially. After running the program, the output files will be saved here.

1. **CSV for battery model creation:** Define a uniquely named CSV file that includes the following measurements: Time [sec], Voltage [V], Current [mA], QDischarge [mA/h], Temperature [°C], and SoC. If multiple models are being created, unique file names will help differentiate them. Save this file in the 'model\_measurements' directory.
2. **CSV for SOC estimation:** Define a uniquely named CSV file that includes the following measurements: Time [sec], Voltage [V], Current [mA], QDischarge [mA/h], Temperature [°C], and SoC. Unique file names will help distinguish between multiple estimations. Save this file in the 'estimation\_measurements' directory.
3. **JSON Configuration File:** Open the attached config.json file in your Python project and configure the parameters as described in the instructions below.

**Json file configuration Overview:**

This JSON file is designed to define all necessary parameters without requiring any changes to the code. The parameters are organized into four sections:

1. **Paths:**

* **Model**: A list of paths to the CSV files containing model measurements.
* **Input\_files**: A list of paths to the CSV files with estimation measurements.
* **output\_directory**: The path to the directory where output files will be saved.

1. **Titles:** This section specifies the titles used in the CSV files.

* **Voltage**: The title for voltage measurements, specified in volts.
* **Current**: The title for current measurements, specified in amperes.
* **q\_counting:** The title for the discharge measurement, indicating the amount of charge the battery consumes, measured in milliampere-hours. This is used when comparing the estimation to the coulomb counting method, especially when a commercial state of charge (SOC) estimation is unavailable.
* **Time**: The title for time measurements, specified in seconds.
* **Temperature**: The title for temperature measurements, specified in degrees Celsius.
* **SOC**: The title for the commercial state of charge (SOC) estimation.

1. **Rint\_model\_params:** Parameters for the Rint battery model (an electrical model) and the type of model to be built.

* **r0**: The resistance, specified in ohms.
* **q**: The amount of charge per battery cell.
* **s**: The number of battery cells connected in series.
* **p**: The number of battery cells connected in parallel.
* **soc\_degree**: The degree of the polynomial approximation for the state of charge. This is a list with two values: the first value is always 1 for linear estimation, and the second value specifies the polynomial degree (default is 6).
* **temperature\_degree**: The degree of the polynomial approximation for temperature. For the Rint model, the default is 0 (indicating that temperature is not used).
* **adjusted\_soc**: Set to 'true' if commercial SOC estimation data is available; otherwise, set to 'false'.
* **temperature\_flag**: Set to 'true' if the model considers the temperature factor (creating a three-dimensional model with SOC as a function of temperature and open-circuit voltage (OCV)); set to 'false' if SOC is modeled only as a function of OCV.
* **Initial\_state**: set the value of initial state of Kalman filter.
* **P:** set the value of covariance matrix of the estimation error.
* **Q:** set the value of covariance matrix of the process noise.
* **R:** set the value of covariance matrix of the measurement noise

1. **thevenin\_model\_params:** Parameters for the Thevenin battery model (another electrical model) and the type of model to be built.

* **r0** The resistance of the series resistor, specified in ohms.
* **r1**: The resistance of the parallel resistor, specified in ohms.
* **c0**: The capacitance, specified in farads.
* **q**: The amount of charge per battery cell.
* **s**: The number of battery cells connected in series.
* **p**: The number of battery cells connected in parallel.
* **Soc\_degree**: The degree of the polynomial approximation for the state of charge. This is a list with two values: the first value is always 1 for a linear model, and the second value specifies the polynomial degree (default is 6).
* **temperature\_degree**: The degree of the polynomial approximation for temperature. This is a list with two values: the first value is always 0 for a non-temperature model, and the second value specifies the polynomial degree (default is 3).
* **adjusted\_soc**: Set to 'true' if commercial SOC estimation data is available; otherwise, set to 'false'.
* **temperature\_flag**: Set to 'true' if the model considers the temperature factor (creating a three-dimensional model with SOC as a function of temperature and OCV); set to 'false' if SOC is modeled only as a function of OCV.
* **Initial\_state**: set the value of initial state of Kalman filter.
* **P:** set the value of covariance matrix of the estimation error
* **Q:** set the value of covariance matrix of the process noise
* **R:** set the value of covariance matrix of the measurement noise

**Code Structure**

The project consists of five Python files and one JSON file:

* **ParseCSV:** This file is responsible for creating a battery model from a CSV file with corresponding measurements. It takes the CSV file containing measurements, the battery model parameters, and the model type parameters, and returns a battery model class instance along with plots of the model. This file utilizes the 'BatteryModel' class from the 'BatteryModel' file to build the model.
* **BatteryModel:** This file defines the 'BatteryModel' class, which holds all the parameters of the model, calculates OCV (Open Circuit Voltage) and SOC (State of Charge), creates vectors of the measurements, fits polynomials to the measurements, and plots graphs. Additionally, it includes two subclasses, 'RintModel' and 'TheveninModel', each implementing specific calculations unique to their respective models.
* **KalmanFilter:** This file defines a 'KalmanFilter' class that implements the Kalman filter algorithm for linear SOC estimation.
* **ExtendedKalmanFilter:** This file defines an 'ExtendedKalmanFilter' class that implements the Extended Kalman filter algorithm for non-linear SOC estimation.
* **Main:** This file manages the entire program. It coordinates the process by calling the other files and building the estimation through several stages. First, it builds the battery models using the 'BatteryModel' file, then it performs predictions using the 'KalmanFilter' and 'ExtendedKalmanFilter' algorithms. It also generates graphs of the results, and finally, it calculates the error and evaluates the quality of the estimation.
* **Config:** json file as described above.