Raw data:

Sample dataset is given below which consists of 9125 rows and 34 columns.

```
data_ =pd.read_csv('/content/drive/My Drive/data/e-cad_data1.csv')
data_.head()
```

	Test Time	Step Time	Cycle	Step	TAH	TWH	Current	Voltage	Power	AHChg	AHDchg	WHChg	WHDchg	Rec#	State	ES	DPt Time	Temp 1	Temp 2	VARx1	V
0	352865.118	0.012	18	9	0.000000	0.000000	0.000001	3.592508	0.000004	0.000000	0.0	0.000000	0.0	8817	C	0	12/22/2016 17:37	30.35	30.22	2.5	
1	352925.118	60.012	18	9	0.000013	0.000049	0.000797	3.669108	0.002923	0.000013	0.0	0.000049	0.0	8818	C	1	12/22/2016 17:38	30.34	30.22	2.5	
2	352985.118	120.012	18	9	0.000027	0.000097	0.000797	3.671244	0.002925	0.000027	0.0	0.000097	0.0	8819	C	1	12/22/2016 17:39	30.31	30.22	2.5	
3	353045.118	180.012	18	9	0.000040	0.000146	0.000797	3.672465	0.002926	0.000040	0.0	0.000146	0.0	8820	C	1	12/22/2016 17:40	30.45	30.30	2.5	
4	353105.118	240.012	18	9	0.000053	0.000195	0.000797	3.673381	0.002926	0.000053	0.0	0.000195	0.0	8821	C	1	12/22/2016 17:41	30.43	30.31	2.5	

First goal of this data analysis is to split the data by Cycle number so that we can use it later for further analysis. One complete cycle consists of a charging cycle, discharging cycle and resting period after each charge and discharge. The Cycle column contains only one value which is 18. This column is not well organized by the right cycle number. So the first task is to assign the right values to the cycle column.

Data Preprocessing:

For the ‘State’ column, we find 6 unique values which is shown below:

```
[ ]  data_['State'].unique()

     array(['C', 'R', 'D', 'P', 'O', 'S'], dtype=object)
```

One complete cycle follow this pattern [‘C’,’R’,’D’,’R’] in “State” Column. But we have 3 other types of values in this column. It is needed to figure out how many values for each type exists and where it exists.

```
df_O=data_[data_['State']=='O']
df_O.head()
```

	Test Time	Step Time	Cycle	Step	TAH	TWH	Current	Voltage	Power	AHChg	AHDchg	WHChg	WHDchg	Rec#	State	ES	DPt Time	Temp 1	Temp 2	VARx1	VARx2	VARx3
9102	891349.782	2700.0	18	19	0.0	0.0	0.0	3.598459	0.0	0.0	0.0	0.0	0.0	17919	O	193	12/31/2016 10:13	30.6	30.47	2.5	0	0

```
[ ]  df_S=data_[data_['State']=='S']
     df_S.head()
```

	Test Time	Step Time	Cycle	Step	TAH	TWH	Current	Voltage	Power	AHChg	AHDchg	WHChg	WHDchg	Rec#	State	ES	DPt Time	Temp 1	Temp 2	VARx1	VARx2	VARx3	VA
9116	891359.778	4.998	18	7	0.0	0.0	0.0	3.598306	0.0	0.0	0.0	0.0	0.0	17933	S	192	12/31/2016 10:14	30.64	30.49	2.5	0	0	

```
df_P=data_[data_['State']=='P']
df_P
```

	Test Time	Step Time	Cycle	Step	TAH	TWH	Current	Voltage	Power	AHChg	AHDchg	WHChg	WHDchg	Rec#	State	ES	DPt Time	Temp 1	Temp 2	VARx1	VARx2	VARx3
9053	888649.680	900.000	18	17	0.0	0.0	0.0	3.593576	0.0	0.0	0.0	0.0	0.0	17870	P	17	12/28/2016 22:27	31.78	31.37	2.5	0	0
9054	888649.782	900.102	18	17	0.0	0.0	0.0	3.598459	0.0	0.0	0.0	0.0	0.0	17871	P	199	12/31/2016 9:28	30.84	30.79	2.5	0	0
9115	891359.778	4.998	18	7	0.0	0.0	0.0	3.598306	0.0	0.0	0.0	0.0	0.0	17932	P	17	12/31/2016 10:14	30.64	30.49	2.5	0	0

We can observe that there are only 5 rows which have ‘O’, ‘S’ and ‘P’ states and all of the values are placed at the end of the dataset and after the last cycle. So we deleted the rows after the last cycle using the following code.

```
data_ = data_[:data_[data_['State'] == 'P'].index[0]]
```

```
data_['State'].unique()
```

```
array(['C', 'R', 'D'], dtype=object)
```

Now we have only 3 states which are required for assigning the cycle number values. Here, ‘C’, ‘R’ and ‘D’ refer to charging state, resting stage and discharge state accordingly. But we have two rest periods in one cycle, one after charging and another after discharge. Renaming two types of resting period will help in further analysis. In this step, we create a loop function which assigns correct values in the Cycle column and rename the resting period stage. Resting period after charging is renamed as ‘RC’ and ‘RD’ for the remaining.

```
items=['C','R','D','R']
pattern_length=len(items)
count=0
current_state=data_.State[0]

#cycle=list()
#temp_df=pd.DataFrame()
cycle_count=0
i=1

for index, row in data_.iterrows():
  data_.at[index, 'Cycle']=i
  if current_state!=row.State[0]:
    count=count+1
    current_state=row.State[0]

    if count==pattern_length:
      data_.at[index, 'Cycle']=i+1
      i=i+1
      count=0
      cycle_count=cycle_count+1
  if count==1:
    data_.at[index, 'State']='RC'
  if count==3:
    data_.at[index, 'State']='RD'
cycle_count=cycle_count+1
```

Here, ‘cycle_count’ gives us the total number of cycles. We found that we have 25 cycles in our dataset.

Cycle 10 analysis:

We can extract data for the cycle which we want to analyze using the pandas function. For example, the following code gives us the data for cycle number 10.

```
df_cycle10 = data_[data_['Cycle']==10]
df_cycle10.reset_index(drop=True, inplace=True)
df_cycle10.head()
```

	Test Time	Step Time	Cycle	Step	TAH	TWH	Current	Voltage	Power	AHChg	AHDchg	WHChg	WHDchg	Rec#	State	ES	DPt Time	Temp 1	Temp 2	VARx1
0	549287.802	0.018	10	9	0.000000	0.000000	0.000797	3.609903	0.002877	0.000000	0.0	0.000000	0.0	12135	C	0	12/25/2016 0:11	30.37	30.27	2.5
1	549347.802	60.018	10	9	0.000013	0.000049	0.000797	3.691386	0.002941	0.000013	0.0	0.000049	0.0	12136	C	1	12/25/2016 0:12	30.53	30.38	2.5
2	549407.802	120.018	10	9	0.000027	0.000098	0.000797	3.693675	0.002943	0.000027	0.0	0.000098	0.0	12137	C	1	12/25/2016 0:13	30.51	30.39	2.5
3	549467.802	180.018	10	9	0.000040	0.000147	0.000797	3.694591	0.002943	0.000040	0.0	0.000147	0.0	12138	C	1	12/25/2016 0:14	30.51	30.39	2.5
4	549527.802	240.018	10	9	0.000053	0.000196	0.000797	3.695201	0.002944	0.000053	0.0	0.000196	0.0	12139	C	1	12/25/2016 0:15	30.57	30.44	2.5

For a specific cycle number, four separate data frames are created for ‘C’, ‘RC’, ‘D’ and ‘RD’ state accordingly. Using those data frames, we can extract charging capacity and voltage as well as discharge capacity and voltage. A separate dataset is created using those extracted data.

```
data = [cycle10_C["AHChg"], cycle10_C["Voltage"], cycle10_D["AHDchg"], cycle10_D["Voltage"]]
headers = ["Charging Capacity", "Charging Voltage","Discharge Capacity", "Discharge Voltage" ]
cycle10_ = pd.concat(data, axis=1, keys=headers)
cycle10_.reset_index(drop=True,inplace=True)
cycle10_
```

	Charging Capacity	Charging Voltage	Discharge Capacity	Discharge Voltage
0	0.000000	3.609903	0.000000	4.418631
1	0.000013	3.691386	-0.000013	4.361563
2	0.000027	3.693675	-0.000027	4.347982
3	0.000040	3.694591	-0.000040	4.336233
4	0.000053	3.695201	-0.000053	4.324941

After that, we take the absolute value of discharge capacity and then we converted both charging and discharge capacity from Ah to mAh by multiplying 1000.

```
cycle10_['Discharge Capacity'] = cycle10_['Discharge Capacity'].abs()*1000
cycle10_['Charging Capacity'] = cycle10_['Charging Capacity']*1000
cycle10_.head()
```

	Charging Capacity	Charging Voltage	Discharge Capacity	Discharge Voltage
0	0.000	3.609903	0.000	4.418631
1	0.013	3.691386	0.013	4.361563
2	0.027	3.693675	0.027	4.347982
3	0.040	3.694591	0.040	4.336233
4	0.053	3.695201	0.053	4.324941

Then, we plotted capacity against voltage for both charging and discharge state. Following is the plotted figure.

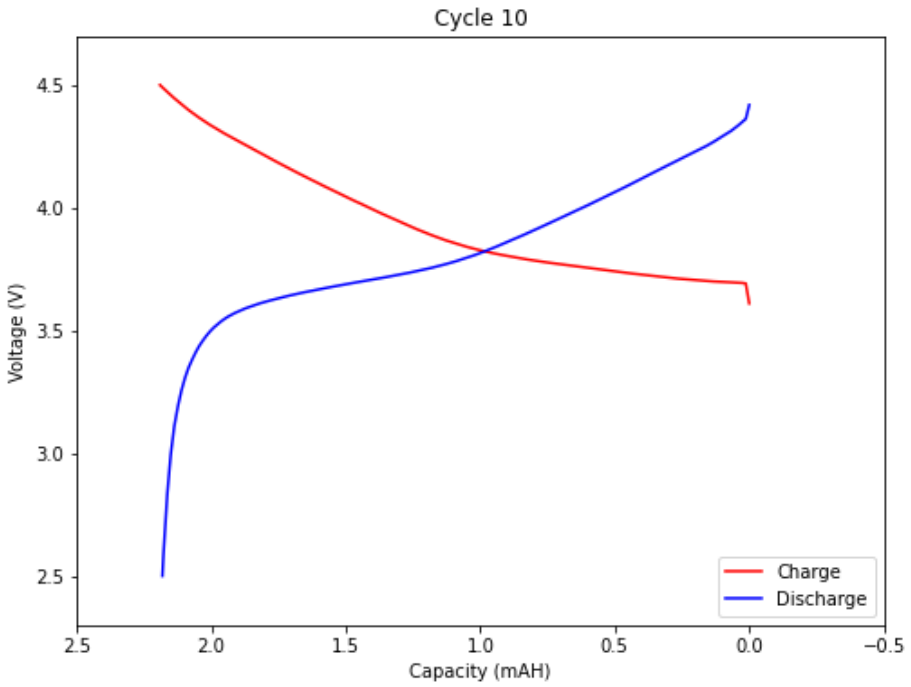


Fig 1: Capacity vs Voltage for cycle 10

Now, we can find specific capacity from the capacity by using the following formula:

$$Charging\ Specific\ Capacity = \frac{charging\ capacity}{mass\ of\ the\ electrode}$$

$$discharge\ specific\ capacity = \frac{Last\ value\ of\ charging\ capacity}{mass\ of\ the\ electrode} \times 1000 - \frac{discharge\ capacity}{mass\ of\ the\ electrode}$$

After finding out the specific capacity, we added this in our data frame.

```
cycle10_['Charging Specific Capacity'] = cycle10_['Charging Capacity']/0.01317
cycle10_['Discharge Specific Capacity'] = (cycle10_C['AHChg'].iloc[-1]/0.01317*1000) -cycle10_['Discharge Capacity']/0.01317
#166.28702
cycle10_.head()
```

	Charging Capacity	Charging Voltage	Discharge Capacity	Discharge Voltage	Charging Specific Capacity	Discharge Specific Capacity
0	0.000	3.609903	0.000	4.418631	0.000000	166.362946
1	0.013	3.691386	0.013	4.361563	0.987092	165.375854
2	0.027	3.693675	0.027	4.347982	2.050114	164.312832
3	0.040	3.694591	0.040	4.336233	3.037206	163.325740
4	0.053	3.695201	0.053	4.324941	4.024298	162.338648

Now, we plotted specific capacity against voltage and following is the output.

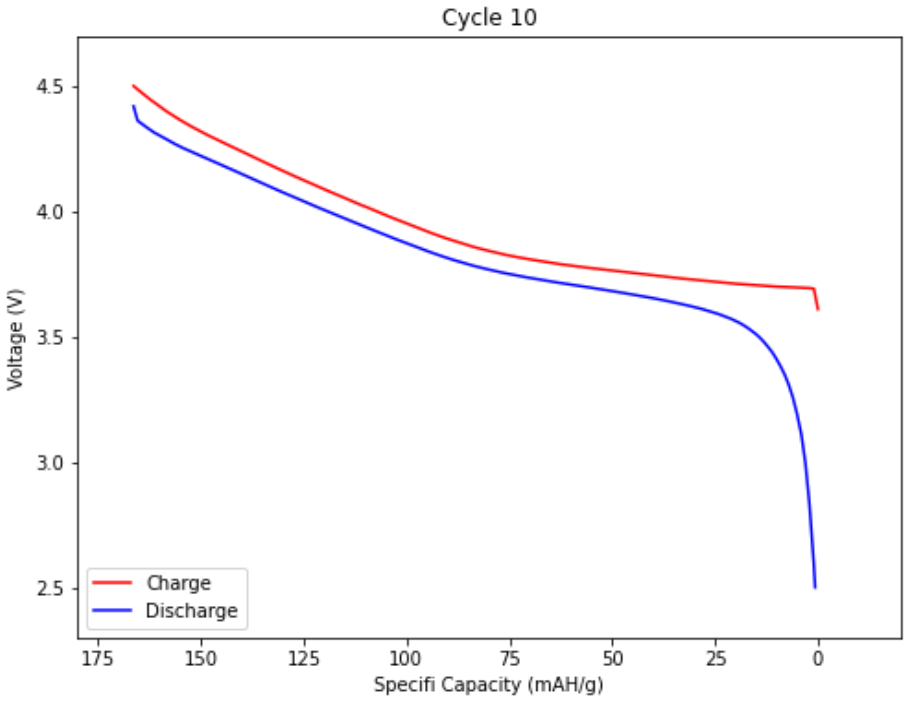


Fig 2: Specific capacity vs Voltage for Cycle 10

Then we need to find out the compensated voltage. The formula for finding the compensated voltage is given below:

$$Compensated\ Charging\ Voltage = Charging\ Voltage - (last\ row\ value\ of\ charging\ voltage - 2nd\ row\ value\ of\ RC\ voltage)/2$$

$$Compensated\ Discharge\ Voltage = Discharge\ Voltage + (last\ row\ value\ of\ RC\ voltage - 2nd\ row\ value\ of\ discharge\ voltage)/2$$

Applying this formula, we found the compensated charging and discharge voltage and added it to the data frame.

```
cycle10_['Compensated Charging Voltage'] = cycle10_['Charging Voltage']-(cycle10_C['Voltage'].iloc[-1]-cycle10_RC['Voltage'].iloc[1])/2
cycle10_['Compensated Discharge Voltage'] = cycle10_['Discharge Voltage']+(cycle10_RC['Voltage'].iloc[-1]-cycle10_D['Voltage'].iloc[1])/2
cycle10_.head()
```

	Charging Capacity	Charging Voltage	Discharge Capacity	Discharge Voltage	Charging Specific Capacity	Discharge Specific Capacity	Compensated Charging Voltage	Compensated Discharge Voltage
0	0.000	3.609903	0.000	4.418631	0.000000	166.362946	3.582666	4.453040
1	0.013	3.691386	0.013	4.361563	0.987092	165.375854	3.664149	4.395972
2	0.027	3.693675	0.027	4.347982	2.050114	164.312832	3.666438	4.382391
3	0.040	3.694591	0.040	4.336233	3.037206	163.325740	3.667353	4.370642
4	0.053	3.695201	0.053	4.324941	4.024298	162.338648	3.667964	4.359350

Now, we import the ideal pseudo SOC (State of charge) dataset which is as follows.

```
data_1 =pd.read_csv('/content/drive/My Drive/data/Ideal_SOC.csv')
data_1=data_1.loc[:,['SOC (%)','Voltage (V)']]
data_1.head()
```

	SOC (%)	Voltage (V)
0	-6.530000e-09	3.03655
1	1.000000e-03	3.09147
2	2.000000e-03	3.13393
3	3.000000e-03	3.17374
4	4.000000e-03	3.21058

Pseudo-OCV-SOC curve is shown in following figure:

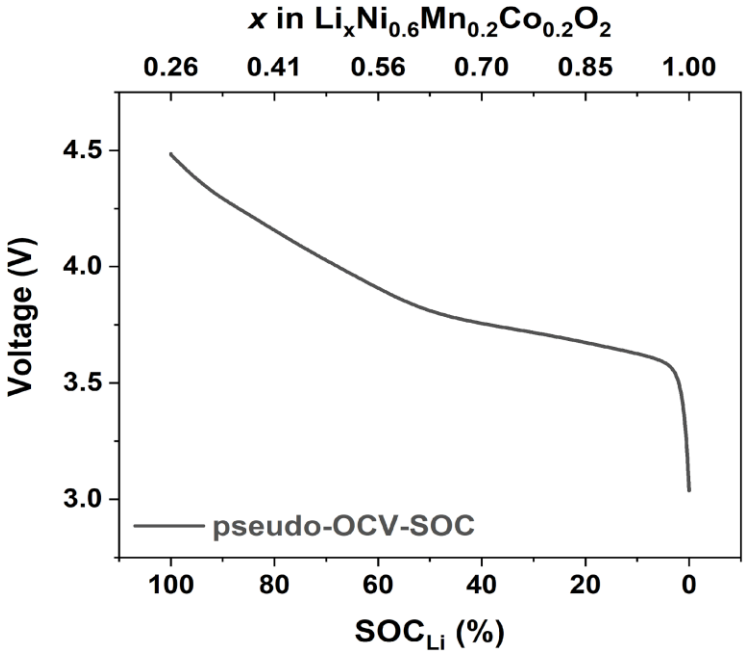


Fig 3: Ideal Pseudo SOC curve

In this stage, we need to do a linear transformation for transforming specific capacity to SOC given that voltage 4.1 to 3.9 will be the same as ideal SOC curve. For doing that, we need to find specific capacity for 4.1 and 3.9. We did that using linear interpolation. Let, A and B are the specific capacity for voltage 4.1 and 3.9 respectively. And P and Q are the SOC for voltage 4.1 and 3.9 respectively.

Now, we need to find out the slope (m) and intersect (c) using following equation:

$$P = mA + c$$

$$Q = mB + c$$

$$\begin{bmatrix} P \\ Q \end{bmatrix} = \begin{bmatrix} A & 1 \\ B & 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix}$$

$$\begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} A & 1 \\ B & 1 \end{bmatrix}^{-1} \begin{bmatrix} P \\ Q \end{bmatrix}$$

Using m and c we can find out the relation between SOC and specific capacity.

$$SOC = m(specific\ capacity) \times c$$

This formula is used for both converting both charging and discharge specific capacity into SOC. We are converting the specific capacity to SOC so that we can plot charging and discharge voltage against SOC and compare the result with ideal SOC.

Following figure shows the plot of SOC against voltage for charging, discharge and pseudo:

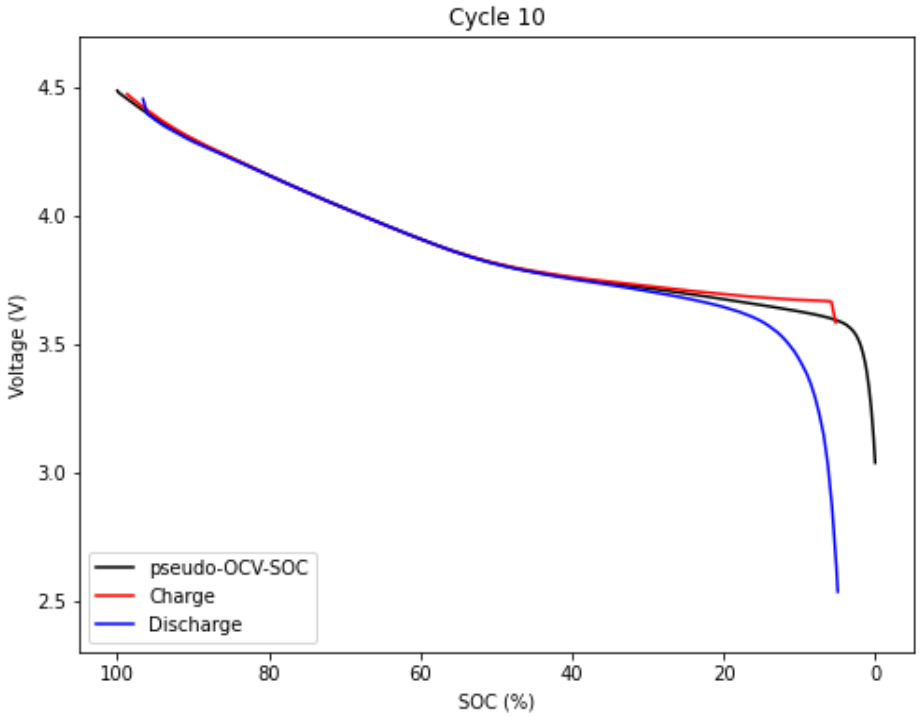


Fig 4: SOC(%) vs Voltage curve for cycle 10

Universal Function generation:

After all of the process, we generate a universal function which will take dataset as input and we need to mention the cycle number (N) we would like to analyze and need to provide the mass of the electrode (M) and we will get all of the desired graph as the output for that desired cycle number. We will also get the total number of cycles that exist on the raw dataset. CC will give the total cycle number and Cycle_ gives the data frame for cycle number N.

```
N=int(input('Enter the cycle number you want to analyze: '))
M=float(input('Enter the mass of elctrode: '))
CC, Cycle_, plot=Cycle(N,data_,M)
```

When we enter N=1 and M=0.01317, we got the following result:

```
Cycle_
```

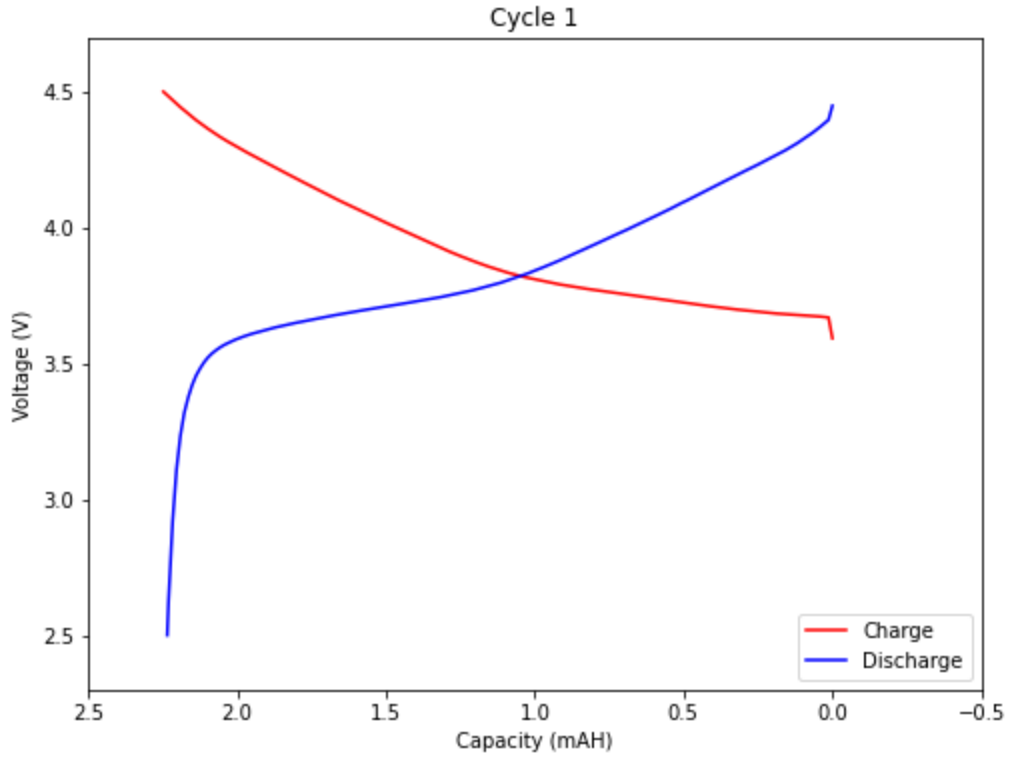
	Charging Capacity	Charging Voltage	Discharge Capacity	Discharge Voltage	Charging Specific Capacity	Discharge Specific Capacity	Compensated Charging Voltage	Compensated Discharge Voltage	Charge SOC(%)	Discharge SOC(%)
0	0.000	3.592508	0.000	4.447929	0.000000	170.690964	3.572213	4.473640	0.043023	0.981991
1	0.013	3.669108	0.013	4.396201	0.987092	169.703872	3.648814	4.421912	0.048504	0.976535
2	0.027	3.671244	0.027	4.383078	2.050114	168.640850	3.650950	4.408789	0.054406	0.970659
3	0.040	3.672465	0.040	4.371328	3.037206	167.653759	3.652171	4.397040	0.059887	0.965203
4	0.053	3.673381	0.053	4.359884	4.024298	166.666667	3.653086	4.385595	0.065369	0.959747
...	...	...	...	...	...	...	...	...	...	...
166	2.204	4.455863	2.204	3.108797	167.350038	3.340926	4.435569	3.134508	0.972278	0.057000
167	2.217	4.468833	2.217	2.918517	168.337130	2.353834	4.448539	2.944228	0.977759	0.051544
168	2.231	4.482414	2.231	2.619135	169.400152	1.290812	4.462119	2.644846	0.983661	0.045668
169	2.244	4.495995	2.235	2.499962	170.387244	0.987092	4.475700	2.525673	0.989143	0.043989
170	2.248	4.499809	NaN	NaN	170.690964	NaN	4.479515	NaN	0.990829	NaN

171 rows × 10 columns

```
CC
```

```
25
```

```
Enter the cycle number you want to analyze: 1
Enter the mass of elctrode: 0.01317
```



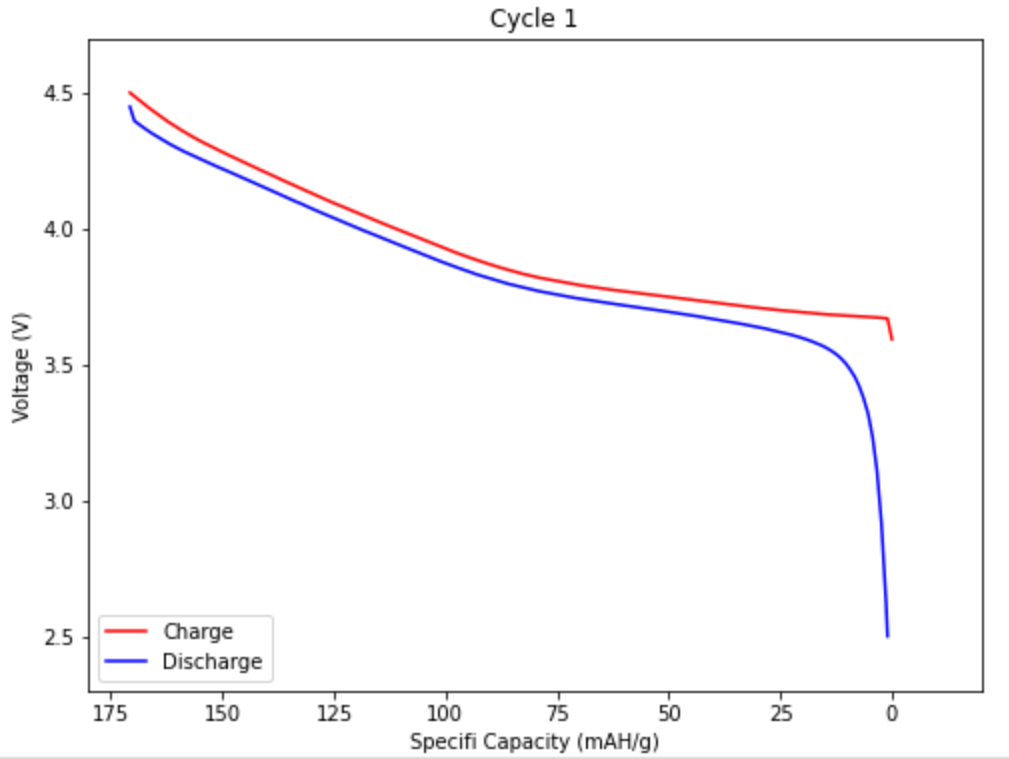


Fig 5: How function is working

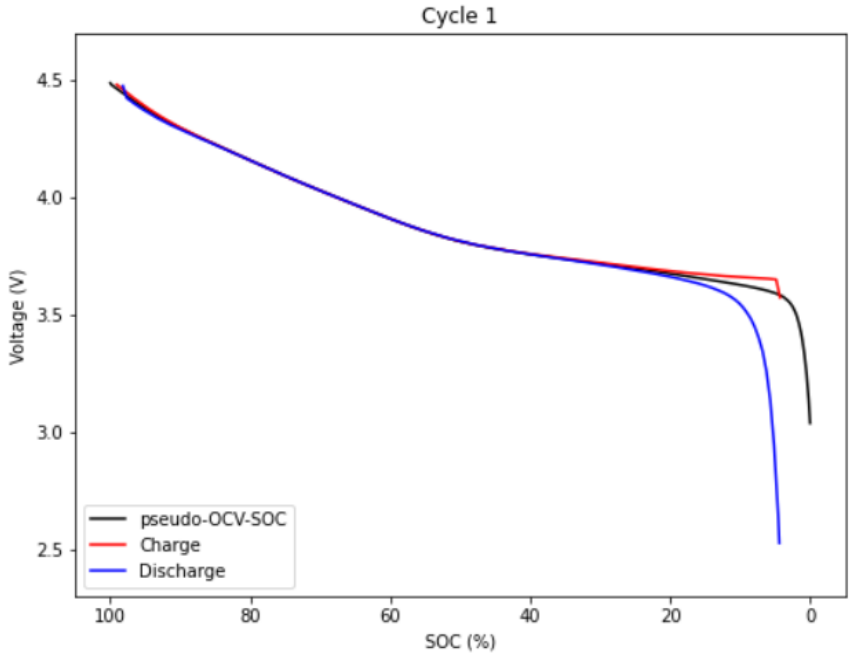


Fig 6: Output of function for cycle number 1

When we enter N=25, it will give us the analysis for cycle number 25.

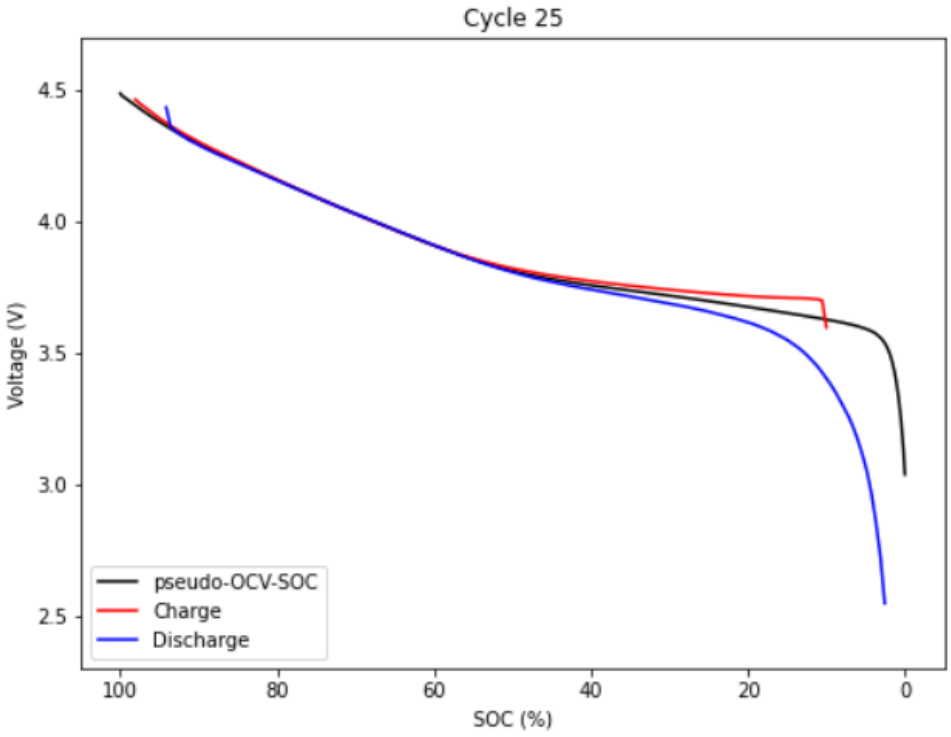


Fig 7: output of function for cycle number 25

Different Cycle Analysis:

It is observed that more deviation is noticeable with the increase of cycle number. For understanding this, we plotted SOC vs Voltage graph for cycle 1,5,10,15,20,25. In the last cycle, we observed the highest deviation.

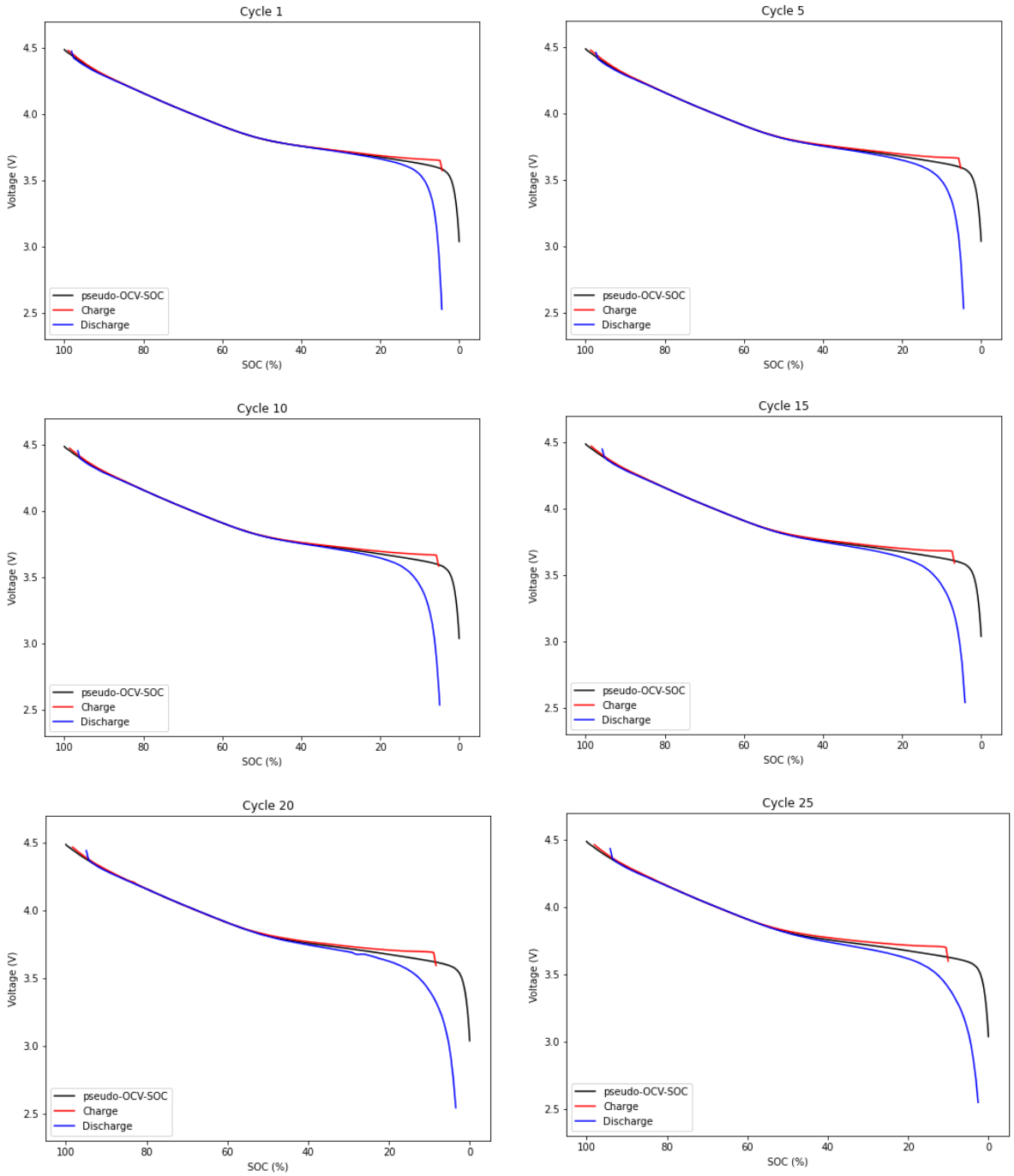


Fig 8: Different cycle analysis

Pseudo SOC Curve Fitting:

After trying lots of different fitting, we found that a smoothing spline is giving us the best fit. The amount of smoothing that smooth.spline applies is controlled by the argument spar. We can examine the effect of spar by producing several plots, each with a different value of spar.

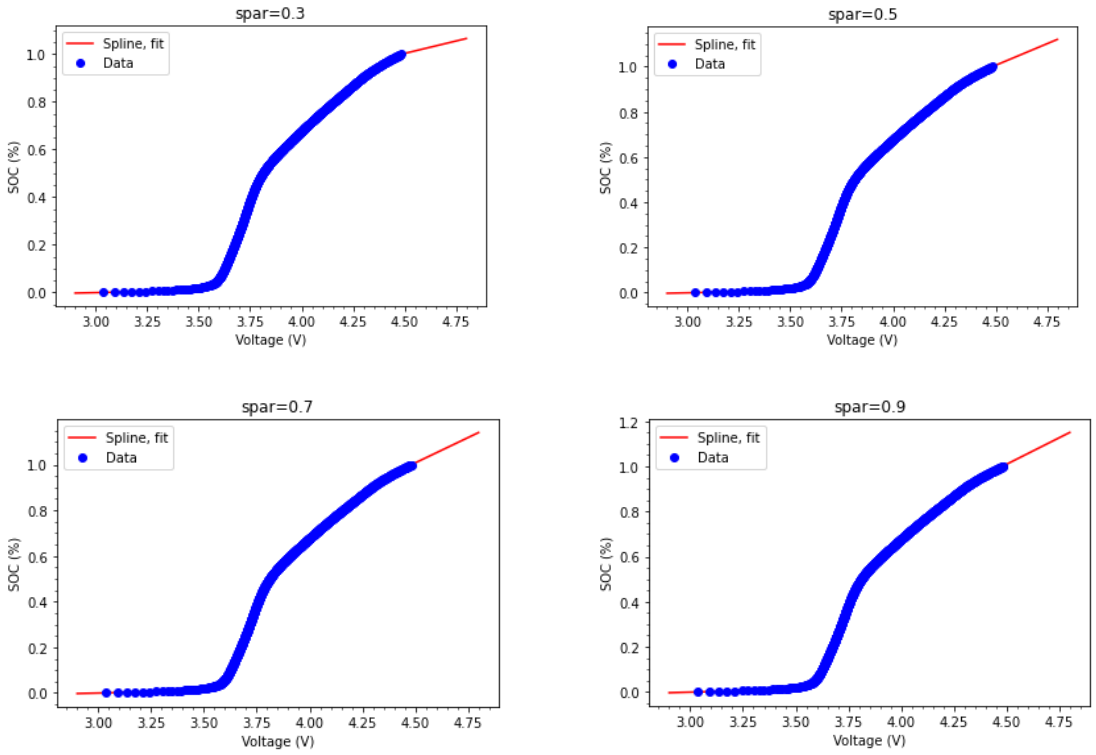


Fig 9: Spline fit with different spar values

Machine Learning Model for SOC prediction:

We build a linear regression model as well as a random forest regression model to predict SOC from the voltage. We used 80% of the data for the model training and the rest 20% is used for testing the model. Following are the results from both regression model:

Model	Training Accuracy	Testing Accuracy
Linear Regression	93.33%	93.9%
Random Forest Regression	99%	99%

From this table, we can say that the random forest model performs better for SOC prediction. Following figure showed the original vs predicted value for the testing data for both of the models.

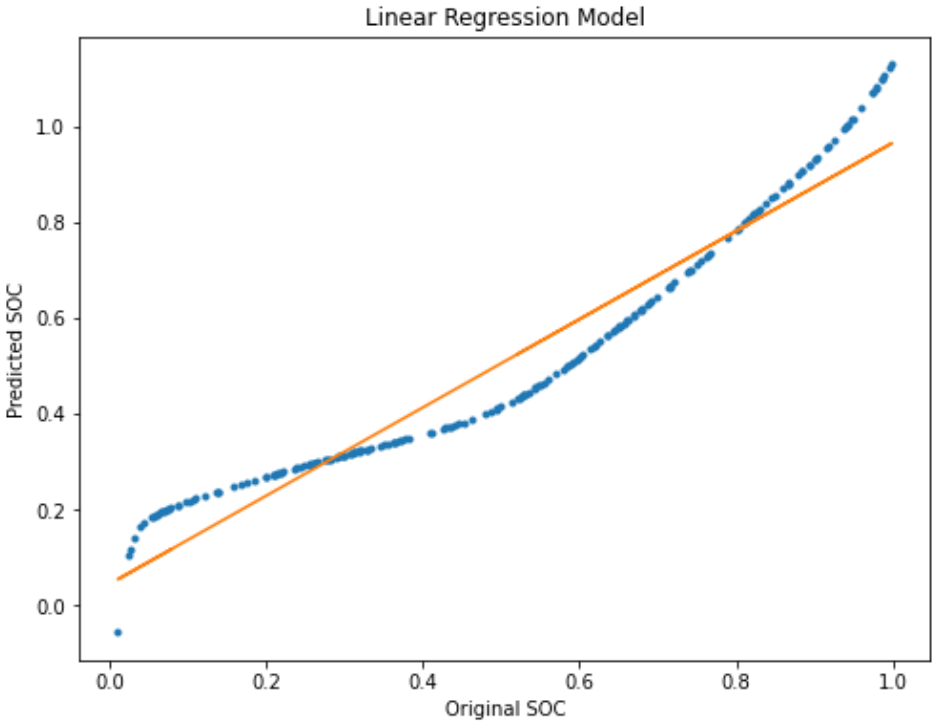


Fig 10: Original vs predicted SOC for Linear regression model

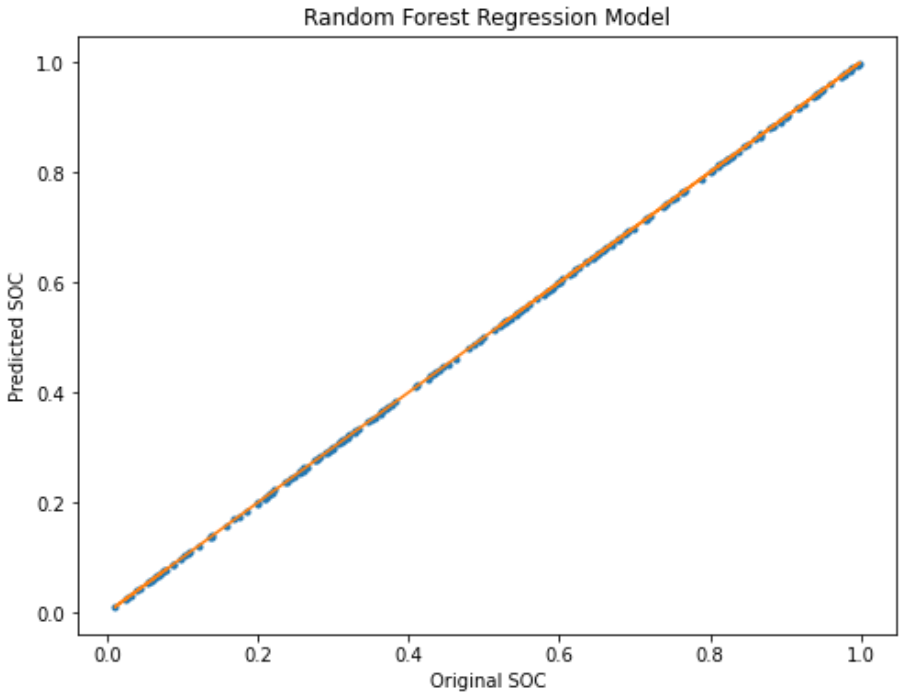


Fig 11: Original vs predicted SOC for random forest regression model

Conclusion:

In this project, we were able to fix the problem in raw data and assigned the right values to the Cycle column. It helps us to work with specific cycles. We developed an application which will give the required graph as output. Previously, this work has been done manually in excel. Now, the python environment is used to make it automated. We just need to give the specific cycle number we want to analyze and the mass of the electrode, it will automatically generate all graphs and data frames. This makes it less time consuming and efficient. In the future, we can build more data analysis and machine learning models to find out why there is deviation with the increase of cycle number.