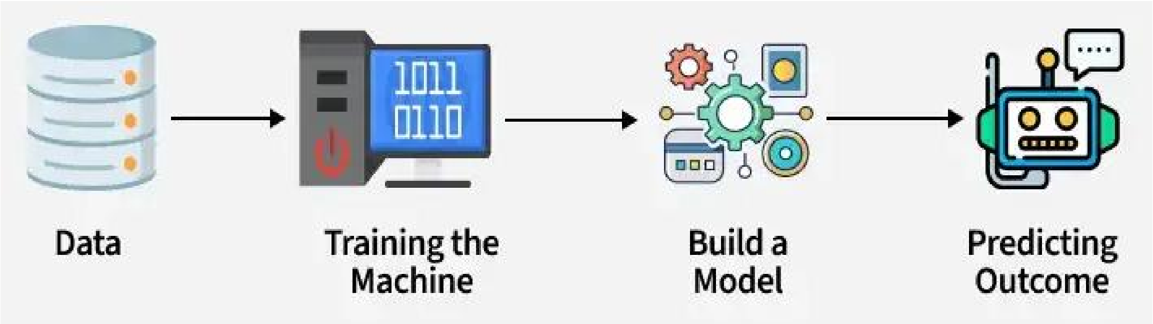
Northwestern Polytechnical University, Xi'an China
Master's Degree in Aerospace Science and technologies
Self Project
Academic Year 2024-2025

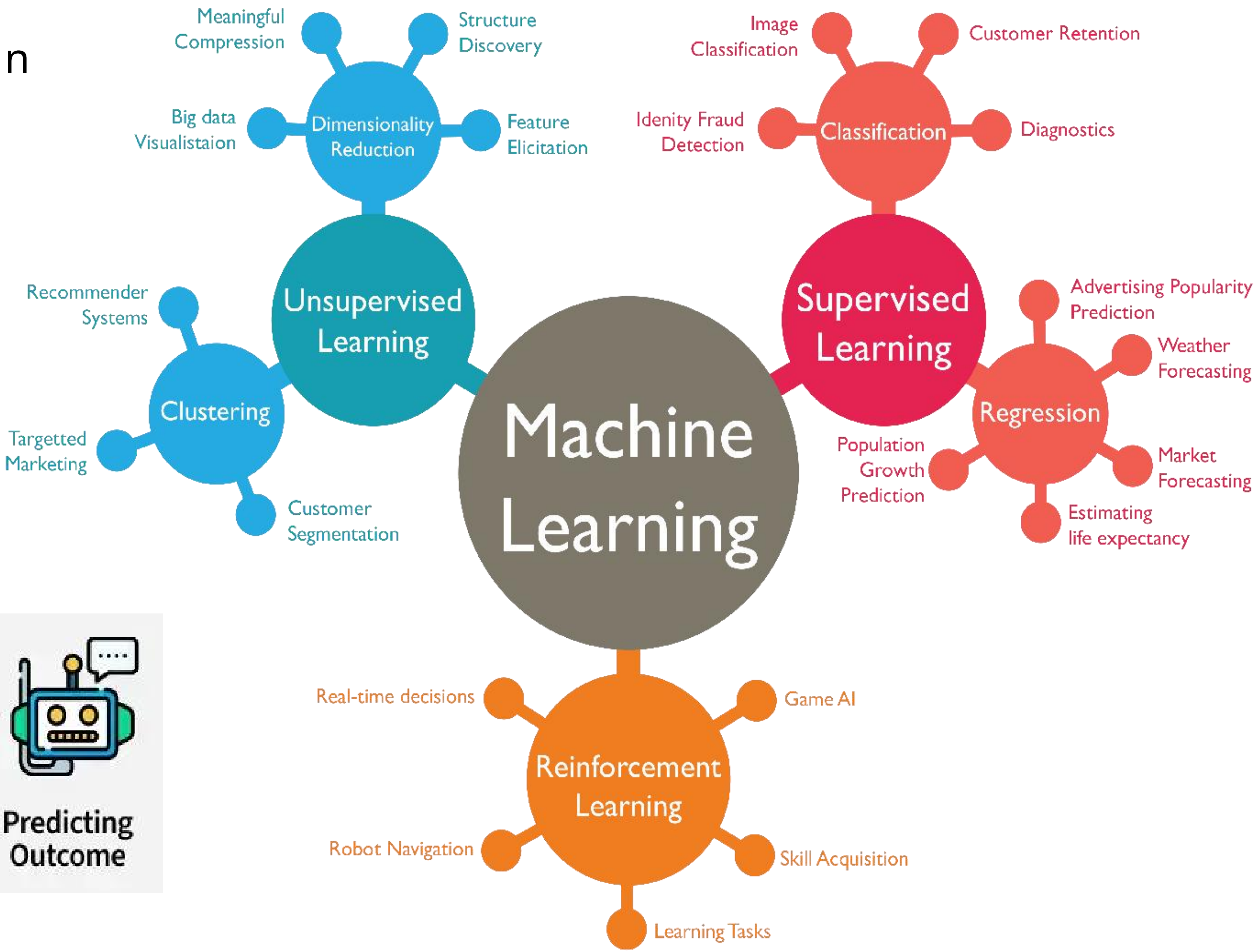
Lithium ion battery State of Charge estimation using Bidirectional Long Short Term Memory(LSTM)

Author:
Amrit Dhakal

Table of Contents

1. Introduction and dataset selection
2. Data processsing and EDA
3. Data preparation functions
4. Model selection and training
5. Model evaluation
6. Conclusion and references





1. Introduction

- LIBs are also widely used in the mobile device industry, aerospace and aviation industry, and defense industry.
- All of these contribute to a rapidly increasing LIB market. However, despite its growing market and relatively good performance, climate change and particularly electric vehicle (EV) applications push for lower costs and higher energy densities over a long lifetime. Unfortunately, these metrics are generally tradeoffs, and therefore, understanding and modeling is critical for optimizing LIB performance.

Technology	Specific Energy (Wh/kg)		Cycle Life (number)	
	State-of-the-art	Future projection	State-of-the-art	Future projection
Ni-Cd battery	50-60	-	2000-2500	-
Li-ion battery	100-265	450	>300	400-450
Li-S battery	250-300	800-950	-	1000
Li-air battery	300-350	1300-1600	>50	500
Supercapacitor	5-15	200-300	∞	∞
Fuel cell	100*	500*	-	-

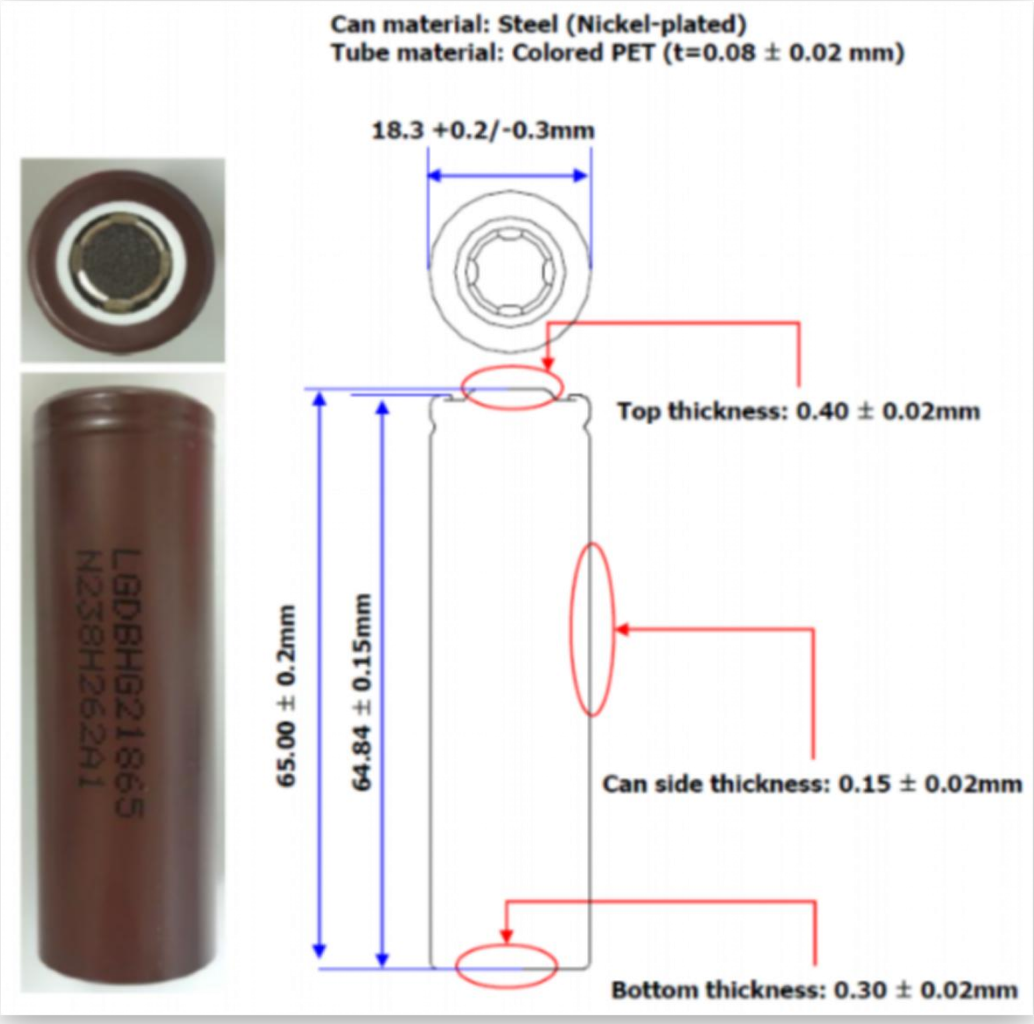
*Specific power (W/kg)

1. Battery Dataset used for this study

- LG_HG2_Original_Dataset_McMasterUniversity_Jan_2020

- A series of tests were performed at six different temperatures, and **the battery was charged after each test at 1C rate to 4.2V, 50mA cut off**, with battery temperature 22degC or greater. The tests were performed as follows:
    - Four pulse discharge **HPPC test** (1, 2, 4, and 6C discharge and 0.5, 1, 1.5, and 2C charge, with reduced values at lower temperatures) performed at 100, 95, 90, 80, 70..., 20, 15, 10, 5, 2.5, 0 % SOC.
    - **C/20 Discharge and Charge test**.
    - **0.5C, 2C, and two 1C discharge tests**. The first 1C discharge test is performed before the UDDS cycle, and the second is performed before the Mix3 cycle.
    - Series of four drive cycles performed, in following order: **UDDS, HWFET, LA92, US06**.
    - A series of **eight drive cycles (mix 1-8)** consist of random mix of UDDS, HWFET, LA92, US06. The drive cycle power profile is calculated for a single LG HG2 cell in a compact electric vehicle.
    - The **previous tests are repeated** for ambient **temperatures of 40degC, 25degC, 10degC, 0degC, -10degC, and -20degC**, in that order. For tests with ambient temperature below 10degC, a reduced regen current limit is set to prevent premature aging of the cells. The drive cycle power profiles are repeated until 95% of the 1C discharge capacity at the respective temperature has been discharged from the cell.

1. Battery specs



Specification	Value
Battery Chemistry	Li[NiMnCo]O2 (H-NMC) / Graphite + SiO
Nominal Voltage	3.6 V
Charge Current (Normal)	1.5 A (constant current), 4.2V, 50 mA End current (constant voltage)
Charge Current (Fast)	4 A (constant current), 4.2V, 100 mA End current (constant voltage)
End Current (Fast)	100 mA (constant voltage)
Discharge	2 V (End Voltage), 20 A Maximum Continuous Discharge Current
Nominal Capacity	3.0 Ah
Energy Density	240 Wh/kg

Data processing and EDA

2. Data Loading using pandas dataframe

- Dataset contains files in ".csv" format so we use *pandas.read_csv("file_path")* to load these files with appropriate file path
- using

```
l = os.listdir(train1)
l = [i for i in l if i.endswith(".csv")]
```

we can see all the files ending with ".csv" file format

- For Explorotory data analysis we choose a single file
- Load it into dataframe and see first few features and label
- We can see

	10/29/2018 3:53:42 AM	39	TABLE	16:50:37.323	00:00:01.673	1	1.1	LG_HG2_CyclesA	4.19155	-0.05108	23.76583	-0.00000	-0.00000.1	5.00000	Unnamed: 14
0	10/29/2018 3:53:42 AM	39	TABLE	16:50:37.423	00:00:01.773	1	1	LG_HG2_CyclesA	4.19088	-0.08173	23.76583	-0.00000	-0.00001	5.0	NaN
1	10/29/2018 3:53:42 AM	39	TABLE	16:50:37.522	00:00:01.872	1	1	LG_HG2_CyclesA	4.19054	-0.08939	23.76583	-0.00000	-0.00002	5.0	NaN
2	10/29/2018 3:53:42 AM	39	TABLE	16:50:37.622	00:00:01.972	1	1	LG_HG2_CyclesA	4.19037	-0.09195	23.76583	-0.00001	-0.00003	5.0	NaN
3	10/29/2018 3:53:42 AM	39	TABLE	16:50:37.723	00:00:02.073	1	1	LG_HG2_CyclesA	4.19037	-0.09195	23.76583	-0.00001	-0.00004	5.0	NaN
4	10/29/2018 3:53:42 AM	39	TABLE	16:50:37.821	00:00:02.171	1	1	LG_HG2_CyclesA	4.19037	-0.09195	23.76583	-0.00001	-0.00005	5.0	NaN

2. Feature names based on type of data

- Next we define following columns to be included in our new datfrrame and see the dataframe's first few elements

	Time Stamp	Step	Status	Prog Time	Step Time	Cycle	Cycle Level	Procedure	Voltage	Current	Temperature	Capacity	WhAccu	Cnt
0	10/29/2018 3:53:42 AM	39	TABLE	16:50:37.423	00:00:01.773	1	1	LG_HG2_CyclesA	4.19088	-0.08173	23.76583	-0.00000	-0.00001	5.0
1	10/29/2018 3:53:42 AM	39	TABLE	16:50:37.522	00:00:01.872	1	1	LG_HG2_CyclesA	4.19054	-0.08939	23.76583	-0.00000	-0.00002	5.0
2	10/29/2018 3:53:42 AM	39	TABLE	16:50:37.622	00:00:01.972	1	1	LG_HG2_CyclesA	4.19037	-0.09195	23.76583	-0.00001	-0.00003	5.0

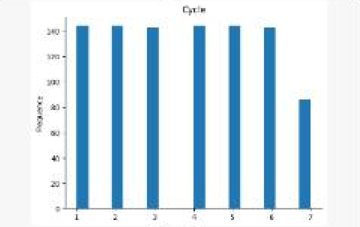
- We include the "DCH" or "TABLE" alias data only in our dataframe like this:

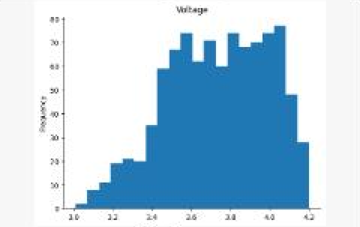
```
df_train1=df_train1[(df_train1["Status"]=="DCH") | (df_train1["Status"]=="TABLE")]
```

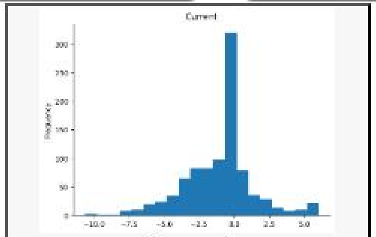
- Next we want to plot and visualize the data

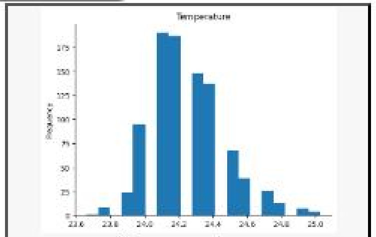
2. Initial Data visualization Overview

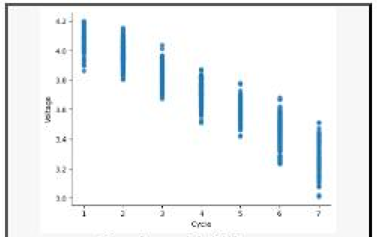
- We use matplotlib and seaborn library for plotting, we try to visulize the data as much as possible to know relationship between features
- Here we have plotted values and their frequency for first four plots, then we have cycle vs voltage, voltage vs current, current vs temperature, temperature vs capacity. Finally, various features Voltage, current and temperature with respect to time cycle

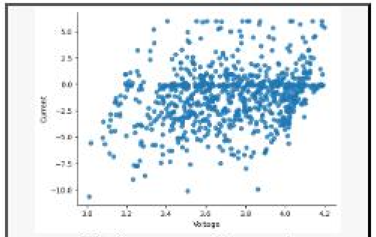

Cycle

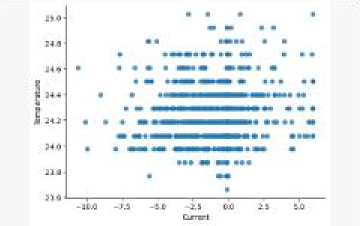

Voltage

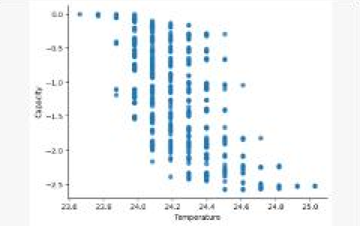

Current

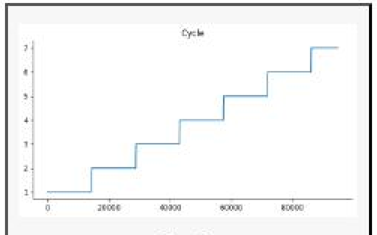

Temperature

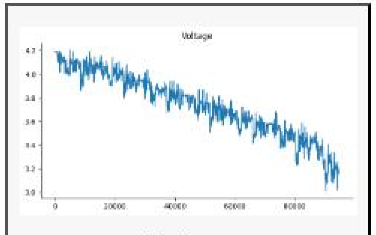

Cycle vs Voltage

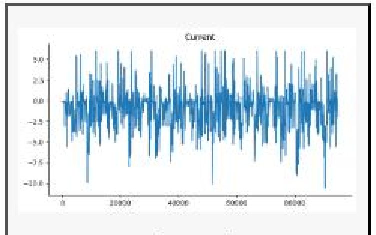

Voltage vs Current

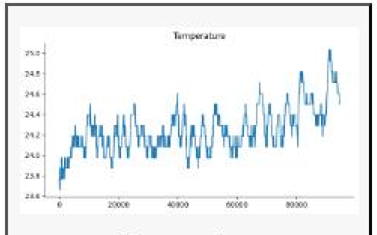

Current vs Temperature


Temperature vs Capacity


Cycle


Voltage


Current


Temperature

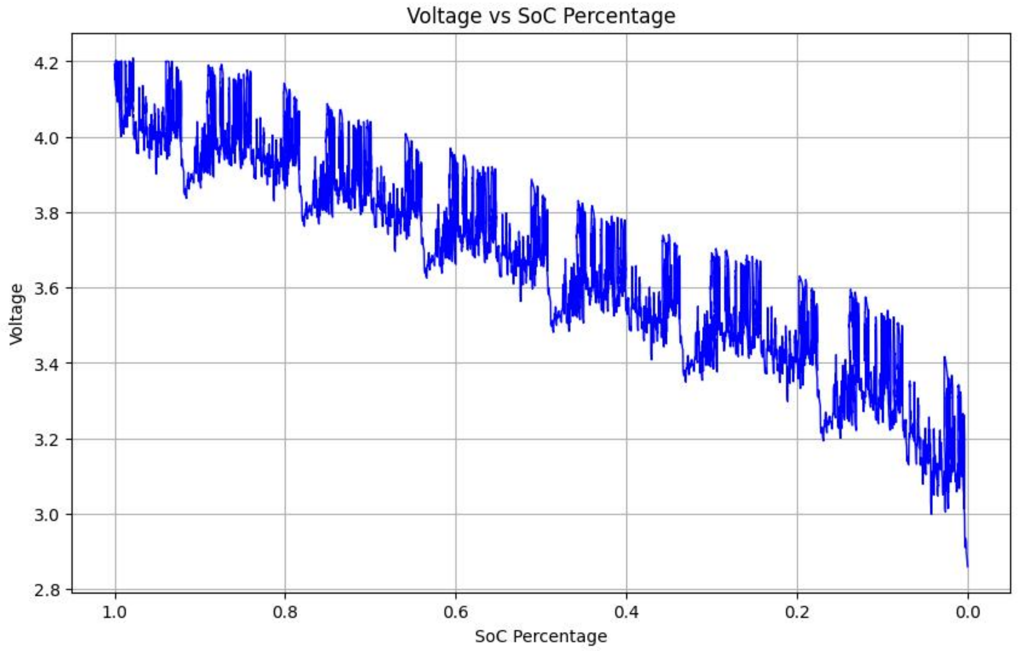
2. Calculating label: State of Charge(SOC)

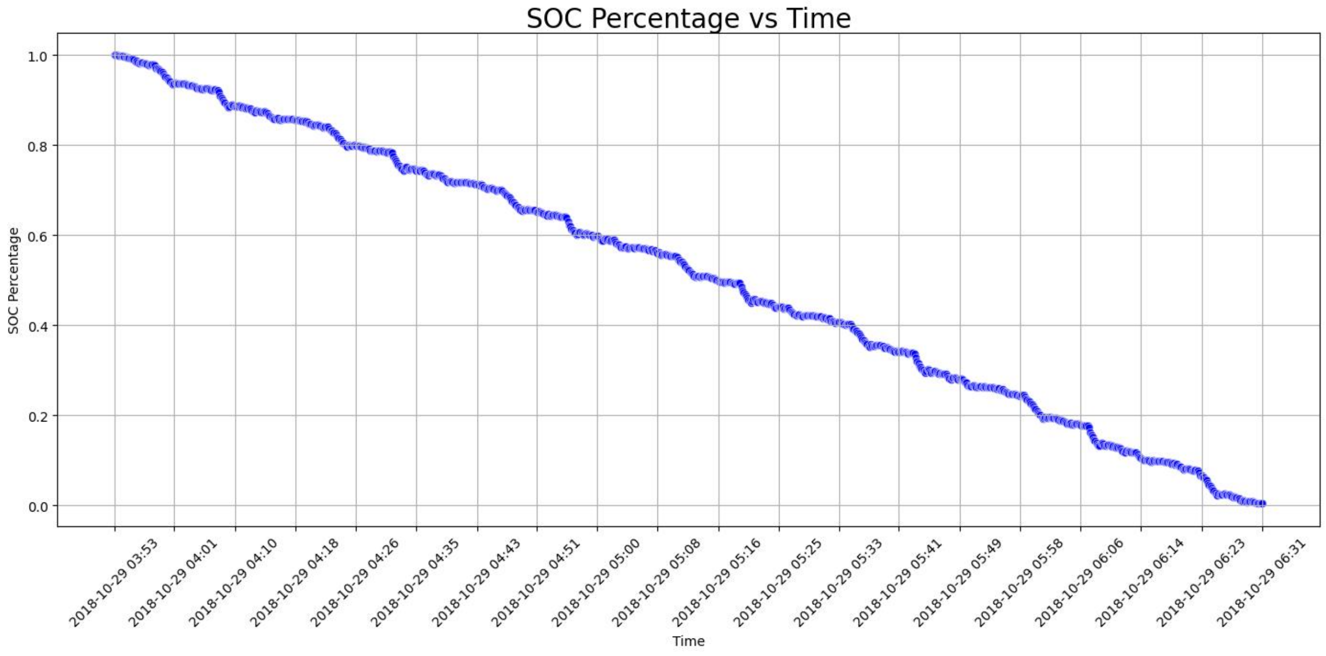
SOC(%) is our **label**, Calculated by using the capacity (Ah) method given below:

- Determine Maximum Discharge Capacity= $C_{max}|min(C_{capacity})|$, Since the capacity values are negative during discharge, take the absolute minimum value.
- Compute SoC Capacity $C_{soc} = C_{max} + C_{capacity}$. This shifts the capacity values to ensure they start from zero.
- Calculate SoC Percentage $SOC_{\%} = \frac{C_{soc}}{max(C_{soc})}$, Normalize the SoC capacity by dividing it by its maximum value to express it as a percentage.

2. Plotting SOC w.r.t. Voltage(V) and time

- These graph show the decresing trend in SOC,
- As the experiment continues, that is time inceases we can see both Voltage and SOC are in decreasing trend, which is to be expected because we are discharging the battery.

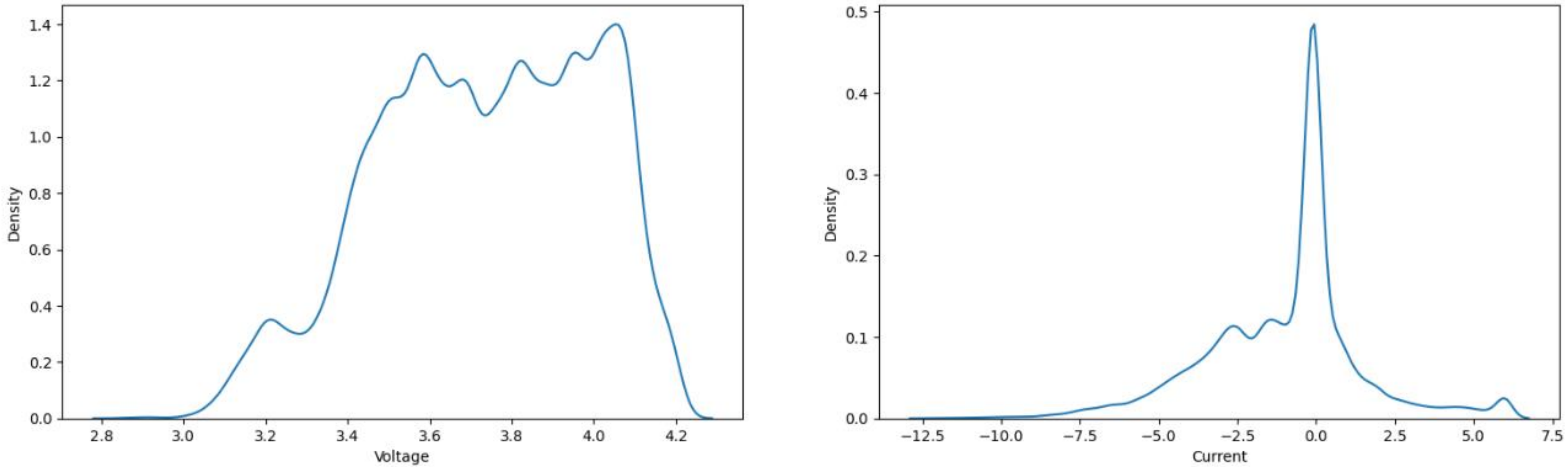




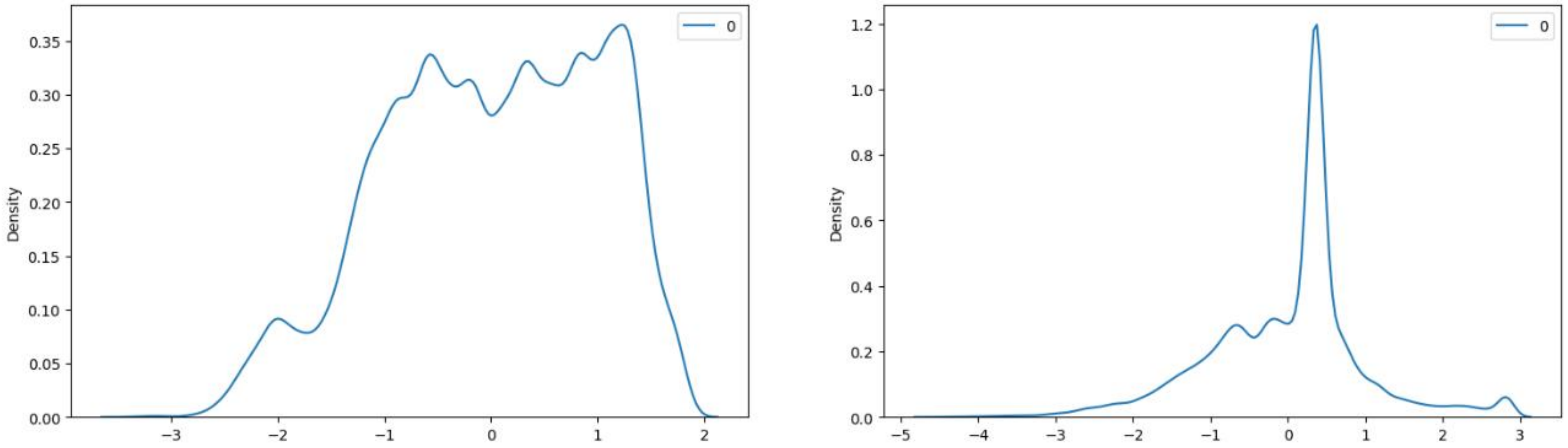
2. Kernel Density Estimation (KDE) plot

- KDE is a smooth, continuous estimate of the distribution of a dataset, showing where the values of each variable are more concentrated

- Original KDE plot:



- Normalized KDE plot:



Data Preparation

3. Features and Label Selection

- x contains the input features: Voltage, Current, and Temperature.
- y contains the target output: SoC Percentage.
- we convert the dataframe to numpy array

```
x = df_train1[["Voltage", "Current", "Temperature"]].to_numpy()
y = df_train1[["SoC Percentage"]].to_numpy()
```

- We create an empty list called cycles. The list is intended **to store pairs of data (features and target values) for each cycle or observation**.

```
cycles = []
cycles.append((x, y))
```

- We appends **a tuple (x, y) to the cycles list**.
- **x represents the feature matrix** containing the columns Voltage, Current, and Temperature.
- **y represents the target vector** containing the SoC Percentage.
- cycles will eventually contain multiple tuples, each representing the input-output pair for different cycles or observations.

3. Functions to load data

Here we define various functions to automate the data loading process for various temperatures and cycles.

- **get_dishcharge_whole_cycle:** returns the train and test data for all cycles according to the file paths provided and scales according to the user input
- **_get_data:** Returns a list of (x, y) tuples for every file path
- **_scale_x:** Flatten train/test data to fit MinMaxScaler using sklearn library
- **_time_string_to_seconds:** to convert all times to seconds
- **get_discharge_multiple_steps**: Splits the given cycle data into multiple time steps for training and testing.
- **_split_to_multiple_step**: Split cycles into multiple steps, returns x and y as numpy arrays containing all the split sequences for training.
- **keep_only_y_end**: This function is used to extract specific parts of the target (y) data based on the provided step size and whether the model is stateful or not.

Model selection and training

4. Tensorflow library and model selection

- *import tensorflow as tf*
- *from tensorflow import keras*
- *from tensorflow.keras import layers*
- *from keras.models import* ***Sequential***
  - where layers are stacked linearly.
- *from keras.layers import* ***Dense, Dropout, Activation, InputLayer***:
  - Dense: Fully connected layer.
  - Dropout: Regularization technique to reduce overfitting.
  - Activation: Defines activation functions (e.g., ReLU, Sigmoid).
  - InputLayer: Layer to define the input shape to the model.
- *from tensorflow.keras.****optimizers*** *import SGD,* ***Adam***
  - Adaptive Moment Estimation, an adaptive optimizer often used for deep learning models.

4. Additional Layers:

- *from keras.layers import LSTM, BatchNormalization, RepeatVector, TimeDistributed, Masking, Bidirectional*:
  - LSTM: Long Short-Term Memory (RNN) layer used for sequence modeling.
  - BatchNormalization: Normalization layer to improve training speed and stability.
  - RepeatVector: Layer to repeat input sequence for decoder in sequence-to-sequence models.
  - TimeDistributed: Applies a layer to each time step in a sequence.
  - Masking: Helps in ignoring padding values during training.
  - Bidirectional: Wrapper for making RNN layers bidirectional.
- *from keras.callbacks import EarlyStopping, ModelCheckpoint, LambdaCallback*
  - EarlyStopping: Stops training early if the model performance doesn’t improve.
  - ModelCheckpoint: Saves model weights during training at checkpoints.
  - LambdaCallback: Allows custom callback functions at certain training points (e.g., after every epoch).

4. Loss function selection and hyperparamers

- *activation = "selu" :* Scaled Exponential Linear Unit (SELU)
- *loss = "huber" :* huber loss
- **The Huber loss** is less sensitive to outliers than MSE and combines both squared loss and absolute loss. It is defined as:

- $$Huber\ Loss(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & for\ |y - \hat{y}| \leq \delta \\ \delta|(y - \hat{y})| - \frac{1}{2}\delta^2, & for\ |y - \hat{y}| > \delta \end{cases}$$

where,

δ is threshold (usually a small positive constant like 1

y is the true value.

$\hat{y}$ is the predicted value.

Layer (type)	Output Shape	Param #
bidirectional_14 (Bidirectional)	(None, 128)	34816
dense_52 (Dense)	(None, 256)	33024
dense_53 (Dense)	(None, 128)	32896
dense_54 (Dense)	(None, 64)	8256
dense_55 (Dense)	(None, 1)	65

Total params: 109,057
Trainable params: 109,057
Non-trainable params: 0

4. Training

Training hyperparameters

- adam optimizer learning_rate=0.00001
- epochs: 120
- batch size = 32
- verbose = 1
- validation split = 0.2
- callbacks = model checkpoints : save best only, early stopping: patience 50

```
%%time
history = model.fit(train_x, train_y,
                    epochs = 120,
                    batch_size = 32,
                    verbose = 1,
                    validation_split = 0.2,
                     callbacks = [mc, es]
                    )
```

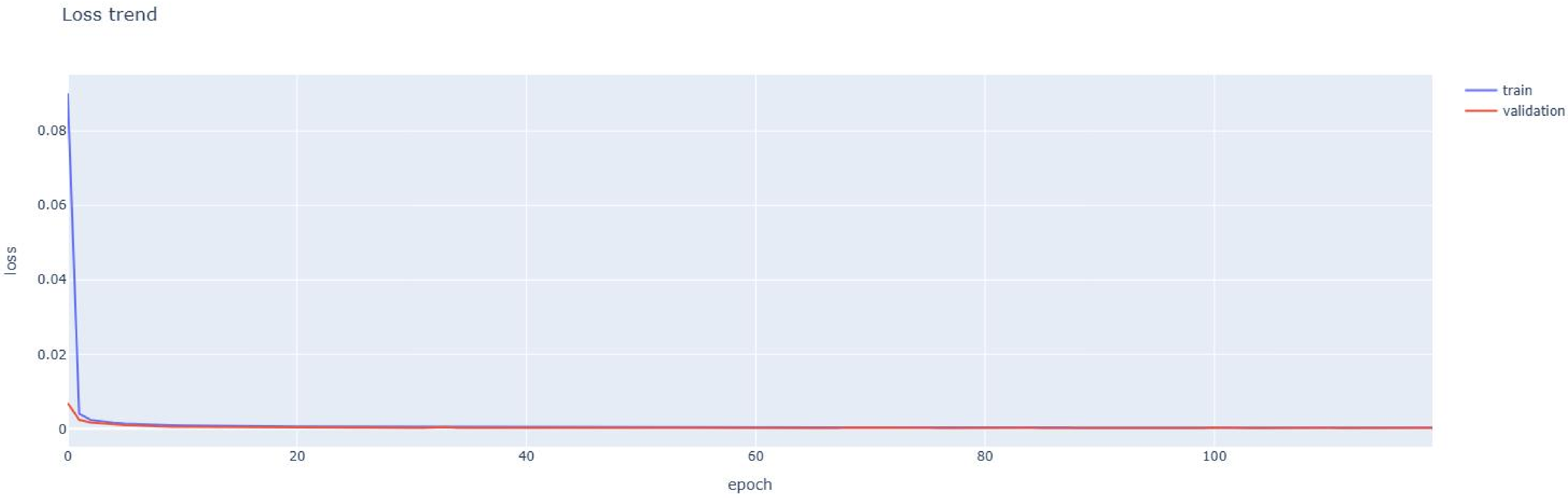
A glimpse of train on local machine with RTX2060 laptop gpu:

```
Epoch 1/120
161/161 [==============================] - 247s 2s/step - loss: 0.0149 - mse: 0.0299 - mae: 0.1246 - mape: 69.5861 - rmse: 0.1728 - val_loss: 0.0031 - val_mse: 0.
0062 - val_mae: 0.0636 - val_mape: 35.5507 - val_rmse: 0.0786
Epoch 2/120
161/161 [==============================] - 242s 2s/step - loss: 0.0025 - mse: 0.0050 - mae: 0.0533 - mape: 34.9492 - rmse: 0.0711 - val_loss: 0.0012 - val_mse: 0.
0024 - val_mae: 0.0363 - val_mape: 27.8892 - val_rmse: 0.0487
Epoch 3/120
161/161 [==============================] - 243s 2s/step - loss: 0.0017 - mse: 0.0033 - mae: 0.0423 - mape: 30.5297 - rmse: 0.0575 - val_loss: 9.1659e-04 - val_ms
e: 0.0018 - val_mae: 0.0305 - val_mape: 27.2327 - val_rmse: 0.0428
Epoch 4/120
 89/161 [===============>..............] - ETA: 1:48 - loss: 0.0014 - mse: 0.0028 - mae: 0.0387 - mape: 32.5272 - rmse: 0.0528
```

Model Evaluation

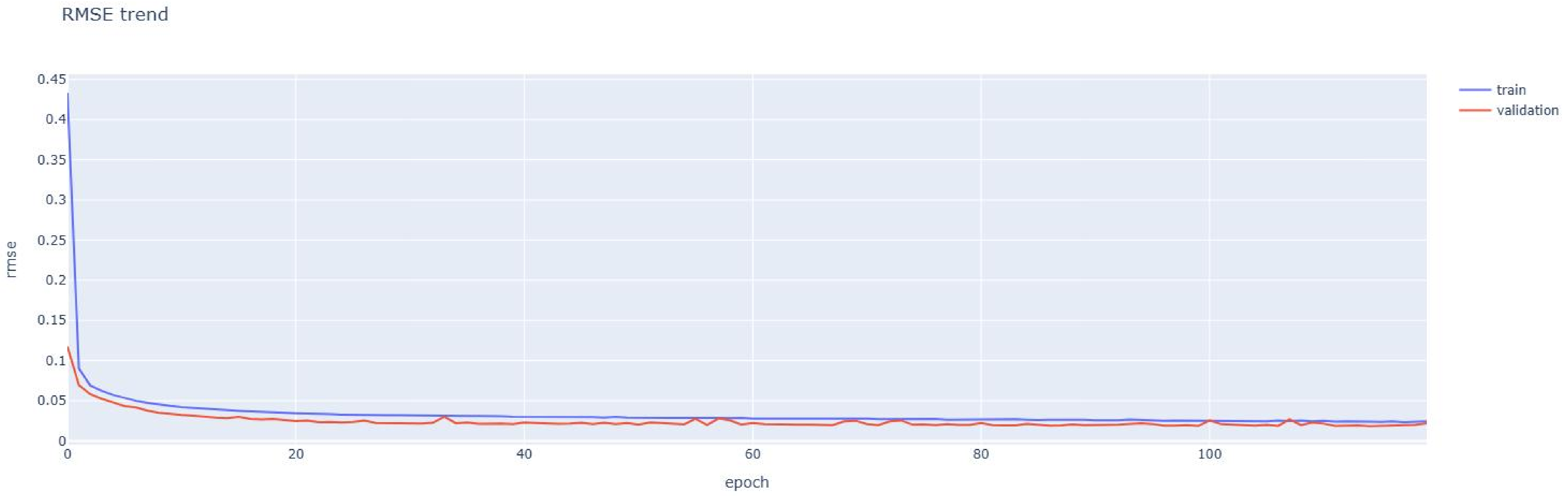
5. Loss trend

- Adam optimizer with learning rate of **1e-5**



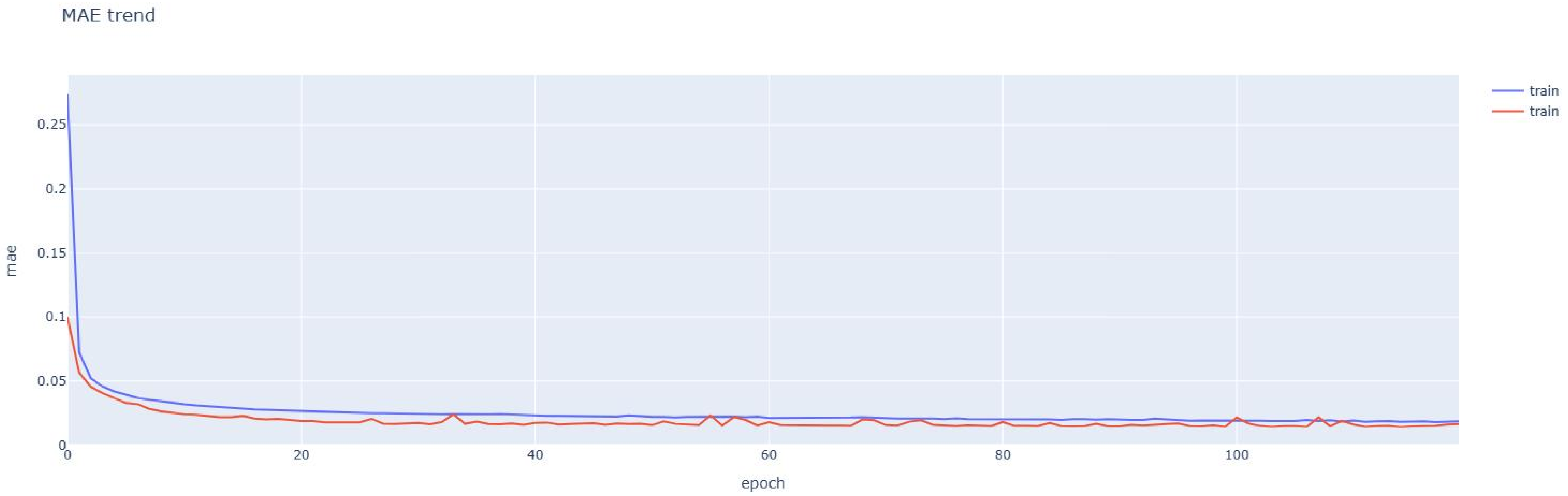
5. RMSE trend

- Root mean Square error (RMSE): **0.0213** on traning data and **0.0368** on testing data



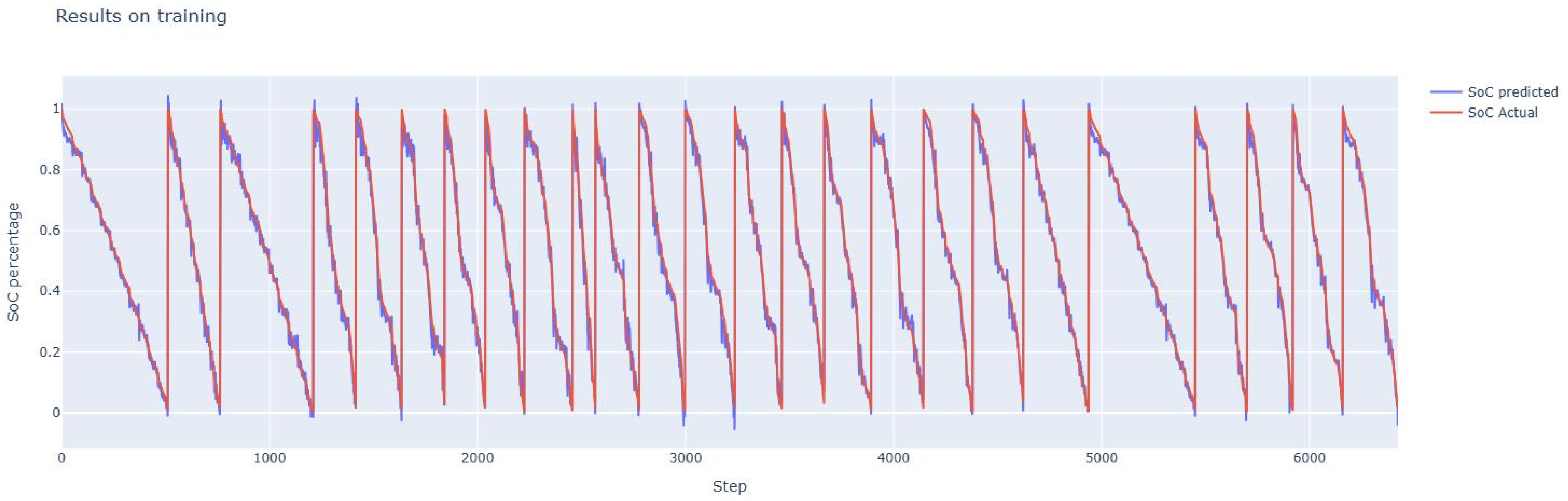
5. MAE trend

- Mean average error (MAE): **0.0176** on training data, **0.0281** on testing data.



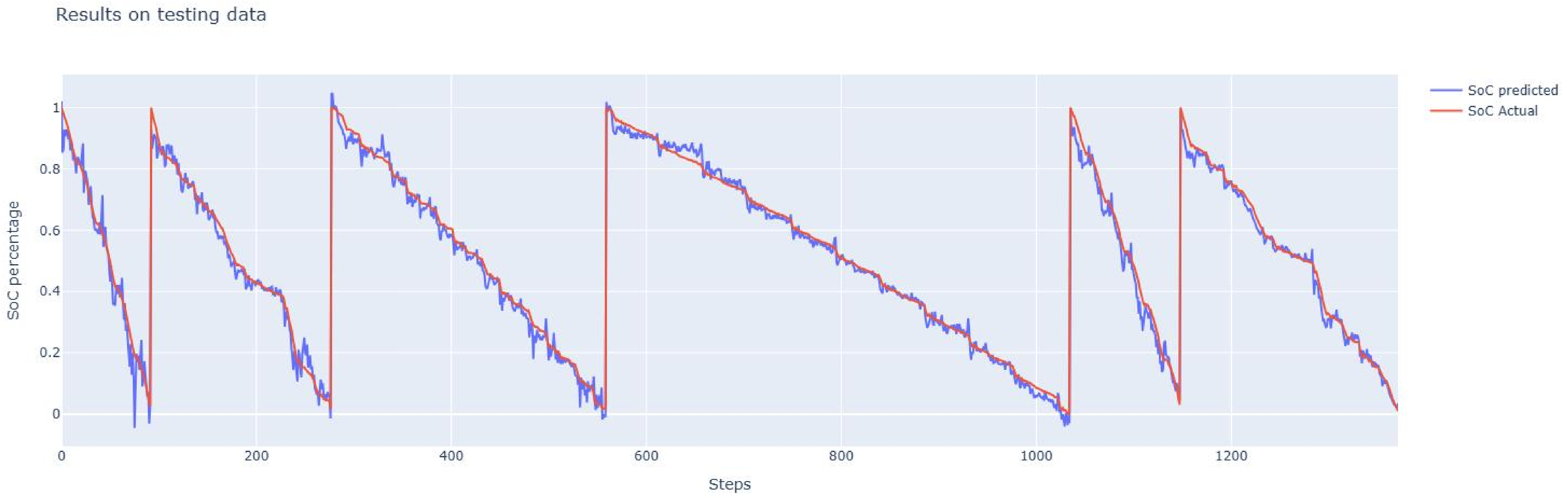
5. Results on training

- State of Charge(%) **predicted on training data** after training is completed Vs Actual State of charge



5. Result on testing data

- State of Charge(%) **predicted on testing data** after training is completed Vs Actual State of charge



6. Conclusion

- Bidirectional LSTM was trained on LG_HG2_Original_Dataset_McMasterUniversity_Jan_2020 dataset to estimate the Li ion battery state of charge.
- The trained model showed rmse of **0.0213** on traning data and **0.0368** on testing data

6. References

1. Nirmal Krishnas. (2020). [Battery SOC Estimation using Bi-LSTM](https://www.kaggle.com/code/nirmalkrishnas/battery-soc-estimation-bilstm) on Kaggle.

2. Vidal, C., Kollmeyer, P., Naguib, M., Malysz, P., Gross, O., & Emadi, A. (2020). Robust xEV Battery State-of-Charge Estimator Design using Deep Neural Networks. [Sae.org](https://www.sae.org/publications/technical-papers/content/2020-01-1181/). Accessed January 28, 2020.