01 – EIS & SOC Exploration

Chouaib (github.com/ChouaibB)

December 19, 2025

Contents

1. Data source: SoC EIS LFP dataset

For this notebook I use the public dataset:

> Mustafa, Hamza; Bourelly, Carmine; Vitelli, Michele; Milano, Fillippo; Molinara, Mario; Ferrigno, Luigi (2024),
> **"SoC Estimation on Li-ion Batteries: A New EIS-based Dataset for data-driven applications"**,
> Mendeley Data, V2, doi: 10.17632/cb887gkmxw.2.
> Available at: https://data.mendeley.com/datasets/cb887gkmxw/2
> License: **CC BY 4.0**

Dataset highlights:

- 11 cylindrical **LFP** cells (B01–B11).
- EIS at 20 SoC levels: 100%, 95%, 90%, 85%, 80%, 75%, 70%, 65%, 60%, 55%, 50%, 45%, 40%, 35%, 30%, 25%, 20%, 15%, 10%, 5%.
- Two discharge cycles per cell, same frequency grid from 0.01 Hz to 1000 Hz (28 frequencies).

Folder structure (after downloading the dataset into `../data/soc_eis_lfp/`):

- One folder per cell: `B01`, `B02`, …, `B11`.
- Inside each `Bxx` folder:
  - `EIS Measurement/` — subfolders with the actual EIS CSV files.
  - `Capacity Measurement/` — capacity test CSV (not used in this first notebook).

2. Load data & quick sanity checks.

```
Shape: (12320, 7)
Batteries: ['B01' 'B02' 'B03' 'B04' 'B05' 'B06' 'B07' 'B08' 'B09' 'B10' 'B11']
Cycles: [np.int64(1), np.int64(2)]
Unique SOCs: [np.int64(5), np.int64(10), np.int64(15), np.int64(20),
np.int64(25), np.int64(30), np.int64(35), np.int64(40), np.int64(45),
np.int64(50), np.int64(55), np.int64(60), np.int64(65), np.int64(70),
np.int64(75), np.int64(80), np.int64(85), np.int64(90), np.int64(95),
np.int64(100)]
Freq min/max: 0.01 → 1000.0
Temp min/max: 20 → 20

   frequency_hz  z_real_ohm  z_imag_ohm  temperature_c battery_id  cycle  soc
0          0.01    0.227844   -0.313990             20        B01      1  100
1          0.02    0.182858   -0.197273             20        B01      1  100
2          0.03    0.163270   -0.160068             20        B01      1  100
3          0.05    0.138709   -0.113159             20        B01      1  100
4          0.08    0.121685   -0.082724             20        B01      1  100

        frequency_hz  z_real_ohm  z_imag_ohm  temperature_c battery_id  \
count      12320.000   12320.000   12320.000        12320.0      12320
unique           NaN         NaN         NaN            NaN         11
top              NaN         NaN         NaN            NaN        B01
freq             NaN         NaN         NaN            NaN       1120
```

```
mean        117.360      0.078     -0.015          20.0        NaN
std         246.764      0.025      0.031           0.0        NaN
min           0.010      0.042     -1.329          20.0        NaN
25%           0.275      0.061     -0.015          20.0        NaN
50%           6.500      0.072     -0.007          20.0        NaN
75%          85.750      0.089     -0.006          20.0        NaN
max        1000.000      0.389     -0.001          20.0        NaN
```

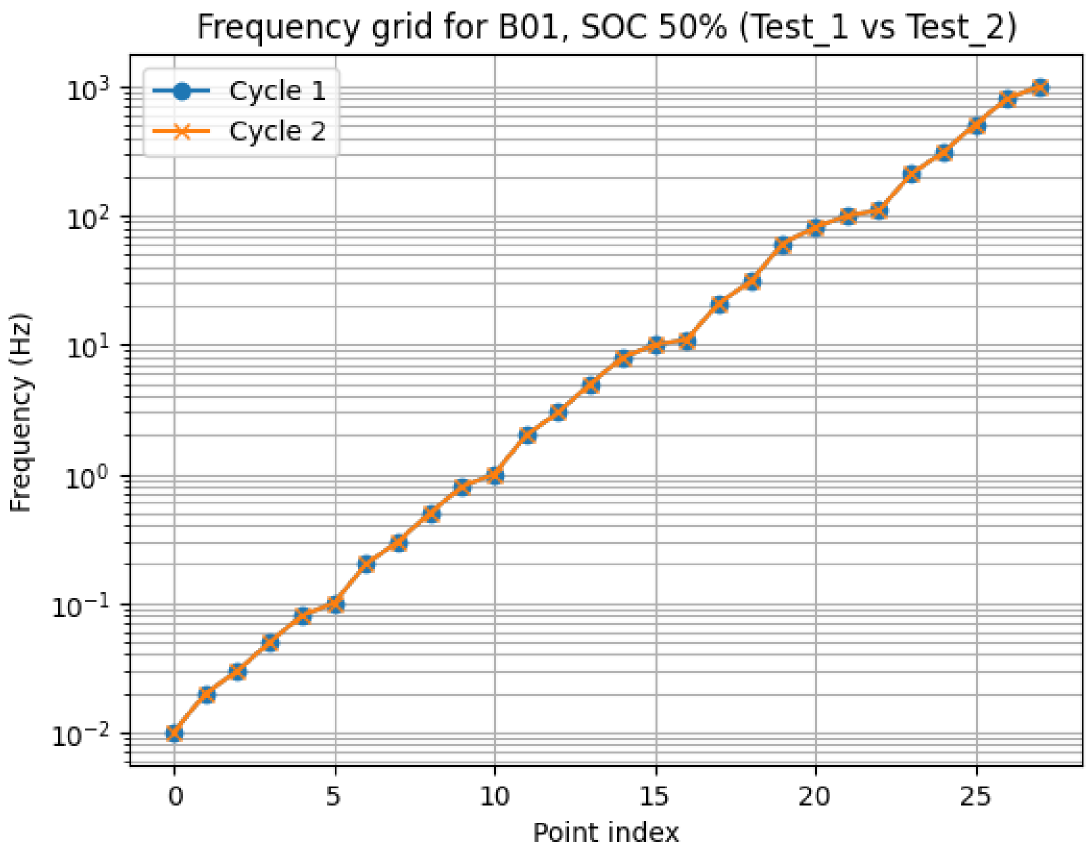
```
          cycle        soc
count   12320.0  12320.000
unique      NaN        NaN
top         NaN        NaN
freq        NaN        NaN
mean        1.5     52.500
std         0.5     28.833
min         1.0      5.000
25%         1.0     28.750
50%         1.5     52.500
75%         2.0     76.250
max         2.0    100.000
```

```
frequency_hz     0
z_real_ohm       0
z_imag_ohm       0
temperature_c    0
battery_id       0
cycle            0
soc              0
dtype: int64
```

```
np.int64(0)
```

```
Unique point counts per spectrum: [28]
```

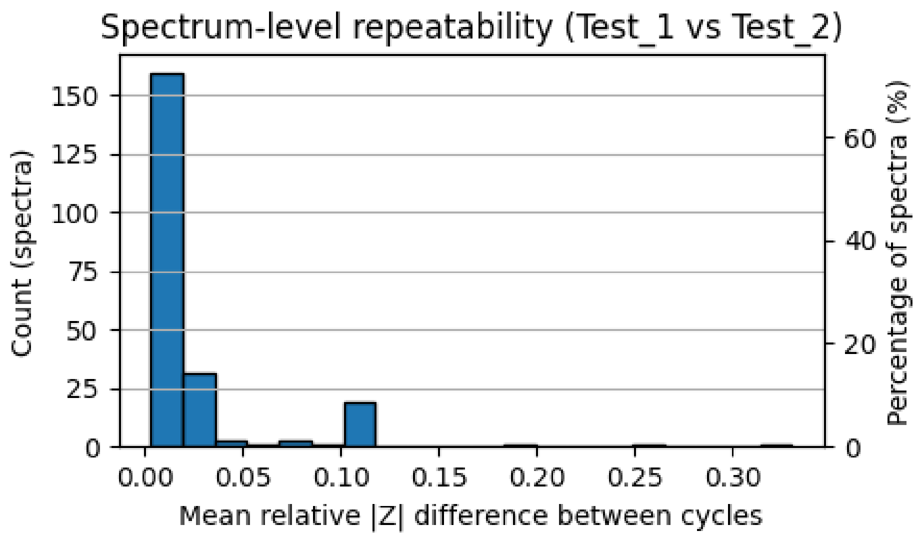
```
count    440.0
mean      28.0
std        0.0
min       28.0
25%       28.0
50%       28.0
75%       28.0
max       28.0
Name: frequency_hz, dtype: float64
```



2. Cycle QC and averaging

Each cell and SOC has two EIS measurements (Test_1 and Test_2).
Here I first check how consistent these repeats are, then average them to
get one "denoised" spectrum per (battery_id, SOC) for the rest of the notebook.

```
count    220.000
mean       0.026
std        0.040
min        0.003
25%        0.009
50%        0.011
75%        0.025
max        0.331
Name: rel_diff, dtype: float64
```



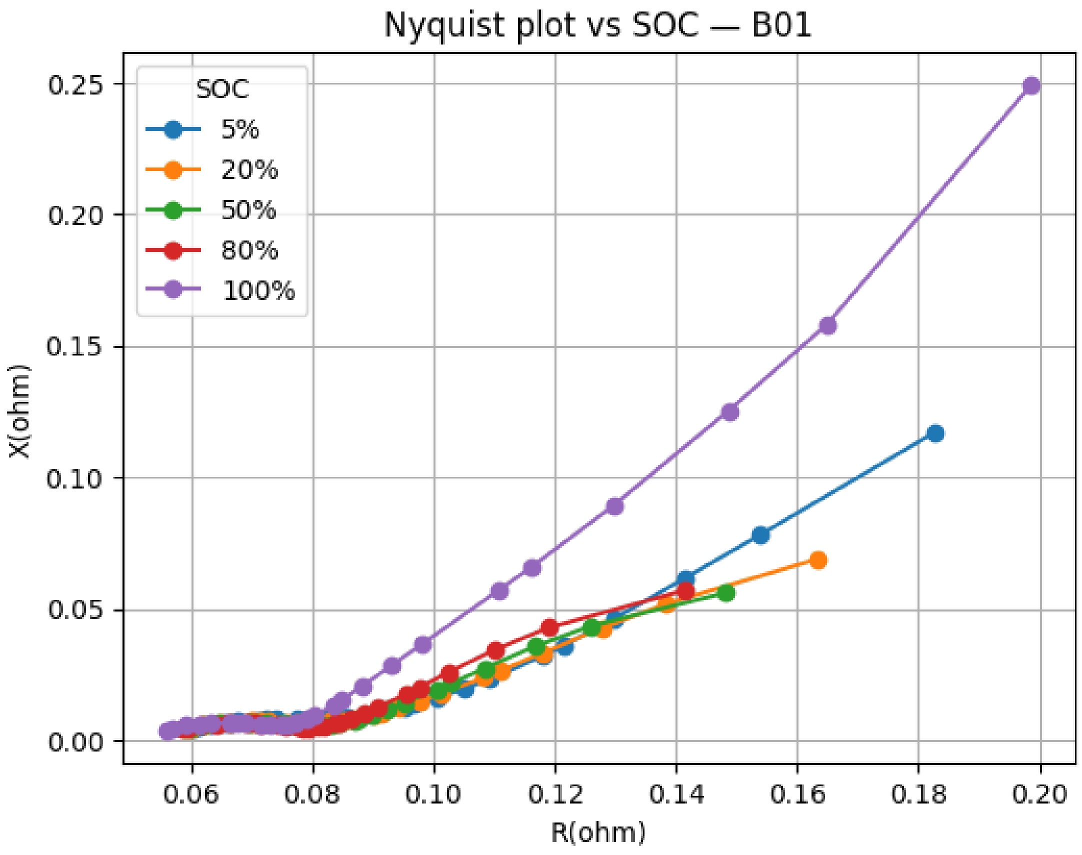
The relative |Z| difference between Test_1 and Test_2 is mostly very small (median   1.1%, 75%   2.5%), with a few larger outliers up to   33%.
In this notebook I keep all spectra and simply average the two cycles per (battery_id, SOC), which is reasonable given the overall good repeatability. In a stricter setting one could drop spectra where the mean relative difference between cycles exceeds, for example, 20%.

```
Averaged EIS table shape: (6160, 6)
```

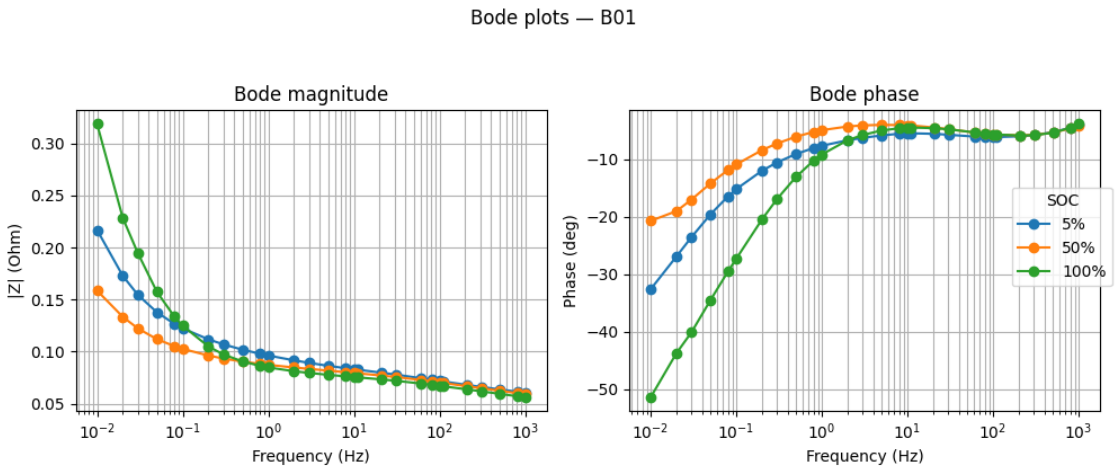
3. EIS visualisation vs SOC

3.1 Add magnitude and phase

3.2 Nyquist plots at different SOCs (one battery, one cycle)



3.3 Bode plots (magnitude & phase vs frequency)



4. Feature engineering: simple impedance fingerprints

Goal: compress each full EIS spectrum into a small feature vector, one row per **(battery_id, SOC)**, that we can later feed to a simple ML model.

In this section I will: - Create one feature row per (battery_id, soc). - Use basic summary features (min/max/mean) of |Z| and phase. - Sample |Z| and phase at a few fixed frequencies.

Why these engineered features?

The goal is to capture simple, physically meaningful aspects of the EIS curve without fitting a full equivalent circuit model:

- `zreal_lowfreq` / `zreal_highfreq`
  - Roughly "low-frequency / polarization resistance" vs "high-frequency / ohmic resistance".

  - These shift with SOC as the cell's effective resistance changes.
- `zmag_mean`, `zmag_max`
  - Overall "size" of the impedance arc and its peak.

  - Higher values usually mean higher resistance / lower conductivity.
- `phase_min`, `phase_max`
  - Range of capacitive / diffusive behaviour across frequency.

  - SOC changes how strongly capacitive or resistive the cell looks.
- `zmag_{f}Hz`, `phase_{f}Hz` at 0.01, 0.1, 1, 10, 100, 1000 Hz
  - These frequencies are chosen as simple, log-spaced points across the available 0.01–1000 Hz band.

  - Together they sample the low-, mid-, and high-frequency parts of the spectrum, giving a compact but informative "fingerprint" for SOC estimation.

4.1 Choose target frequencies from the measured grid I sample |Z| and phase at a few log-spaced frequencies that exist in the data, to capture the overall shape of the spectrum in a compact way.

```
Measured frequency grid:
 0: 0.01 Hz
 1: 0.02 Hz
 2: 0.03 Hz
 3: 0.05 Hz
 4: 0.08 Hz
 5: 0.1 Hz
 6: 0.2 Hz
 7: 0.3 Hz
 8: 0.5 Hz
 9: 0.8 Hz
10: 1 Hz
11: 2 Hz
```

```
12: 3 Hz
13: 5 Hz
14: 8 Hz
15: 10 Hz
16: 11 Hz
17: 21 Hz
18: 31 Hz
19: 61 Hz
20: 81 Hz
21: 100 Hz
22: 110 Hz
23: 210 Hz
24: 310 Hz
25: 510 Hz
26: 810 Hz
27: 1000 Hz

Target frequencies:
 0: 0.01 Hz
 1: 0.1 Hz
 2: 1 Hz
 3: 10 Hz
 4: 100 Hz
 5: 1000 Hz
```

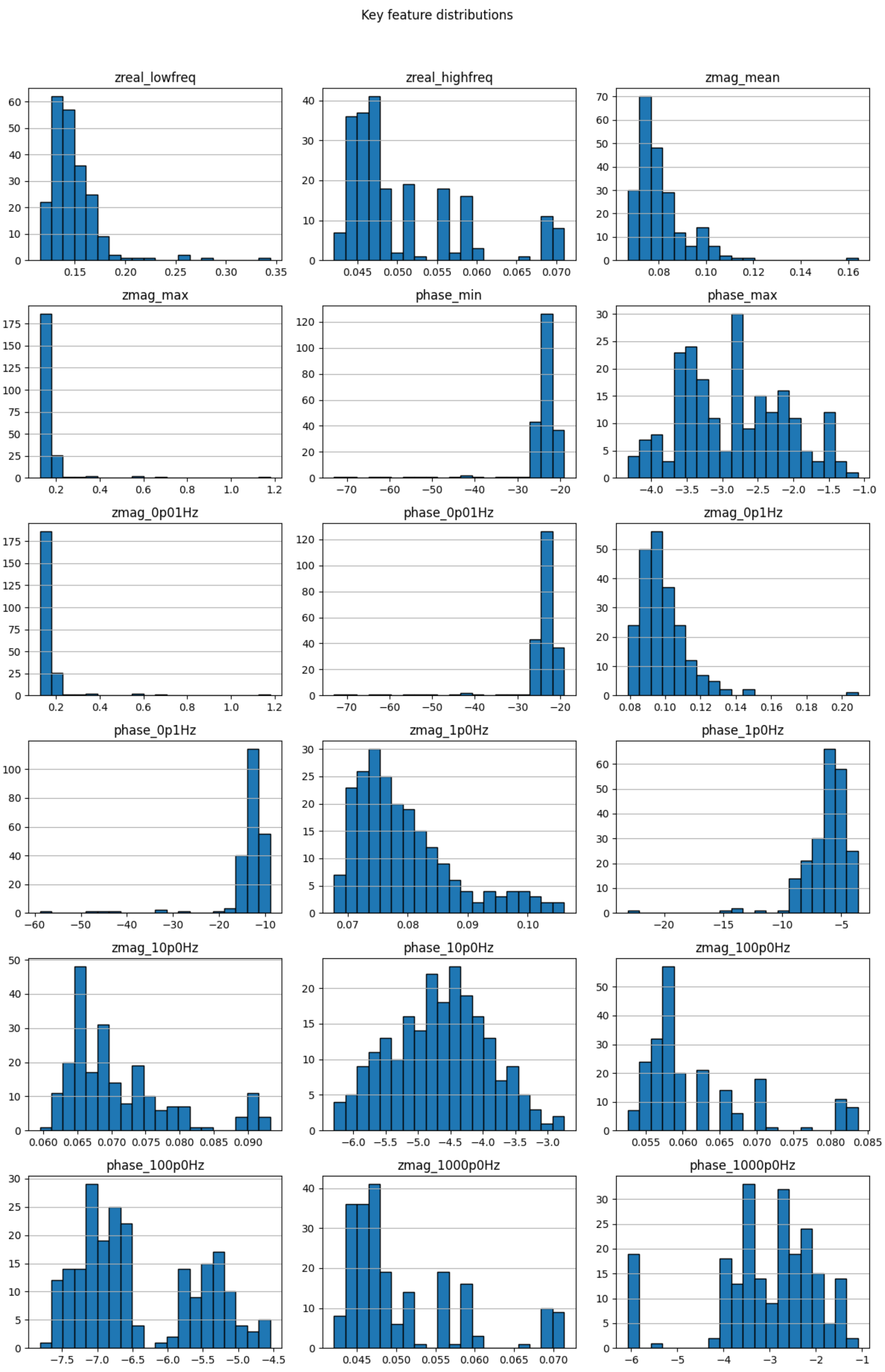
4.2 Build one feature vector per (battery_id, SOC) For each averaged EIS spectrum I create a single feature row.
I keep a few global summary statistics (extremes and averages), and I take |Z| and phase directly at a set of fixed frequencies (0.01, 0.1, 1, 10, 100, 1000 Hz).

```
  battery_id  soc  zreal_lowfreq  zreal_highfreq  zmag_mean  zmag_max  \
0        B01    5       0.182653        0.060269   0.097467  0.216863
1        B01   10       0.170783        0.059821   0.092930  0.187566
2        B01   15       0.166635        0.059541   0.091769  0.180553
3        B01   20       0.163275        0.059271   0.090735  0.177173
4        B01   25       0.160309        0.059097   0.089930  0.172943

   phase_min  phase_max  zmag_0p01Hz  phase_0p01Hz  zmag_0p1Hz  phase_0p1Hz  \
0 -32.621081  -4.004392     0.216863    -32.621081    0.122336   -15.181986
1 -24.422087  -3.960382     0.187566    -24.422087    0.115161   -13.161971
2 -22.644222  -3.972768     0.180553    -22.644222    0.113091   -12.632649
3 -22.845436  -3.991457     0.177173    -22.845436    0.110929   -12.400498
4 -22.036235  -4.016717     0.172943    -22.036235    0.109430   -12.100670

   zmag_1p0Hz  phase_1p0Hz  zmag_10p0Hz  phase_10p0Hz  zmag_100p0Hz  \
0    0.096208    -7.636961     0.083010     -5.468509      0.071766
1    0.092719    -6.811176     0.081175     -4.923678      0.071196
2    0.091900    -6.489224     0.080937     -4.856404      0.070999
```

```
3    0.090892    -6.178033     0.080575     -4.715034       0.070769
4    0.090236    -5.955735     0.080347     -4.614360       0.070625

   phase_100p0Hz  zmag_1000p0Hz  phase_1000p0Hz
0      -6.121537       0.060417       -4.004392
1      -5.845588       0.059965       -3.960382
2      -5.846668       0.059685       -3.972768
3      -5.840951       0.059415       -3.991457
4      -5.832845       0.059243       -4.016717
```

4.3 Visualise key feature distributions Before modelling, I check a few key feature distributions to see:

- that values look reasonable,
- and that there is some spread across samples (so the model has signal to learn from).



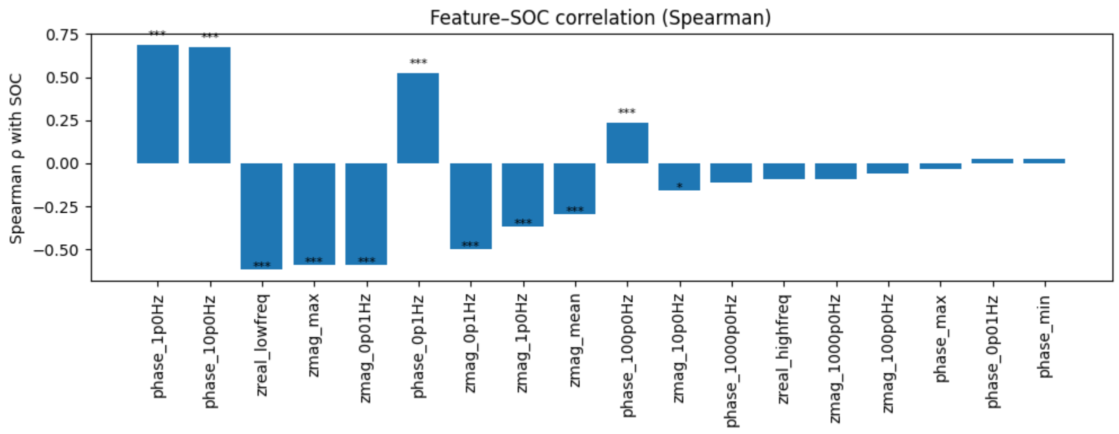
Key feature distributions
zreal_lowfreq
zreal_highfreq
zmag_mean
zmag_max
phase_min
phase_max
zmag_0p01Hz
phase_0p01Hz
zmag_0p1Hz
phase_0p1Hz
zmag_1p0Hz
phase_1p0Hz
zmag_10p0Hz
phase_10p0Hz
zmag_100p0Hz
phase_100p0Hz
zmag_1000p0Hz
phase_1000p0Hz

4.4 Correlation of engineered features with SOC To keep things readable, I only look at how each engineered feature correlates with **SOC** (rather than a full feature–feature heatmap).

I use **Spearman rank correlation** between SOC and all numeric features and mark statistically significant correlations:

- * : $p < 0.05$
- ** : $p < 0.01$
- ***: $p < 0.001$

This highlights which features are most informative for SOC and which ones are largely redundant.

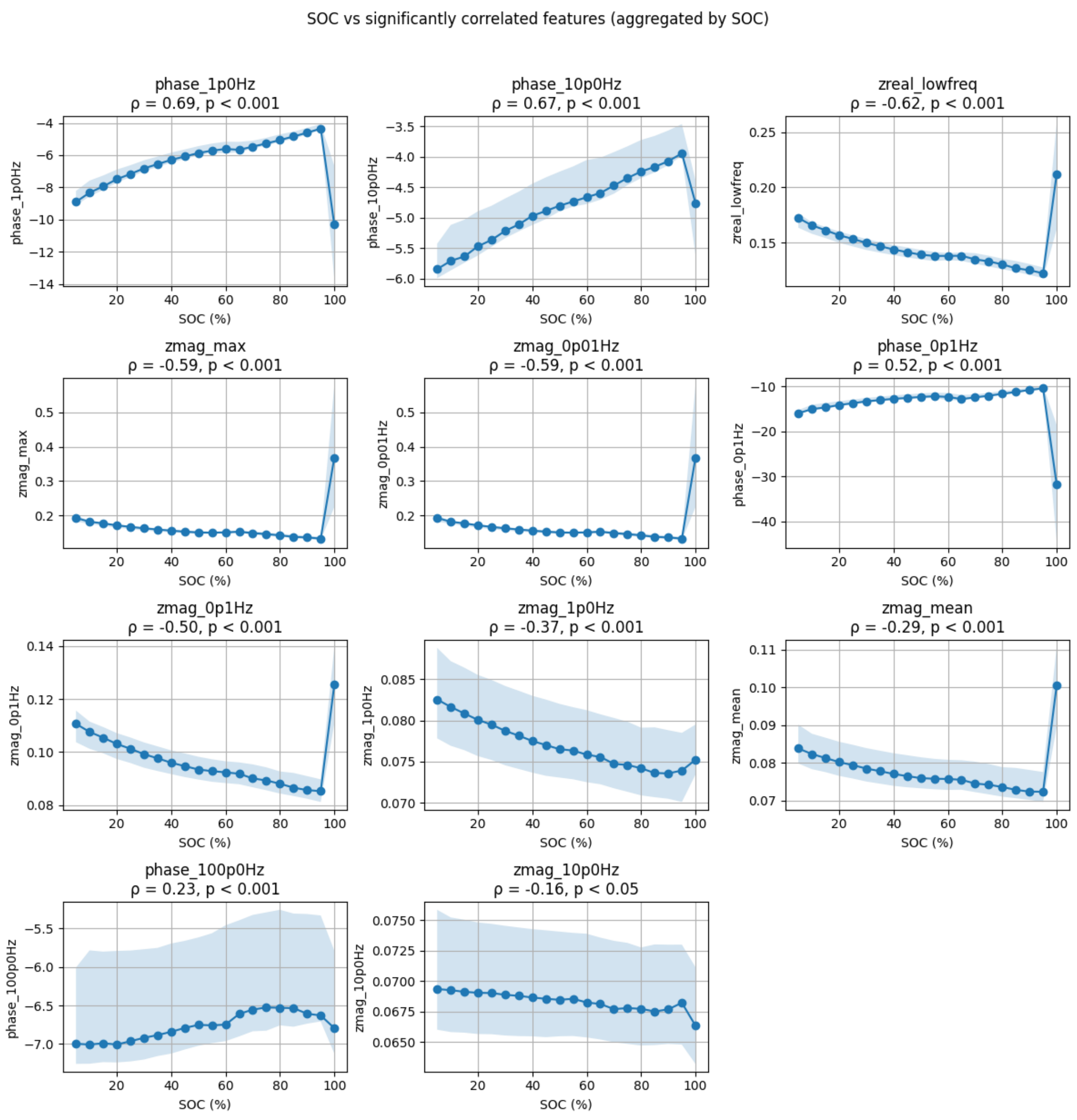


4.5 SOC vs significantly correlated features (aggregated by SOC) Scatter plots were noisy, so here I aggregate by SOC and plot, for each
significant feature ($p < 0.05$):

- median value vs SOC (line),
- 25–75% range vs SOC (shaded band).

This makes monotonic trends much easier to see.

```
Significant features (p < 0.05): ['phase_1p0Hz', 'phase_10p0Hz',
'zreal_lowfreq', 'zmag_max', 'zmag_0p01Hz', 'phase_0p1Hz', 'zmag_0p1Hz',
'zmag_1p0Hz', 'zmag_mean', 'phase_100p0Hz', 'zmag_10p0Hz']
```



5. SOC prediction target & problem setup

5.1 Define targets Main target (regression): - `soc` in %, e.g. 5–100.

Optional side target (bins): - Coarse SOC bands (0–20, 20–40, 40–60, 60–80, 80–100%) for an easier "are we in the right band?" view.

```
soc_bin
0     44
1     44
2     44
3     44
```

```
4    44
Name: count, dtype: int64
```

5.2 Feature subsets: all vs significant I compare two feature sets:

- **all**: all engineered numeric features except identifiers and targets
- **sig**: only features with significant Spearman correlation to SOC ($p < 0.05$)

Both will be evaluated inside the same nested GroupKFold CV.

```
All features: 18
Significant features: 11
Significant feature names: ['zreal_lowfreq', 'zmag_mean', 'zmag_max',
'zmag_0p01Hz', 'zmag_0p1Hz', 'phase_0p1Hz', 'zmag_1p0Hz', 'phase_1p0Hz',
'zmag_10p0Hz', 'phase_10p0Hz', 'phase_100p0Hz']
```

5.3 Models and nested GroupKFold setup I compare two models:

- **RandomForestRegressor** (non-linear, tree ensemble)
- **ElasticNet** (linear model with L1/L2 regularisation in a scaled pipeline)

Nested grouped CV:

- **Outer**: GroupKFold(n_splits=5) on `battery_id`
  - ~80% of cells for training, ~20% for testing in each fold.
- **Inner**: GroupKFold(n_splits=3) on the outer-train cells
  - used only for hyperparameter tuning (MAE as objective).

5.4 Nested GroupKFold CV: run and collect metrics For each outer fold (5 folds):

- Use only the outer-train cells in a 3-fold GroupKFold inner CV to tune hyperparameters for each (feature set, model) combination.
- Refit the best model on all outer-train data.
- Evaluate on the outer-test cells:
  - MAE (mean absolute error) in SOC %
  - RMSE (root mean squared error) in SOC %
  - SOC-bin accuracy (using the coarse bins defined earlier)

I also record how many samples and batteries are in train/test per fold.

```
Nested CV:   0%|          | 0/20 [00:00<?, ?it/s]

Fold 0, fs=all, model=rf | MAE=3.27, RMSE=4.88, bin_acc=81.67%,
best={'max_depth': None, 'n_estimators': 300}
Fold 0, fs=all, model=enet | MAE=4.02, RMSE=6.90, bin_acc=81.67%,
best={'model__alpha': 0.1, 'model__l1_ratio': 1.0}
Fold 0, fs=sig, model=rf | MAE=3.86, RMSE=5.11, bin_acc=83.33%,
best={'max_depth': None, 'n_estimators': 300}
Fold 0, fs=sig, model=enet | MAE=4.79, RMSE=8.13, bin_acc=80.00%,
best={'model__alpha': 0.1, 'model__l1_ratio': 1.0}
```

```
Fold 1, fs=all, model=rf | MAE=3.75, RMSE=5.39, bin_acc=90.00%,
best={'max_depth': None, 'n_estimators': 150}
Fold 1, fs=all, model=enet | MAE=4.05, RMSE=5.98, bin_acc=87.50%,
best={'model__alpha': 0.1, 'model__l1_ratio': 1.0}
Fold 1, fs=sig, model=rf | MAE=4.41, RMSE=6.56, bin_acc=85.00%,
best={'max_depth': None, 'n_estimators': 150}
Fold 1, fs=sig, model=enet | MAE=4.11, RMSE=5.43, bin_acc=85.00%,
best={'model__alpha': 0.1, 'model__l1_ratio': 1.0}
Fold 2, fs=all, model=rf | MAE=4.45, RMSE=7.33, bin_acc=72.50%,
best={'max_depth': None, 'n_estimators': 300}
Fold 2, fs=all, model=enet | MAE=2.89, RMSE=3.74, bin_acc=85.00%,
best={'model__alpha': 0.1, 'model__l1_ratio': 1.0}
Fold 2, fs=sig, model=rf | MAE=4.53, RMSE=10.12, bin_acc=70.00%,
best={'max_depth': None, 'n_estimators': 300}
Fold 2, fs=sig, model=enet | MAE=2.83, RMSE=3.96, bin_acc=77.50%,
best={'model__alpha': 0.1, 'model__l1_ratio': 1.0}
Fold 3, fs=all, model=rf | MAE=10.59, RMSE=14.73, bin_acc=55.00%,
best={'max_depth': None, 'n_estimators': 300}
Fold 3, fs=all, model=enet | MAE=2.62, RMSE=3.32, bin_acc=80.00%,
best={'model__alpha': 0.1, 'model__l1_ratio': 1.0}
Fold 3, fs=sig, model=rf | MAE=12.89, RMSE=18.38, bin_acc=52.50%,
best={'max_depth': None, 'n_estimators': 300}
Fold 3, fs=sig, model=enet | MAE=3.66, RMSE=4.71, bin_acc=82.50%,
best={'model__alpha': 0.1, 'model__l1_ratio': 1.0}
Fold 4, fs=all, model=rf | MAE=7.56, RMSE=11.39, bin_acc=60.00%,
best={'max_depth': None, 'n_estimators': 150}
Fold 4, fs=all, model=enet | MAE=5.89, RMSE=8.62, bin_acc=67.50%,
best={'model__alpha': 0.1, 'model__l1_ratio': 1.0}
Fold 4, fs=sig, model=rf | MAE=7.69, RMSE=11.97, bin_acc=60.00%,
best={'max_depth': None, 'n_estimators': 300}
Fold 4, fs=sig, model=enet | MAE=7.80, RMSE=16.60, bin_acc=65.00%,
best={'model__alpha': 0.1, 'model__l1_ratio': 1.0}

   fold  n_train_samples  n_test_samples  n_train_batteries  n_test_batteries  \
0     0              160              60                  8                 3
1     1              180              40                  9                 2
2     2              180              40                  9                 2
3     3              180              40                  9                 2
4     4              180              40                  9                 2

                                train_batteries test_batteries
0       B02, B03, B04, B05, B07, B08, B09, B10  B01, B06, B11
1  B01, B02, B03, B04, B06, B07, B08, B09, B11       B05, B10
2  B01, B02, B03, B05, B06, B07, B08, B10, B11       B04, B09
3  B01, B02, B04, B05, B06, B07, B09, B10, B11       B03, B08
4  B01, B03, B04, B05, B06, B08, B09, B10, B11       B02, B07
```

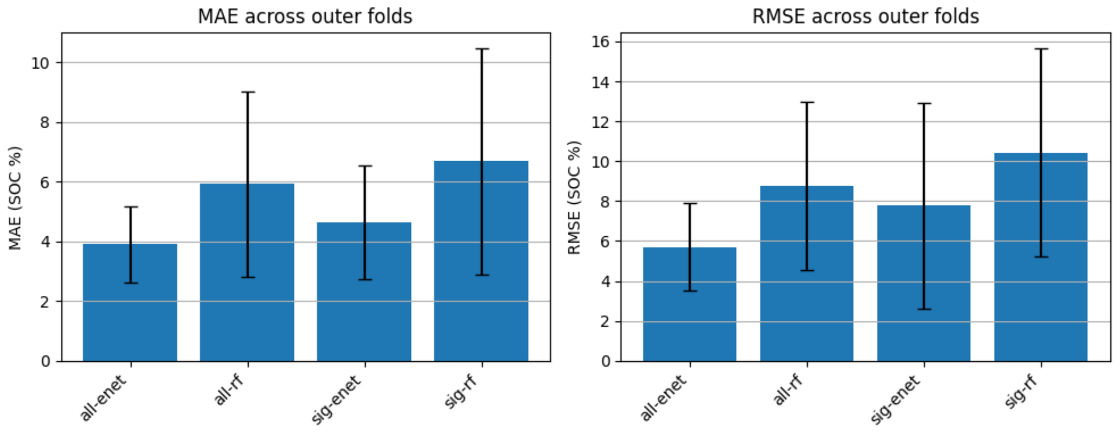
5.5 Compact summary of performance I summarise performance across the 5 outer folds by feature set and model:

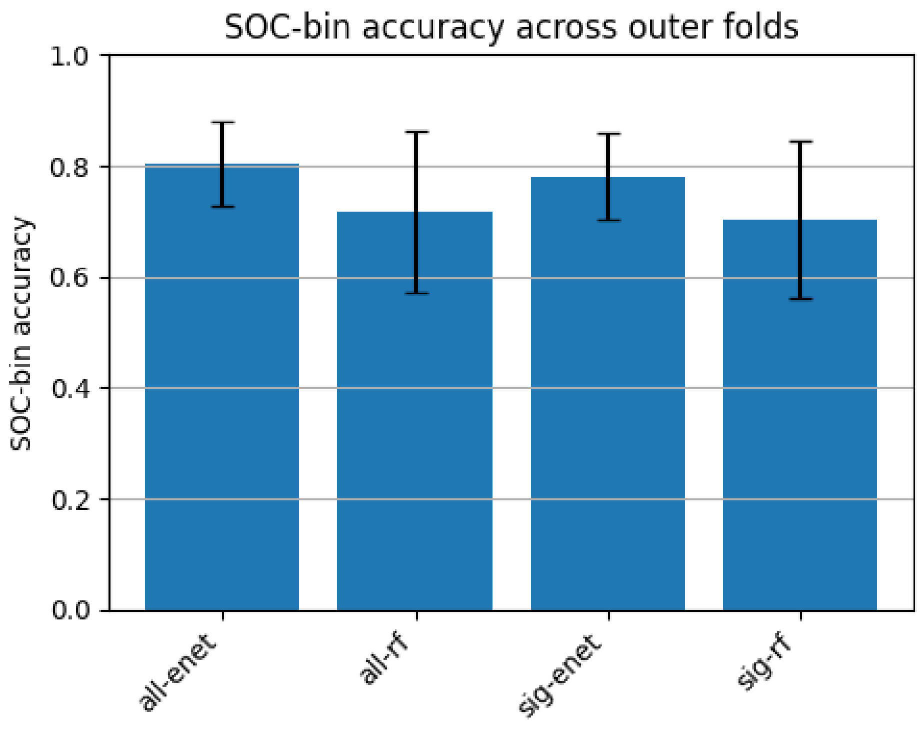
- Mean ± std of MAE and RMSE in SOC %
- Mean SOC-bin accuracy

This lets me compare:

- RandomForest vs ElasticNet (non-linear vs linear)
- All features vs only the statistically significant ones

```
Summary metrics across outer folds:

  feature_set model  mae_mean   mae_std  rmse_mean  rmse_std  bin_acc_mean  \
0         all  enet  3.897597  1.289881   5.711748  2.211639      0.803333
1         all    rf  5.923722  3.097528   8.742880  4.215280      0.718333
2         sig  enet  4.637253  1.906821   7.763793  5.182472      0.780000
3         sig    rf  6.676111  3.784897  10.429521  5.219661      0.701667

   bin_acc_std
0     0.077415
1     0.145821
2     0.077862
3     0.142205
```





6. Summary of SOC modelling experiment

This section briefly sums up the SOC prediction experiment based on the engineered EIS features and the nested, grouped cross-validation.

6.1 Aggregated metrics table First I summarise the nested GroupKFold results across outer folds for each (feature set, model) combination.

```
<pandas.io.formats.style.Styler at 0x7265add66e50>
```

6.2 Setup (reminder)

- **Input:** engineered EIS features per `(battery_id, SOC)`, including:
  - global stats: `zreal_lowfreq`, `zreal_highfreq`, `zmag_mean`, `zmag_max`, `phase_min`, `phase_max`
  - `|Z|` and phase at 0.01, 0.1, 1, 10, 100, 1000 Hz
- **Target:** SOC in %, evaluated as:
  - **regression**: MAE / RMSE in SOC %
  - **coarse SOC bins**: [0–20, 20–40, 40–60, 60–80, 80–100]% for bin accuracy
- **Splits:** nested, grouped by `battery_id`
  - **Outer:** `GroupKFold(n_splits=5)`
    → ~80% of cells for training, ~20% for testing in each fold

  - **Inner:** `GroupKFold(n_splits=3)` on outer-train cells
    $\rightarrow$ used only for hyperparameter tuning (MAE as objective)
- **Feature sets:**
  - **all** – all engineered features (except identifiers and targets)
  - **sig** – only features with Spearman $p < 0.05$ vs SOC
- **Models:**
  - **rf** – `RandomForestRegressor`
  - **enet** – `ElasticNet` (in a scaled pipeline, tuned `alpha` and `l1_ratio`)

The `summary` table aggregates metrics across the 5 outer folds for each (feature set, model) combination.

6.3 Quantitative results (mean ± std across outer folds) From the aggregated results (`summary`):

- **all + enet** (ElasticNet on all features):
  - MAE **3.9 ± 1.3** % SOC
  - RMSE **5.7 ± 2.2** % SOC
  - SOC-bin accuracy **0.80 ± 0.08**
- **all + rf** (RandomForest on all features):
  - MAE 5.9 ± 3.1 % SOC
  - RMSE 8.7 ± 4.2 % SOC
  - SOC-bin accuracy 0.72 ± 0.15
- **sig + enet** (ElasticNet on significant features only):
  - MAE 4.6 ± 1.9 % SOC
  - RMSE 7.8 ± 5.2 % SOC
  - SOC-bin accuracy 0.78 ± 0.08
- **sig + rf** (RandomForest on significant features only):
  - MAE 6.7 ± 3.8 % SOC
  - RMSE 10.4 ± 5.2 % SOC
  - SOC-bin accuracy 0.70 ± 0.14

6.4 Main observations

1. **Best overall configuration**
   - The best-performing configuration is **ElasticNet with all features** (`feature_set = "all"`, `model = "enet"`):
     - MAE **3.9 % SOC**
     - RMSE **5.7 % SOC**
     - SOC-bin accuracy **80%** (correct 20%-wide SOC band)

    - This is the natural “baseline SOC model” to highlight in this notebook.

2. **Linear vs non-linear model**

    - On both feature sets, **ElasticNet beats RandomForest**:
        - With all features:
            - MAE improves from ~5.9 → ~3.9 % SOC
            - RMSE improves from ~8.7 → ~5.7 % SOC
            - SOC-bin accuracy improves from ~72% → ~80%
    - Interpretation:
        - The EIS features are smooth and highly correlated.
        - A regularised **linear model** already captures most of the SOC signal; the non-linear RandomForest does not add value here and may overfit given only 11 cells.

3. **All features vs “significant-only” features**

    - For ElasticNet:
        - Restricting to only the “significant” features (Spearman $p < 0.05$) **slightly worsens** MAE, RMSE and bin accuracy versus using all features.
    - For RandomForest:
        - The same pattern holds: using only the significant subset is not better.
    - Interpretation:
        - Dropping features based purely on per-feature correlation with SOC does not help in this case.
        - Given strong multicollinearity and the use of regularisation, it is actually safe (and slightly better) to keep the full engineered feature set.

4. **Variability across folds**

    - For the best config (all + enet), variability across outer folds is moderate:
        - MAE std **1.3 % SOC**
        - RMSE std **2.2 % SOC**
        - SOC-bin accuracy std **0.08**
    - This reflects the small number of cells (11) and the fact that each fold holds out a different subset of batteries.
    - Despite this, the ranking of configurations is consistent: **all+enet** best, then **sig+enet**, both RandomForest variants clearly worse.

6.5 Takeaways Overall, the experiment shows that:

- With a compact set of physically motivated EIS features, a **regularised linear model (ElasticNet)** can estimate SOC with:

    - typical errors around **4% SOC** (MAE),
    - and ~**80% accuracy** in coarse 20%-wide SOC bands,
    - when evaluated with 5-fold `GroupKFold` that holds out entire cells.

- A more complex **RandomForest** model does not improve performance on this dataset and is consistently worse across all metrics.

- Restricting to only “statistically significant” features (Spearman $p < 0.05$) slightly hurts performance; the full engineered feature set works best, likely because:

  - redundant but informative features are handled well by ElasticNet's regularisation,
  - and single-feature correlation is a crude way to do feature selection on smooth spectral data.

Given the small number of cells (11), these results are a **proof-of-concept** rather than a final production model, but they already demonstrate that:

> **Impedance fingerprints contain enough information to recover SOC reasonably well, and simple, interpretable models can leverage that information effectively.**