Predicting State of Health in Li-Ion Batteries

Muhammad Usman - 12036782

COS7045-B Advanced Machine Learning

Submitted to Dr. Paul Trundle

May 12, 2020

Abstract

Predicting State of Health (SoH) in Lithium-Ion (Li-Ion) batteries has been a widely explored research problem because of its huge implications. Various machine learning techniques and data mining approaches have been used for this purpose. In this work, a real dataset, containing 50 features and 50,00,000 records, provided by an automobile company is analyzed to get meaningful details out of it. With the aim to predict SoH from the dataset we have, the dataset is statistically analyzed first before implementing two different approaches using Decision Tree and Logistic Regression. Prediction results from both approaches revealed that Decision Tree produced better prediction results due to its ability to handle continuous and categorical features which our dataset had. The work can be extended to predict and analyze other important features as well, for instance 'imbalance'.

1 Introduction

The rapidly evolving technologies around the world are progressing to make the human lives easier by proposing solutions to the previously impossible problems or introducing ways to introduce efficiency in the already developed systems. But, on the other hand, these innovative systems bring new challenges as well which were not considered as an 'issue' in the past. On such example is the transport industry which can undoubtedly be considered as one of the highly advanced industries, drastically enhanced with the help of information technology. Introduction of electric vehicle was a revolutionary development to save the planet by reducing the ever-increasing burden from petroleum fuels. This new development brought new challenges regarding batteries to be used in future vehicles and to find the factors which can enhanced battery lives. One of such problem is correct prediction of battery's health state so that it can be fixed at the best time maintaining a smooth customer experience. Different solutions have been proposed for this in the past and one such effort is also done in this work to predict state of health from the dataset of a company.

1.1 Data Set Overview

In this work, the dataset of an automobile company is analysed containing information about Lithium-Ion (Li-Ion) batteries in their electric cars. The data is composed of 50 columns and 50,00,000 records of instances. Each instance shows details about the periodic state of battery, composed of different features, being sent from different sensors. The goal is to extract all those features which can impact the battery life. Some of the features which can potentially be of importance are described in following Table 1:

Table 1: Data Features Explanation

Feature Name	**Description**
Actual_time	Signal Capturing Time and Date
Powermode	Describes the power mode at the time when signal was sent
Odometer	Vehicle odometer value
Ambient_temp	Ambient temperature
Vehicle_speed	Speed of vehicle
Balancing_status	Battery cell balancing status
Min_voltage	Minimum voltage of cell
Imbalance_percent	Percentage of difference between best and worst supercell
soh	State of health
Fast_charge_count	Frequency of battery being charged
Num_cycle	Cycle count of vehicle

1.2 Problem Statement

The purpose of this report is to answer the question, 'Can we successfully predict state of health (soh) of the Li-Ion batteries from the data features we have and which machine learning techniques can be used for that?' In particular, the goal would be to identify all those features which influence state of health in lithium-ion batteries. For machine learning perspective the aim would be to find those tools and technologies which can help us to estimate a particular target in such datasets. This would help the engineers to focus on those key features while manufacturing reliable batteries. Separately, there is no consensus among the research community for defining 'State-of-health', but for this work state-of-health refers to one of the data features with higher values representing better health in the battery.

1.3 Literature Review

State of health in the Li-Ion batteries has been a popular research problem among the research community. Landi and Gross proposed two methodologies for correct estimation of state of health in Li-Ion batteries in smart grids as well as in vehicles. Following the data driven approaches in their work, they have proposed prediction systems based on Fuzzy Logic and Neural Networks (Landi & Gross (2014)). Both of their proposed methods showed promising results in prediction with less than 5% of errors in estimated and experimental values.

Lin et al. have also used Neural Networks for estimating state of health in Li-Ion batteries (Lin et al. (2012)). This work used the data, composed of 110 batteries, obtained from the recharging/discharging and battery's life-cycle to estimate the health of the battery. The probabilistic Neural Network (PNN) was trained on 100 batteries and validated against 10 when it produced results with prediction error of 0.28% and with standard deviation of 1.15%. The PNN was reported as 'fast' with training time of 62.5ms but low number of validation result could be a reason for such efficient results.

Apart from this, modeling the combination of capacity degradation and internal resistance growth has also been used to estimate remaining useful life in batteries (Guha & Patra (2017)). In this work, capacity degradation is decrease in battery's charging capacity due to ageing. In their work, curve fitting was used to initialize the mathematical model parameters. After that, using particle filtering algorithm based on Bayesian learning, the parameters of the degradation models were upgraded so that the degradation trend can be tracked. The range of errors in predicted and test state of health in the proposed work was 0.25 - 3.25 with more tuning producing less errors.

Tang et al. have followed a pure data mining approach by proposing a novel efficient algorithm which predicted state of health with less than 2.5% errors(Tang et al. (2018)). Their work makes used of incremental capacity of cells, and of the delta in peak voltage and lowest value of individual battery cells, this approach is influenced by the work proposed by Bhide and Shim (Bhide & Shim (2011)) which was a simulation based proposal of the same idea.

Moreover, use of Random Forest Regression has also produced high accuracy of soh prediction. The works of Li et al. and Tao et al. are examples of such use (Li et al. (2018)),(Tao & Lu (2017))). The former work fed the raw data to the Random forest (RF) whereas the other did some feature extraction before using this algorithm. The work in which the raw data was used yielded Root-Mean-Square-Error (RMSE) of 1.2% and extracting useful features before applying RF had produced even better results with average RMSE of 0.03.

Apart from this, mathematical regression model has also been used to estimate remaining useful health in batteries (Tseng et al. (2015)). The work used Particle Swarm optimization to find optimum weights for the models with discharged voltage and internal resistance being used as a factor of aging parameters. This work yielded an R^2 value of 0.98 with the values closer to 1 being considered as very good.

For most of the mentioned work, the batteries' cycles were experimentally controlled which is not the case with this study as it contains data from real usage of vehicle batteries making it a more complex problem. Analyzing all the mentioned approces for soh prediction on the basis of accuravy and simplicity. It was found that Random Forests Regression was a simple yet accurate solution for this problem. Therefore, this solution would be one of the used approaches in this work as it has already produced promising results on similar datasets/problem.

2 Data Analysis

Initial analysis of data features revealed some challenges in it including data outliers (figure 2) and some highly-biased features (figures 1,6). This motivated us to work further on data analysis to clean the data by discarding records, and attributes, if needed. Therefore, some important features of the data were statistically analysed with respect to the context of the problem to find more insights of the data.

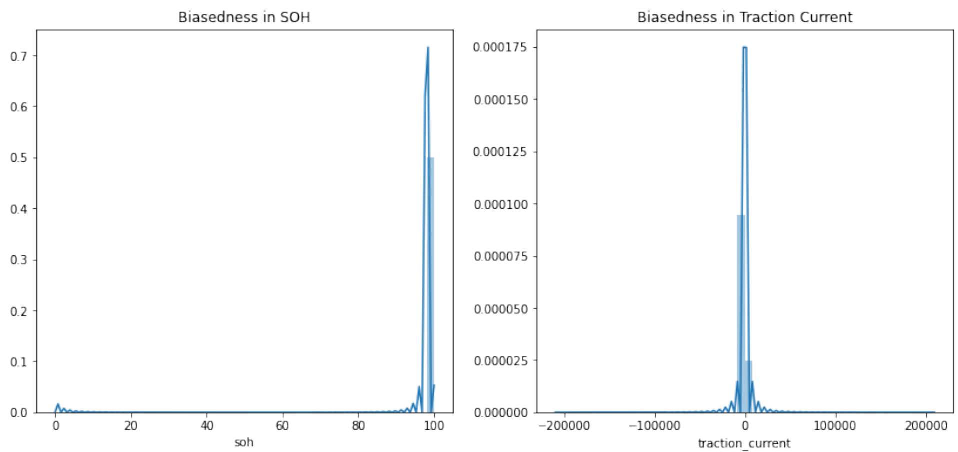

Figure 1: Highly Skewed Potential Target Variables

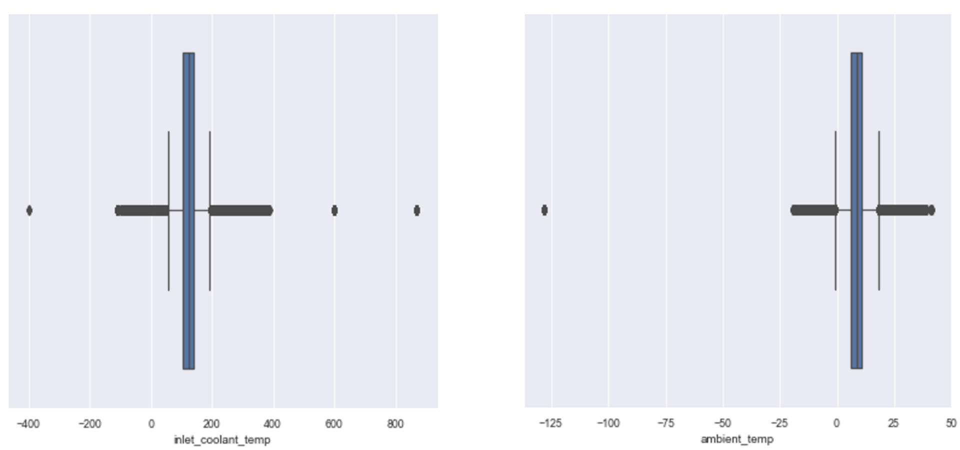

Figure 2: Outliers in the data

Table 2: Statistical Summary of Important Data Features

Feature	Minimum Value	Maximum Value	Mean	Standard Deviation
Odometer	0	142256	6022.74	4665.96
Ambient Temp	-128.0	46.5	8.16	11.57
Vehicle Speed	0	200.13	9.1	25.75
Max Voltage	0	4.18	3.89	0.17
Imbalance	-0.02	1.14	0.02	0.03
Power SOH	0	100.0	98.45	4.45
Inlet Coolant Temperature	-400	870	130.56	72.3
Imbalance Percentage	-0.0053	0.311	0.0059	0.0065
Minimum SOH	0	100.0	92.41	9.78
SOH (State of Health)	0	100.0	98.08	2.259
Maximum SOH	0	100.0	94.73	8.20

The table 2 shows that the important feature are not in the same scale which could cause challenges for the predictive models in the later stage. There are multiple features in the data, including minimum soh, maximum soh and imbalance, which could potentially be used as output variables for other studies. Though, soh would be predicted in this study, but the initial data analysis shows min. and max. soh have more variance than this variable.

2.1 Applying Principal Component Analysis (PCA)

It was found that the high number of input feature will impact the predictive model in later stages so it was decided to apply some dimensionality reduction technique on the dataset. This technique is mostly applied on the data to find independent features which can losslessly represent the data. PCA is one of such techniques which is mostly used to obtain those independent features. Using PCA as the main tool, a brief experiment comprising the activities shown in figure 3 was conducted to find most independent features (with least variance).

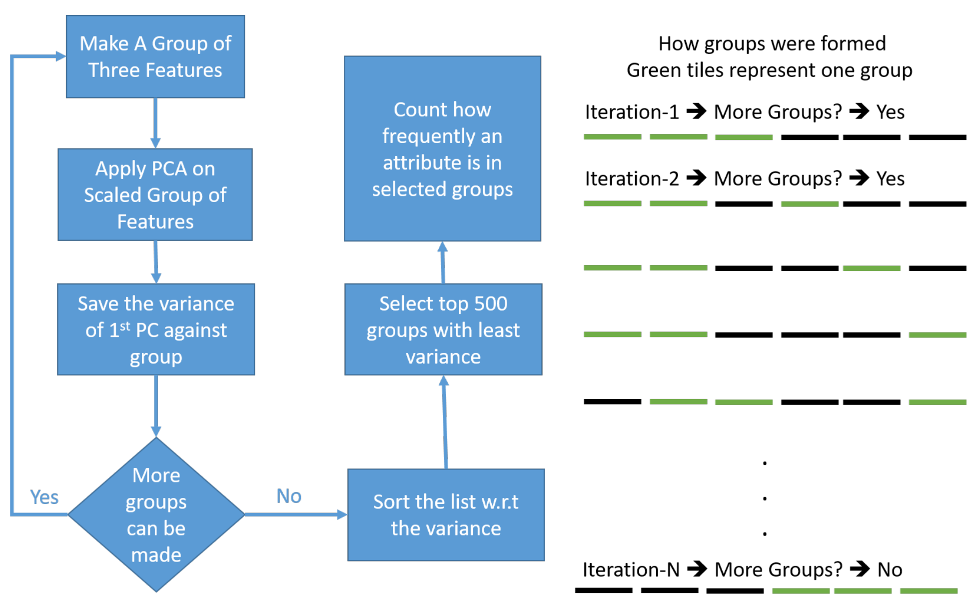

Figure 3: PCA Experiment Flow

At the completion of this, countplot was used to filter the top features as shown in figure 4. This idea of using principal component analysis for filtering important features is novel and was not inspired from any previous research work.

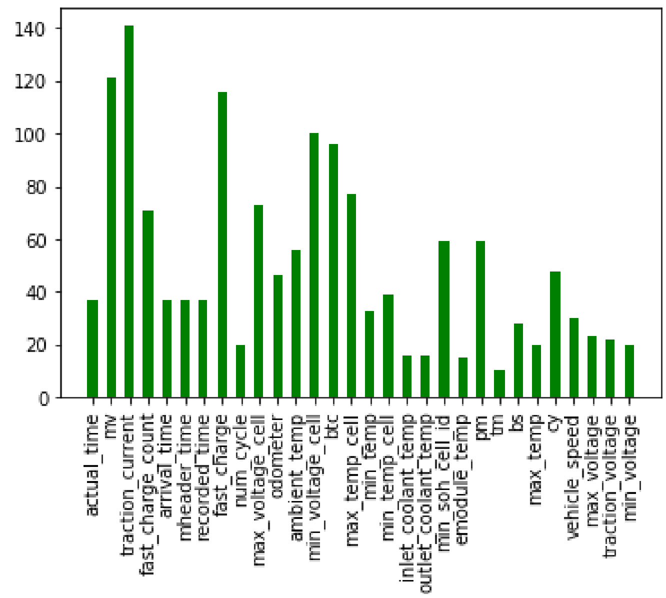

Figure 4: Most Independent Features

Figure 4 shows that 'traction_current' is the feature which was most frequently found in those groups which had low variance of first principal component. Which means, when PCA was applied on a group having 'traction_current' as one of them, their first principal component was unable to describe most of the details which implies this feature's independent nature. In this way, this countplot helped us to identify important features which must not be discarded.

2.2 Features Correlations

After finding independent features, the next task was to find those features which are highly dependent on each other. Therefore, correlation heatmap of the data features was plotted to find correlations of features with each other.

It can be seen from Figure 5 that there are some features which are highly correlated. An important observation was found for features 'fuse_temp' and 'cooling_energy_used' showing white lines. Exploring these features revealed that they only have one value throughout the dataset. This guided us to the idea of checking unique values in all the features. Following table 3 shows all the features which have less than 4 values:

Table 3: Number of unique values in features

Feature Name	Unique Values
Mheader_time	1
Balancing_status	3
Fuse_temp	1
Cat4	2
Cat6	2
Cat7	2
Cooling_energy_used	1
Cycle	3
Fast_charge	2

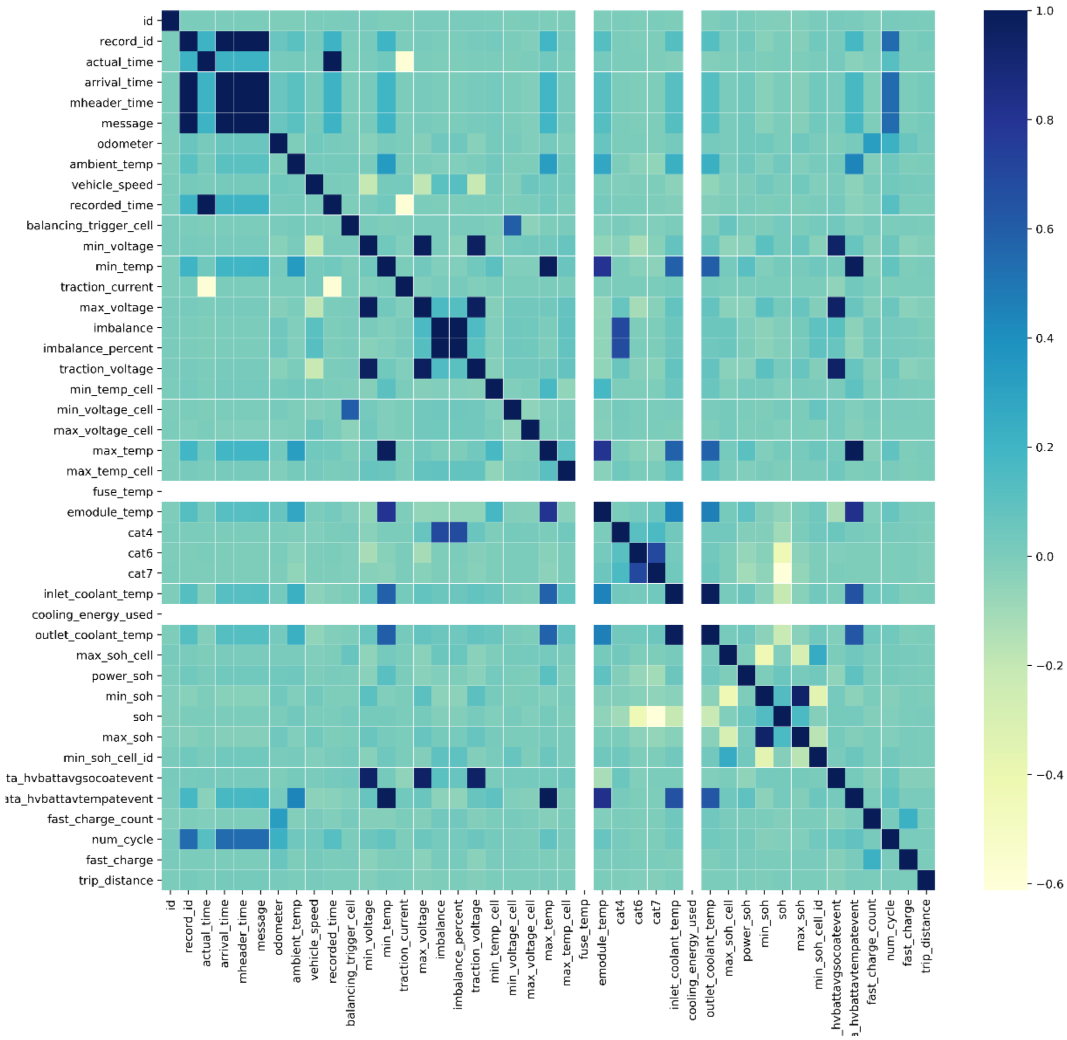

Figure 5: Correlation Heatmap of Features

Out of these ‘cycle’ feature is supposed to have three values so this feature was left as it is. Whereas cooling energy used and fuse temp having only one values shows their limitations as per the context of data so these two features will be removed right away. Also, count of values in ‘cycle’ features was plotted which clearly showed biasedness in the data towards charging state as it can be seen in figure 6:

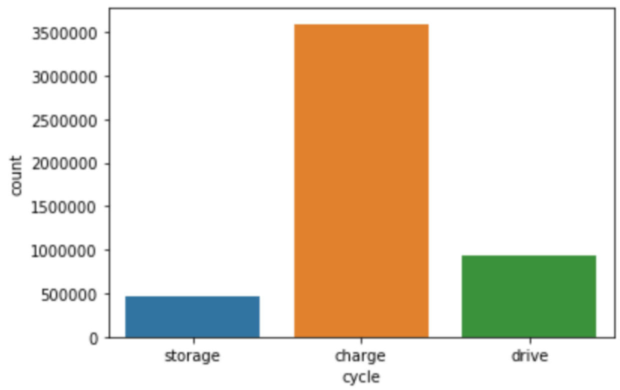

Figure 6: Biasedness in cycle feature

2.3 Discussion on Statistical Data Analysis

One of the important observation in this part was the biasedness of the data towards one state of the battery which is 'charging'. This observation guided us to have a second phase of the experimentation in which we individually applied the machine learning algorithms on the data clusters with respect to this features. Also, the statistical analysis, helped in discarding variables which are not relevant to the context or were highly correlated with each other (top left corner of figure 5). Imbalance percentage and imbalance are highly correlated so one of them will be removed. Similarly, minimum SOH and maximum SOH are highly correlated, and inlet coolant temperature is correlated with outlet temperature, so one of those will be removed as well. After this analysis, following features were discarded from the data:

- Record ID
- Arrival Time
- MHeader Time
- Message
- Cooling Energy Used
- Fuse Temp
- Cat 4,6,7
- Inlet/Outlet coolant temperature
- Minimum/Maximum SOH
- Imbalance

3 Methodology

The purpose of experimentation is to understand the factors influencing state of health and to find those machine learning techniques which works better on this type of data. With this in mind, experimentation is divided into two main phases using different pre-processing and predictive models. Phase-I will be composed of the following three main steps:

- Pre-Processing on the data and removal of the mentioned attributes
- Apply ML algorithm (Decision Tree Regression) on complete data with SOH as target variable
- Test the model on testing data to predict SOH

After this, second phase of experimentation will be started. At this time, we will have a predictive model which can predict state of health using the data features, but to better understand which feature is influencing more towards a good/bad state of health and to make this whole experimentation more relatable for the Engineers, we will apply labelling on the data in this phase. This will also help us to quantify good or bad state of health as well. To obtain rules in the format of "the feature A in range X – Y is more prone to resulting a bad state of health" we will apply Association Rule Mining (ARM) on the labelled data. Moreover, our data is highly biased towards the charging state (figure 2) so we will apply these rules on three subsets (on the basis of cycle feature) of data separately to avoid biasedness. On this labelled data, we will apply some prediction algorithm again to predict soh and will compare our prediction results with phase-I to conclude which machine learning techniques are better to predict soh using this type of data. Following figure 7 explains these activities in a flow chart:

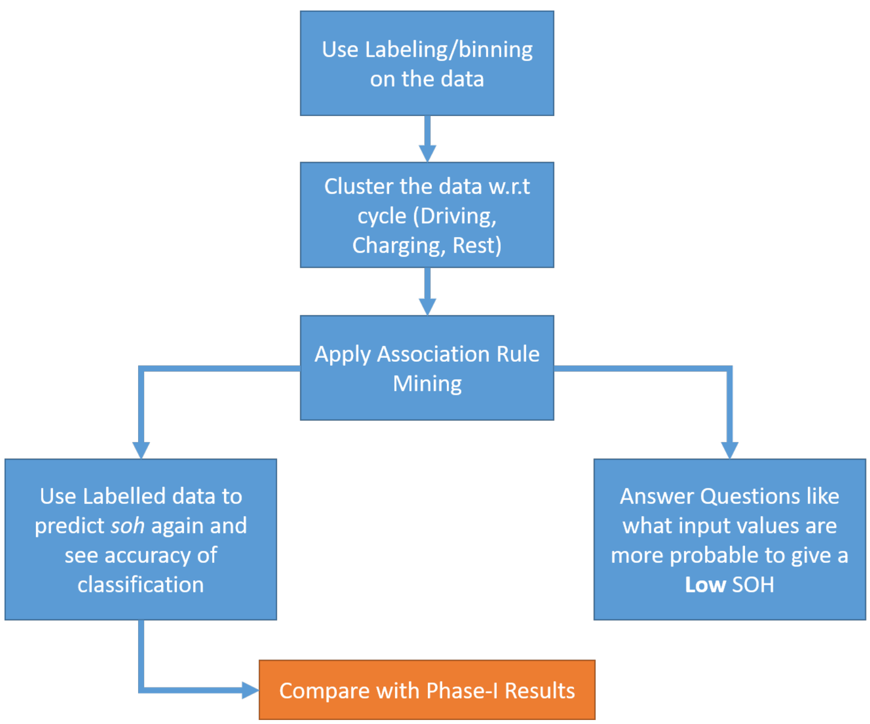

Figure 7: Activities in Phase-II

3.1 Phase-I

As described above, phase-I of the experiment had three main steps which are described below. As the target output in this phase is continuous, so the prediction task was regression.

Step-I – Pre-processing of the data

The first step in pre-processing was to remove the un-necessary features from the data. These features were identified by analysing the problem context and correlations among them. For instance, according to the context of problem, state of health is believed to be independent of record_id, so it should be removed. After removing such features, the next task was to cater the categorical features in the data as they were supposed to become a problem for our regression predictor. For this task, Ordinal encoding was used which maps each of the categorical string value to an integer value (*Ordinal Encoding using Scikit-Learn* (2020)). Following code was written in Python for that purpose:

```
balancing_kvp = {'noBalancing' : 1, 'passiveBalancing' : 2, 'initialValue' : 3}
thermal_kvp = {'idle' : 1, 'activeHeating' : 2, 'passingCooling' : 3, 'thermalBalancing' : 4,
 'initialValue' : 5, 'activeCooling' : 6}
cycle_kvp = {'charge' : 1, 'drive' : 2, 'storage' : 3}
fast_charge_kvp = {False : 0, True : 1}
```

After this, the data was split into training and testing datasets with a ratio of 80% and 20% respectively. This was done using train_test_split() method of Python libraries Scikit-learn.

Step-II – Prediction Algorithm

As the output we wanted to predict was continuous we had multiple options of regression algorithms for this task. Out of those choices, it was decided to apply Decision Tree Regression for this task as decision trees are considered to be more flexible when dealing with both categorical and continuous input variables. This was further enforced by the relevant work when Parkka et al. have compared the results of decision tree with Neural Networks for a relatively similar healthcare data gathered from sensors (Parkka et al. (2006)). Using the built-in Python libraries' functions a decision tree regression model was trained on the training data.

Step-III – Model Evaluation

The trained model was then used to do predictions against the testing dataset inputs. Finally, the predicted outputs were compared to the actual test outputs to calculate coefficient of determination R^2 (*R2 Score Scikit-Learn* (2020)).

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Where

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

and

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}\epsilon_i^2$$

Best value of R^2 for a predictor is 1.0 and value in our case was 0.95 which can be considered as good. Apart from this, Root-Mean-Square-Error (RMSE) has been found to be the performance metric in the literature so it was computed for the implemented model and the RMSE in this case was **0.46**. To further analyse the results, predicted and actual test values were plotted using scatter plots:

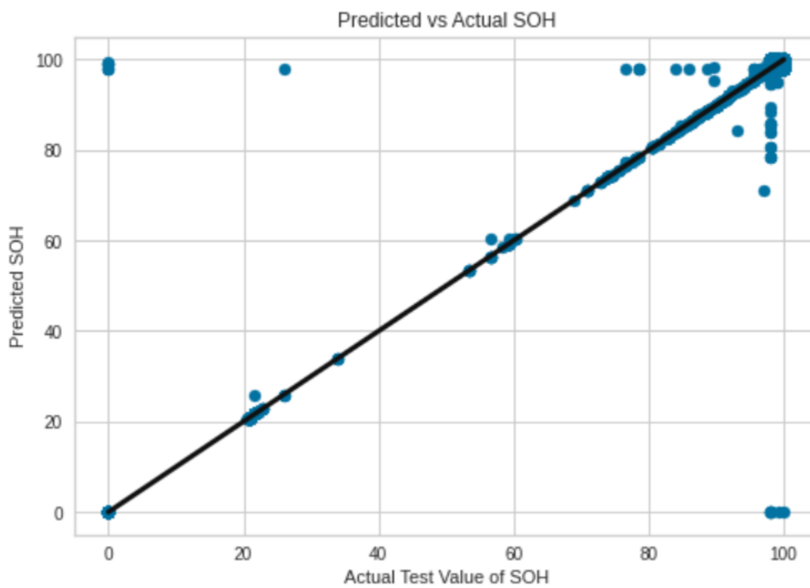

Figure 8: Actual Vs Predicted SOH Phase-I

It can be seen in the scatter plot (figure 8) as well, for most of the items, predicted and actual values are same. Still, it can be seen that there are relatively high number of errors at specific values of soh (0, 20, 80<). To dig deep into this issue distribution of soh values were plotted and zoomed in for values less than 95.

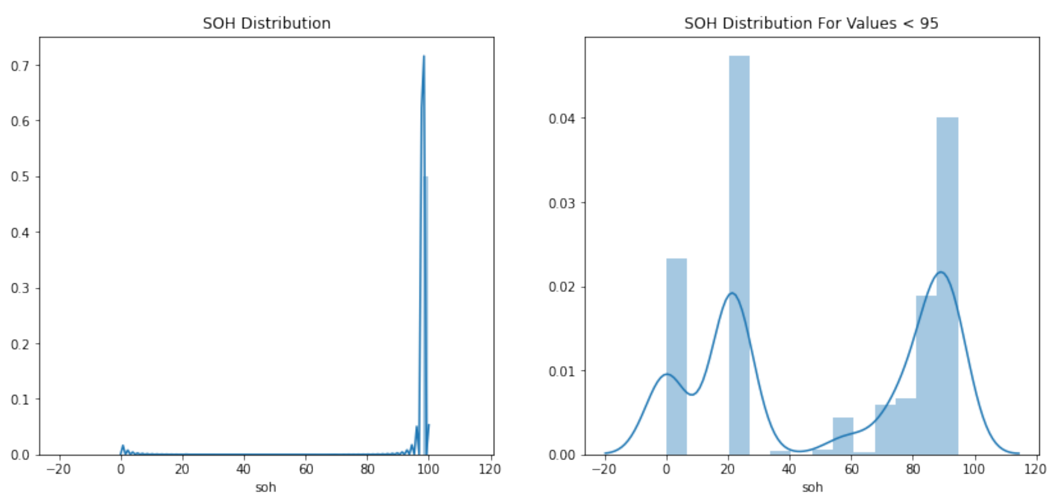

Figure 9: SOH feature Distributions

Left side of figure 9 represents distribution of complete feature whereas the right side is the distribution for values less than 95. It can be seen that there are spikes in the distribution at those locations which actually have high rate of prediction errors in figure 8. This implies, that for these highly concentrated values, similar combination of input features are producing different target output which the decision tree is unable to tackle.

3.2 Phase-II

As shown in figure 7, various activities were involved in this phase of experimentation which are going to be described below. The purpose of this phase was to see if labelling can help us in achieving more meaningful results which can later be shared with the concerned engineers. So, there were two major aims of this phase:

- Predict Labelled State of Health and compare the results with Phase-I
- Applying Association Rule Mining on the data

For both of these aims, the data binning was needed so it was done as the first step for both aims.

Step-I - Converting continuous features to categorical features

Binning is a process of converting continuous data into categorical data. This is done by defining ranges for each of the data features and denoting a particular value with the range it corresponds. This process is usually done in two major ways:

1. **Fixed-Width Binning**: In this type of binning, the number of bins are predefined by the user and the ranges are equal for a particular feature. This type of binning is easier to implement but is not considered a good option for the data having diverse features.
2. **Adaptive Binning**: This is more generic type of binning in which the number of bins and the ranges are are calculated through the distribution of a particular feature. This type of binning can work for diverse nature features but is usually more costly as compared to the fixed binning in terms of computation.

For our work, we have used Quantile binning on our dataset which is an implementation of adaptive binning. Following python code line performed this binning using predefined qcut() function:

```
new_data[col], boundaries[col] = pd.qcut(new_data[col], 4,duplicates='drop', labels=False,
 retbins=True)
```

To make the data ready for predictive algorithm, the remaining categorical variables with string values were also converted to numerical categories using the following code:

```
d3 = {'charge': 0, 'drive' : 1, 'storage' : 2}
d4 = {False: 0, True: 1}

X['cycle'] = X['cycle'].map(d3)
X['fast_charge'] = X['fast_charge'].map(d4)

d1 = { 'noBalancing' : 0, 'passiveBalancing' : 1, 'initialValue' : 2}
d2 = {'idle' : 0, 'activeHeating' : 1, 'passingCooling' : 2, 'thermalBalancing' : 3,
      'initialValue' : 4, 'activeCooling' : 5 }


X['thermal_manager_mode'] = X['thermal_manager_mode'].map(d2)
X['balancing_status'] = X['balancing_status'].map(d1)
```

Step-II - Applying Prediction Algorithm

The dataset was distributed in training/testing subsets with 75/25 ratio respectively. Out of the options we had for those predictive algorithms which can predict the class of target variable, Logistic regression was chosen. This was mainly because of its wide use while working with categorical data when the target is a class as well. Following Python code was written for that:

```
logisticRegr = LogisticRegression()
logisticRegr.fit(x_tr, y_tr)
```

Step-III - Evaluation of Results

To evaluate the performance of the regression model, the remaining 25% of the dataset was fed to the model to predict the class of soh, which was our target variable. To compare the results with Phase-I, R^2 score of these predictions was calculated which came as **0.78**.

As it was a binary classification problem with state of health being only low or high and the algorithm was trained to predict the class on the basis of input variables.

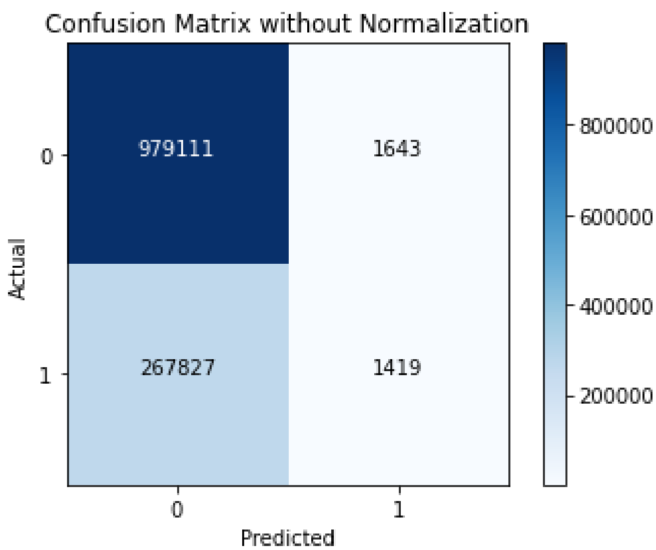

Figure 10: Phase-II Confusion Matrix for Binary Classification of SOH

Figure 10 shows confusion matrix to evaluate the results of our process. There was a huge amount of errors when the actual value was 1 but the model predicted as 0, this can be because of highly biased nature of labelled soh variable (Figure 11).

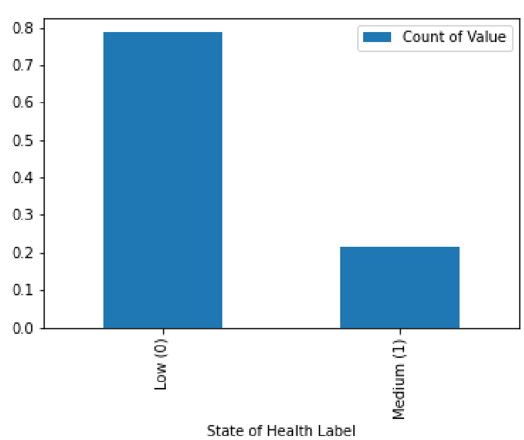

Figure 11: Distribution of Labelled SOH After Quantile Binning

The large difference in actual and predicted values guided us to look for the cause and it was decided to implement another binning technique on the target variable. Because, it was found that the same combination of input variable was resulting high soh and the same combination was resulting low at some different instance. This lead us to apply some other binning technique on the target variable before using the data for prediction again.

Step-I(A) - Binning Using Bayesian Block Algorithm

A fuzzy binning algorithm, Bayesian Block Binning, was applied on the target variable (Scargle et al. (2013)).

Bayesian Block Binning algorithm provides the boundaries for bins and when soh variable was passed to this which gave following boundaries for soh variable:

```
[  0.,    68.25,  84.25,  98.05, 100.  ]
```

On the basis of these boundaries, the soh feature was labelled again and the dataset was fed to logistic regression algorithm to be trained on the training data for predictions on the later stage. The distribution of target variable at this stage is shown in figure 12.

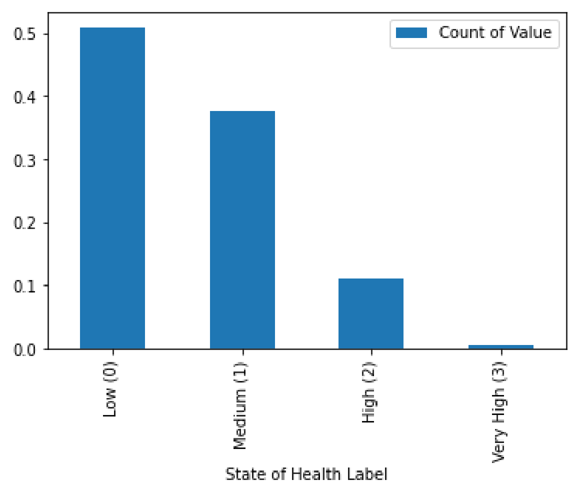

Figure 12: Distribution of Target Feature After Bayes Binning

Step-III(A) - New Prediction Results

This new labelling also showed similar results as the previously used binning which can be seen in the confusion matrix (Figure 13).

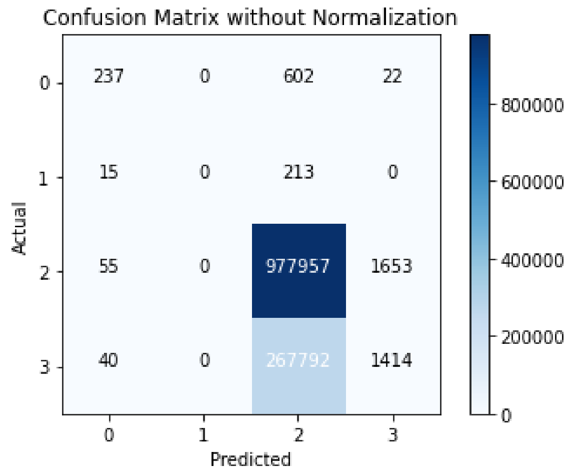

Figure 13: Phase-II Confusion Matrix After Bayesian Block Binning

Discussion on Phase-II Predictions

It was seen in both the binning types that the model results are relatively bad. Table 4 shows a comparison of evaluation metrics of predictive models after both binning algorithms.

Table 4: Effect of Different Binning

Binning Technique	Accuracy	Recall	F1-Score	Precision
Quantile	78.44	0.52	1.04	46.34
Bayesian Blocks	78.36	78.36	69.10	71.4

It can be seen from Table 4 that the accuracy of the model is minimally changed with the new binning. Whereas, there has been a huge improvement in terms of Recall and F1-score. This result was further analyzed by exploring the biasedness in *soh* variable. It was seen that most of the values in soh variable are 98, therefore, it was decided to plot the count of records for which soh was 98 and all the others. Following code was used to plot the number of records for which soh was 98.0:

```
sns.countplot(filtered_data['soh'] == 98.0)
```

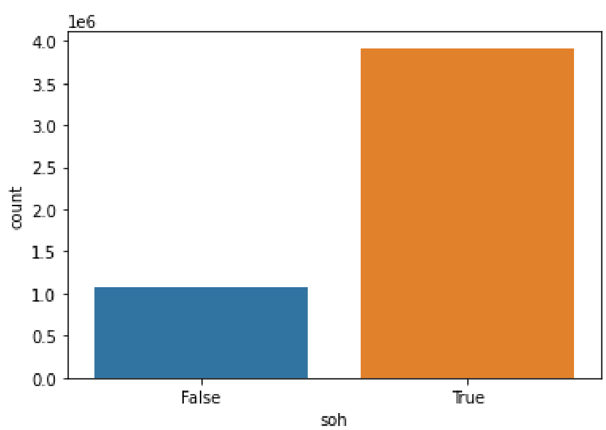

Figure 14: SOH Biasedness

In figure 14, it can be clearly seen that the feature is highly biased for just one value of soh which is 98. The block labeled as 'True' is denoting the number of records for which soh is 98, whereas the 'False' block is showing all the remaining values.

Phase-II - Association Rule Mining (ARM)

The next aim in this phase was to apply ARM on the data to get meaningful rules for the engineers. For this purpose, already implemented quantile binning was used on the raw data followed by these two actions:

- Labelling the data (Low, Medium, High, Very-High)
- Make three clusters/subsets of the data on the basis of cycle variable
- Encoding the data for ARM

Labelling the data

The above-mentioned code converted the continuous variables in values ranging from 0-3 with 0 being the lowest value range of that feature and 4 being the highest. To make the association rules more meaningful for the engineers, the dataset was labelled as Low, Medium, High, and Very-High labels corresponding values from 0-3 respectively. Following Python code was used for this labelling:

```
d = {0 : 'Low', 1 : 'Medium', 2 : 'High', 3 : 'Very_High'}
d1 = { 'noBalancing' : 0, 'passiveBalancing' : 1, 'initialValue' : 2}
d2 = {'idle' : 0, 'activeHeating' : 1, 'passingCooling' : 2, 'thermalBalancing' : 3,
      'initialValue' : 4, 'activeCooling' : 5 }
for c in float_cols.columns:
  filtered_data[c] = filtered_data[c].map(d)
```

This code replaced the annotated data from 0-3 as labels from Low to Very-High with Low denoting the lowest range of that feature (0) and Very-High denoting the highest range (3).

Clustering the data w.r.t cycle feature

As shown in Figure 6 that the data was highly biased towards one value of 'cycle' feature (charging), so the data was partitioned into three subset with respect to values of this feature to avoid biasedness in the rules. At the end of this step, we had three subsets of data representing the data records while battery was charging, driving, and storage (at rest).

Encoding the dataset for ARM

Association rule mining in Python has to be applied on encoded data in such a way that the dataset should only be composed of 0s and 1s. The data at this stage has only categorical features and these categorical features were encoded using One-hot encoding algorithm, already implemented in Python libraries. This was done using the following code:

```
encoded_storage = pd.get_dummies(bat_storage_data)
encoded_charge = pd.get_dummies(bat_charge_data)
encoded_drive = pd.get_dummies(bat_drive_data)
```

Applying ARM and Results

Association Rule Mining was applied on each of the data subset using **Apriori Algorithm** and rules with minimum support of 0.6 and minimum confidence of 0.8 were extracted (Al-Maolegi & Arkok (2014)). Following were the top five rules for Medium SOH of storage dataset with respect to confidence and support:

Table 5: Rules from Storage Data subset

antecedents	consequents	support	confidence	lift
(vehicle_speed_Low)	(soh_Medium)	1.00	1.0	1.0
(balancing_status_noBalancing)	(soh_Medium)	0.95	1.0	1.0
(traction_current_Very_High)	(soh_Medium)	0.99	1.0	1.0
(max_temp_cell_Low)	(soh_Medium)	0.69	1.0	1.0
(thermal_manager_mode_idle)	(soh_Medium)	0.908	1.0	1.0

Similarly, following were the top five rules for Medium SOH of drive dataset with respect to confidence and support:

Table 6: Rules from Drive Data subset

antecedents	consequents	support	confidence	lift
(vehicle_speed_Low)	(soh_Medium)	1.00	1.0	1.0
(balancing_status_noBalancing)	(soh_Medium)	0.979	1.0	1.0
(traction_current_Very_High)	(soh_Medium)	0.804	1.0	1.0
(max_temp_cell_Low)	(soh_Medium)	0.801	1.0	1.0
(thermal_manager_mode_idle)	(soh_Medium)	0.978	1.0	1.0

Finally, Medium SOH rules charging subset were:

Table 7: Rules from Charging Data subset

antecedents	consequents	support	confidence	lift
(vehicle_speed_Low)	(soh_Medium)	1.00	1.0	1.0
(fast_charge_count_Low)	(soh_Medium)	1.0	1.0	1.0
(balancing_status_noBalancing)	(soh_Medium)	0.976	1.0	1.0
(max_temp_cell_Low)	(soh_Medium)	0.768	1.0	1.0
(thermal_manager_mode_activeHeating)	(soh_Medium)	0.645	1.0	1.0

It can be seen from the tables 5, 6 and 7 that the rules generated from each of the subset are almost similar with more similarity between storage and drive rules. Relating again with the distribution of labelled soh variable in figure 11, there were almost 4 times 'Low' values than 'Medium'. Therefore, there were no rules with high confidence for 'Medium' soh. Also, for all the 'Low' rules, 'Lift' was always more than 0.95 or greater. These values of Lift almost equal to 1 every time indicates that the consequent and antecedent were independent despite high confidence value of the rules.

Distributions of four of the most frequently occurring features (vehicle_speed, fast_charge_count, balancing_status, traction_current) in these rules were found to analyze the usefulness of these rules. Two of the features i.e. vehicle_speed and fast_charge_count had one value throughout the labelled dataset which resulted rules with high characteristic values. Similarly, balancing_status was also highly biased towards one value (Figure 15) which was the reason to have its rule with high confidence.

On the other hand, traction_current was a uniformly distributed feature against SOH as it can be seen in the figure 16.

But, analyzing the distributions against cycle variable revealed some more details about its absence from charging rules. Here the actual details about 'traction current' helped to explain this behavior as this current

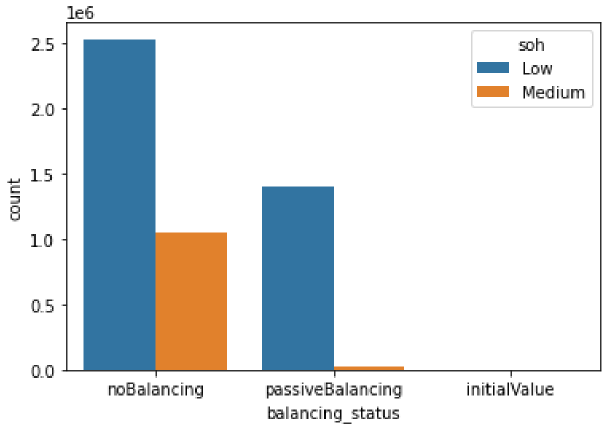

Figure 15: Distribution of Balancing Status Against SOH

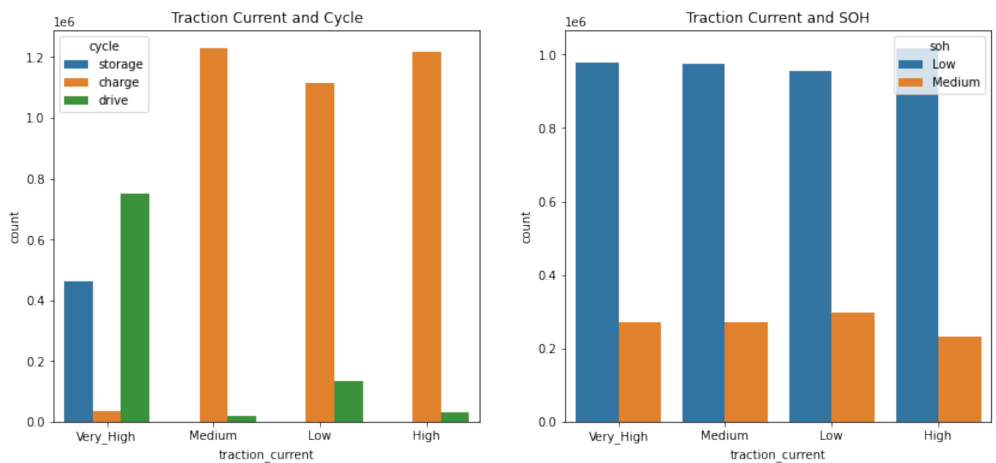

Figure 16: Distribution of Traction Current

is at its highest level when the batteries are not being charged. This is a common behavior with all the batteries so it **cannot** be used to imply that whenever traction current would be very high, it would result a good state of health.

4 Critical Review of Used Techniques

In this study, a real-life dataset with a real problem was being investigated and choice of tool (Python language) was a one of the requirements imposed by the company for which this project was being done. Following Python libraries were used during the experimentation:

1. **Pandas:** For importing the data in the form of data frames (*Pandas* (2020))
2. **Scikit-learn:** For Machine Learning algorithms including regression models (*Scikit-Learn* (2020))
3. **Seaborn:** For appealing data visualizations (*Seaborn* (2020))

Despite being a requirement, using Python for the implementation was proved to be a very good choice because of its rich libraries ranging from seamless data import to visualizations and easy to use APIs (Application Program Interface) for machine learning algorithms.

Phase-I of the experiment has produced relatively good results and are comparable with the previous works using similar methods on similar data. For instance, the work of Tseng et al. had yielded an R^2 score of 0.98 which was 0.96 in our case (Tseng et al. (2015)). Similarly, both of the mentioned works using Random Forest had produced RMSE of 1.2 and 0.03 which was 0.46 in our results (Li et al. (2018))(Tao & Lu (2017)). The selection of this algorithm for this phase was due to the flexible nature of decision trees for effectively handling categorical and continuous data. The results actually proved the choice to be a good one with some errors at specific locations. As the data was collected from real-life scenario, these kind of errors were expected with large amount of records having same relationship yielding different outcome.

Phase-II was aimed to see whether labelling this kind of data can produce better results or not. The results of binning were poor as compared to the first phase, mainly because labels of such a biased target value were not uniform in any of the applied binning algorithm (Quantile and Bayesian Blocks). The challenge of the data again came into play for this as almost 4/5 of the soh records had value of 98 (Figure 14). As the problem at this phase was at the pre-processing of the data, any of the classifier would have shown similar kinds of results. Still, the choice of Logistic regression for this phase was because of its effectiveness for classification problems.

From the results of both phases, it was observed that directly applying prediction algorithm on the dataset, without any major transformation, has yielded better prediction results as compared to applying predictive algorithms on only categorical data. Though it cannot be concluded that categorizing the data before feeding it to the predictive model would always produce poor results as the current observation is because of the kind of dataset we had in the study. To illustrate, when the highly biased features in the dataset were binned in limited ranges, there were various records when the similar combination of input features were yielding different targets. This kind of scenario is preferred to be avoided in prediction problems.

Another important result from the study was the rules we obtained at the end of phase-II which are yet to be explored by mapping them with the domain knowledge.

Challenges

The major challenge was the data itself due to a high number of data features and highly skewed (biased) important features. Another major challenge was the interpretations of data features requiring some knowledge of the battery functioning. These challenges were tackled by reading about the domain and relevant literature first, to get a better idea about the domain. Then, techniques like Principal Component Analysis (PCA) and Correlation heat-map were used to identify most important and redundant data features. Moreover, to cater the problem of biasedness, the dataset was labelled as well as partitioned to analyze the impact.

Word count: 4856

References

Al-Maolegi, M. & Arkok, B. (2014), 'An improved apriori algorithm for association rules', *arXiv preprint arXiv:1403.3948* .

Bhide, S. & Shim, T. (2011), 'Novel predictive electric li-ion battery model incorporating thermal and rate factor effects', *IEEE Transactions on vehicular technology* **60**(3), 819–829.

Guha, A. & Patra, A. (2017), 'State of health estimation of lithium-ion batteries using capacity fade and internal resistance growth models', *IEEE Transactions on Transportation Electrification* **4**(1), 135–146.

Landi, M. & Gross, G. (2014), 'Measurement techniques for online battery state of health estimation in vehicle-to-grid applications', *IEEE Transactions on Instrumentation and Measurement* **63**(5), 1224–1234.

Li, Y., Zou, C., Berecibar, M., Nanini-Maury, E., Chan, J. C.-W., van den Bossche, P., Van Mierlo, J. & Omar, N. (2018), 'Random forest regression for online capacity estimation of lithium-ion batteries', *Applied energy* **232**, 197–210.

Lin, H.-T., Liang, T.-J. & Chen, S.-M. (2012), 'Estimation of battery state of health using probabilistic neural network', *IEEE Transactions on Industrial Informatics* **9**(2), 679–685.

Ordinal Encoding using Scikit-Learn (2020), https://scikit-learn.org/stable/modules/preprocessing.html. Accessed: 2020-05-07.

Pandas (2020), https://pandas.pydata.org/. Accessed: 2020-05-07.

Parkka, J., Ermes, M., Korpipaa, P., Mantyjarvi, J., Peltola, J. & Korhonen, I. (2006), 'Activity classification using realistic data from wearable sensors', *IEEE Transactions on information technology in biomedicine* **10**(1), 119–128.

R2 Score Scikit-Learn (2020), https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2$_{s}core.html.Accessed : 2020 - 05 - 07.$

Scargle, J. D., Norris, J. P., Jackson, B. & Chiang, J. (2013), 'Studies in astronomical time series analysis. vi. bayesian block representations', *The Astrophysical Journal* **764**(2), 167.

Scikit-Learn (2020), https://scikit-learn.org/. Accessed: 2020-05-07.

Seaborn (2020), https://seaborn.pydata.org/. Accessed: 2020-05-07.

Tang, X., Zou, C., Yao, K., Chen, G., Liu, B., He, Z. & Gao, F. (2018), 'A fast estimation algorithm for lithium-ion battery state of health', *Journal of Power Sources* **396**, 453–458.

Tao, L. & Lu, C. (2017), Random forest based li-ion battery capacity prediction subject to capacity recovery effect, *in* '2017 IEEE Vehicle Power and Propulsion Conference (VPPC)', IEEE, pp. 1–6.

Tseng, K.-H., Liang, J.-W., Chang, W. & Huang, S.-C. (2015), 'Regression models using fully discharged voltage and internal resistance for state of health estimation of lithium-ion batteries', *energies* **8**(4), 2889–2907.

5 Appendix - PCA Code

```
# -*- coding: utf-8 -*-
"""PCA-cleaned.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1vqqgF1590x32EJDSizbk8OYCMtpm8W1e

## Using Principal Component Analysis (PCA) for Dimenstionality Reduction
Working with large dataset having multiple features affects the efficiency of prediction models in machine learning. Therefore, dimensionality reduction is mostly applied on the data to find independent features which can losslessly represent the data. PCA is one of such techniques which is mostly used to obtain those independent features. Following code applies PCA technique for finding low variant features in the given batteries' dataset.
"""

# Commented out IPython magic to ensure Python compatibility.
'''
authored by Muhammad Usman
'''
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline
import seaborn as sns

FILE_PATH = 'C:/Users/musman14/Desktop/DS Research/train_naive_encode_corr.csv'
bat_data = pd.read_csv(FILE_PATH)    # reading the data from csv file

"""Implementing PCA in Python primarily composed of three main functions:
1. Scaling the data (Features of different scale can be projected on the same scale)
2. Separate out target features
3. Use PCA to project the data in less dimenstions (1D in our case)
Details about each of these can be found [here](https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60).
"""

from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# initializing standard scaler, to be used for data scaling
scaler = StandardScaler()

# dropping the probable target attributes from the data
output_data = bat_data.loc[:, ['max_soh_cell', 'min_soh', 'soh', 'max_soh', 'imbalance_percent']]
bat_data = bat_data.drop(columns = ['Unnamed: 0', 'max_soh_cell', 'min_soh', 'soh', 'max_soh', 'imbalance_percent'])

# backing up the data and scaling it to be used in PCA
bat_data_scaled = bat_data.copy()
bat_data_scaled[bat_data_scaled.columns] = scaler.fit_transform(bat_data_scaled)

```

```
# selecing only one principal component from the analysis as we are already dealing with a group of three
pca = PCA(n_components=1)

# iteratubg the list of attributes to get combinations of three
attr_dict = dict()   # this dictionary will hold the variance of group of attributes in dict[attributes-name] = 1st PCA comp

# making group of three attributes
for i in range (len(bat_data_scaled.columns) - 3):
    for j in range (i+1, len(bat_data_scaled.columns) - 2):
        for k in range (j+1, len(bat_data_scaled.columns) - 1):
            input_cols = [bat_data_scaled.columns[i], bat_data_scaled.columns[j], bat_data_scaled.columns[k]]
            col_str = "" + bat_data_scaled.columns[i] + "," + bat_data_scaled.columns[j] + "," + bat_data_scaled.columns[k]
            pca_batt = pca.fit_transform(bat_data_scaled[input_cols]) # applying PCA
            var = np.round(pca.explained_variance_ratio_, decimals=3) * 100 # variance on the first component
            print(col_str,  "=" , var)
            attr_dict[col_str] = var

import operator
# sorting the dictionary with respect to values on variance
sorted_dict = {k: v for k, v in sorted(attr_dict.items(), key=lambda item: item[1])}
single_attr_dict = dict()

count = 0

# going through top 500 results, with least variance
for key in sorted_dict:
    temp = key.split(",")  # Splitting the keys in single attributes
    for val in temp:       # for each attribute, count how many times it is present in the top 500 results
        if val in single_attr_dict:
            single_attr_dict[val] += 1
        else:
            single_attr_dict[val] = 1
    if count == 500:
        break
    count+=1

# plotting the results in bargraphs
df = pd.DataFrame([single_attr_dict])

plt.xticks(rotation='vertical')
plt.bar(single_attr_dict.keys(), single_attr_dict.values(), width=0.5, color='g')

#sns.countplot(x=df[:])
```

6 Appendix - Phase-I Code

```
# -*- coding: utf-8 -*-
"""Decision_Tree_Complete_Data.ipynb

Automatically generated by Colaboratory.

Original file is located at
```

```
    https://colab.research.google.com/drive/1VPVC0TAnr_7-hqMh8mLsB0VxWOKlyIAu

## Applying Decision Tree Regression for Predicitng State of Health in Li-Ion Batteries
Following are the list of activities being performed in this notebook:
* Analysis of Complete Battery Data
* Data Cleaning
* Data Pre-processing for Prediction (Coverting String categorical features to Numeric)
* Spliting the data into Test/Train Subbsets
* Applying Decision Tree on the Data
* Evaluating Decision Tree Results on the Test Data

### Statistical Analysis of Battery Data
"""

from google.colab import drive
drive.mount('/content/drive')

import os
os.chdir("/content/drive/My Drive/Colab Notebooks")
!ls

import platform
import dill
dill.load_session('dec_tree_complete_data.db')
#platform.architecture()

#!pip install tensorflow==2.0.0-alpha0

# Commented out IPython magic to ensure Python compatibility.
'''

!pip install pandas
!pip install numpy
!pip install matplotlib
!pip install seaborn
'''

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline
import seaborn as sns

FILE_PATH = "batteries_processed.csv"
bat_data = pd.read_csv(FILE_PATH, error_bad_lines = False)    # reading the data from csv
 file

print(bat_data[bat_data['soh'] == 0].shape, bat_data[bat_data['soh'] == 100].shape)

low_soh = bat_data[bat_data['soh'] < 95]
f, axes = plt.subplots(1, 2, figsize=(14, 6), sharex=True)
ax1 = sns.distplot(bat_data['soh'], ax = axes[0])
ax2 = sns.distplot(low_soh['soh'], ax = axes[1])
ax1.set_title("SOH Distribution")
ax2.set_title("SOH Distribution For Values < 95")

f, axes = plt.subplots(1, 2, figsize=(14, 6), sharex=False)
sns.distplot(bat_data['soh'], ax = axes[0])
sns.distplot(bat_data['traction_current'], ax = axes[1])
```

```
axes[0].set_title('Biasedness in SOH')
axes[1].set_title('Biasedness in Traction Current')

data_columns = bat_data.columns
bat_data.shape

"""### Removal of Extra Features"""

input_data = bat_data.copy()
columns_to_drop = ['id', 'vin', 'vin_prefix', 'record_id','recorded_time', 'actual_time', 'arrival_time', 'powermode', 'mheader_vin', 'mheader_time', 'mheader_type', 'message','cat4', 'cat6', 'cat7', 'inlet_coolant_temp','cooling_energy_used', 'max_soh_cell', 'power_soh', 'min_soh', 'max_soh', 'soh','min_soh_cell_id','imbalance_percent', 'parkingdata_hvbattavgsocoatevent', 'parkingdata_hvbattavtempatevent']
input_data = input_data.drop(columns=columns_to_drop)
input_data.head()

## keep vin_prefix for input variables
## min_voltage is highly correlated with traction voltage and max_voltage
##

output_data = bat_data.soh

#!pip install sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

print(input_data.info())

input_data.describe()

"""### Replacing String Categorical Variables to Numeric Categorical Features"""

object_attr = ['balancing_status', 'thermal_manager_mode', 'cycle', 'fast_charge']
for attr in object_attr:
    print(attr)
    print(bat_data[attr].value_counts())

balancing_kvp = {'noBalancing' : 1, 'passiveBalancing' : 2, 'initialValue' : 3}
thermal_kvp = {'idle' : 1, 'activeHeating' : 2, 'passingCooling' : 3, 'thermalBalancing' : 4, 'initialValue' : 5, 'activeCooling' : 6}
cycle_kvp = {'charge' : 1, 'drive' : 2, 'storage' : 3}
fast_charge_kvp = {False : 0, True : 1}

input_data[object_attr[0]] = input_data[object_attr[0]].replace(balancing_kvp)
input_data[object_attr[1]] = input_data[object_attr[1]].replace(thermal_kvp)
input_data[object_attr[2]] = input_data[object_attr[2]].replace(cycle_kvp)
input_data[object_attr[3]] = input_data[object_attr[3]].replace(fast_charge_kvp)
#fast_charge_kvp = {False : -1, True : 1}

#input_data = input_data.drop(columns=['recorded_time'])
input_data.head()

input_data.replace([np.inf, -np.inf], np.nan)

input_data.isnull().sum().sum()
#input_data['imbalance_percent'].value_counts()
```

```
"""### Test/Train Split of 20/80"""

X_train, X_test, y_train, y_test = train_test_split(input_data, output_data, test_size=0.2,
random_state=0)

"""### Applying Decision Tree Regression"""

from sklearn.tree import DecisionTreeRegressor

reg_tree = DecisionTreeRegressor()



reg_tree.fit(X_train, y_train)
reg_tree_pred = reg_tree.predict(X_test)

import sklearn
sklearn.tree.plot_tree(reg_tree)

"""### Evaluating Decision Tree Results"""

from sklearn.model_selection import cross_val_score

score = reg_tree.score(X_test, y_test)
print(score)

from sklearn import metrics
import math

mse = metrics.mean_squared_error(y_test, reg_tree_pred)
rmse = math.sqrt(mse)
print(mse, rmse)

"""This score is coefficient of determination R<sup>2</sup>. The coefficient R<sup>2</sup> is
 defined as (1 - u/v), where u is the residual sum of squares (y_true - y_pred)<sup>2</sup>
and v is the total sum of squares (y_true - y_true.mean()<sup>2</sup>. The best possible
score is 1.0 and it can be negative (because the model can be arbitrarily worse)"""

# https://scikit-learn.org/stable/modules/model_evaluation.html
from sklearn import metrics
#reg_tree_pred.value_counts()
#metrics.accuracy_score(y_test, reg_tree_pred)
#metrics.f1_score(y_test, reg_tree_pred)
mse = metrics.mean_squared_error(y_test, reg_tree_pred)
print(mse)

from yellowbrick.regressor import PredictionError

fig, ax = plt.subplots()
ax.scatter(y_test, reg_tree_pred)
ax.set_title('Predicted vs Actual SOH')
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k-', lw=3)
ax.set_xlabel('Actual Test Value of SOH')
ax.set_ylabel('Predicted SOH')
plt.show()

#def f(x):
#    x = x.ravel()
#   return np.exp(-x ** 2) + 1.5 * np.exp(-(x - 2) ** 2)
```

```
'''
plt.figure(figsize=(10, 6))
#plt.plot(X_test, f(X_test), "b")
plt.scatter(X_test, y_test, c="b", s=20)
plt.plot(X_test, reg_tree_pred, "g", lw=2)
plt.xlim([-5, 5])
#plt.title("Decision tree regressor, MSE = %.2f" % (np.sum((y_test - reg_tree_pred) ** 2) /
n_test))
plt.show()
'''

#bat_data['soh'].value_counts()
sns.distplot(bat_data['soh'])

sns.distplot(bat_data['imbalance'])

#regressor = LinearRegression()
#regressor.fit(X_train, y_train)

#sns.countplot(bat_data['powermode'])

dill.dump_session('dec_tree_complete_data.db')
```

7 Appendix - Phase-II Code

```

# -*- coding: utf-8 -*-
"""Phase-II-ML_Final.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1z3vfAD4TFaORH2Td5y6saZFHoS2ZGQyf

## Phase-II
This phase would have the following keys steps to perform:
* Apply labelling/binning on the data (yet to decided the technique)
* Make three clusters/subsets of the data on the basis of cycle variable
* Apply Association Rule Mining on the subsets
* Apply prediction technique to classify the target variable soh
* Evaluate Models Results

More explanation for the process and what I want to acheive
"""

from google.colab import drive
import os
drive.mount('/content/drive')
os.chdir("/content/drive/My Drive/Colab Notebooks")

import platform
import dill
dill.load_session('phase-II-ml-copy.db')

# Commented out IPython magic to ensure Python compatibility.
'''
authored by Muhammad Usman
```

```
!pip install pandas
!pip install numpy
!pip install matplotlib
!pip install seaborn
'''
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline
import seaborn as sns

FILE_PATH = "batteries_processed.csv"
bat_data = pd.read_csv(FILE_PATH, error_bad_lines = False)    # reading the data from csv
file

"""### Removing the same features which were removed in Phase-I as well"""

filtered_data = bat_data.copy()
columns_to_drop = ['id', 'vin', 'vin_prefix', 'record_id','recorded_time', 'actual_time', '
arrival_time', 'powermode', 'mheader_vin', 'mheader_time', 'mheader_type', 'message','cat4',
'cat6', 'cat7', 'inlet_coolant_temp','cooling_energy_used', 'max_soh_cell', 'fuse_temp', '
power_soh', 'min_soh', 'max_soh', 'min_soh_cell_id','imbalance_percent', '
parkingdata_hvbattavgsocoatevent', 'parkingdata_hvbattavtempatevent']
filtered_data = filtered_data.drop(columns=columns_to_drop)
filtered_data.head()

sns.countplot(filtered_data['soh'] == 98.0)

for c in filtered_data.columns:
  print("Column", c)
  print(filtered_data[c].value_counts(normalize='true'))

float_cols

"""### Binning the data
We have to choices for Binning the data here:
* Fixed-Width Binning: Specific fixed widths for each of the bins which are usually pre-
defined
* Adaptive Binning: Such binning in which we use the data distribution itself to decide bin
ranges

&rightarrow; Quantile Binning is one of the type of Adaptive Binning which helps in
partitioning the continuous valued distribution of a specific numeric field into discrete
contiguous bins or intervals. We will use this binning to convert out continuous variables
into categorical variables.
"""

int_cols = filtered_data.select_dtypes('int64') ## finding all the continuous variables
float_cols = filtered_data.select_dtypes('float')
categ_data = pd.DataFrame()
int_cols = int_cols.loc[:, int_cols.nunique() > 4]
float_cols = float_cols.loc[:, float_cols.nunique() > 4]
#float_cols = float_cols.drop(columns=['soh'])

## try fuzzy binning on the SOH
## print the boundaries as well
boundaries = dict()
for col in int_cols.columns:
  print("Col in progress is ", col)
```

```
    filtered_data[col], boundaries[col] = pd.qcut(filtered_data[col], 4,duplicates='drop',
labels=False, retbins=True)
  print("Int Cols Now")
  for col in float_cols.columns:
    print("Col in progress is ", col)
    filtered_data[col], boundaries[col] = pd.qcut(filtered_data[col], 4, duplicates='drop',
labels=False, retbins=True)
  filtered_data.head()

  ## This output will show the boundary cut for binning
  ## the output should be read in the following way e.g. for [      0.    2813.    5412.    8233.
142256.]
  ## 0. to  2813. --> Low
  ## 2813. to 5412. --> Medium
  ## 5412. to  8233. --> High
  ## 8233. to 142256. --> Very High
  for col in boundaries:
    print(col)
    print(boundaries[col])

  soh_boundaries = [   0.,     68.25,   84.25,   98.05,  100.   ]
  labelled_soh = []
  #count = 0
  for val in filtered_data['soh']:
    if soh_boundaries[1] > val >= soh_boundaries[0]:
      labelled_soh.append(0)
    elif soh_boundaries[2] > val:
      labelled_soh.append(1)
    elif soh_boundaries[3] > val:
      labelled_soh.append(2)
    else:
      labelled_soh.append(3)
    #count += 1
  filtered_data.insert(loc=len(filtered_data.columns), column='new_soh', value=labelled_soh)
  filtered_data.head()

  filtered_data = filtered_data.drop(columns=['soh'])
  filtered_data.head()

  print(filtered_data['new_soh'].value_counts(normalize=True))

  d = {0 : 'Low', 1 : 'Medium', 2 : 'High', 3 : 'Very_High'}
  d1 = { 'noBalancing' : 0, 'passiveBalancing' : 1, 'initialValue' : 2}
  d2 = {'idle' : 0, 'activeHeating' : 1, 'passingCooling' : 2, 'thermalBalancing' : 3,
        'initialValue' : 4, 'activeCooling' : 5 }
  for c in float_cols.columns:
    filtered_data[c] = filtered_data[c].map(d)
  for c in int_cols.columns:
    filtered_data[c] = filtered_data[c].map(d)
  #filtered_data['thermal_manager_mode'] = filtered_data['thermal_manager_mode'].map(d2)
  #filtered_data['balancing_status'] = filtered_data['balancing_status'].map(d1)
  filtered_data.head()

  sns.countplot(x='vehicle_speed', hue='soh', data=filtered_data);

  #plt.figure(figsize=(10,5))
  #chart = sns.countplot(
  #     filtered_data[filtered_data['soh'] == 'Medium']['balancing_status'],
  #    palette='Set1'
```

```
#)
#chart.set_xticklabels(chart.get_xticklabels(), rotation=45)
#sns.countplot(
#    filtered_data[filtered_data['soh'] == 'Medium']['balancing_status'], palette='Set1')
sns.countplot(x='balancing_status', hue='soh', data=filtered_data);

sns.countplot(x='fast_charge_count', hue='soh', data=filtered_data);



f, axes = plt.subplots(1, 2, figsize=(14, 6), sharex=True)
ax_1 = sns.countplot(x='traction_current', hue='cycle', data=filtered_data, ax = axes[0]);
ax_2 = sns.countplot(x='traction_current', hue='soh', data=filtered_data, ax = axes[1]);
ax_1.set_title("Traction Current and Cycle")
ax_2.set_title("Traction Current and SOH")

encoded_data = pd.get_dummies(filtered_data)
encoded_data.head()

#filtered_data = filtered_data.drop(columns=['fuse_temp'])
for c in filtered_data.columns:
  print("Column", c)
  print(filtered_data[c].value_counts(normalize='true'))

"""### Creating subsets of the data on the basis of cycle variable"""

sns.countplot(filtered_data['cycle'])

bat_storage_data = filtered_data[bat_data['cycle'] == 'storage']
bat_charge_data = filtered_data[bat_data['cycle'] == 'charge']
bat_drive_data = filtered_data[bat_data['cycle'] == 'drive']
print("storage data dimensions are ", bat_storage_data.shape)
print("charge data dimensions are ", bat_charge_data.shape)
print("drive data dimensions are ", bat_drive_data.shape)

"""#### Encoding the data to apply Association rule Mining"""

encoded_storage = pd.get_dummies(bat_storage_data)
encoded_charge = pd.get_dummies(bat_charge_data)
encoded_drive = pd.get_dummies(bat_drive_data)

encoded_storage.head()

"""### Applying Association Rule Mining"""

from mlxtend.frequent_patterns import apriori, association_rules

#association_rules = apriori(filtered_data, min_support=0.005)
#association_results = list(association_rules)


arm = apriori(encoded_data,min_support = 0.6, use_colnames = True)

assoc_rules = association_rules(arm, metric ="confidence", min_threshold = 0.8)
assoc_rules = assoc_rules.sort_values(['confidence', 'lift'])

arm_storage = apriori(encoded_storage, min_support = 0.6, use_colnames = True)

assoc_rules_storage = association_rules(arm_storage, metric ="confidence", min_threshold =
```

```
0.8)
assoc_rules_storage = assoc_rules_storage.sort_values(['confidence', 'lift'])

arm_charge = apriori(encoded_charge, min_support = 0.6, use_colnames = True)

assoc_rules_charge = association_rules(arm_charge, metric ="confidence", min_threshold = 0.8)
assoc_rules_charge = assoc_rules_charge.sort_values(['confidence', 'lift'])

arm_drive = apriori(encoded_drive, min_support = 0.6, use_colnames = True)

assoc_rules_drive = association_rules(arm_drive, metric ="confidence", min_threshold = 0.8)
assoc_rules_drive = assoc_rules_drive.sort_values(['confidence', 'lift'])

#assoc_rules
assoc_rules["antecedent_len"] = assoc_rules["antecedents"].apply(lambda x: len(x))
assoc_rules[ (assoc_rules['antecedent_len'] >= 3) &
      (assoc_rules['confidence'] > 0.75)  ]

"""There was no rule with State of Health Low or High in the consequents but there were some
rules with soh_Low in the antecedents."""

assoc_rules_charge[(assoc_rules_charge['confidence'] > 0.75) & (assoc_rules_charge['
antecedents'] == {'soh_Medium'}) & (assoc_rules_charge['lift'] != 1.0)]

assoc_rules_drive[(assoc_rules_drive['confidence'] > 0.75) & (assoc_rules_drive['antecedents'
] == {'soh_Medium'}) & (assoc_rules_drive['lift'] != 1.0)]

assoc_rules_storage[(assoc_rules_storage['confidence'] > 0.75) & (assoc_rules_storage['
antecedents'] == {'soh_Medium'}) & (assoc_rules_storage['lift'] != 1.0)]

"""### ARM on the complete Encoded Dataset"""

#assoc_rules["antecedent_len"] = assoc_rules["antecedents"].apply(lambda x: len(x))
assoc_rules[(assoc_rules['confidence'] > 0.75) & (assoc_rules['antecedents'] == {'soh_Low'})]

"""### Further subsets of each of the cycle - W.R.T State of Health"""

encoded_storage_sh = encoded_storage[encoded_storage['soh_Medium'] == 1]
encoded_storage_sl = encoded_storage[encoded_storage['soh_Low'] == 1]

arm_storage_high = apriori(encoded_storage_sh, min_support = 0.6, use_colnames = True)

assoc_rules_storage_high = association_rules(arm_storage_high, metric ="confidence",
min_threshold = 0.8)
assoc_rules_storage_high = assoc_rules_storage_high.sort_values(['confidence', 'lift'])

arm_storage_low = apriori(encoded_storage_sl, min_support = 0.6, use_colnames = True)

assoc_rules_storage_low = association_rules(arm_storage_low, metric ="confidence",
min_threshold = 0.8)
assoc_rules_storage_low = assoc_rules_storage_low.sort_values(['confidence', 'lift'])

"""The following rules are extracted from the storage cycle dataset of the complete data.
Support, Confidence, and Lift are three of the most important features of the rules which
explain how strong/valid the rule is for the applied dataset.
Also, Antecedends and Consequents are the features of the data. The rule should be read in
the following way:
For this dataset, occurance of "Antecedents" results "Consequents" with this support,
confidence and Lift.
```

```
E.g. the first rule would be read as:

**For the subset of data with cycle as Storage, Whenever "Vehile Speed is Low" the State of Health is Medium. **
"""

assoc_rules_storage_high[(assoc_rules_storage_high['confidence'] > 0.75) & (assoc_rules_storage_high['consequents'] == {'soh_Medium'})]

assoc_rules_storage_low[(assoc_rules_storage_low['confidence'] > 0.75) & (assoc_rules_storage_low['consequents'] == {'soh_Low'})]

encoded_drive_sh = encoded_drive[encoded_drive['soh_Medium'] == 1]
encoded_drive_sl = encoded_drive[encoded_drive['soh_Low'] == 1]

encoded_charge_sh = encoded_charge[encoded_charge['soh_Medium'] == 1]
encoded_charge_sl = encoded_charge[encoded_charge['soh_Low'] == 1]

arm_drive_high = apriori(encoded_drive_sh, min_support = 0.6, use_colnames = True)

assoc_rules_drive_high = association_rules(arm_drive_high, metric ="confidence", min_threshold = 0.8)
assoc_rules_drive_high = assoc_rules_drive_high.sort_values(['confidence', 'lift'])

arm_drive_low = apriori(encoded_drive_sl, min_support = 0.6, use_colnames = True)

assoc_rules_drive_low = association_rules(arm_drive_low, metric ="confidence", min_threshold = 0.8)
assoc_rules_drive_low = assoc_rules_drive_low.sort_values(['confidence', 'lift'])

arm_charge_high = apriori(encoded_charge_sh, min_support = 0.6, use_colnames = True)

assoc_rules_charge_high = association_rules(arm_charge_high, metric ="confidence", min_threshold = 0.8)
assoc_rules_charge_high = assoc_rules_charge_high.sort_values(['confidence', 'lift'])

arm_charge_low = apriori(encoded_charge_sl, min_support = 0.6, use_colnames = True)

assoc_rules_charge_low = association_rules(arm_charge_low, metric ="confidence", min_threshold = 0.8)
assoc_rules_charge_low = assoc_rules_charge_low.sort_values(['confidence', 'lift'])

assoc_rules_drive_high[(assoc_rules_drive_high['confidence'] > 0.75) & (assoc_rules_drive_high['consequents'] == {'soh_Medium'})]

assoc_rules_drive_low[(assoc_rules_drive_low['confidence'] > 0.75) & (assoc_rules_drive_low['consequents'] == {'soh_Low'})]

assoc_rules_charge_high[(assoc_rules_charge_high['confidence'] > 0.75) & (assoc_rules_charge_high['consequents'] == {'soh_Medium'})]

assoc_rules_charge_low[(assoc_rules_charge_low['confidence'] > 0.75) & (assoc_rules_charge_low['consequents'] == {'soh_Low'})]

del bat_data, bat_drive_data, bat_storage_data
bat_data.head()

"""## Applying Predictive Algorithms to predict State of Health after Bayes Binning on the SOH Feature"""
```

```

import statsmodels.api as sm

y = filtered_data['new_soh']
X = filtered_data.copy()
X = X.drop(columns=['new_soh'])


d3 = {'charge': 0, 'drive' : 1, 'storage' : 2}
d4 = {False: 0, True: 1}

X['cycle'] = X['cycle'].map(d3)
X['fast_charge'] = X['fast_charge'].map(d4)

d1 = { 'noBalancing' : 0, 'passiveBalancing' : 1, 'initialValue' : 2}
d2 = {'idle' : 0, 'activeHeating' : 1, 'passingCooling' : 2, 'thermalBalancing' : 3,
      'initialValue' : 4, 'activeCooling' : 5 }


X['thermal_manager_mode'] = X['thermal_manager_mode'].map(d2)
X['balancing_status'] = X['balancing_status'].map(d1)

from sklearn.model_selection import train_test_split
x_tr, x_tt, y_tr, y_tt = train_test_split(X, y, test_size=0.25, random_state=0)

from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression()
logisticRegr.fit(x_tr, y_tr)
predictions = logisticRegr.predict(x_tt)

"""### Evaluating the model"""

score = logisticRegr.score(x_tt, y_tt)
print(score)

from sklearn import metrics
cm = metrics.confusion_matrix(y_tt, predictions)
print(cm)

import itertools
plt.imshow(cm,cmap=plt.cm.Blues,interpolation='nearest')
plt.colorbar()
plt.title('Confusion Matrix without Normalization')
plt.xlabel('Predicted')
plt.ylabel('Actual')
tick_marks = np.arange(len(set(y_tt))) # length of classes
class_labels = ['0','1','2','3']
tick_marks
plt.xticks(tick_marks,class_labels)
plt.yticks(tick_marks,class_labels)
# plotting text value inside cells
thresh = cm.max() / 4.
for i,j in itertools.product(range(cm.shape[0]),range(cm.shape[1])):
    plt.text(j,i,format(cm[i,j],'d'),horizontalalignment='center',color='white' if cm[i,j] >
thresh else 'black')
plt.show();

acc = metrics.accuracy_score(y_tt, predictions)
recall = metrics.recall_score(y_tt, predictions, average='weighted')
```

```
f1 = metrics.f1_score(y_tt, predictions, average='weighted')
prec = metrics.precision_score(y_tt, predictions, average='weighted')
print("Following are the details of model evaluations:")
print("Accuracy is", acc*100)
print("Recall is", recall*100)
print("F1-Score is", f1*100)
print("Precision is", prec*100)

import dill
dill.dump_session('phase-II-ml-copy.db')



"""## Prediction"""

import statsmodels.api as sm

y = filtered_data['soh']
X = filtered_data.copy()
X = X.drop(columns=['soh'])


d3 = {'charge': 0, 'drive' : 1, 'storage' : 2}
d4 = {False: 0, True: 1}

X['cycle'] = X['cycle'].map(d3)
X['fast_charge'] = X['fast_charge'].map(d4)

d1 = { 'noBalancing' : 0, 'passiveBalancing' : 1, 'initialValue' : 2}
d2 = {'idle' : 0, 'activeHeating' : 1, 'passingCooling' : 2, 'thermalBalancing' : 3,
      'initialValue' : 4, 'activeCooling' : 5 }


X['thermal_manager_mode'] = X['thermal_manager_mode'].map(d2)
X['balancing_status'] = X['balancing_status'].map(d1)

from sklearn.model_selection import train_test_split
x_tr, x_tt, y_tr, y_tt = train_test_split(X, y, test_size=0.25, random_state=0)

from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression()
logisticRegr.fit(x_tr, y_tr)
predictions = logisticRegr.predict(x_tt)

score = logisticRegr.score(x_tt, y_tt)
print(score)

from sklearn import metrics
cm = metrics.confusion_matrix(y_tt, predictions)
print(cm)

import itertools
plt.imshow(cm,cmap=plt.cm.Blues,interpolation='nearest')
plt.colorbar()
plt.title('Confusion Matrix without Normalization')
plt.xlabel('Predicted')
plt.ylabel('Actual')
tick_marks = np.arange(len(set(y_tt))) # length of classes
class_labels = ['0','1']
```

```
    tick_marks
    plt.xticks(tick_marks, class_labels)
    plt.yticks(tick_marks, class_labels)
    # plotting text value inside cells
    thresh = cm.max() / 2.
    for i,j in itertools.product(range(cm.shape[0]),range(cm.shape[1])):
        plt.text(j,i,format(cm[i,j],'d'),horizontalalignment='center',color='white' if cm[i,j] >
    thresh else 'black')
    plt.show();

    acc = metrics.accuracy_score(y_tt, predictions)
    recall = metrics.recall_score(y_tt, predictions)
    f1 = metrics.f1_score(y_tt, predictions)
    prec = metrics.precision_score(y_tt, predictions)
    print("Following are the details of model evaluations:")
    print("Accuracy is", acc*100)
    print("Recall is", recall*100)
    print("F1-Score is", f1*100)
    print("Precision is", prec*100)
```