Electric Vehicle Battery Status Estimation

A Project Report
Presented to
The Faculty of the Department of Applied Data Science
San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Data Analytics

By
Arick Althoff
Xueliu Fan
Joanna Kuo
Yiting Li

May 20, 2022

APPROVED FOR DEPARTMENT OF APPLIED DATA SCIENCE

_____

Dr. Eduardo Chan, Project Advisor

_____

Dr. Lee C. Chang, Department Chair

ABSTRACT

Electric Vehicle Battery Status Estimation

By

Arick Althoff, Xueliu Fan, Joanna Kuo, Yiting Li

Purposes

The lithium ion battery was developed in the 1970s and had remarkable progress in the 1980s, but it was not widely used in vehicles until the 1990s. The Li-ion battery market has been driven by the high demand for electric vehicles, fast iteration of new plug-in electric vehicles models, and concerns over environmental issues from using conventional vehicles. As the core component in electric vehicles, batteries directly correlate with a vehicle's performance such as the driving range and safety. The users' range anxiety still exists despite the rising number of electric vehicle users and the continuous innovation of battery technology. The main concern is the accuracy of battery status shown to users on the screen inside the vehicle. Often users worry about the battery being depleted before reaching the intended destination despite seeing that there should be sufficient charge remaining on the battery. Therefore, it is crucial to accurately estimate the battery percentage, remaining mileage available when driving and the plug-in duration when charging.

Tasks

In this project, novel machine learning approaches are proposed to predict the battery capacity which is used to derive the battery state of charge, plug-in duration, and remaining mileage. Five models are designed and optimized on the battery data from the NASA Prognostics Center of Excellence including XGBoost, support vector regression (SVR), k-nearest neighbor (KNN), long short-term memory (LSTM) with time-series transformation, and multilayer perceptron (MLP). Statistical model SARMIX (seasonal auto-regressive integrated moving average with exogenous factors) is applied as a benchmark to compare with the machine learning models since it is able to do time-series forecasting for battery status prediction. Features measured directly relating to the battery such as multiple loading profiles, energy consumption speed, voltage dropped speed, and current change speed as well as environment data such as temperatures are used to simulate driving scenarios. The models are trained, evaluated and compared based on performance on the validation dataset before the three most optimal models are tested on the test dataset. Model performances are compared and chosen based on low error metrics, specifically MAE (mean absolute error) and RMSE (root mean squared error) and high R^2 coefficient values. The model with the best metrics after predicting on the test dataset will be used to output real time predictions on remaining mileage, plug-in duration and battery percentage based on input entered by users on a web application.

Outcomes

The project begins with transforming the electric vehicle lithium ion battery dataset which is split for training and validating six prototype models and testing three optimal models before selecting the best model for the final product which is an interactive web application that displays remaining mileage, plug-in duration and battery percentage predictions for both driving and charging conditions. In addition, presentation slides, demo videos, progress reports and drafts of the final report in the form of workbooks are completed to thoroughly document the project.

The results demonstrate that XGBoost, LSTM, and MLP models achieved excellent performances with low error and high efficiency. Among them, LSTM with time-series transformation has the best performance on both sequential prediction and single-cycle prediction with low error metrics including a MAE of 0.0331, RMSE of 0.0432 and high R^2 score of 0.9860.

Applications

The application is delivered by defining user cases, designing user interfaces and functions, preparing, coding, testing, reviewing, and launchment. One application is the dashboard that displays the prediction results using cloud platforms so that users can interact with the forecast data. The other one is the web application that provides users with real-time information on battery states. Both applications can help users to overcome range anxiety by accurate battery state prediction. Moreover, the potential new data and new elements are considered to adapt to the rising numbers of electric vehicles on the road, the progressive self-driving technology, and the innovation in battery industries.

ACKNOWLEDGMENTS

We would like to express our sincere gratitude to our advisor Dr. Eduardo Chan for his knowledgeable support, encouragement, and guidance throughout the entire thesis project. We would also like to thank Dr. Lee C. Chang as well as all other staff in the Department of Applied Data Science for organizing a comprehensive program.

We would like to thank Ms. Dawn McIntosh from NASA Prognostics Center of Excellence for providing the open source battery cell data used to complete this project as well as Matthias Steinsträter from Technical University of Munich for providing battery pack data used in the initial first half of our project.

Table of Contents

1. Introduction

1.1 Project Background and Executive Summary

Electric vehicles (EVs) have developed for decades and have become a common form of transportation. Electric vehicles have unbeatable benefits compared to gas-powered vehicles such as high-speed acceleration, almost zero noise, and few emissions (IEA, 2021). In the meantime, the performances are continuously improving with a breakthrough to the innovation of renewable energy storage and conversion technology (Xu et al., 2020).

There is always an amount of uncertainty when operating a battery-powered device, but a motorized vehicle has little room for that uncertainty. For over a hundred years, humans have been testing and operating gas-powered vehicles, but electric vehicles are still new and developing. Even though the electric vehicles markets expanded, there are a lot of negative reactions towards electric vehicle ownership. The top reason that people are reluctant to adopt electric vehicles is range anxiety (Pevec et al., 2020). On one hand, the drivers are worried that the electric vehicles will be unable to drive farther distances that are out of the maximum mileage range the powers can provide. They will become frustrated if they cannot arrive at the charging station. On the other hand, the drivers would doubt the accuracy of the fleet analysis, which makes drivers fear becoming stranded (Sautermeister et al., 2017). Sautermeister et al. investigated prediction estimate uncertainties to find that they require a safety margin of 12% to 23%. They also found “that uncertainty more than doubles over lifetime of the vehicle” (Sautermeister et al., 2017, p. 2624)and that certain driving conditions could ultimately lead to a stranded electric vehicle. Given that the real-time prediction of mileage range is costly and uncertain, an accurate estimation of battery performances in electric vehicles is challenging.

Rechargeable batteries are the core components to power electric vehicles. Most of them are lithium-ion batteries (LIBs) (IEA, 2021). LIBs are the most efficient that can provide power quickly and store large amounts of energy. LIBs are packed into several modules and connected in the series, parallelled, or combined ways to offer the vehicle powers. In general, hundreds or thousands of LIBs are installed per electric vehicle. Battery management systems (BMS) are developed to monitor the batteries (Ali et al., 2019). BMS in the vehicles worked as an electric and thermal management system to maximize the battery lifespan and performance and enhance security.

The main functions of BMS include: monitor the batteries' voltage and current to avoid overcharging or over-discharging, monitor the temperature of batteries to avoid thermal runaway or explosion, manage the batteries' working condition via electric signal communication, diagnose and detect fault, and estimate the batteries' remaining amount of energy in the way of displaying the rest mileage range on the interface (Gabbar et al., 2021). Despite the technology improvement in BMS, the uncertainty of the rest mileage range remains due to the batteries degradation and instability under various environments. Commonly the amount of energy left is referred to as the state of charge (SoC), and the battery degradation is referred to as the state of health (SoH) and the remaining useful life (RUL). It is vital to make an accurate online and off-line battery state estimation in electric vehicles.

State of charge (SoC) is defined by a ratio of remaining capacity to the available capacity of battery while state of health (SoH) is defined as the percentage of current nominal capacity to the nominal capacity of the fresh cell (Zhang & Fan, 2020). The mathematical representation of the SOC and SOH are shown in equation 1.1 and equation 1.2.

$$SOC = \frac{Q}{Q_r} * 100\% \quad (1.1)$$

$$SOH = \frac{Q_r}{Q_{max}} * 100\% \quad (1.2)$$

where Q, Q_r, and Q_{max} are the residual capacity, the nominal capacity at current state, and the nominal capacity respectively.

Several approaches have been reported to estimate the battery status in electric vehicles. Generally, they can be categorized into three types: 1) physical properties based measurements such as coulomb counting, open circuit voltage, electrochemical impedance spectroscopy, 2) the statistic derivation, such as kalman filter and particle filter, 3) data-driven methods mainly including machine learning and deep learning (Hannan et al., 2017). Most of these methods are analyzed in the laboratory without considering the driving condition. The derived outputs are difficult to be applied in the real journey, which lead to less accurate predictions and increase the uncertainty.

In this project, we deploy five machine learning algorithms to make the estimation of electric vehicles battery capacity. We combine neural networks and data mining to simulate a draining battery under dynamic conditions. The simulated battery management system is built up including the components of driving conditions such as environment conditions and loading profiles to simulate driving condition..

The goal is to make a platform to estimate real-time EV battery status online along with the plug-in duration in parking and the remaining mileage during the trip. Using the data-driven approaches, we can provide the drivers with virtual services before buying an electric vehicle and bring more positive experiences against the range anxiety. The polished model can be used to improve the battery management system in the current electric vehicle industry.

1.2 Project Requirements

We aim to provide a platform for electric vehicle battery capacity prediction by utilization of machine learning. There are functional and artificial intelligence requirements in this project.

1.2.1 Data Requirements

The most ideal data for this application should be collected from real life driving trips, specifically from electric vehicles powered by lithium ion batteries. The most essential features to complete for this attempt would be electrical based ones such as voltage, current, and capacity. However to implement a more dynamic and realistic solution, features such as environment conditions and battery operation conditions should be included. Furthermore, the dataset should be as diverse and large as possible to not only achieve accurate predictions but to ensure that the implementation can apply to all electric vehicles.

Unfortunately, most electric vehicle data is confidential from competitors and it may be challenging to find real-trip data. In that case, the bare minimum amount of data required are voltage, current, time, and capacity. The number of cycles or another time-based feature is used to keep track of changes in battery status which will be predicted with time series forecasting. Finally, to implement dynamic estimations, the environment temperature should also be collected. The battery consistency is monitored and controlled by the battery management system so that the electric vehicle battery status can be represented by single battery cell states.

1.2.2 Model Requirements

The data from electric vehicle battery operation are trained by models and objective functions are created for model optimization, such as kernel functions for capturing the input feature properties, loss functions and maximum-likelihood functions for iteration convergence,

and the error functions for model evaluation. Models must be able to work with nonlinear data as batteries do not perform linearly and also must make accurate predictions so users can reach charging stations before running out of battery. Since vehicle and environmental factors influence how a battery is drained, models must efficiently train or predict on many features.

Both machine learning and deep learning models will be tested as some may excel in time series prediction, while others may be better at predictions by reducing error. Finally, a wide variety of the types of models should be considered because each machine learning model has a different approach to each problem. Each type may or may not estimate numerical values, but they should still be considered for a comparative analysis. The types of machine learning models considered are tree-based, distance-based, and neural networks.

1.2.3 Prediction Requirements

Depending on the data, SoC may not be the primary estimated value. But, it can be derived from making predictions on capacity. For a successful attempt, the machine learning model should have a low prediction time while having the highest possible accuracy. There are other considerations to think about as well, such as, its ability to predict while the battery loading profiles are changed. In those real-life situations, a driver must decide on how far they can drive before they charge and if a model overestimates the battery percentage, the driver may become stranded.

The prediction quality is controlled and evaluated by the statistical estimator measurements tools. The results should be interpretable. The final application is reserved for the users through the battery percentage, remaining charging time, and remaining mileage. The prediction is used to remind users of the remaining amount of battery but it is up to the user to charge the electric vehicles in time.

1.3 Project Deliverables

This project spans from August 2021 to May 2022. Throughout the project, progress checks will be made in the form of workbooks, progress reports, and a formal middle presentation. These checks are to be submitted and verified by the graduate advisor. In the second half of the project, demo videos of the web application are included in demo presentations. The final deliverables include the successful deployment and implementation of an online web-based system, final report, and final presentation.

Table 1.1

Project Deliverables Schedule

Deliverable Name	**Due Date**	**Description**
Project Abstract (Draft)	9/17/2021	An abstract and short summary of the entire project's purpose, goals, and expected outcomes. Will be updated after the completion of the project (May 2022) with specific project results.
Progress Report #1	10/1/2021	A summary of the first month of project progress quantified by approximating the percentage of completion for each section of the report. Detailed notes on achievements, issues encountered and solutions taken.
Workbook #1	10/8/2021	A first draft of the Chapter 1: Introduction and Chapter 2: Data and Project Management Plan sections of the report.
Transformed Dataset	11/4/2021	Generate new features from the charge and discharge datasets and complete feature extraction.
Progress Report #2	11/12/2021	Documenting progress since Progress Report #1 in the same format as previous progress reports.
Workbook #2	11/24/2021	A first draft of the Data Engineering and Model Development chapters of the report after the dataset has been cleaned and transformed and model research is complete.

Deliverable Name	Due Date	Description
Mid-Project Presentation	12/07/2021	A presentation on the progress of the first half of the project which covers the first four chapters of the project.
Mid-Report	12/13/2021	A complete mid-progress report on the first four chapters of the project.
Project Funding Proposal	2/27/2022	Itemized plan of required resources and the corresponding costs.
Workbook #3	2/27/2022	A first draft of Chapter 5 Data Analytics System and updates to Chapter 4 Model Development.
Prototype Models	2/20/2022	Initial prototypes of models predicting battery capacity based on model proposals.
Project Demo #1	3/13/2022	Present prototype models and metrics.
Final Models	4/1/2022	Final battery capacity estimation models with optimized parameters tuned for the best performance based on evaluation metrics.
Workbook #4	4/12/2022	A first draft of Chapter 6 System Evaluation and Visualization and Appendices as well as updates to Chapter 4 Model Development.
Project Demo #2	4/24/2022	Present model updates and first prototype of web application.
Successful Deployment of Web-based application	5/17/2022	The successful deployment of a website to interact with users under multiple user cases.
Final Presentation	5/18/2022	Final comprehensive presentation and demonstration of the final version of the web application.
Final Report	5/20/2022	Final report detailing all the work accomplished during the project.

1.4 Technology and Solution Survey

The most common approaches for estimating the battery states are state of charge (SoC), state of health (SoH), and remaining useful life (RUL). Some research involves battery capacity prediction that can derive SoC, SoH, and RUL.

Conventional methods are to make direct experiment-based estimation from measurable values such as coulomb counting, open circuit voltage (OCV), and electrochemical impedance spectroscopy (EIS) (Hannan et al., 2017; Snihir et al., 2006; Densmore & Hanif, 2015). The coulomb counting method integrates the current over time as the battery is charging and discharging (Hannan et al., 2017). It is a fast way to obtain the battery state of charge from collecting current data through the equipment sensor and comparing the charge going in vs the charge going out but measured cell current is not the same as true cell current because the measured cell current also includes sensor noise which can lead to inaccuracy. Coulomb counting is best used for short time periods and when initial conditions are known because this method has a disadvantage of accumulating error over time due to sensor noise. A solution to this problem is to constantly reset a new initial condition but this is complex which means it might be too slow for constant real time battery state of charge estimations.

OCV is also a simple method to estimate SoC where the measurable terminal voltage is used in place of OCV which cannot be directly measured in the OCV vs SoC function. It tests the open circuit voltage when the battery has finished cycling and reaches the balanced state (Snihir et al., 2006). The disadvantage for OCV is the low efficiency and the uncertainty in the OCV-SoC relationship because terminal voltage is not the same as OCV but it is measurable. Overall, OCV is a simple method to implement but performs poorly since it does not account for ohmic losses or hysteresis voltage and also because small changes in voltage can actually have

large changes in SoC which makes this voltage-based SoC prediction method unreliable. There is also an improved version of OCV that adds the product of ohmic drop and resistance to the terminal voltage but this would still neglect hysteresis while adding more time between SoC predictions as a tradeoff for slightly better but still inconsistent SoC predictions. Besides the measurements of current and voltage, the internal resistances of the battery can also be applied to estimate SoC. It is always used to explore the electrochemical mechanism inside the battery. By measuring the impedance under the different frequencies, the capacity of the battery can be derived (Densmore & Hanif, 2015). The impedance results are complex and not as interpretable as other methods on EV batteries.

As the Li-ion battery is charging and discharging repeatedly, the battery can be treated as a dynamic system since the current state of the battery can be affected by the previous cycle. Thus, the implementation of filter algorithms can improve the estimation of the battery state (Hannan et al., 2017). The researchers develop a branch of statistical model based methods to estimate SoC. One of the common filters called Kalman filter (KF) is a recursive filter that estimates the current state based on the previous state over time-scale (Shrivastava et al., 2019). It is widely used for the uncertain parameters by calculating and minimizing the error until convergence. Various battery cell dynamic models are established to explain the nonlinear behavior of battery charging and discharging such as extended Kalman filter (EKF) for the first and second order models and unscented Kalman filter (UKF) for high order models (Shrivastava et al., 2019; Hu et al., 2012). Using the KF algorithm, the battery state can be estimated accurately as it is adapting as new data comes in but the drawback is that this method has a high computational cost.

Considering the low efficiency in experimental measurements and the complex calculations in the statistical models, researchers often use machine learning (ML) and deep learning (DL) models which can use large amounts of battery cycling data to predict the battery capacity (Vidal et al., 2020). ML and DL approaches have powerful learning capacity which means faster SoC estimations and also require less knowledge of the battery internal characteristics which means it can be used for various battery systems instead of being limited to a specific kind. SoC estimation can be considered as a time series prediction where previous data is used to forecast the future battery states. ML models involving regression such as Gaussian Process Regression of Support Vector Regression or DL models such as sequence-to-sequence recurrent neural network (RNN) which takes in a sequence of previous data and then outputs a sequence of predictions or long short term memory (LSTM) which makes future predictions based on short term and long term data are all able to make time series predictions. Other common issues with battery states estimation include noisy data or unreliable predictions which can be solved with multilayer perceptron which adjusts weights between different neurons to reduce error or autoencoders which focus on finding patterns and are able to denoise data.

1.5 Literature Survey of Existing Research

Many research studies have been conducted on prediction of battery states by using machine learning approaches. For example, Chandran et al have reported using linear regression, Gaussian process regression, ensemble bagging and boosting, support vector machine, as well as artificial neural network to estimate the SoC of Panasonic 18650FP battery cell (Chandran et al., 2021). The optimization of parameters is conducted to improve the performance which is evaluated by the error analysis. The corresponding results demonstrate that Gaussian process regression (GPR) and artificial neural network (ANN) outperform the other algorithms such as support vector machine, linear regression and ensemble methods in case of battery state estimation with lower root mean square errors (RMSE) of 0.00170 for GPR and 0.04118 for ANN while other methods had higher RMSE values over 0.1.

Battery state estimation has also been attempted using deep learning approaches. A paper by Chemali et al. generated real data in a lab environment by applying drive cycles at various temperatures to a single battery and used deep feedforward neural networks for estimates and predictions (Chemali et al., 2018). According to Chemali et al., the benefits from this method is its accuracy and efficiency owing to a multitude of simple calculations. For data collection, they applied nine different drive cycles on Panasonic NCR18650PF with varied ambient temperature between -20° C and 25° C. Chemali and others made SoC estimations at every point of the batteries charge and mean absolute error was calculated at each individual point. At max, an error of 4% was reached, but on average, error remained around 2% (Chemali et al., 2018). This attempt showed that deep feedforward neural networks are highly effective when estimating SoC.

Another attempt by Gruosso et al. combined conventional and machine learning methods achieved a low root-mean-square-error (RMSE) using Support Vector Regressions (SVR), principal component analysis (PCA), and a dual polarization battery model. The collected LiFePo4 battery data contained a total of four routes with different traffic conditions and included speed, acceleration, battery voltage, and current (Gruosso et al., 2020, p. 1). PCA was used to reduce dimensions and pass through the SVRs that utilized the polynomial and gaussian kernels. Finally, the estimated current signal from SVRs is the input for the dual polarization model. This model copies the behavior of batteries and is used to estimate battery state of charge. To minimize chi squared error, a nonlinear least square adaptive algorithm estimates individual parameters of battery state estimation. Gruosso et al. calculated a 4th order regression polynomial that estimated battery state based on the open circuit voltage of the battery. Final root-mean-squared-error did not exceed 0.06 for the gaussian kernel and the polynomial kernel did not exceed 0.13 for each route (Gruosso et al., 2020, p. 13).

Another attempt that utilized neural networks was made by Zahid et al. They used an adaptive neuro-fuzzy inference system known as ANFIS. ANFIS is a powerful artificial neural network estimator that is able to accomplish high accuracy on most estimation problems (Zahid et al., 2018, p. 873). On top of ANFIS, a subtractive clustering approach was used. This method assumes each data point is a potential center of a cluster then calculates the probabilities of each point that is true (p. 875). Data collection for this attempt was done using NREL's Advanced Vehicle Simulator that collected battery data from ten different drive cycles. Each drive cycle continuously ran until the electric vehicle batteries ran out of charge to acquire the entire range of data. Three of the ten cycles were combinations of the others to mimic real world driving conditions. In comparison to back propagation models, the resultant SoC results were much

improved. The model predicted battery state with a maximum absolute estimation error of 0.1% or lower for each of the ten constructed cycles (p. 881).

There has also been research using hybrid models such as combining Extended Kalman Filter (EKF) for estimating battery hysteresis, which is when rest voltage varies depending on if the battery has recently been charging or discharging, with a neural network that uses estimations from EKF to estimate battery state. Zhihang et al. proposed a hybrid EKF and Neural Network model to estimate SoC on both a 12V and 40 Ah UI-12XP lithium-ion battery and 1.2 V and 3.4 Ah nickel metal hydride (NiMH) battery as both methods can be applied to any battery system and is able to used online (Zhihang et al., 2011). The neural network model input parameters include voltage, current, temperature, and estimation of previous battery state from EKF and outputs the current battery state estimation. Four different EKF methods were tested and the method that used a split OCV curve, which splits charge and discharge into two separate curves, resulted in the lowest RMSE for SoC at 2.0428% (Zhihang et al., 2011, p. 2161). Using only a neural network without any previous SoC estimations from EKF resulted in a less accurate model with a higher RMSE of 5.02% compared to using only the split curve EKF but combining both the split curve EKF and a neural network together resulted in a RMSE of 0.89% which was concluded as the most optimal model for estimating battery state (Zhihang et al., 2011, p. 2162). Charkhgard et al. also proposed a hybrid model combining both EKF and a radial basis function (RBF) neural network to estimate SoC of a 1.2-Ah lithium ion battery. However, results from this experiment may not be applicable to estimating battery state of real life electrical vehicle trips as this was done at room temperature which eliminates outside temperature as a factor influencing EV battery state (Charkhgard et al., 2010). The input parameters of the proposed RBF neural network include previous battery voltage, estimated SoC and battery current and outputs the

estimated current battery terminal voltage (Charkhgard et al., 2010). After training the neural network with a subset of the full dataset, the model is used with EKF and resulted in a RMSE of 2% when tested offline and RMSE of 3% when tested online (Charkhgard et al., 2010, p. 4183 - 4184).

Since the battery nominal capacity fades over time, the time-series-based models are employed for estimation. Dong et al proposed a neural network algorithm named recurrent nonlinear autoregressive with exogenous inputs (RNARX) (Hannan et al., 2020). The data under constant discharge test and hybrid pulse power characterization test are collected for estimation. The hyperparameters are optimized by getting the minimized value of objective functions. The model shows the generalization capability and adaptability and the estimation has high accuracy under various environmental conditions.

Despite progress in the development of ML and DL models, it is limited due to estimation being highly dependent on the quality of the data. The disturbing and the noise in the data would affect the estimation performance and lead to overfitting. Also the confined testing environment conditions would limit the applicability of battery state estimation on the fleet analysis in electric vehicle manufactures, especially the lack of consideration of various factors like engine temperature, regenerative braking, altitude, and many other non-laboratory setting variables.

In this project, we propose using machine learning models to estimate the electric vehicle battery capacity, plug-in duration, and remaining mileage. The model is designed to capture the batteries' characteristics. The battery behavior will be interpreted and the EV battery environment will be simulated under different driving conditions.

2. Data and Project Management Plan

2.1 Data Management Plan

The data management in this project includes local management and data on cloud management. Datasets are first collected from the real trip. After data preparation, datasets are compared and selected for the model implementation. These steps are implemented in a personal computer environment. For simulating a real time BMS, data is then stored onto Google Cloud Platform (GCP). On GCP, streaming data ingestion, data preparation, data exploration, and modeling are performed. Finally, the web application is connected to GCP for visualization in order to create an interactive BMS dashboard.

2.1.1 Data Collection

In this project, data is collected from public resources about the electric vehicle battery. The most ideal dataset for this project would include the battery and vehicle data covering voltage, current, temperature, discharging profile as well as external environment conditions. Data with complete charging and discharging are the most suitable option as they can gather all the battery behavior including the battery for electric vehicle usage.

2.1.2 Management Methods

The local data management here is defined as a process to find suitable datasets to carry out research. The first step is to search and choose datasets. There are some criteria to select the datasets: data size, data quality, features are considered in data selection. For example, the data with instances smaller than 100 cannot be considered. The second step is local data storage. The

selected datasets are stored both locally and on cloud as a backup. The third step is to explore deeper into the datasets: compare size, features and data quality to decide which dataset to use. The data management on cloud is done on Google Cloud Platform with data being stored in Google Cloud Storage and BigQuery. Finally, the web application is connected to BigQuery as the data source for predictions.

2.1.3 Storage Methods

Datasets are stored locally on personal computers and also on Google Drive as a backup. On GCP, datasets are first stored on Google Cloud Storage. Then, Google Cloud BigQuery is used for storing the cleaned and transformed datasets which will be used for streaming data during model training.

2.1.4 Usage Mechanisms

To evaluate the usage, features and data quality in datasets are examined. The tools used here are Python within the Jupyter notebook. Descriptive features and target features are listed, and distributions of features are visualized which are then used to analyze and decide which datasets are suitable for SoC estimation. Final decision of datasets relies on the following project development methodology.

2.2 Project Development Methodology

To manage the development and planning of this project, the data industry standard, CRISP-DM will be utilized. CRISP-DM stands for cross industry standard process for data mining (CRISP-DM, n.d.). This method takes into account different aspects of data management, testing, and modeling and is non-linear. It allows for transitioning between its six phases to ensure a well-formed and complete project.

The six phases of CRISP-DM are:

1. Business Understanding - To successfully complete this phase, a complete understanding of the problem, what is required, and how to solve the problem is needed. This phase is one of the most critical phases as an improper understanding of the problem is common and can easily derail a project. The entirety of the first chapter is dedicated to getting a better understanding of what it will take to accomplish utilizing machine learning to construct a BMS.

2. Data Understanding - This phase is about comprehension of what is required to solve the problem from a data standpoint. In terms of implementing a BMS, the data that is collected needs to be able to fulfill the project's requirements and solve the problem. As mentioned above, the CRISP-DM process allows for backtracking between phases. And, if there is no available data that could solve the problem, the business understanding phase could be re-entered to re-evaluate the problem or what it would take to solve the problem.

3. Data Preparation - The data that is collected must be cleaned and analyzed before further steps can be made. Features would need to be derived from raw data in this stage and some derived features could also be created. A thorough understanding of the structured and cleaned data should be established at this step with data summarizing visualizations and statistics as well.

4. Modeling - Not only will multiple machine learning models need to be tested in this phase, but two models may be needed for SOC estimation and duration till a full charge. This step includes thoroughly describing the models that should be considered as well as the models attempted. Results from any successfully trained model can be compared in this stage as well. Often, data errors or new data preparation steps can be detected in this step, so the CRISP-DM method allows its users to backtrack to the previous stage to fix or integrate any changes.

5. Evaluation - This stage tests for accuracy metrics such as accuracy, F2 score, ROC Curves and so on. Furthermore, it compares each of the machine learning models that were sufficient in the modeling stage using these metrics to select the best performing model. If either SOC estimation or duration till a full charge model are not accurate enough to solve the business problem or be more accurate than the Kalman filter, CRISP-DM allows the backtracking to the business understanding phase. In doing so, using the knowledge gained during the past phases, a new outlook or understanding of the problem may yield better results.

6. Deployment - The final product, an online implementation of the battery management system should be deployed in this phase. The web-based application should be operable by users without further management by project members. Although, there is always room for improving the system's accuracy and usability which may require continued upkeep.

Although the CRISP-DM method is the primary project management style that this project will utilize, a "waterfall" project management plan will be used in tandem. This style focuses on a very linear framework where one task must be finished before another is started. Combining these two methods will keep the cyclical nature of CRISP-DM while incorporating flexible deadlines that need to be met before progression onto the next task may start.

2.3 Project Organization Plan

A work breakdown structure was created on top of other organizational techniques to generalize the required tasks for this project as shown in Figure 2.1. This hierarchical format is essential for every waterfall management scheme as it lays out tasks that must be completed before the next task can be started, with some exceptions. There is also the possibility of revisiting important steps such as cleaning/formatting data as needed during the model development process or proposing new models if tested models do not fulfill project requirements.

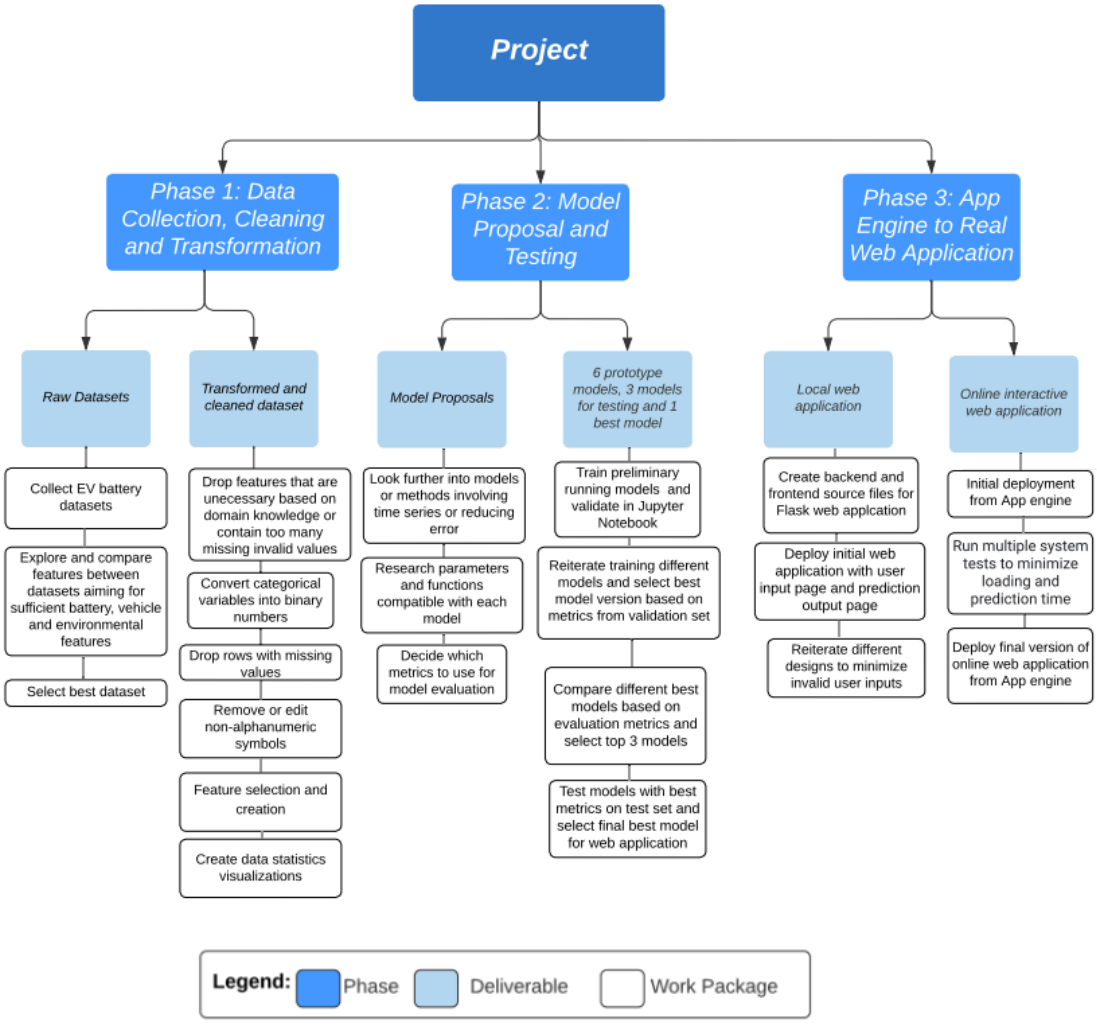
The project is split into three phases starting with Data Collection, Data Cleaning and Transformation as Phase 1. We start by collecting and comparing open source EV battery datasets based on the data quality of battery, vehicle and environmental features before selecting the best dataset and removing missing invalid values, irrelevant features and converting categorical variables into numeric variables using one hot encoding to create the transformed and cleaned dataset.

Phase 2 starts with statistical, machine learning and deep learning model proposals with the focus on models that can handle nonlinear data, time series prediction and reducing error. After proposing models, the evaluation metrics will also be decided. Models will be made in Jupyter notebook where validation and parameter tuning is completed before comparing models to identify the best performing models for SoC estimation which are tested on the test set.

Finally in Phase 3, a web application is developed locally, and then deployed onto Google Cloud Platform.

Figure 2.1

Work Breakdown Structure



2.4 Project Resource Requirements and Plan

The whole project is built locally with personal computers and on cloud with Google Cloud Platform.

For the local environment, the minimum requirement of the CPU is 1.6 GHz Dual core Intel Core i5. The minimum requirement of memory is 8GB 1600 MHz DDR3. The OS environment is macOS Catalina 10.15.4. The collected datasets are stored locally and on Google Drive as backup. For data exploration, Python 3.7 and Anaconda platform are needed. The language used in modeling is Python 3.7. The compiler environment is also Python 3.7. The data preparation and exploration are done in Python within Jupyter Notebook.

The cloud environment is built with Google Cloud Platform. On GCP, both compute engine and app engine pipelines are tried. In compute engine pipeline, GCP Compute engine, Cloud Storage, GCP Cloud Pub/Sub, GCP Cloud dataflow, GCP BigQuery, GCP AI Platform and Data Studio are used. In the web-based application pipeline, GCP App Engine and Flask platform are used. Meantime, model performance comparison is implemented with Data Studio on Google Cloud Platform.

Cost can be separated into two parts: local environment and Google Cloud Platform. The local hardware and software are at personal cost. All the services on Google Cloud Platform are a pay-as-you-go pricing structure[1]. Besides the $300 free credit for new users on GCP within 3 months, the estimated GCP cost based on estimated usage is $1500. In sum, the budget for this project is $1500.

[1] https://cloud.google.com/pricing

Table 2.1

Planned Required Resources Function and Estimated Cost

Function	Resource Type	Resource	Time Duration	Cost
Project implementation on local machine	Hardware	Personal computer	08/22/2021 - 05/31/2022 (10 mos)	Personal cost
Project implementation on local machine	Software	OS environment with macOS Catalina 10.15.4	08/22/2021 - 05/31/2022 (10 mos)	Personal cost
Data preparation & exploration	Software	Anaconda Jupyter Notebook	10/01/2021 - 05/31/2022 (8 mos)	Anaconda is free
Software development	Software	Python 3.7	08/22/2021 - 05/31/2022 (10 mos)	Free
Data storage backup	Hardware	Google Drive	08/22/2021 - 05/31/2022 (10 mos)	Free
Real time demo on cloud service	Google Cloud Platform (GCP)	Google Cloud Platform	01/01/2022 - 05/31/2022 (5 mos)	$1500 based on estimated usages
Compute engine on cloud	Hardware	GCP Compute Engine	01/01/2022 - 05/31/2022 (5 mos)	(Included in GCP cost)
Data storage on cloud	Hardware	GCP Cloud Storage	01/01/2022 - 05/31/2022 (5 mos)	(Included in GCP cost)
Data ingestion on cloud	Software	GCP Cloud Pub/Sub	01/01/2022 - 05/31/2022 (5 mos)	(Included in GCP cost)
Data processing on cloud	Software	GCP Cloud Dataflow	01/01/2022 - 05/31/2022 (5 mos)	(Included in GCP cost)
Data exploration on cloud	Software	GCP BigQuery	01/01/2022 - 05/31/2022 (5 mos)	(Included in GCP cost)
Visualization	Software	GCP Data Studio	01/01/2022 - 05/31/2022 (5 mos)	(Included in GCP cost)
Application	Software	Flask GCP App Engine	01/01/2022 - 05/31/2022 (5 mos)	(Included in GCP cost)

2.5 Project Schedule

This project schedule was designed with the waterfall project management scheme, but some changes were made to account for the multi-tasking ability that can be done by a four person group.

Table 2.2

Team Roles and Responsibilities

Name	**Role**	**Description of Role**
Yiting Li	Project Team Leader	Confirm project resource requirements Finalize cost estimation report Lead agenda for team meetings
Xueliu Fan	Machine learning Expert + Website developer	Lead machine learning research and testing Lead the web-based application design and deployment
Arick Althoff	Data Manager + Google Cloud Platform Expert	Handle feature selection and finalize transformed dataset Assist with issues or difficulties on Google Cloud Platform
Joanna Kuo	Logistics Coordinator + Deep Learning Expert	Finalize progress reports, book meetings and submit assignments Lead deep learning model research and testing

The first Gantt Chart, Figure 2.2 represents the first half of the project completed in 2021 while Figure 2.3, covers the second half of the project that will be completed in 2022. Red is used to label project deliverables with hard deadlines directly related to the thesis report or presentations. Deadline dates listed in Figure 2.3 are estimated based on the timeline of deadlines from the first half of the project and will be updated early 2022 when final deadlines are known. Orange represents tasks that require a prior task to be completed before it can be started. This is representative of the waterfall style as one task will remain a priority until completed. Green tasks can be started after previous tasks have been completed but these are often writing or compilation tasks that are not prerequisites for future tasks.

This project schedule also takes into account the CRISP-DM methodology implemented. During the break or a small extended period of time before the final project is due, CRISP-DM backtracking for any beneficial purposes should be considered.

Figure 2.2

Gantt Chart for DATA 298A in 2021

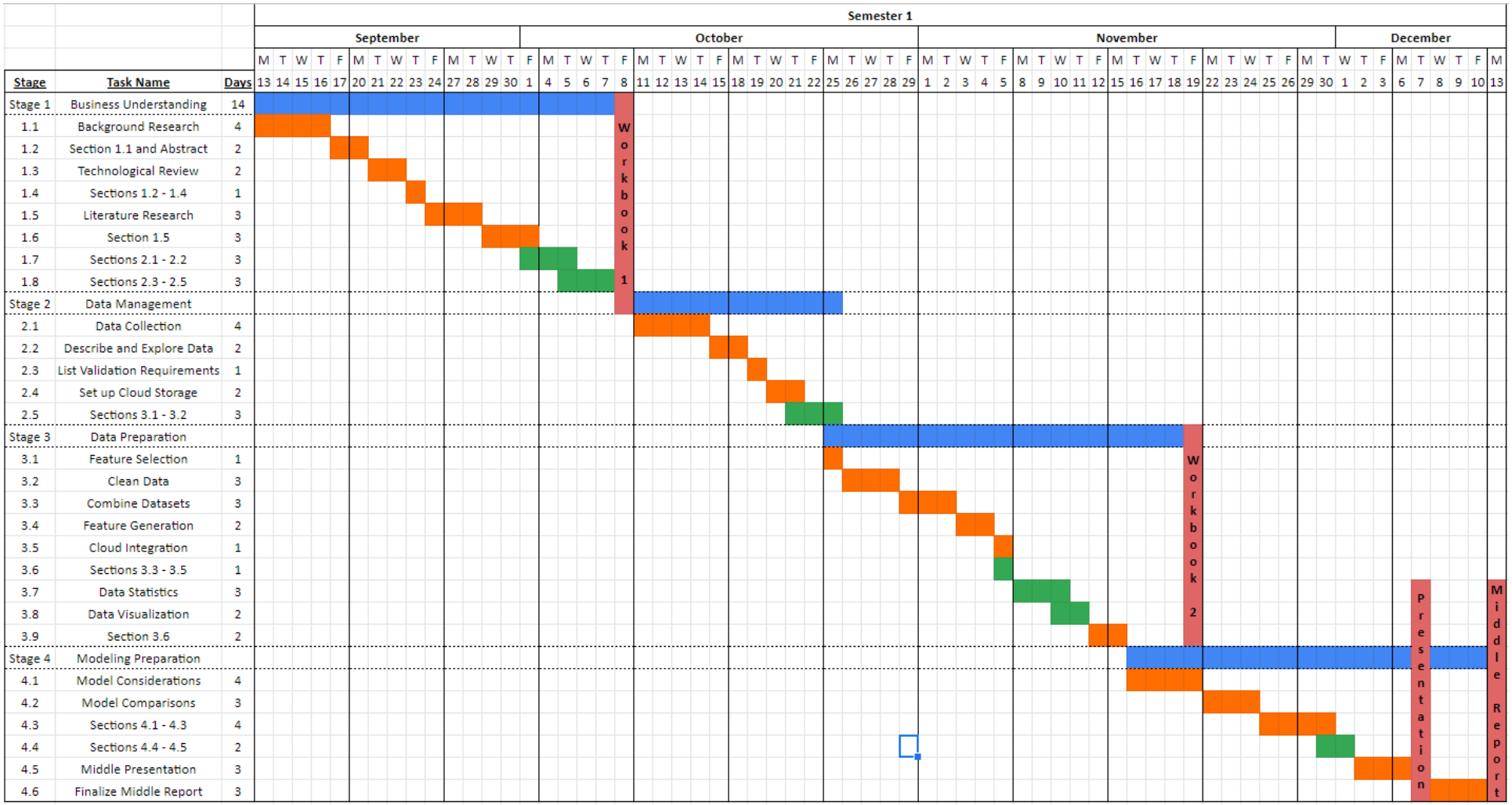


Figure 2.3

Gantt Chart for DATA 298A in 2022

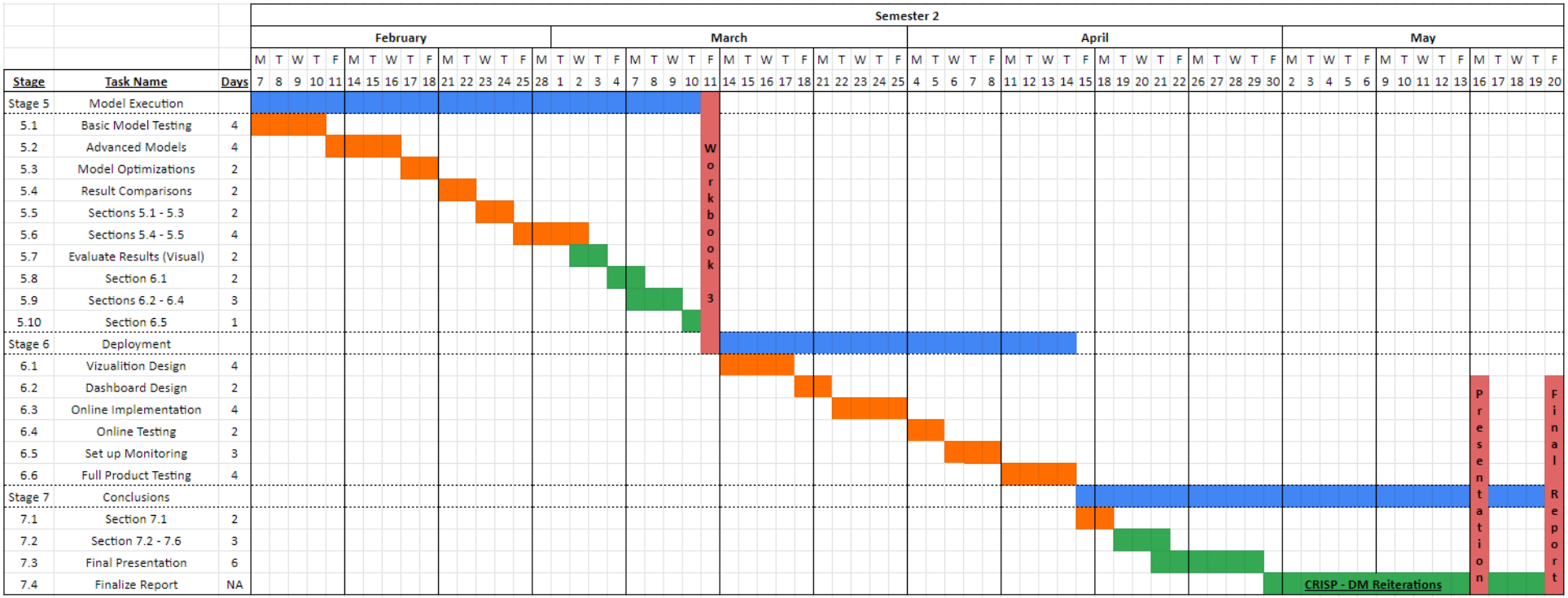


Figure 2.4

PERT Chart for DATA 298A Tasks in 2021

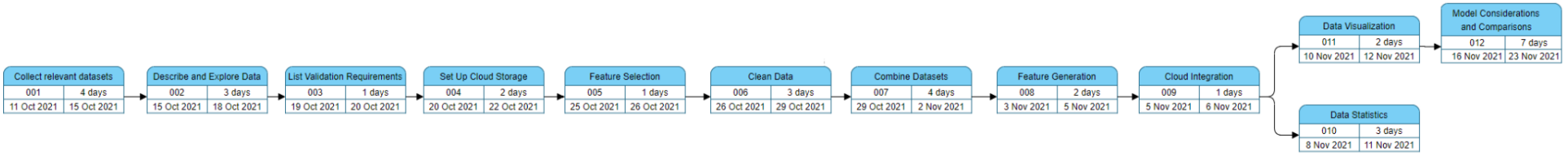
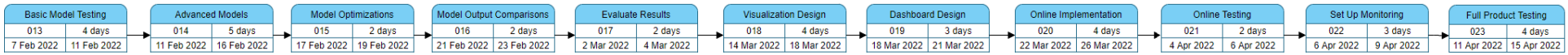


Figure 2.5

PERT Chart for DATA 298B Tasks in 2022



3. Data Engineering

3.1 Data Process

Data processing encompasses data collection, pre-processing, transformation, and preparation. Each of these steps are integral to a well-performing model and final implementation.

First, data must be collected from a source that includes battery and environmental data via battery aging tests. Those tests should be conducted by charging to 100% and discharging to 0% with multiple loading profiles and varied environmental temperatures before electric vehicles are sold to customers. The manufacturers can use the data to provide an accurate prediction of battery capacity, battery SoC, plug-in duration during chagrin, and remaining mileage during driving to the users. Also they have to guarantee that the batteries are operated in a safe way to monitor the suspicious behavior and to prevent the failure of the battery such as thermal runaway or shortcut. Therefore, the Li-ion Battery Aging Datasets from NASA Ames Prognostics Center of Excellence (PCoE) (B. Saha and K. Goebel, 2007) are fetched for this project to predict the battery status.

The dataset extensively recorded raw data collected from 18650 Li-ion batteries under several tests. The behavior of the lithium ion batteries and the performances of the vehicle are reserved as well as the various environmental conditions. This dataset was selected because it would be perfect in estimating capacity using real-world conditions as well as whole battery pack data as it contains both battery and environmental data.

Next, the data pre-processing stage covers the data integration and data cleaning steps. Data integration is required to uniformly store the data as well as create new features or connections that aren't present in the original dataset format. This dataset in particular had stored

each of its tests in separate files, so these needed to be combined into one dataset. Finally, data cleaning is performed to remove any missing values or determine if any features should be removed. If a column has too many missing values or the same value throughout, they likely would not assist in any future model. Rows of data can also be removed if they are missing important sections of data points that would be able to be accurately filled.

The data transformation step is an extremely important step in the data engineering stage. Here, feature generation would take place which is essential when working with time-series data. Unlike conventional datasets, each data point in a time-series is usually reliant on the previous data point and so on. So, to accurately capture this, features must be generated that captures the trends in the data. On top of feature generation, other tasks to transform the data into a model-ready dataset must be taken. Data normalization will allow any model to disregard the scaling of different features and one-hot encoding will transform text-based categorical features into something that is more readable.

Finally, data preparation encompasses splitting the data into training, testing, and validation datasets. There are two purposes for this. First, the validation dataset will allow for iterative testing of the model without it becoming biased towards the testing dataset. In doing so, model parameters and other settings may be optimized. Next, the testing dataset provides a way to generate accuracy metrics of each model. This is perfect for comparative analysis as well as determining general model performance.

3.2 Data Collection

The datasets of the high-voltage lithium-ion batteries of electric vehicles are collected from NASA PCoE, an open data repository. Those data are provided from research groups in NASA Ames Research Center, Moffett Field CA. They collected the charging data by charging the batteries under 1.5A constant current until the battery capacity reaches around 80% and then under constant voltage until 100%. The discharge data are collected under a variety of parameters including the multiple loading profiles from 1A, 2A, and 4A; the environment temperatures changing from 4 degree C, room temperature, to 44 degree C, and the discharge voltage from 2.7V, 2.5V, to 2.2V. In each test, the cycling is carried out until there is 30% fade for the capacity. To estimate the battery status, comprehensive test conditions are selected.

The entire data consists of three groups of test results stored in 10 excel files which cover the electric vehicle battery usage range. Each dataset consists of two parts: the battery charge data and the battery discharge data in which the battery voltage, the battery current, the loading voltage, the loading current, the temperature, the cycle number, the capacity are included. The features in the raw data are listed in Table 3.1.

Table 3.1

List of features from raw dataset

Data	**Description**
Cycle	One cycle contains charge and discharge
Type	Operation type, can be charge, discharge
Current_measured	Battery output current (Amps)
Voltage_measured	Battery terminal voltage (Volts)
Temperature_measured	Battery temperature (degree C)
Current_charge	Current measured at charger (Amps)
Voltage_charge	Voltage measured at charger (Volts)
Time	Time vector for the cycle (secs)

Figure 3.1 shows the sample data from the raw dataset of a single trip. The raw data are presented in the .xlsx files that will be transformed into data frames by Python. The battery measurement records the battery signals with 2.5 second intervals. It captures every change of the battery status as well as the temperature.

It is considered that the electric vehicle is driving under various environmental conditions such as cold weather in winter or hot one in summer. Not only are the battery temperatures provided but the outdoor temperatures are also recorded from 4°C to 44°C. The performance of the battery is significantly affected by the temperatures. The low temperature would limit the battery current and lead to the short mileage range and the high temperature would cause the overheating inside the battery resulting in the dangerous situation.

The loading profile is another feature that can cause fast or slow energy consumption and will be reflected by the battery current. It is defined by the ratio of current to the nominal

capacity and denoted as C. In our case, the maximum nominal capacity for this 18650 battery is around 2000mAh. When the loading profile is 1A, the discharge rate is 0.5C. The loading profile of 2A means 1C, and so on. In real driving, the acceleration is equivalent to the high loading profile in terms of high negative discharge current, vice versa.

At last, the most important and straightforward features to represent the battery status are the battery data, including voltage, current, temperature, and the capacity which is the target. The terminal voltage and current represent the voltage and the current of the battery and the loading voltage and the loading current are the external circuit conditions which are not used in this project.

Figure 3.1

The sample data from the raw dataset of a single test: (a) input, (b) target

	cycle	type	Voltage_measured	Current_measured	Temperature_measured	Current_charge	Voltage_charge	Time
0	1	charge'	3.873017	-0.001201	24.655358	0.000	0.003	0.000
1	1	charge'	3.479394	-4.030268	24.666480	-4.036	1.570	2.532
2	1	charge'	4.000588	1.512731	24.675394	1.500	4.726	5.500
3	1	charge'	4.012395	1.509063	24.693865	1.500	4.742	8.344
4	1	charge'	4.019708	1.511318	24.705069	1.500	4.753	11.125
...	...	...	...	...	...	...	...	...
591453	170	charge'	0.236356	-0.003484	23.372048	0.000	0.003	0.000
591454	170	charge'	0.003365	-0.001496	23.369434	0.000	0.003	2.547
591455	170	charge'	4.985137	0.000506	23.386535	0.000	5.002	5.500
591456	170	charge'	4.984720	0.000442	23.386983	-0.002	5.002	8.312
591457	170	charge'	4.213440	-0.000734	23.385061	-0.002	4.229	12.656

capacity
1.856487
1.846327
1.835349
1.835263
1.834646
...
1.293464
1.288003
1.287453
1.309015
1.325079

(a) (b)

3.3 Data Pre-Processing

This step encompasses data cleaning and integration. Data cleaning is an important step that removes unnecessary columns and empty values. This step is an extremely long and comprehensive section that also analyzes outliers and summary statistics of the data to validate and ensure healthy data. For data integration, the goal is to combine datasets from various datasets into a singular dataset for modeling, feature generation, and summary statistics. This may also be used to create new features on a dataset from other sources.

3.3.1 Data Cleaning

As seen in Figure 3.1, there are symbols in the string fields such as ‘`’ or ‘\’ that make the data inconsistent with each other so that they are replaced with the space. There are missing values in every file marked as ‘NaN’ in the numerical fields of battery voltage, battery current, and temperatures. Those missing values always appear at the bottom of the dataset so that it is hard to use interpolations to fill in. The entire rows are removed.

The abnormal outliers appear in the voltage and the current data due to the unusual sensitivity in the recording system. One single 18650 battery can charge to 4.2V and discharge down to 3V or even 2V based on the cells’ cutoff voltage. It is not safe to overcharge to melt the battery or over-discharge the battery to damage the internal materials. As normal the current should keep positive the charging state and would be negative in the discharging state depending on the discharging status. In our case, the data are removed in which the voltage is higher than 4.2V or lower than 2V, or the current in the charging state is exceptionally negative. The current with negative values in the charging state can be rounded to 0.0000001 if they are close to 0.

Two other features in the dataset are processed for the future processing. One is the time that is recorded by the equipment automatically and returned to zero at the beginning of each

charge or discharge state. There is still some data that is misfilled with non-zero values which will make the capacity calculation shifting to the wrong values. The other is the type indicating the battery is charging or discharging. The cycle starts from charge to 100% and then discharge to almost empty. Some cycles contain only charge data or discharge data and some cycles are in the reverse order. Those issues are addressed by checking through the entire dataset to make sure that every state starts from zero in the charging state. In the meantime, the categorical feature 'type' is converted to the numerical features by replacing 'charge' with 0 and 'discharge' with 1.

Finally, on examination of the summary description of the data, it was revealed that the capacity value is provided only when the battery is charged fully. Since the capacity is the charges stored in the battery, it can be derived by integration of current flowing over the battery.

The sample of the cleaned dataset is shown in Figure 3.2.

Figure 3.2

Cleaned and Integrated Sample

idx	cycle	type	Voltage_measured	Current_measured	Temperature_measured	Time	capacity
1	1	charge	3.325055	3.020467e-04	29.341851	0.000	0.000000e+00
2	1	charge	3.001951	1.000000e-07	29.335723	2.516	6.988889e-11
3	1	charge	3.434644	1.508670e+00	29.334717	5.500	1.250520e-03
4	1	charge	3.454857	1.510043e+00	29.341949	8.391	2.463168e-03
5	1	charge	3.468788	1.508704e+00	29.331462	11.266	3.668036e-03
...	...	...	...	...	...	...	...
1	152	charge	0.236356	1.000000e-07	23.372048	0.000	0.000000e+00
2	152	charge	0.003365	1.000000e-07	23.369434	2.547	7.075000e-11
3	152	charge	4.200000	5.056083e-04	23.386535	5.500	4.148100e-07
4	152	charge	4.200000	4.417751e-04	23.386983	8.312	7.598854e-07
5	152	charge	4.213440	1.000000e-07	23.385061	12.656	7.600060e-07

3.3.2 Data Integration

Each test dataset was stored in individual files with an overview text file giving a summary of the test. The first step would be to concatenate all the cleaned datasets into a cohesive dataset for informational, training, and testing purposes. The data of nominal capacity (rated capacity) for each cycle are reserved in a separate dataset and concatenated into one file for SoC, plugin duration, and remaining mileage calculation.

The overview dataset is the combined data containing battery data and the environment data, as shown in Figure 3.3.

Figure 3.3

Pre-processed data of all tests combined for total of 2,528,129 instances

	cycle	Time	type	Voltage_measured	Current_measured	Temperature_measured	capacity
0	1	0.000	0	3.325055	3.020467e-04	29.341851	0.000000e+00
1	1	2.516	0	3.001951	1.000000e-07	29.335723	6.988889e-11
2	1	5.500	0	3.434644	1.508670e+00	29.334717	1.250520e-03
3	1	8.391	0	3.454857	1.510043e+00	29.341949	2.463168e-03
4	1	11.266	0	3.468788	1.508704e+00	29.331462	3.668036e-03
...	...	...	...	...	...	...	...
2528124	78	9945.640	0	4.190090	7.918553e-02	3.688650	1.228380e+00
2528125	78	9952.343	0	4.190191	8.018630e-02	3.625333	1.228529e+00
2528126	78	9963.750	0	4.190323	7.971176e-02	3.585707	1.228782e+00
2528127	78	9970.453	0	4.190293	7.816658e-02	3.501019	1.228928e+00
2528128	78	9981.781	0	4.190374	7.739590e-02	3.383375	1.229171e+00

2528129 rows × 7 columns

3.4 Data Transformation

Before modeling can begin, the data needs to be modified to provide a more comprehensive view of the data. This step includes calculating additional features to better represent the data, adjusting the scale of each feature to be similar to one another, and finalizing the data for training.

3.4.1 Feature Generation

For time-series data, feature generation is an important step to correctly construct an accurate machine learning model. As mentioned above, this step needs to comprehensively capture each data point as a moment in time, not just a single outlying point in time. To do so, multiple features should be created to better represent each individual data point.

The first feature to be generated is the energy consumption speed which is related to the driving speed. The more energy consumed in one hour, the higher speed the user is driving. This feature is derived by the differential of capacity over time. Both the discharge rate or the energy consumption can only happen in the discharge state.

The second one is the environment temperature as mentioned 3.2. The extremely high or low temperature varying can lead to poor performances or serious safety problems. The battery can show a low capacity, poor cycle, or die in the cold weather because of the deterioration of the electrolyte, the separators and the abnormal reaction between the lithium ions and the electrode materials. On the other hand, the high temperature would result in the melting inside the batteries, thermal runaway, or explosion. Though a slightly higher temperature than room temperature can improve the capacity. In our case, the environment temperature is set up before or during the battery cycling. Thus it can be obtained by the mean temperature in each cycle.

In addition to the cleaned samples in Figure 3.3, the samples for the newly generated features are shown in Figure 3.4.

Figure 3.4

The sample data after feature generation

	cycle	Time	type	Voltage_measured	Current_measured	environment_temperature	Temperature_measured	consumption_per_second	capacity
0	1	0.000	0	3.325055	3.020467e-04	28.0	29.341851	0.0	0.000000e+00
1	1	2.516	0	3.001951	1.000000e-07	28.0	29.335723	0.0	6.988889e-11
2	1	5.500	0	3.434644	1.508670e+00	28.0	29.334717	0.0	1.250520e-03
3	1	8.391	0	3.454857	1.510043e+00	28.0	29.341949	0.0	2.463168e-03
4	1	11.266	0	3.468788	1.508704e+00	28.0	29.331462	0.0	3.668036e-03
...	...	...	...	...	...	...	...	...	...
2528124	78	9945.640	0	4.190090	7.918553e-02	6.0	3.688650	0.0	1.228380e+00
2528125	78	9952.343	0	4.190191	8.018630e-02	6.0	3.625333	0.0	1.228529e+00
2528126	78	9963.750	0	4.190323	7.971176e-02	6.0	3.585707	0.0	1.228782e+00
2528127	78	9970.453	0	4.190293	7.816658e-02	6.0	3.501019	0.0	1.228928e+00
2528128	78	9981.781	0	4.190374	7.739590e-02	6.0	3.383375	0.0	1.229171e+00

2528129 rows × 9 columns

3.4.2 Data Normalization

In Figure 3.4, the values for each feature vary widely in magnitude. Some are values fixed between zero and one, while others commonly have three digits. This difference in magnitude presents issues for most machine learning models as larger scaling of the data may make the model misinterpret the importance of some features. This imbalance of scaling may be solved by using a normalization method. Normalization rescales each feature down to a specific minimum and maximum depending on the minimum and maximum values present in the data. Often, zero and one values are used as the new minimums and maximums of the data.

The formula for normalizing data into a zero through one range is below with x_i equal to the data point needing normalization with *min(x)* and *max(x)* equal to the minimum and maximum values of the feature.

$$norm\, x_i = [x_i - min(x)] \,/\, [max(x) - min(x)] \tag{3.6}$$

After normalizing the dataset, the data transformation step is completed. Figure 3.5 shows samples pulled from the data after normalization.

Figure 3.5

Sample of normalized dataset at 5 randomly selected indexes

type	Voltage_measured	Current_measured	environment_temperature	consumption_per_second	capacity
0	0.266793	0.000199	0.371429	0.996473	0.000000e+00
0	0.000000	0.000000	0.371429	0.996473	3.494444e-11
0	0.357283	0.995704	0.371429	0.996473	6.252598e-04
0	0.373974	0.996611	0.371429	0.996473	1.231584e-03
0	0.385477	0.995727	0.371429	0.996473	1.834018e-03
...	...	...	...	...	...

3.5 Data Preparation

To be able to assess the accuracy of any model, the data will be split into three pieces; training, testing, and validation. The most popular split to use is the 50/15/35 split where 50% is designated to training data, 15% into validation data, and the last 35% for tests. The training data will be used to train the model. The validation data is used to evaluate the model by fitting the trained model to compare for errors in the predicted value and actual values. This data should be separate from the training data to prevent overfitting. Finally, the test dataset will be used for the outperformed models to select the optimal one and treated as unseen data. Multiple models can be created and tuned by checking for accuracy against the validation data. This helps prevent overfitting, helps tune hyperparameters, and increases accuracy.

The following are abbreviated final samples of the separated training & validation datasets, and test dataset. The training & validation dataset contains a total of 1,477,114 values while the test dataset contains 760,571 values.

Figure 3.6

Data preparation by splitting data into training & validation data and test data

```
Training & Validataion
```

	timestamp	type	Voltage_measured	Current_measured	environment_temperature	consumption_per_second	capacity
0	2005-01-01 00:00:00.000	0	0.266793	0.000199	0.371429	0.996473	0.000000e+00
1	2005-01-01 00:00:02.516	0	0.000000	0.000000	0.371429	0.996473	6.988889e-11
2	2005-01-01 00:00:05.500	0	0.357283	0.995704	0.371429	0.996473	1.250520e-03
3	2005-01-01 00:00:08.391	0	0.373974	0.996611	0.371429	0.996473	2.463168e-03
4	2005-01-01 00:00:11.266	0	0.385477	0.995727	0.371429	0.996473	3.668036e-03
...	...	...	...	...	...	...	...
1477109	2044-03-18 02:45:45.640	0	0.980027	0.052770	0.057143	0.996473	1.228380e+00
1477110	2044-03-18 02:45:52.343	0	0.980206	0.053437	0.057143	0.996473	1.228529e+00
1477111	2044-03-18 02:46:03.750	0	0.980443	0.053121	0.057143	0.996473	1.228782e+00
1477112	2044-03-18 02:46:10.453	0	0.980389	0.052091	0.057143	0.996473	1.228928e+00
1477113	2044-03-18 02:46:21.781	0	0.980534	0.051577	0.057143	0.996473	1.229171e+00

1477114 rows × 7 columns

```
Test
```

	timestamp	type	Voltage_measured	Current_measured	environment_temperature	consumption_per_second	capacity
0	2006-01-01 00:00:00.000	0	0.242540	0.000000	0.371429	0.996473	0.000000e+00
1	2006-01-01 00:00:02.516	0	0.000000	0.000000	0.371429	0.996473	6.988889e-11
2	2006-01-01 00:00:05.500	0	0.326846	0.997400	0.371429	0.996473	1.256260e-03
3	2006-01-01 00:00:08.391	0	0.345428	0.996931	0.371429	0.996473	2.472795e-03
4	2006-01-01 00:00:11.266	0	0.358125	0.997308	0.371429	0.996473	3.683054e-03
...	...	...	...	...	...	...	...
760566	2043-03-13 02:45:45.640	0	0.998698	0.055892	0.057143	0.996473	1.284322e+00
760567	2043-03-13 02:45:52.343	0	0.998619	0.055385	0.057143	0.996473	1.284476e+00
760568	2043-03-13 02:46:03.750	0	0.998747	0.053952	0.057143	0.996473	1.284731e+00
760569	2043-03-13 02:46:10.453	0	0.998977	0.052355	0.057143	0.996473	1.284877e+00
760570	2043-03-13 02:46:21.781	0	0.999160	0.053155	0.057143	0.996473	1.285127e+00

760571 rows × 7 columns

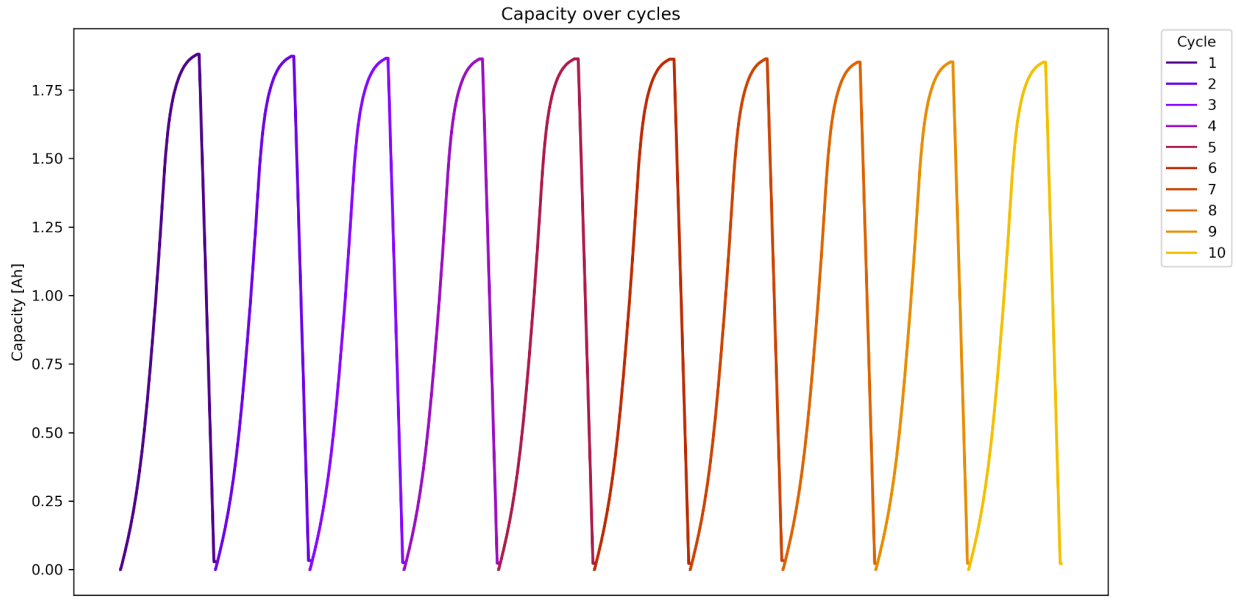
3.6 Data Statistics

3.6.1 Feature Overview

The most important feature is the target feature ‘capacity’. It can determine the battery status such as SoC and SoH, the plug-in duration in charging, and the remaining mileage in discharge.The overall capacity is shifting from around 2Ah to 0Ah in each charge and discharge cycle. It will rise in the charging state and fall in the discharging state which is shown in Figure 3.7. The combination of the capacity rise and fall is called cycling.

Figure 3.7

Capacity changes over cycles. The sample data is ten cycles.



Besides capacity, the features are also crucial to characterize the battery status such as current and voltage. When the charging starts, the current will increase to near 1.5A and keep constant until the battery is charged to around 80% then drop to close to zero to finish the rest of the charging cycle as seen in Figure 3.8. In the discharging state, the current will remain at 2A

when the discharge rate is 1C until the battery depletes to empty. Figure 3.9 depicts the overall change to the battery current over ten cycles.

Figure 3.8

Current change in the charge state over cycles

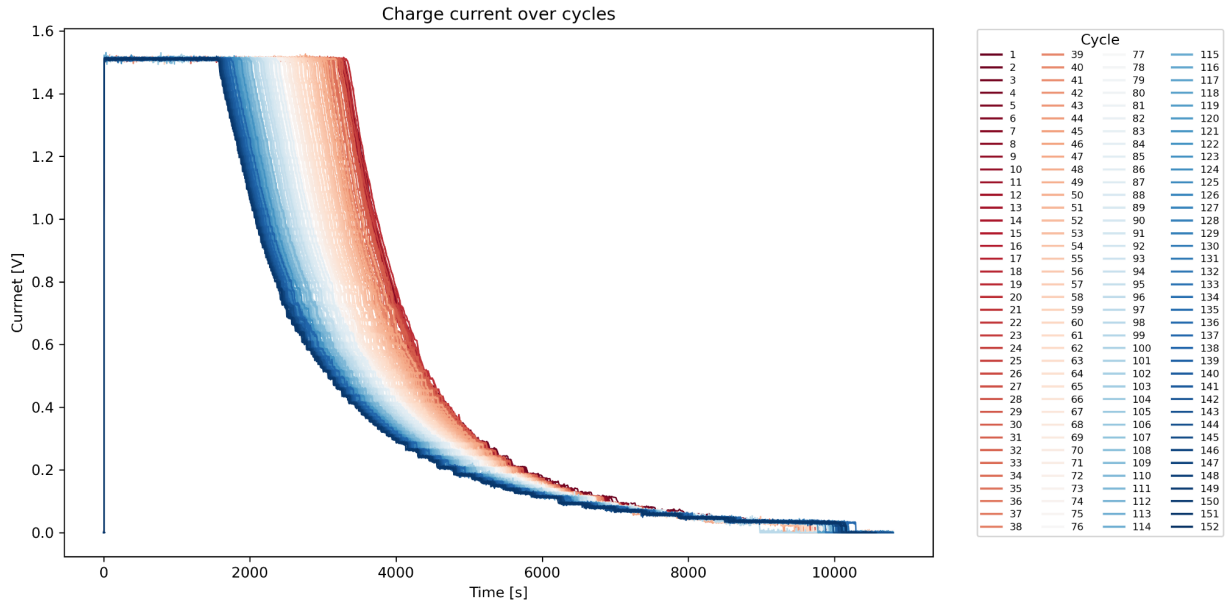
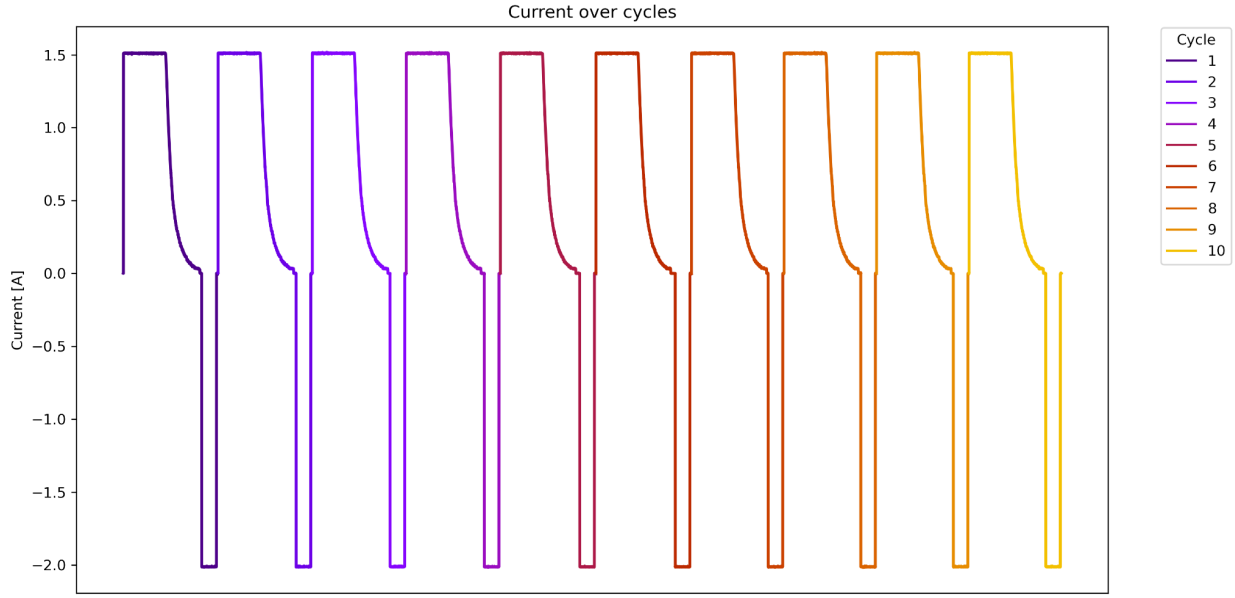


Figure 3.9

Current over cycle. The sample data is ten cycles.



The battery voltage is much more straightforward as it will increase to 4.2V and keep constant to charge the battery with the remaining 20% energy, and gradually decrease from 4.2V to 2.2V, 2.5V or 2.7V depending on the test requirement. The voltage's cycling behavior is displayed in Figure 3.10. The voltage changes in the charge and the discharge states over cycles are demonstrated in Figure 3.11 and 3.12 respectively.

The voltage drops faster and faster as the cycle progresses. So that the time spent for battery depletes to empty would be shorter than the fresh cell. The phenomenon can be observed in the left shifting of the discharge voltage curves and the shortened time spent in Figure 3.13. That indicates battery degradation.

Normally, the battery voltage is between 3.7V to 4.2V as the deep discharge will damage the battery. Thus the range of data is large enough for the electric vehicle.

Figure 3.10

Voltage over cycle. The sample data is ten cycles.

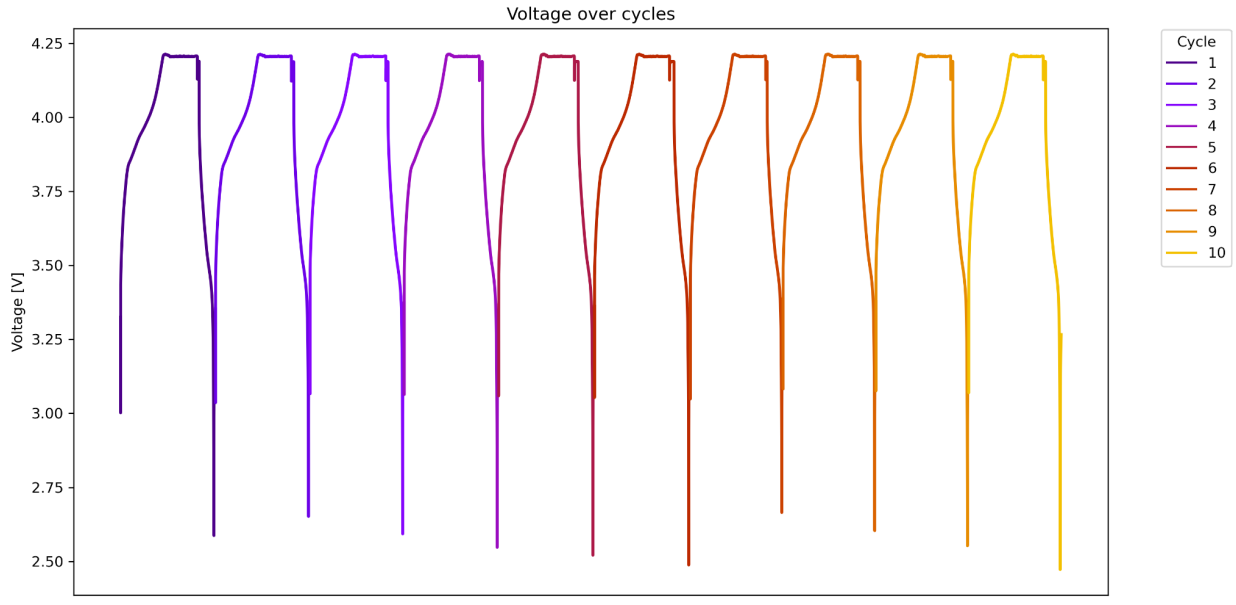


Figure 3.11

Voltage change in the charge state over cycles.

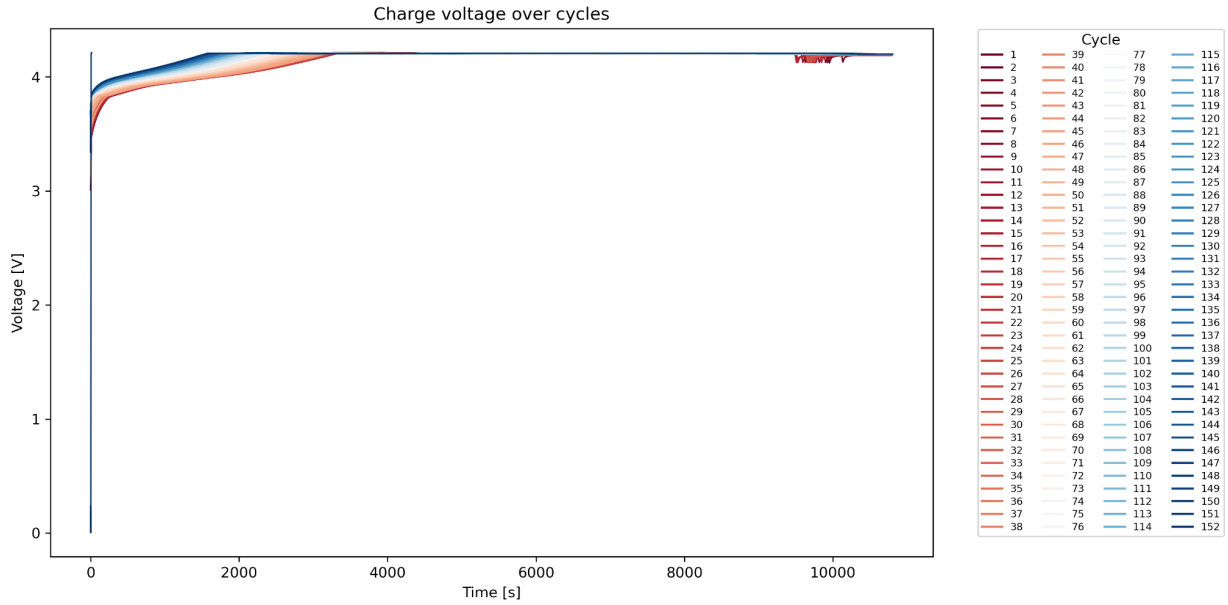


Figure 3.12

Voltage change in the discharge state over cycles.

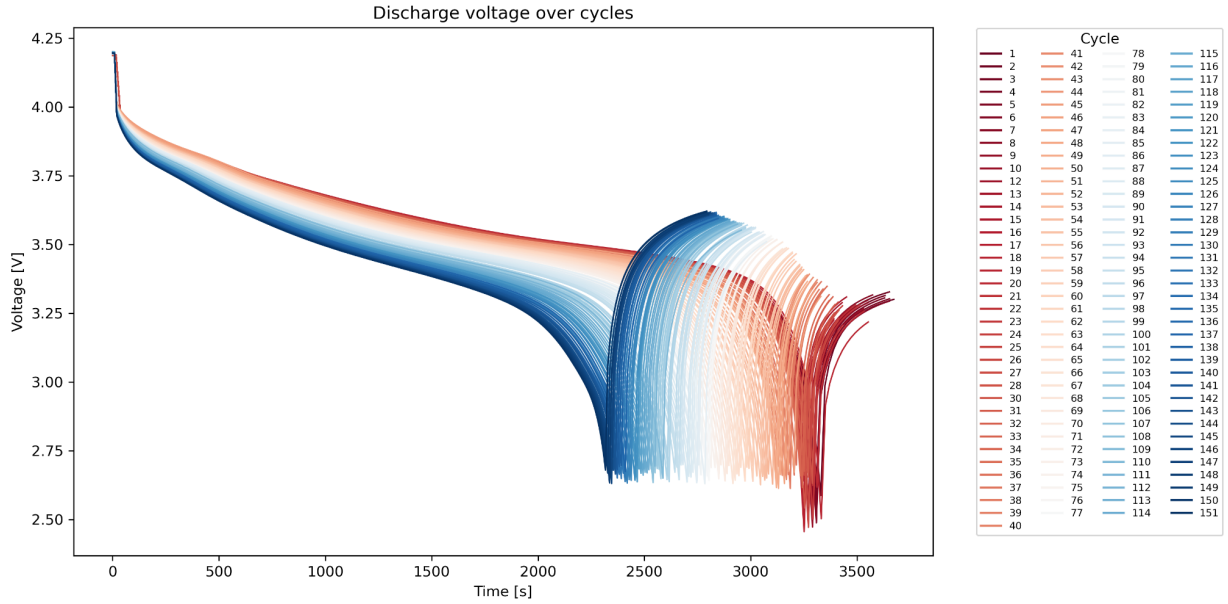


Figure 3.13

Time spent to discharge the battery completely over cycles.

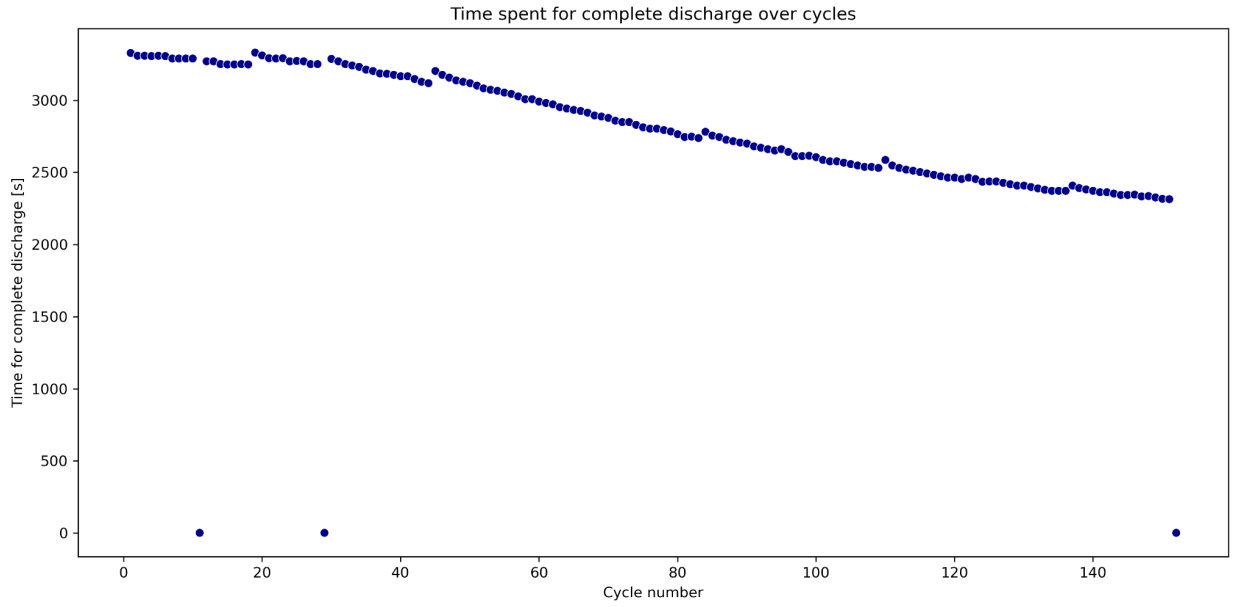
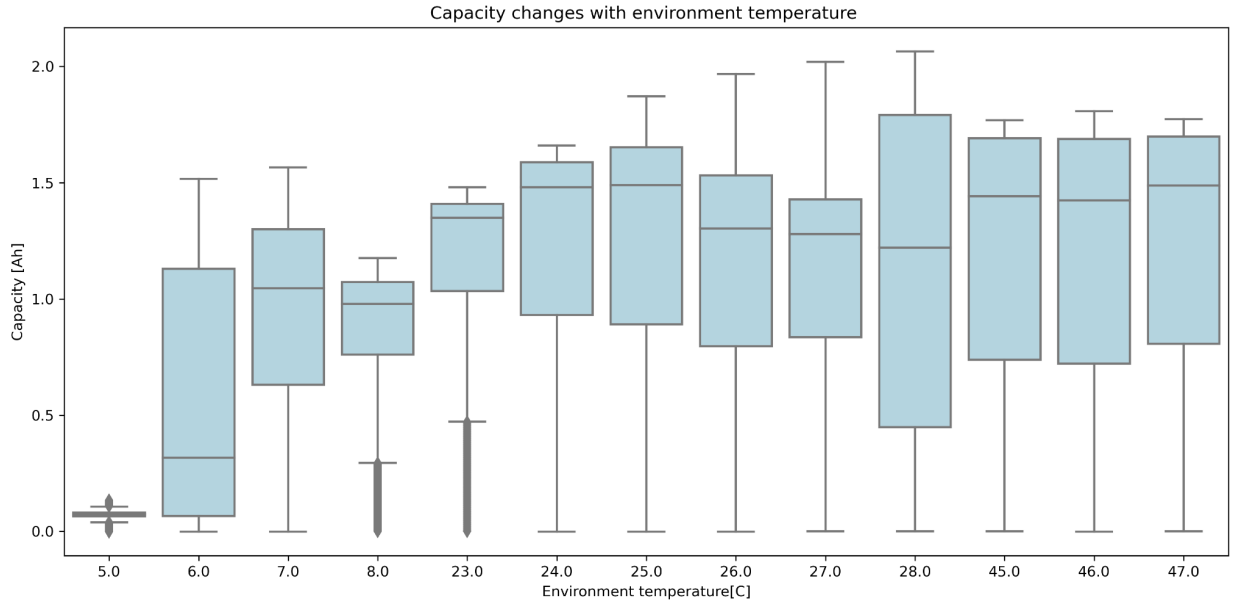


Figure 3.14

Environment Temperature Distributions

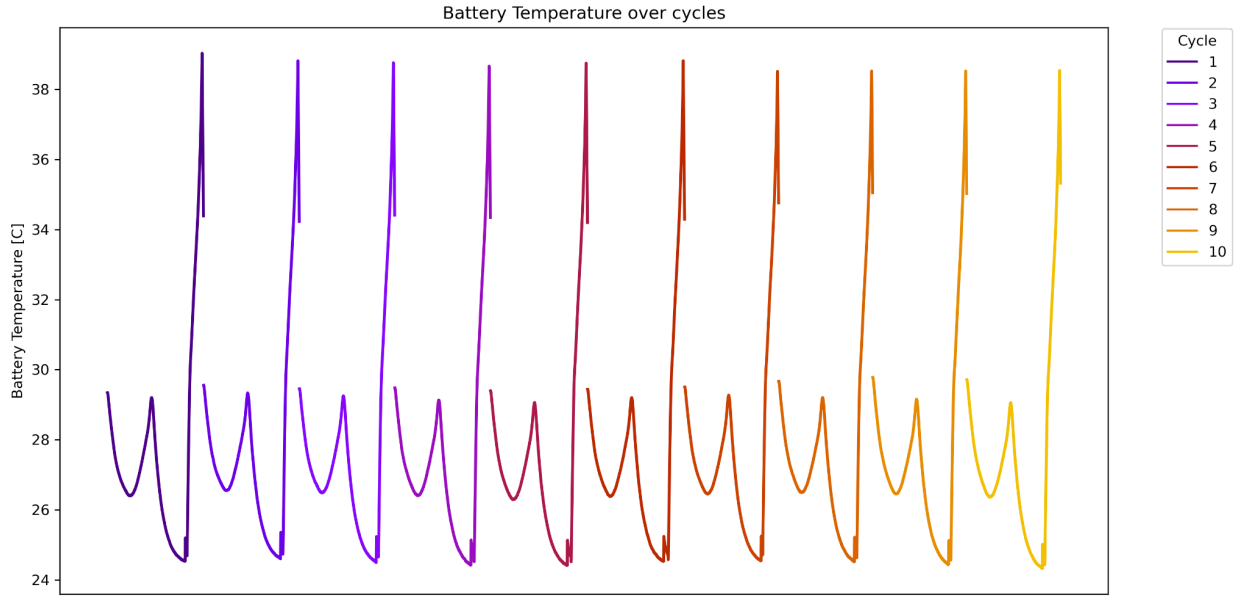


The environment temperature distributions are demonstrated in Figure 3.14. As mentioned above, the low temperature and the high temperature are set for the battery cycling. That is necessary as the users would drive the car in different regions and under different weather which can affect the battery performances.

Though the battery temperature in Figure 3.15 is the results from the battery cycling and cannot be utilized as the input feature, the trend of temperatures rising in each cycle can make a warning that there is safety issues when the battery is exposed to the hot weather.

Figure 3.15

The battery temperature changes over cycle. The sample data is ten cycles.



3.6.2 The Data

The exploratory data analysis exhibits the patterns of capacity change over cycles and time. The capacity will rise in charge and fall in discharge. The nominal capacity per cycle will gradually decrease as the battery is in charge and discharges in turn repeatedly. As the loading profile or the environment temperature changes, the patterns for cycles shift correspondingly. That is to say, the battery data demonstrate trends over cycles and changed seasonality among cycles.

The time series decomposition is conducted by applying Seasonal and Trend decomposition using Loess (STL). The period is set as the mean point numbers per cycle. The data can be split into trend-cycle, seasonality, and residues. One independent dataset has the similar trend that the capacity will drop over cycles while the capacity would dramatically decrease with the low temperature and increase with the high temperature.

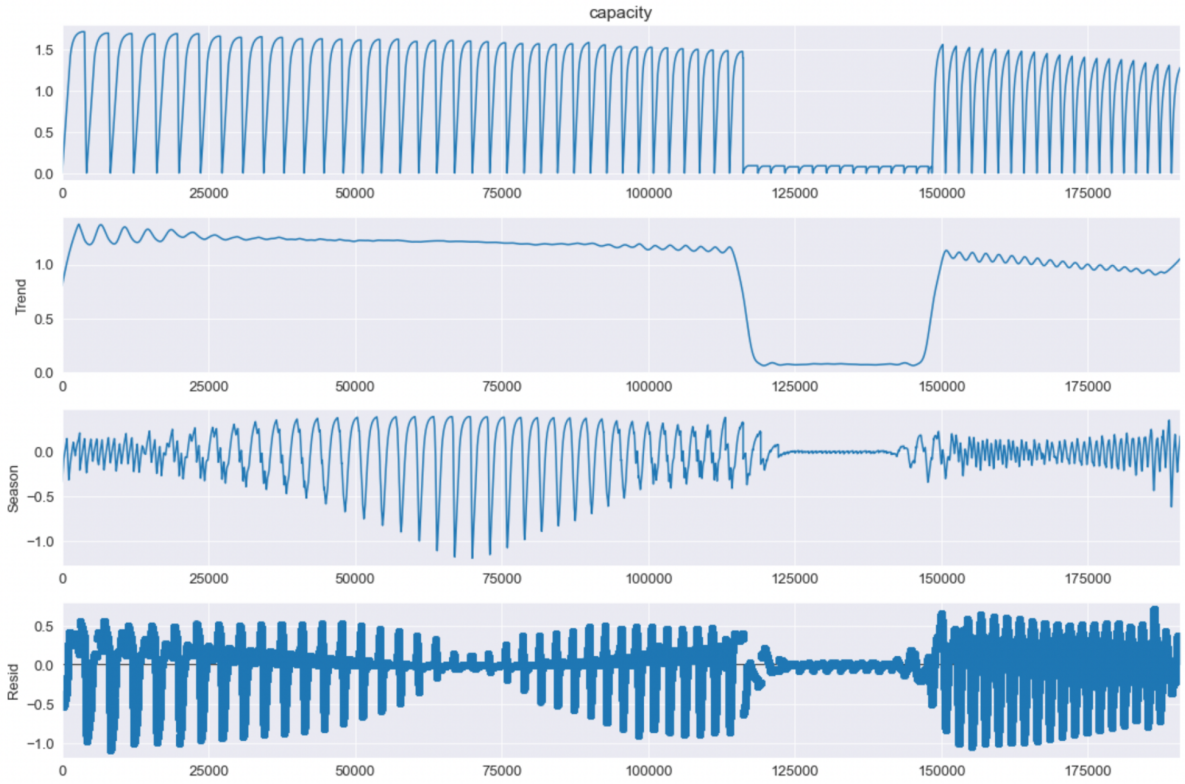
The patterns are listed below: capacity under low temperature and multip load, capacity under high temperature and multiple loads, and capacity under room temperature and constant load.

Figure 3.16: Time-series decomposition

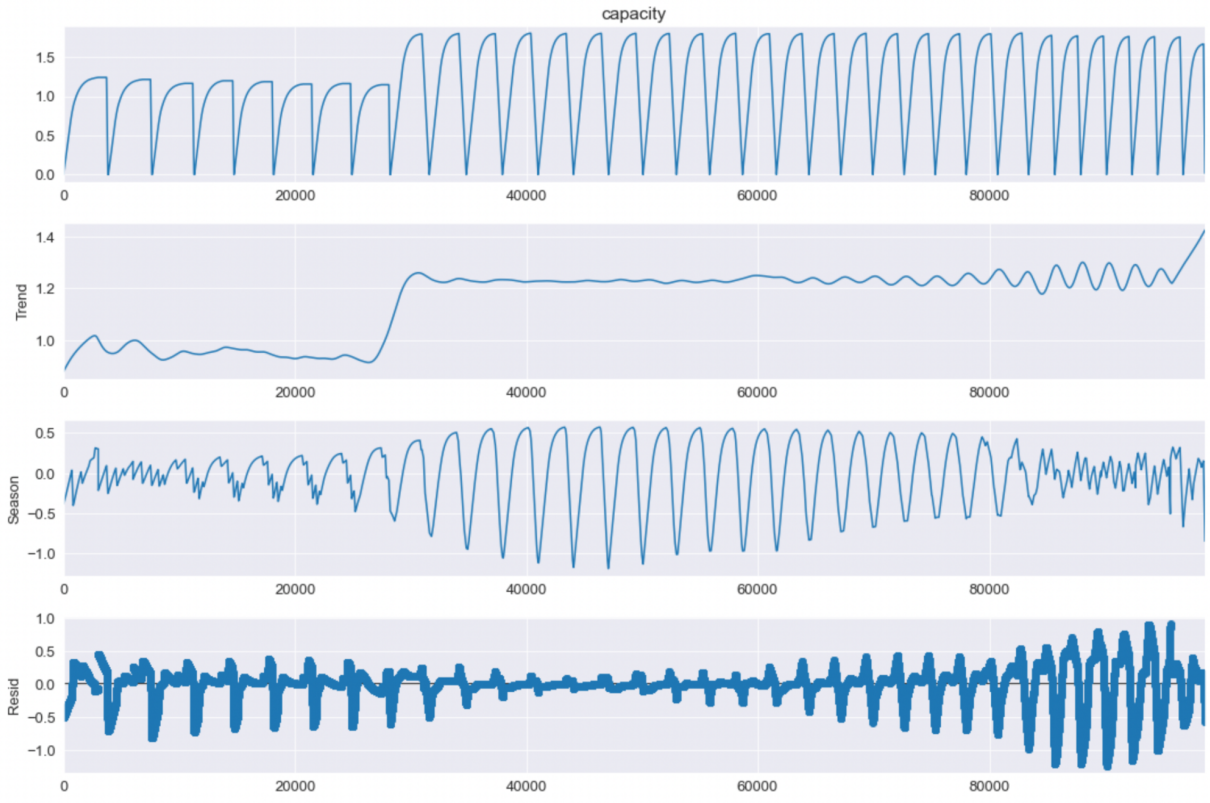
(a) capacity under low temperature and multiple loads

(b) capacity under high temperature and multiple loads

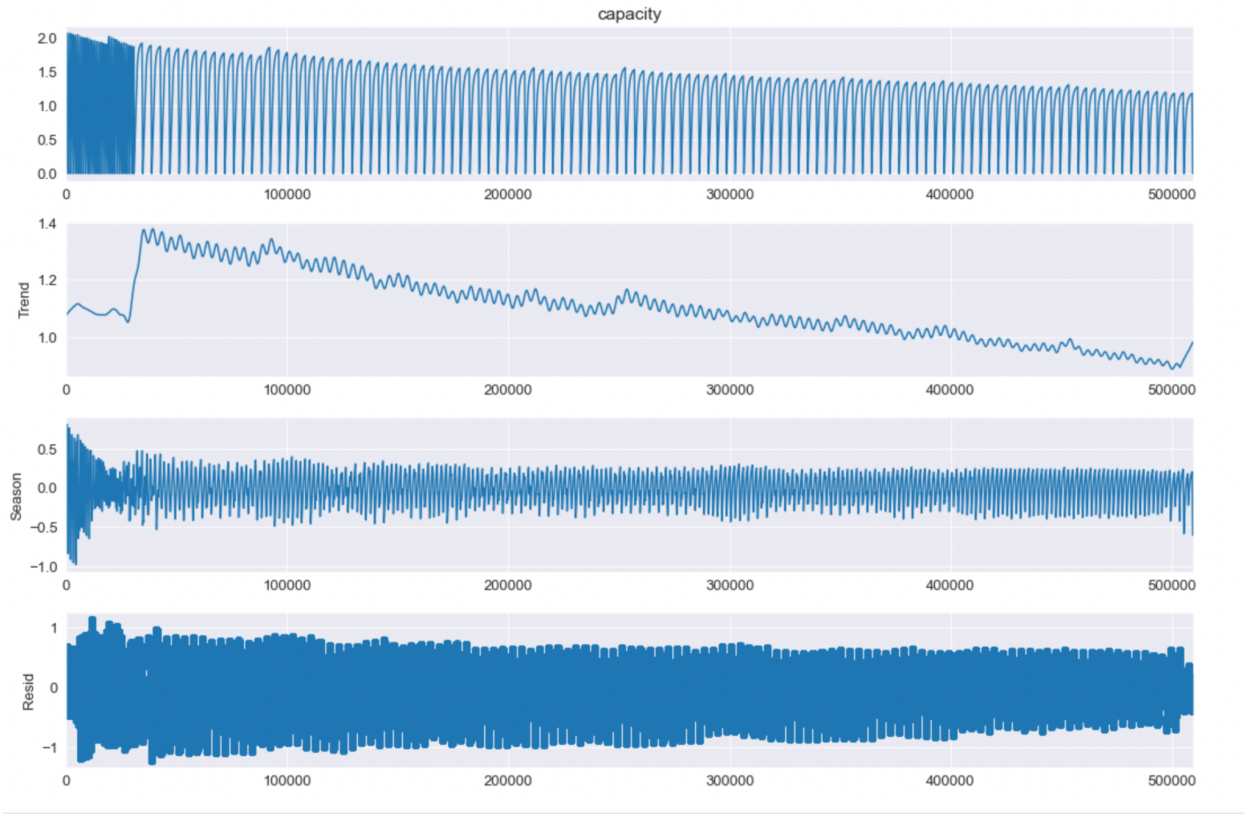
(c) capacity under room temperature and constant load



(a)



(b)



(c)

The figures above further confirm the analysis that there is the trend of the capacity over cycles. The frequency of the seasonality is changed by loading profile, environment temperature, and the cycle progresses. It is reasonable to do the time-series prediction.

3.6.3 Summary

18650 Li-ion battery data are collected for the battery status prediction. The raw data contains 10 files and consists of battery data and environmental data. With domain knowledge of lithium ion batteries, the features with significant impact on battery status are reserved. The data exploration shows the cyclical behavior of battery, fast charging profile, changed discharging profile under various environment conditions. Those factors will determine capacity in different

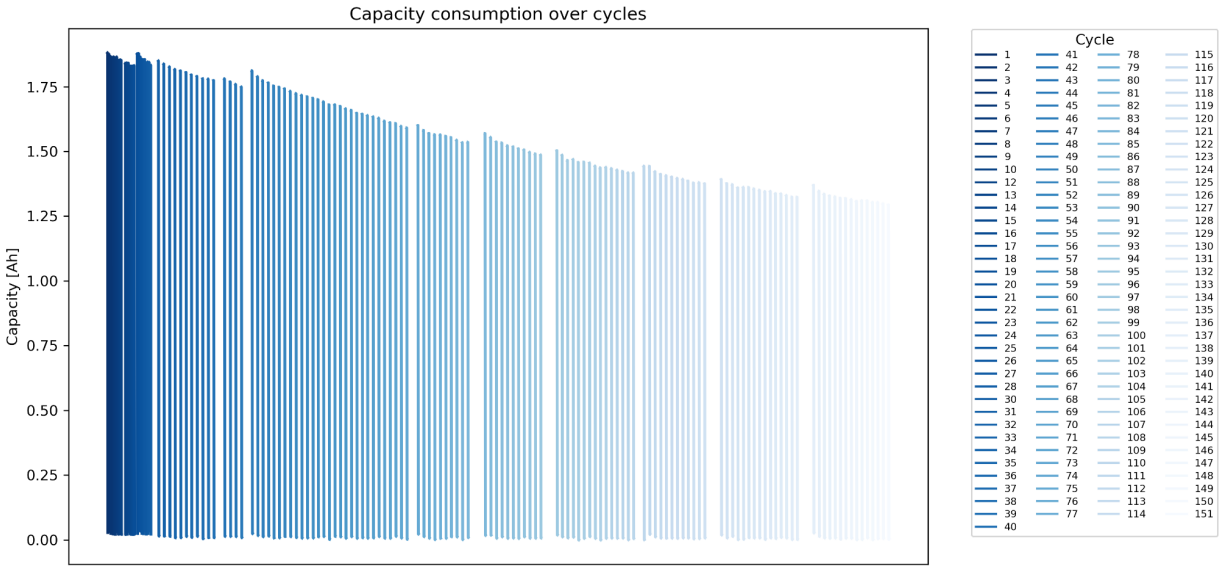
ways. The features are generated to capture the characteristics of energy consumption and varied environment conditions. The data processing is executed by data integration, data cleaning, and data normalization. The obtained dataset is splitted into training, testing, and validation with the portion of 50%, 15%, and 35%.

3.7 Data Analytics Results

The data analytics results demonstrate the cyclical and period behavior over time. Figure 3.17 shows the capacity distribution over the cycle overall. The maximum capacity in each cycle is reduced gradually over cycles. That's the nominal capacity per cycle (rated capacity). This phenomenon is caused by battery degradation. Hence there should be a life span for the battery. The estimation of the battery states will be defined as time-series prediction which will involve the time-series based features and approaches in the machine learning and the deep learning deployment.

Figure 3.17

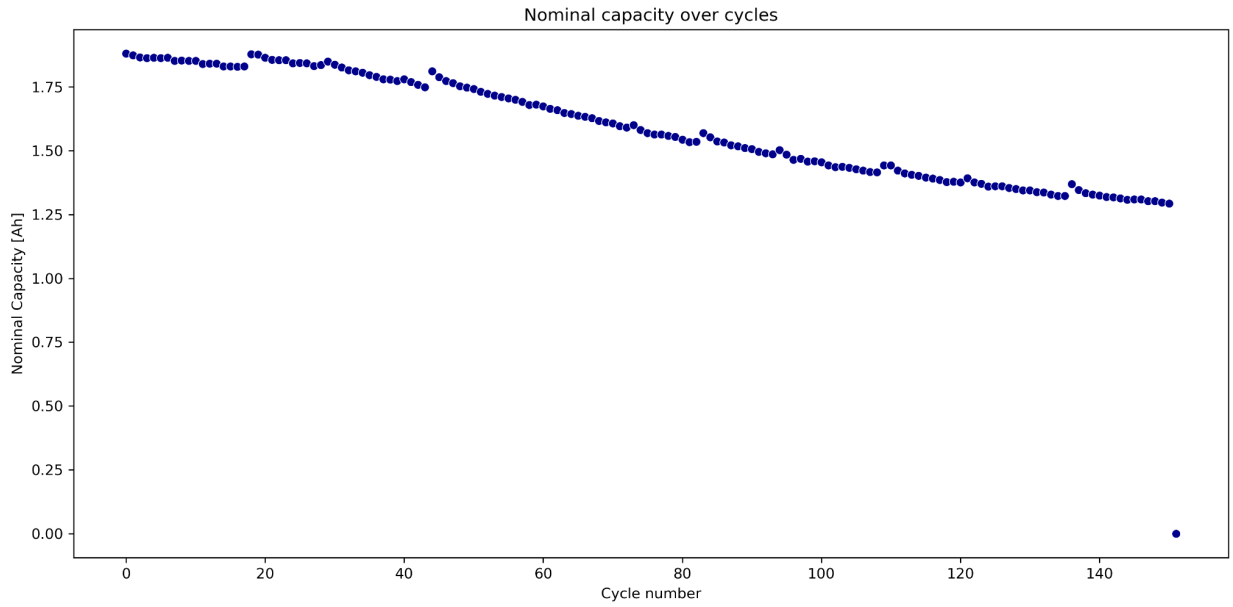
Capacity consumption over cycles



The nominal capacity can be extracted so that we can see the trend over cycles in Figure 3.18. The nominal capacity is declining along the cycle number in a near linear way.

Figure 3.18

Nominal capacity over cycles



4. Model Development

4.1 Model Proposals

4.1.1 Feature Selection

The battery cell data are used for training, evaluation, and tests. The raw data contain the battery cycling data under multiple loading profiles and the environment data. After data processing, the dataset covers most of the application scenarios such as driving/parking mode or hot/cold weather.

Theoretically, battery capacity and SoC can be determined by the integration of nominal voltage (V), current (A) and time (s) (Zhang & Fan, 2020). However, the change from the environment condition and the driving condition would lead to various ways for energy consumption. Accordingly, the battery capacity and SoC for EV in the trip would drop faster or slower than that resulting from the laboratory measurements. So the input features should have the ability to capture the change.

First of all, the features of battery states are selected from original feature lists that have a significant influence on the battery capacity. Table 4.1 details those features that are required for battery capacity estimation.

Table 4.1

Original Battery Features

Feature Name	Description
Current_measured	Battery current (Amps)
Voltage_measured	Battery voltage (Volts)
Temperature_measured	Battery temperature (degree C)
Cycle	Number of cycles
type	Charge or Discharge
Time	Seconds
Capacity	Charge stored in the battery (Ah)

Several features are generated to capture the changes that affect the battery status. Acceleration and deceleration during driving can lead to the current variation and are reflected in the discharging data. Thus the speed of the voltage drop is represented by the differential of the loading profile over each time-step. The energy consumption speed indicated the driving speed. Normally the energy will be consumed faster on the highway than on local. This feature is derived by differentiation of capacity over each time-step. The environment condition is given by the battery testing setup that includes the low temperature, room temperature, and high temperature. The nominal capacity or rated capacity is extracted from the capacity value when the battery is fully charged.

All features are passed through a random forest decision tree regressor model and are given an importance score. This score estimates the impact that feature has in estimating the regression. Lower scores may be eliminated as they likely have a lower impact in a machine learning model.

The way a decision tree functions is that it maximizes information gain from splitting features into two or more categories that the data will fall into, called child nodes (Kelleher et al., 2015). To do so, it calculates the entropy of each possible child node using the equation

$$E(A) = -\sum_{k=1}^{n} p_k log_2(p_k) \quad (4.1)$$

where p_k is the probability of falling into the n-th split. Information gain is then individually calculated by subtracting the entropy of each child node with its parent, and the child node that would gain the most information would be selected.

However, decision trees can be prone to overfitting as each feature is meticulously selected and split into perfect nodes for the training data. So, a random forest bagging method is introduced to increase accuracy and prevent overfitting. Through the use of random forest, a large number of decision trees are created where random features are selected for each decision node. And, the features that would provide little to no value have their importance decreased while the features that continuously outperform others through high information gain will be labeled with high importance.

Through the use of the random forest decision tree regressor, even features that are similar to one another can be distinguished. Table 4.2 outlines the final feature list. These were the features the random forest decision tree regressor deemed most important. The generated feature ‘Discharge_speed [V/s]’ will be dropped as it shows the least impact on the target feature. These features in addition to the features named above, will be included in the modeling process.

Figure 4.1

Feature importance of input features

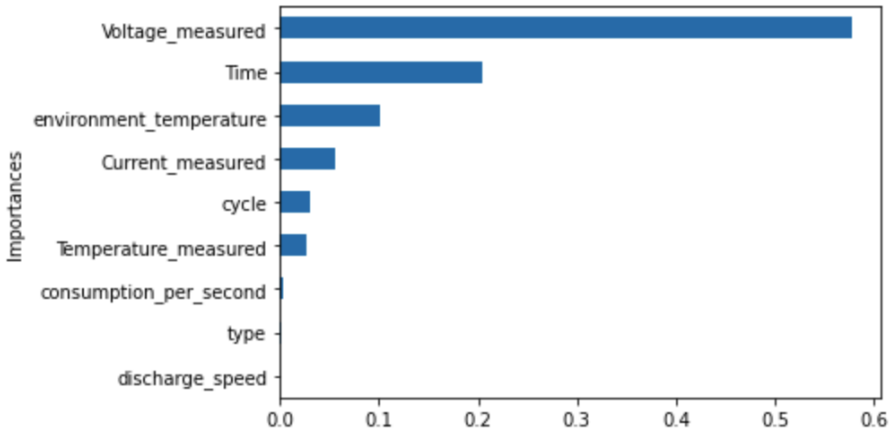


Table 4.2

Random Forest Feature Importance

	feature	importance
7	discharge_speed	0.000001
1	type	0.001521
8	consumption_per_second	0.002932
5	Temperature_measured	0.026377
0	cycle	0.029674
3	Current_measured	0.056169
4	environment_temperature	0.100601
6	Time	0.204782
2	Voltage_measured	0.577943

One important aspect of this process is that the CRISP-DM project management cycle for data analytics allows for reworking previous steps to improve upon the final product. If the modeling aspect provides mediocre results, additional scrutiny can be made on the feature generation and feature selection steps. Additional features may be created to capture the time-series data better, and additional features could be removed to increase training times and

increase accuracy. While the random forest decision tree regressor may provide some insight on the importance of features, only the modeling process can truly reveal the strength of the finalized feature list.

4.1.2 Model Proposal Introduction

The project is aiming to predict the battery capacity and deliver the battery state of charge (SoC), plug-in duration, and remaining mileage in electric vehicles. The capacity and SoC estimation is part of the fleet analysis which provides the remaining mileage information to the drivers. Accurate prediction provides benefits for the consumer such as reducing range anxiety so drivers can maximize battery usage and extending the battery lifespan by providing drivers with charging reminders which prevents overcharging and over discharging. From a manufacturer's perspective, accurate prediction reduces cost because it means more reliability and consistency. This makes it easier to define warranty and also allows for smaller battery packs with the right amount of cells for the intended purpose instead of adding extra cells as a preemptive measure that would cost more and result in heavier battery packs.

In this section, an assumption is proposed that the current battery state is determined by the previous battery state. Therefore, the relevant features should be selected that can capture the battery change in time-series. Then the fitting models are elected for the capacity, SoC, plug-in duration, and remaining mileage estimation.

The prepared data contains multivariable input features and a target feature 'Capacity'. The data are splitted into training, validation, and test sets for machine learning. Machine learning is divided into two parts: offline training and online prediction. The offline training is carried out on the whole training data with k-fold cross-validation resampling. The parameters

will be updated for model optimization. Then the online prediction is executed by inputting one set of data. The real-time capacity will be predicted.

Five models are proposed for capacity prediction: eXtreme Gradient Boosting (XGBoost), Support Vector Regression (SVR), k-Nearest Neighbor regression (KNN), Long Short-Term Memory (LSTM), and Multilayer perceptron (MLP). Specifically, the GPR with the designed kernel is proposed to model the characteristics of the features and the gating network is built to speed up the modeling process and make the results interpretable. At last, The Seasonal Auto-Regressive Integrated Moving Average with eXogenous factors (SARIMAX) model is set as a baseline so that we can compare the prediction accuracy and consuming time of convergences.

4.1.3 *eXtreme Gradient Boosting (XGBoost)*

XGBoost is an ensemble decision tree model that has been widely used for many machine learning problems. Unlike random forest that builds fully grown trees in parallel and gets the performances by majority votes, XGboost builds a tree in sequentially called weak learner. The loss function is applied and updated by adding a new weak learner. By gradient descent, the loss function is minimized over the weak learners to reach the maximum performance by differential functions (Chen & Guestrin, 2016). With the benefit of the decision tree approach, the results from XGBoost are also interpretable while the iteration over the loss functions makes the training of input data efficient by implementing the training paralleling. The algorithm of XGBoost is described as follows.

For the dataset D with n instances and m features, the initial model can be set as:

$$\widehat{f}_0(x) = arg\,min \sum_{i=1}^{n} L(y_i, \theta) + \Omega(\theta) \tag{4.2}$$

where θ is the parameter, L is the differential loss function, and Ω is the regularization term . The gradients and the hessians at k_{th} step are calculated as the first order and the second order differentiation of the loss function and denoted as g and h:

$$\widehat{g}_k(x_i) = \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \tag{4.3}$$

$$\widehat{h}_k(x_i) = \frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2} \tag{4.4}$$

Thus the loss function is obtained:

$$\widehat{L}_k(x_i) = \sum_{i=1}^{n} [\widehat{g}_k(x_i) f(x_i) + \frac{1}{2}\widehat{h}_k(x_i) f(x_i)^2] + \Omega(\theta) \tag{4.5}$$

The model is updated as:

$$\widehat{f}_k(x) = \widehat{f}_{k-1}(x) + \widehat{f}_k(x) \tag{4.6}$$

Table 4.3

Pseudocode for XGBoost

Input: $X \in \{training\ data\}$, $X^* \in \{test\ data\}$
Output: $\widehat{y}$ battery capacity
1. For dataset D, initiate model:

$$\widehat{f}_0(x) = arg\ min \sum_{i=1}^{n} L(y_i, \theta) + \Omega(\theta)$$

2. For k in range(K), calculate gradient statistics

Gradient $\widehat{g}_k(x_i)$

Hessians $\widehat{h}_k(x_i)$

Update loss function and object function

3. **Iterating** until minimized object function.

4.1.4 *Support Vector Regression*

The Support Vector Machine (SVM) was created in 1995 by Cortes and Vapnik; it was designed for classification models. The main goal of the SVM model is to create a hyperplane that optimizes the best decision boundary that separates two or more classes of data (Cortes & Vapnik, 1995, p. 278). They also outlined the ability to adapt this hyperplane to predict continuous values as well. This technique is widely known as the Support Vector Machine Regression or SVR. While SVM optimizes the hyperplane for the best decision boundary, SVR attempts to optimize a hyperplane that best estimates the numerical outcome of the data. This hyperplane will have its own decision boundaries to attempt to capture as many data points within as it can while reducing the distance between those points and the hyperplane. These predictions are error-based and mathematically represented by the following equation.

$$Y_i = \sum_{i=1}^{N} W \bullet K(x_i, x) + B \tag{4.7}$$

From this equation, Y_i is equal to the predicted value, W and B is equal to the matrix of trained weights as well as the bias matrix. Finally, the $K(x_i, x)$ is known as a kernel function that is used to convert the simpler linear functions into advanced hyperplanes. However, the linear version of SVR can produce accurate results when the predicted value follows a linear pattern. This may be the case for SoC, but that remains to be seen.

According to Parashar, to mathematically find the optimal hyperplane for a multi-dimensional dataset, it must be mapped to a multi-dimensional space using a mapping function. However, this is nearly impossible to do for extremely high dimensions and computationally expensive even for lower ones. To solve this, the kernel trick is used to calculate data into higher dimensions. To apply the kernel trick, the dot product of two vectors is replaced

by a kernel function. The optimal hyperplane in the *n*th-dimension can be calculated by this equation.

$$max_a(\sum_{i=1}^{n} a_i - 1/2 \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j y_i y_j K(X_i^T, X_j) \quad (4.8)$$

The following is a rough summary of how SVR can be trained for a model. With each new data point, a new hyperplane is calculated which recalculates the error of the regression. The updated optimal hyperplane is the hyperplane that minimizes error within the dataset and its decision boundaries. Table 4.4 represents the pseudocode for SVR.

Table 4.4

Pseudocode for Support Vector Regression

Input: Training dataset with classifications 1. Set Coefficients and Bias to 0 2. Finding a suitable hyperplane a. Find hyperplane candidates b. Select hyperplane that minimizes error c. Calculate decision boundaries d. Calculate error from the squared sum of each point outside the hyperplane. 3. Update Coefficients and Bias 4. When new training data is entered, repeat steps 2 and 3

4.1.5 *K-Nearest Neighbor Regression (KNN)*

As a similarity-based learning model (Kelleher et al., 2015), KNN's basic logic is that the target should be similar to its k nearest ones (Chaka et al., 2018). This project applies the KNN regression model to estimate the continuous value.

KNN regression algorithm is to first measure the distances between test data and training data points, then find k nearest neighbors in the training dataset for the test case, and then the average of these neighbors is taken to predict the target value (Kelleher et al., 2015). KNN has

the advantage of versatility. The descriptive features form a multi dimensional space. When a new test data comes in, the distance between this test data and each training data is calculated, and the average of the nearest K neighbor will be returned as the battery capacity. Although KNN is easy to interpret, one drawback is that it only computes when new test data requests, which makes prediction step expensive, especially for large datasets.

In KNN, selection of K value, distance metrics and weight affect the prediction performance (Kelleher et al., 2015). About the selection of K, the larger the K is, the less the effect of the noise. However, large K makes the boundaries less distinct. About distance metrics, this paper uses the commonly used Euclidean distance. According to Kelleher et al., the Euclidean distance between two points "a" and "b" in m-dimensional feature space is:

$$Euclidean\ (a, b)\ = \sqrt{\sum_{i=1}^{m} (a_i - b_i)^2} \tag{4.9}$$

Both points have m descriptive features. "∑" is to calculate the sum of square of each feature's difference (Kelleher et al., 2015). Then, the square root is returned as Euclidean distance. About the weight, one approach is to calculate the average of the K nearest neighbors; another method is to use the inverse distance weighted average. The average or weighted average is returned as the prediction. The pseudocode for KNN regression is presented in Table 4.5.

Table 4.5

Pseudocode for K-Nearest Neighbor Regression

Input: X: training data, Y: battery capacity for X, x: test data

1. for each element in X:
    a. compute Euclidean distance between this element and test data
2. Get the K nearest elements for the test data
3. **Output:** Return the average / weighted average of Y of these K nearest elements

4.1.6 *Long short-term memory (LSTM)*

Long short-term memory is a type of recurrent neural network (RNN) that is able to forecast predictions in the near future based on previous historical data. It uses three dimensions as input which represent batch size, number of time steps and number of input features per time step. One advantage of this method is that the number of timesteps used to train a model can be specified and how far to forecast predictions can also be set in advance. Each prediction relies on the specified number of previous timesteps so a prediction farther in the future may include previous predictions as part of the previous timesteps to generate the next prediction. Although there is a simplified variant of LSTM known as Gated Recurrent Unit (GRU), it has a tradeoff of reducing accuracy in exchange for faster performance which may be useful if instant estimations are the priority but for the scope of this project, the focus is accurate SOC estimation. LSTM uses two state vectors, one for short term state and one for long term state as the overall purpose of this network focuses on learning what to keep, remove or use from the long term state.

LSTM begins at current timestep t with the previous long term cell state c_{t-1} which goes through the forget gate, f_t, to remove some information from the long term state. Then the long term state gains information based on what information was selected by the input gate, i_t which results in current long term cell state c_t which is defined as:

$c_t = f_t \otimes c_{t-1} + i_t \otimes g_t$ where f_t is the forget gate, c_{t-1} is the previous long term cell state, i_t is the input gate and g_t is the output from the main layer (4.10)

A copy of the updated long term state goes through a tanh function before being filtered by the output gate, o_t , to generate the current short term hidden state h_t which is defined as:

$h_t = o_t \otimes tanh\ (c_t)$ where o_t is the output gate and c_t is the long term state (4.11)

Once h_t has been generated from c_t , previous short term hidden state h_{t-1} and current input vector x_t will pass through the main layer resulting in the output:

$$g_t = tanh(W_g \times x_t + U_g \times h_{t-1} + b_g) \quad (4.12)$$

Gate controllers which are activated with a logistic function will filter what should be used from c_t and also what to modify based on h_{t-1} and x_t. Forget gate f_t determines what to remove, input gate i_t decides which parts of g_t will be added to c_t and output gate o_t controls what to output from c_t into h_t and/or cell output y_t .

Forget gate is defined as:

$f_t = \sigma\ (W_f \times x_t + U_f \times h_{t-1} + b_f)$ where W_f is the weight matrix for the layer connected to the input vector x_t and U_f is the weight matrix for the layer connected to the previous short term hidden state h_{t-1} and b_f is the bias term for this layer (4.13)

Input gate is defined as:

$i_t = \sigma\ (W_i \times x_t + U_i \times h_{t-1} + b_i)$ where W_i is the weight matrix for the layer connected to the input vector x_t and U_i is the weight matrix for the layer connected to the previous short term hidden state h_{t-1} and b_i is the bias term for this layer (4.14)

The output gate is defined as:

$$o_t = \sigma\,(W_o \times x_t + U_o \times h_{t-1} + b_o)$$

where W_o is the weight matrix for the layer connected to the input vector x_t and U_o is the weight matrix for the layer connected to the previous short term hidden state h_{t-1} and b_o is the bias term for this layer (4.15)

Overall, this method is able to use both short term and long term states to determine patterns in time series and also adjust these states at different time points as new input data is collected.

Table 4.6

Pseudocode for LSTM

Input: current input vector x_t, previous short term hidden state h_{t-1}, previous long term cell state c_{t-1}

Output: current long term state c_t, current short term state h_t, cell output y_t

Previous long term state modified into current short term state

1. c_{t-1} enters forget gate f_t to remove information from c_{t-1}
2. c_{t-1} gains information selected by input gate i_t resulting in c_t

   $c_t = f_t \otimes c_{t-1} + i_t \otimes g_t$
3. c_t copied and passes through output gate o_t which results in

   $h_t = o_t \otimes tanh\ (c_t)$

Passing previous short term state through main layer and gate controllers

4. x_t and h_{t-1} analyzed in main layer resulting in output

   $g_t = tanh(W_g \times x_t + U_g \times h_{t-1} + b_g)$
5. Using logistic function, gate controller will open if the output is 1 and close if the output is 0
6. Forget gate f_t determines what should be removed from c_t

   $f_t = \sigma\ (W_f \times x_t + U_f \times h_{t-1} + b_f)$
7. Input gate i_t decides which parts of g_t will be added to c_t

   $i_t = \sigma\ (W_i \times x_t + U_i \times h_{t-1} + b_i)$
8. Output gate o_t controls what to output from c_t into h_t and/or y_t

   $o_t = \sigma\ (W_o \times x_t + U_o \times h_{t-1} + b_o)$

   $y_t = h_t = o_t \otimes tanh\ (c_t)$

4.1.7 *Multilayer Perceptron (MLP)*

Multilayer perceptron is an artificial neural network (ANN) with multiple layers including an input layer and hidden layers which include a bias neuron and an output layer. It is also referred to as deep neural network (DNN) when there is more than one hidden layer and also backpropagation neural network (BPNN). During the forward pass, each neuron in each layer will calculate a total net input which is the sum of the product of corresponding initial weights and inputs along with adding a bias term before using an activation function to generate an output at each layer until the output layer is reached. Table 4.7 outlines the pseudocode for this method.

$net_h = w_k * i_j + w_{k+1} * i_{j+1} \ldots + b$ where h is a specific hidden layer neuron, w is the weight associated with a specific input neuron and hidden neuron, i is the value of the input neuron and b is the corresponding bias for the hidden layer neuron (4.16)

$net_o = w_k * h_j + w_{k+1} * h_{j+1} \ldots + b$ where o is a specific output layer neuron, w is the weight associated with a specific hidden neuron and output neuron, h is the value previously calculated for the hidden neuron and b is the corresponding bias for the output neuron (4.17)

After calculating total error, backpropagation from the output layer to the input layer will reduce error by adjusting the weights between neurons of different layers.

$E_{total} = \sum \frac{1}{2}(target - output)^2$ total error for the each output neuron (4.18)

$\frac{\partial E_{total}}{\partial w_k} = \frac{\partial E_{total}}{\partial out_n} * \frac{\partial out_n}{\partial net_n} * \frac{\partial net_n}{\partial w_k}$ chain rule to determine how much total error will change when weight is adjusted with n referring to a specific hidden or output neuron (4.19)

$\frac{\partial E_{total}}{\partial out_h} = \frac{\partial E_{ok}}{\partial out_h} + \frac{\partial E_{ok+1}}{\partial out_h}$ derivative of total error with respect to a specific hidden neuron based on multiple output neurons (4.20)

$\frac{\partial E_o}{\partial out_h} = \frac{\partial E_o}{\partial net_o} * \frac{\partial net_o}{\partial out_h}$ derivative of error for an output neuron with respect to the output of a hidden neuron (4.21)

$\frac{\partial E_o}{\partial net_o} = \frac{\partial E_o}{\partial out_o} * \frac{\partial out_o}{\partial net_o}$ derivative of error for an output neuron with respect to the total net input of the same output neuron (4.22)

$w_{new} = w_{old} - \eta * \frac{\partial E_{total}}{\partial w_k}$ new adjusted weight where η is the learning rate (4.23)

Although LSTM is expected to perform better than MLP, neural networks are considered to be black box techniques making it difficult to understand how an output is generated so including MLP as a deep learning model to compare with LSTM can help with identifying the most suitable hyperparameters and also look into the tradeoff of accuracy and prediction time between simpler models and more complex deep models (Khalid et al., 2019).

Table 4.7

Pseudocode for MLP

Input: i as input neurons, w as weights between neurons of different layers, b as the bias term, h as hidden layer neurons and o as output layer neurons
Output: updated weights and total error
Forward Pass

1. Set initial weights between neurons of different layers and bias term
2. Calculate total net input for each neuron in the hidden layer
   $net_h = w_k * i_j + w_{k+1} * i_{j+1} \ldots\ldots + b$
3. Insert total net input of hidden neuron into activation function to generate hidden neuron output
4. Calculate total net input for each neuron in the output layer
   $net_o = w_k * h_j + w_{k+1} * h_{j+1} \ldots\ldots + b$
5. Insert total net input of output neuron into activation function to generate output neuron output

Backpropagation

6. Calculate total error for each output neuron
   $E_{total} = \sum \frac{1}{2}(target - output)^2$
7. Calculate derivative of total error with respect to a specific weight
   $\frac{\partial E_{total}}{\partial w_k} = \frac{\partial E_{total}}{\partial out_n} * \frac{\partial out_n}{\partial net_n} * \frac{\partial net_n}{\partial w_k}$
8. Adjust weight to reduce total error
   $w_{new} = w_{old} - \eta * \frac{\partial E_{total}}{\partial w_k}$

Repeat until total error stops decreasing

4.1.8 SARIMAX

Seasonal Auto-Regressive Integrated Moving Average with eXogenous factors, named as SARIMAX, is a classic statistical model for time-series forecasting. It is used as a benchmark to compare with machine learning and deep learning models. Different from the ARIMA model without seasonality, this model can detect the trend and seasonality. Since it has a backshift movement during the prediction, the model has a wide range of forecasting (Rob J Hyndman & George, 2018).

The battery data has shown the cyclical behaviors as seen in chapter 3. However, there are significant trends for the capacity loss over time. Also the battery status is affected by the temperature, the charging speed, and the discharge loading profile. It is not marked as stationary time-series data and can have a long-term prediction by this statistical model.

The ARIMA model consists of three parts: 1) auto-regression; 2) moving average; and 3) integration. The auto-regression model is can be written as:

$$y_t = c + \phi_1 y_1 + \phi_2 y_2 + \ldots + \phi_n y_n + \varepsilon \tag{4.24}$$

where ϕ is a parameter indicating the time series pattern.

The moving average function is written as:

$$y_t = c + \varepsilon_1 + \theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q} \tag{4.25}$$

where ε is the white noise.

Combined by differencing and multiplying the non-seasonal term to the additional seasonal term, we can obtain the SARIMAX model for prediction.

Table 4.8

Pseudocode for SARIMAX

Input: $X \in \{training\ data\}, X^{*} \in \{test\ data\}$
Output: $\widehat{y}$ battery capacity
1. For dataset D, Time Series Decomposition
2. Dickey-Fuller Test to detect stationary characteristic
3. Optimize non-seasonal parameters(p, d, q) and seasonal parameters(P, D, Q)
4. In-sample prediction for validation
5. In-sample prediction for X^{*}

4.2 Model Supports

4.2.1 Tools

The offline model training and evaluation work is done on the local PC. The coding language used is Python. Python will be run on a Jupyter notebook. The libraries for the deployment and execution for all the models include:

- Pandas: for data exploration and analysis;
- Numpy: for working with arrays, matrices, and scientific computing;
- Matplotlib: for visualization and plotting;
- Scikit-learn: for machine learning models implementations
    - sklearn.neighbors.KNeighborsClassifier for KNN
    - sklearn.svm.SVR for SVR
    - sklearn.model_selection.GridSearchCV
    - sklearn.model_selection.cross_val_score and sklearn.cross_validation.KFold for cross-validation
    - sklearn.metrics for model evaluation using confusion_matrix, classification_report, accuracy_score, recall_score, precision_score
- xgboost: for XGboost model
- Tensorflow Keras version 2.6.0: for deep learning models
    - keras.models Sequential for initializing model
    - keras.layers Embedding, LSTM, Bidirectional, Dense, Flatten, Dropout, BatchNormalization, Activation for LSTM and MLP
    - keras.optimizers SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax
- statsmodels: for statistic benchmark model

- statsmodels.tsa.seasonal for time series decomposition
- statsmodels.tsa.stattools.adfuller for Dickey-Fuller Test
- statsmodels.api for error calculation
- statsmodels.tsa.statespace.sarimax for SARIMAX

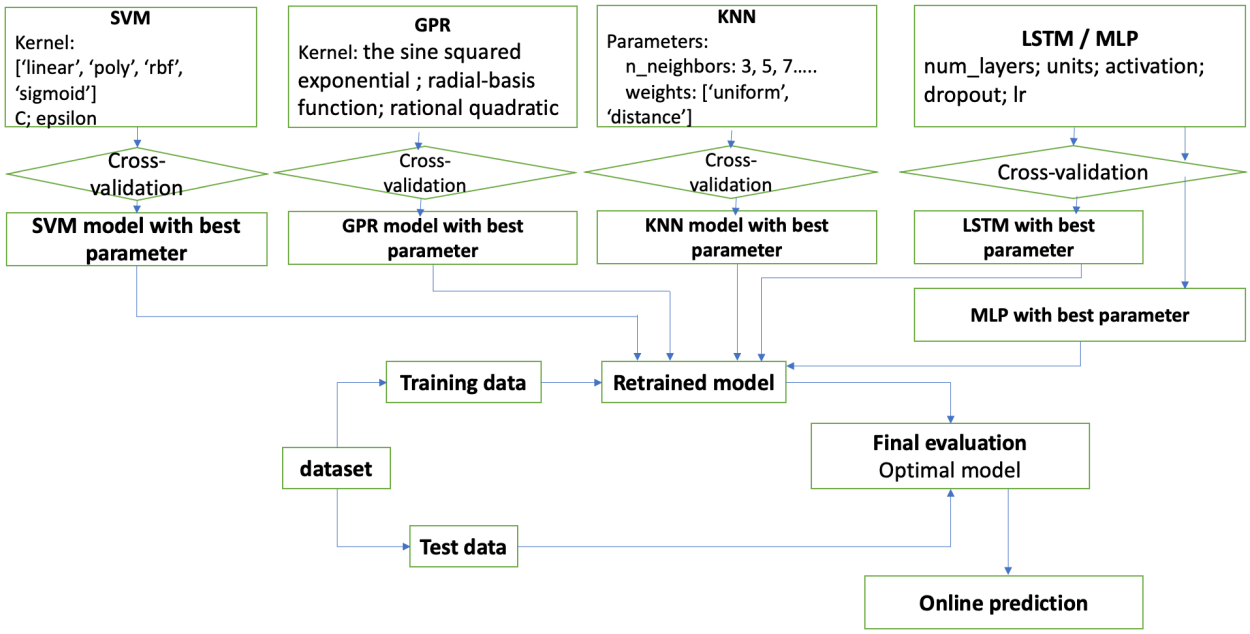
The online real-time model implementation and prediction is done on Google Cloud Platform. On GCP, the models are run with Python on Google AI Platform.

4.2.2 Workflow

The process of model evaluation and selection for our 5 proposed models will follow the workflow shown in Figure 4.2.

Figure 4.2

Flowchart of Machine Learning for SoC Estimation



During model testing, each model will be iterated several times, with different hyperparameters for each model variation in order to find the most optimal model for each method. For XGBoost, the tree based parameters such as minimum leaf, maximum depth, learning rate, and the gradient methods are defined. For SVM, each iteration will use a different kernel, C, or epsilon. For KNN, each iteration uses a different K and will either use an unweighted or weighted distance. For LSTM or MLP, different number of layers, neuron units per layer, activation function, learning rate and dropout or no dropout is tested in each iteration to find the best parameters. For the SARIMAX model, the non-seasonal and the seasonal parameters are chosen in different combinations. The best models from each method will be evaluated based on the metrics listed in section 4.4, and the model with the best performance will be chosen to be deployed on the online real time platform.

4.3 Model Comparison and Justification

The summarized properties of all models proposed are listed in Table 4.9. All models are able to handle non-linear battery data.

XGBoost has fast training speed due to the sequential building of small trees. It can fit the data and is not limited to the data type. The SVR model can adopt kernel functions, which means they are flexible when working with high-dimensional data, but this can also be a limitation that results in models that may lack adaptability. The kernel design and the gated network means the GPR model is good at adapting to large datasets for time-series predictions. SVR and KNN both use distance-based predictions. The goal of SVR is to create a clear margin so performance would be strongly and negatively influenced by noisy data. KNN is easy to interpret after selecting the best k factor but it does not compute efficiently when given a large dataset. LSTM is suitable for forecasting and time series predictions but may be too time

consuming for real time capacity predictions. In contrast, MLP is less complex allowing for faster training and prediction but is not as suitable for time series predictions. SARIMAX is applied straightforwardly for time series forecasting. It can detect the trend and seasonality.

Table 4.9

Model Comparisons

Model	**Advantage**	**Disadvantage**
XGBoost	Great learning skill, fast learning speed	Not eligible for long term prediction
SVR	Flexibility, play with the kernel function Suitable for small datasets Efficient for high-dimensional data	Lower performance when the target data are overlapping Less efficiency with large datasets
KNN	Easy to interpretable Suitable for the time-series prediction	Highly dependent on k factor Inefficient computation
LSTM	Working well with the sequential data Handles vanishing gradient problem Able to forecast predictions and handle both short term and long term states	Issues with exploding gradients or time consuming convergences Requires standardized input data between 0 and 1
MLP	Suitable for nonlinear problems Backpropagation reduces error	Not as suitable for time series
SARIMAX	Classic time-series forecasting, high accuracy for stationary and non-stationary data	Time consuming for optimization, high demand for manually analysis

4.4 Model Evaluation Methods

The goal is to find the model with the most accurate predictions for battery capacity. The basic process is that the test dataset contains instances with correct target values, and these data instances also have a set of predictions by a model (Kelleher et al., 2015). The measurement of prediction accuracy is how these predictions match the correct target values. Since battery capacity is a continuous target, this project selects the evaluation tools for continuous targets: MAE (mean absolute error), RMSE (root mean squared error), and R^2 coefficient.

4.4.1 MAE

MAE is the mean absolute error. MAE is the mean of the absolute difference between prediction and actual value (Wikipedia Contributors, 2019). The formula of MAE of N values is:

$$MAE = \frac{\sum_{i-=1}^{N} \left| Predicted_i - Actual_i \right|}{N} \tag{4.26}$$

MAE is a common evaluation tool of prediction error in time series analysis.

4.4.2 RMSE

RMSE is the root mean squared error (Kelleher et al., 2015). RMSE is the root of the mean squared difference between prediction and actual value. The formula of RMSE of N values is:

$$RMAE = \sqrt{\frac{\sum_{i-=1}^{N} (Predicted_i - Actual_i)^2}{N}} \tag{4.27}$$

RMSE tells how far the actual value is from the prediction points. In other words, it shows how concentrated the actual value is around the predictions.

4.4.3 R^2 coefficient

R^2 coefficient is a domain independent evaluation tool (Kelleher et al., 2015). R^2 coefficient compares the performance of a model with the performance of an imaginary model which always predicts the mean value of the dataset. The formula is:

$$R^2 = 1 - \frac{\sum_{i-=1}^{N} (Predicted_i - Actual_i)}{\sum_{i-=1}^{N} (Actual_i - Actual_{mean})} \tag{4.28}$$

R^2 has a range [0, 1). The larger the value, the better the model performance.

4.4.4 Comparisons

MAE, RMSE and R^2 coefficient are all common performance evaluation tools for continuous targets (Kelleher et al., 2015). MAE and RMSE work directly with errors between prediction and actual value (Chai & Draxler, 2014). However, while MAE is a simple average and gives each error the same weight, RMSE involves squaring the error and gives larger absolute errors more weight (Pontius et al., 2007). Compared with the other two, R^2 coefficient not only considers the errors, but measures the portion that variation of dependent variable is decided by independent variables. R^2 coefficient has a baseline model that always predicts the average of actual values.

The results for MAE and RMSE are arbitrary and can range from 0 to infinite. Lower result, which represents a smaller error, is better. By definition, RMSE's result is always larger than MAE (Kelleher et al., 2015). By contrast, R^2 coefficient's result range is [0, 1). Larger results represent better results.Since MAE, RMSE and R^2 coefficient have been used as evaluation tools for many years and there is no consensus on the best one, this paper applies all three methods for evaluations and analysis.

4.5 Model Validation and Evaluation Results

Validation for SVR can be tuned to optimize hyperparameters using K-fold cross validation and GridSearchCV. The hyperparameters for each model tested in GridSearchCV is included in Table 4.10. The mean cross validation R^2 or cross_val_score from the model_selection module can also be used as a rough measurement on SVR.

Due to the large size of the whole dataset and the constraints of the local machine's computing power, the validation and evaluation for model KNN does not work on the whole dataset directly. However, the validation process is similar to the idea of K-fold cross validation. The cycles are divided into 10 sets. Each set is split into training data and test data and the average R square is calculated for comparison among models. Meanwhile, some other experiments are also designed with KNN to explore deeper about its performance.

For models created using keras, suchs as LSTM and MLP, KerasTuner can be used to tune hyperparameters such as number of layers, number of neuron units in each layer, type of activation function used, if dropout is implemented, and learning rate.

After tuning the hyperparameters for each model, evaluation will be done based on MAE, RMSE and R^2 coefficient generated after testing the validation dataset. The top 3 models with the low error and high R^2 score will be used on the testing data and the same evaluation metrics will be compared to determine the most suitable model for estimating capacity, SoC, plug-in duration, and remaining mileage.

Table 4.10

Hyperparameters for Proposed Models

Model	Hyperparameter Details
XGBoost	n_estimators Min_child_weight gamma subsample colsample_bytree max_depth learning_rate alpha min_child_weight
Support Vector Regression	kernel: [‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’] C epsilon
KNN	n_neighbors weights: [‘uniform’, ‘distance’]
LSTM and MLP	num_layers units activation dropout lr
SARIMAX	(p, d, q) (P, D, Q) period

4.5.1 XGBoost Results

The cross-validation with the parameter optimization is conducted using the embedded mean squared error. Each training time is ranging from 1.6s to 20s depending on the learning rate eta, the partition of the leafs gamma, and the subsample ratio of the training instances. The smaller learning rate, the more conservative algorithm, and the larger ratio of training instances can lead to higher accuracy but consumes more time.

Considering the cost efficiency and the performance of the prediction, the optimal parameters are picked for the single cycle prediction and multi cycles prediction. As seen in Figure 4.2, the first 74 cycles are trained and the 75th cycle is predicted. The prediction is very close to the observed capacity values. The metrics of RSME, MAE, and R2 score are 0.0707, 0.0582, and 0.9783 respectively. The running time is 13 seconds.

The long-term prediction is carried out by training the entire training dataset while the predictions are on the cycling at room temperature or in various environments.The prediction shows in Figure 4,3 and Figure 4.4 with the metrics of RSME, MAE, and R2 score of 0.1147, 0.1042, and 0.9545 for battery capacity at room temperature, and 0.2254, 0.3573, and 0.5107 for battery capacity at various temperatures under multi loading profiles.

Figure 4.2

XGBoost Battery Capacity Prediction on a Single Cycle at Room Temperature

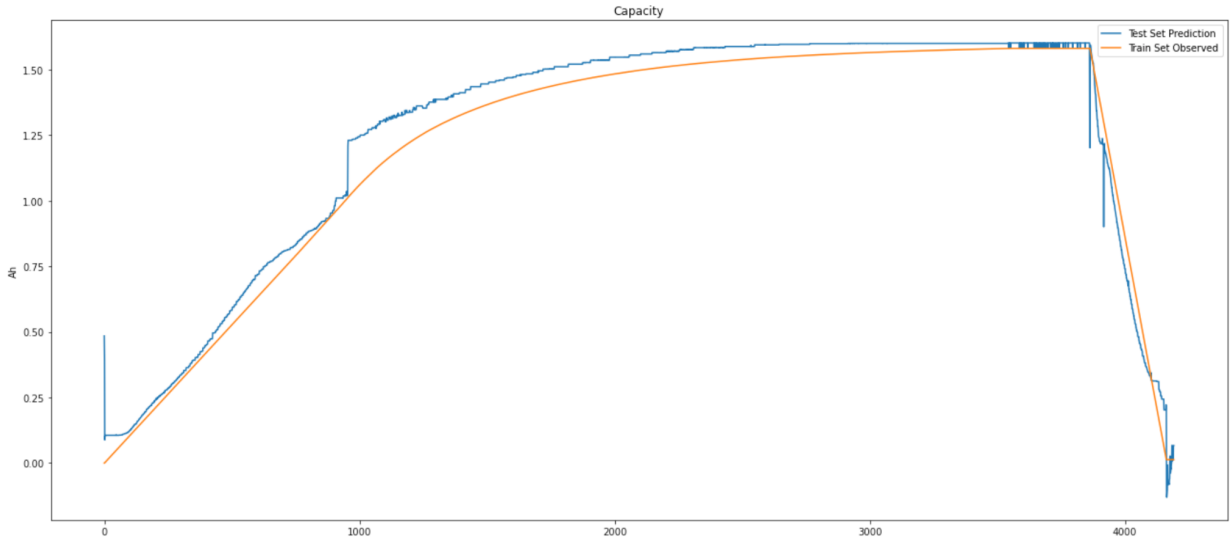


Figure 4.3

XGBoost Long-term Prediction of Battery Capacity at Room Temperature

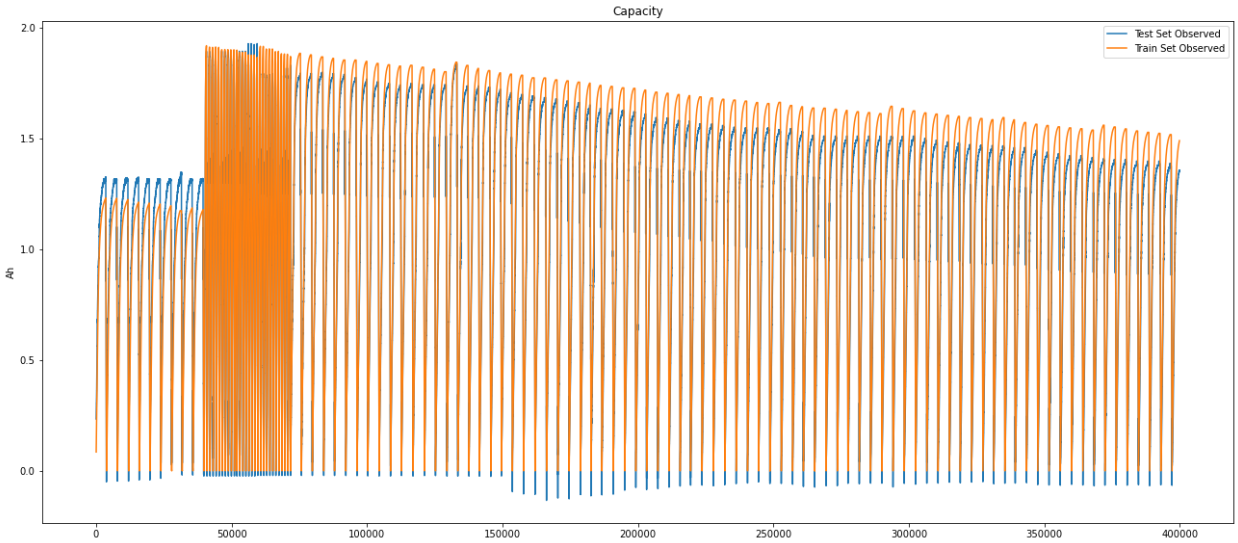
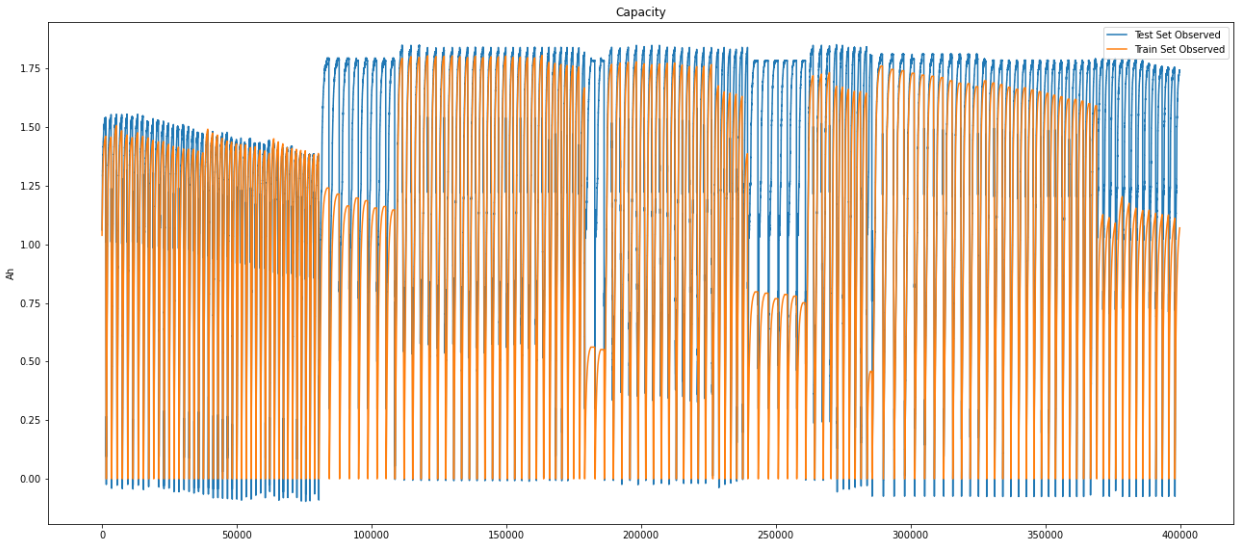


Figure 4.4

XGBoost Long-term Prediction of Battery Capacity at Various Temperatures Under Multi Loading Profiles



Overall, the XBGoost model has exhibited fast training and prediction. It can forest short-term capacity change as well as the long-term prediction under stable environmental conditions. However, it shows inadequate performance when handling varied environments such as low/high temperature or multi-loading profiles. Also, the prediction tends to shift away from the observed values when extending the prediction range. This phenomenon indicates the need for self-adjusting models.

4.5.2 SVM Results

The SVM's results are heavily dependent on the kernel used to estimate values. Changing the kernel of an SVR drastically changes its performance metrics such as accuracy and speed of training and predictions. The four kernels considered to be tested are Radial Basis Function (RBF), Linear, Sigmoid, and Polynomial. In early testing using small sample sizes, the linear kernel achieved very low results in both speed and accuracy. The sigmoid function also had terrible accuracy with root mean square error values in the millions. The two kernels that achieved decent results were RBF and Polynomial.

Table 4.11 outlines the error derived from the RBF and polynomial kernels. In a direct comparison between these two kernels, RBF performed better in most metrics. RBF had a lower error and trained more than 33% faster than the polynomial kernel. However, the polynomial kernel achieved almost four times as fast prediction speeds. This is important when designing a dashboard for a user as large amounts of predictions may increase the latency in the final implementation.

Table 4.11

SVR Comparison Between Polynomial and RBF Kernels

	Radial Basis Function	**Polynomial**
	Accuracy	
MAE	0.3908	0.4413
MSE	0.2391	0.3148
RMSE	0.489	0.561
	Speed (per 1000 points)	
Training	0.156 s	0.215 s
Prediction	0.234 s	0.066 s

According to our prediction and use case, prediction speed isn't a priority over accuracy as the rate of incoming data would likely be around one second. As such, RBF is chosen for any future testing and evaluation in regards to comparing SVR with other models.

Figure 4.5 outlines the accuracy of the prediction made from the RBF kernel and the actual values. This plot reveals a few errors made by SVR. SVR begins to predict negative values even when the data has none. This is likely to greatly affect the error of any SVR function. Improving this aspect of the model may produce significantly greater results. While a simple prediction rule changing negatives into zero values may help with errors over all, SVR does not seem well suited for this dataset as a whole.

Figure 4.5

SVR Predictions and Actual Values

4.5.3 KNN Results

KNN's model implementation includes four parts and the main goal is to evaluate KNN's accuracy and efficiency. For accuracy, first applying KNN inside one single cycle to make predictions; then a set of cycles are used to predict some other cycles; and then a set of cycles are used to predict several single cycles to see how the accuracy changes over time. For efficiency, samples of different sizes are used to see how the run time changes.

Prediction in one single cycle is used to see how KNN works to predict the capacity inside a cycle. 16 cycles are selected, including 1, 11, 21, …, 151. Each cycle is split into training data and test data, as shown in Figure 4.6. Best parameters are found through GridSearchCV: number of neighbors is 3 or 5; parameter "Weights" is "distance". Each cycle is trained with its own best parameters. As a result, the average R2 is 0.998, which means that inside one cycle, KNN has an excellent performance.

Figure 4.6

KNN prediction in one single cycle

training	test									
1		2	3	4	5	6	7	8	9	10
11		12	13	14	15	16	17	18	19	20
21		22	23	24	25	26	27	28	29	30
31		32	33	34	35	36	37	38	39	40
41		42	43	44	45	46	47	48	49	50
51		52	53	54	55	56	57	58	59	60
61		62	63	64	65	66	67	68	69	70
71		72	73	74	75	76	77	78	79	80
81		82	83	84	85	86	87	88	89	90
91		92	93	94	95	96	97	98	99	100
101		102	103	104	105	106	107	108	109	110
111		112	113	114	115	116	117	118	119	120
121		122	123	124	125	126	127	128	129	130
131		132	133	134	135	136	137	138	139	140
141		142	143	144	145	146	147	148	149	150
151		152								

Prediction between cycles is done in different sets of cycles. The 1 to 150 cycles are divided into 10 sets. Each set has 10 cycles as training data, 5 cycles as test data, as shown in Figure 4.7. Same as prediction inside one single cycle, the best parameters are first found with the number of neighbors is 5 or 7 and “Weights” is “distance” . Then each cycle uses its own best parameters to make predictions. As a result, average R2 is 0.92, not as good as prediction in one single cycle.

Figure 4.7

KNN prediction between cycles

CYCLES	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 8	Set 9	Set 10
TRAINING (10 cycles)	1	2	3	4	5	6	7	8	9	10
	11	12	13	14	15	16	17	18	19	20
	21	22	23	24	25	26	27	28	29	30
	31	32	33	34	35	36	37	38	39	40
	41	42	43	44	45	46	47	48	49	50
	51	52	53	54	55	56	57	58	59	60
	61	62	63	64	65	66	67	68	69	70
	71	72	73	74	75	76	77	78	79	80
	81	82	83	84	85	86	87	88	89	90
	91	92	93	94	95	96	97	98	99	100
TEST (5 cycles)	101	102	103	104	105	106	107	108	109	110
	111	112	113	114	115	116	117	118	119	120
	121	122	123	124	125	126	127	128	129	130
	131	132	133	134	135	136	137	138	139	140
	141	142	143	144	145	146	147	148	149	150
	151	152								

KNN's performance between cycles is not as good as inside one cycle. Since the battery cycle has a time series characteristic, one possible reason is that the performance of KNN models goes down over time. To prove this assumption, another experiment is implemented. The 1 to 150 cycles are also divided into 10 sets and each set has 15 cycles. Among the cycles, 10 cycles are training data and are used to predict the other 5 cycles separately, as shown in Figure 4.8. In this experiment, R squared of cycle 101 to cycle 150 are recorded and shown in Figure 4.9. As cycle number increases, the R squared decreases, which means the prediction is more and more inaccurate over the time. In conclusion, KNN is good for prediction inside one cycle. However, it may not be a good model for this time series data.

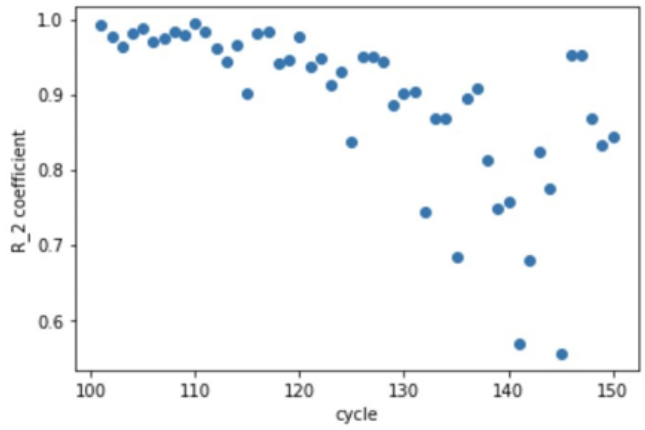
Figure 4.8

KNN prediction over time

CYCLES	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 8	Set 9	Set 10
TRAINING (10 cycles)	1	2	3	4	5	6	7	8	9	10
	11	12	13	14	15	16	17	18	19	20
	21	22	23	24	25	26	27	28	29	30
	31	32	33	34	35	36	37	38	39	40
	41	42	43	44	45	46	47	48	49	50
	51	52	53	54	55	56	57	58	59	60
	61	62	63	64	65	66	67	68	69	70
	71	72	73	74	75	76	77	78	79	80
	81	82	83	84	85	86	87	88	89	90
	91	92	93	94	95	96	97	98	99	100
TEST (5 cycles seperately)	101	102	103	104	105	106	107	108	109	110
	111	112	113	114	115	116	117	118	119	120
	121	122	123	124	125	126	127	128	129	130
	131	132	133	134	135	136	137	138	139	140
	141	142	143	144	145	146	147	148	149	150
	151	152								

Figure 4.9

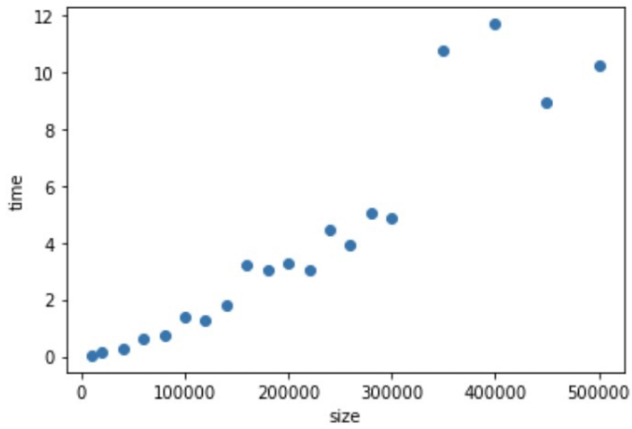
KNN prediction R_2 coefficient over time



Different sizes of samples are selected to test the prediction running time to evaluate the efficiency of the KNN model. As shown in Figure 4.10, as data size increases, the running time increases rapidly. One feature of KNN is that a lot of computation occurs during prediction. Although high speed processors can improve efficiency, it will also increase costs. It is worth mentioning that prediction is not a one-time expense, but an ongoing cost. If applying KNN in the industry, this will be a problem for either the user or the manufacturer. Due to the lack of efficiency without high speed processors and possible high costs, KNN is not the best choice.

Figure 4.10

KNN prediction running time comparison



4.5.4 LSTM Results

Out of all 5 models, LSTM had the best metrics. LSTM models require input data to be scaled between 0 and 1 and then the prediction output needs to be converted back using inverse scaling. Due to the slow speed of complex LSTM models, an univariate model was tested with an input data of arrays of the 1 previous capacity values prior to the capacity value being predicted but the resulting predictions were very inaccurate even for the first cycle which would negatively impact predictions of future cycles. The multivariate model includes 1 LSTM layer and 1 dense output layer with 5 input variables. Early stopping is used to avoid overfitting by terminating training when validation loss begins to increase.

Since it is a time series prediction, the dataset is converted by adding the lag time step for predicting one step ahead. 30 lags and 1 lag have applied that make the dataframe with 185 and 11 input variables respectively. The training time for 30 lags took 2763 seconds with 14 epochs with a training dataset of 1397467 rows and prediction time on the validation dataset took approximately 1 second for 200000 records. Training time was 1.48 s/1000 records and

prediction time was 0.05 s/1000 records. The metrics generated from the validation dataset include a RMSE of 0.0027, MAE of 0.0065 and a R2 score of 0.9990. The training time for 1 lags took 426 seconds with 14 epochs with a training dataset of 886268 rows and prediction time on the validation dataset took approximately 5 seconds for 295417 records. Training time was 6.78 s/1000 records and prediction time was 0.074 s/1000 records. The metrics generated from the validation dataset include a RMSE of 0.022, MAE of 0.012 and a R2 score of 0.996.

When comparing predictions for charge and discharge, both the prediction curves for capacity are very close to the actual curves. When comparing LSTM predictions on the validation set in Figure 4.12, the predictions are more accurate for cycles with higher capacity values but there is slight overprediction on cycles with lower capacity values. As seen in Figure 4.16, there is some overprediction of capacity initially at the start of the cycle where actual capacity is lower and then underprediction of capacity when the actual capacity is higher but the overall prediction trend matches the trend of the actual test data cycle

Table 4.12

LSTM model structure with 1 lag time steps and 1 time step ahead

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 50)                11400
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 11,451
Trainable params: 11,451
Non-trainable params: 0
```

Figure 4.11

Loss function converges for training and validation data with 1 lag time steps and 1 time step ahead

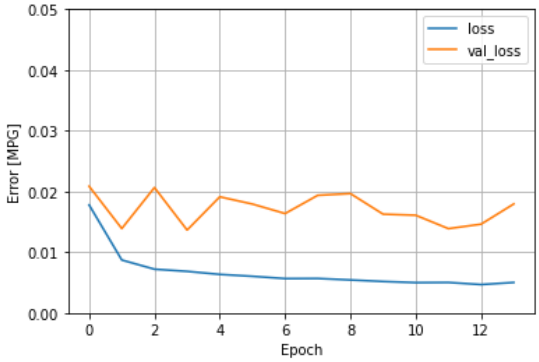


Figure 4.12

LSTM prediction of battery capacity with 1 lag timestep for validation data

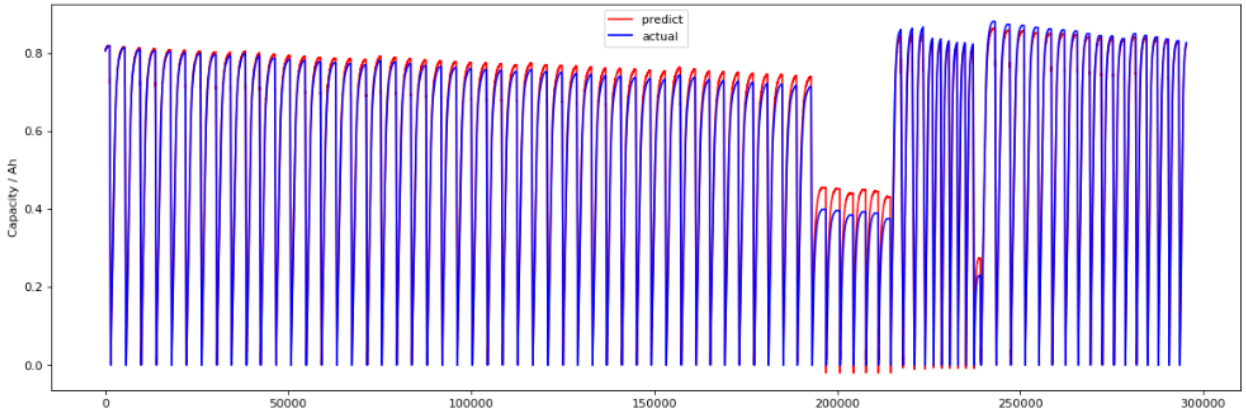


Figure 4.13

LSTM prediction of battery capacity with 1 lag timestep for test data

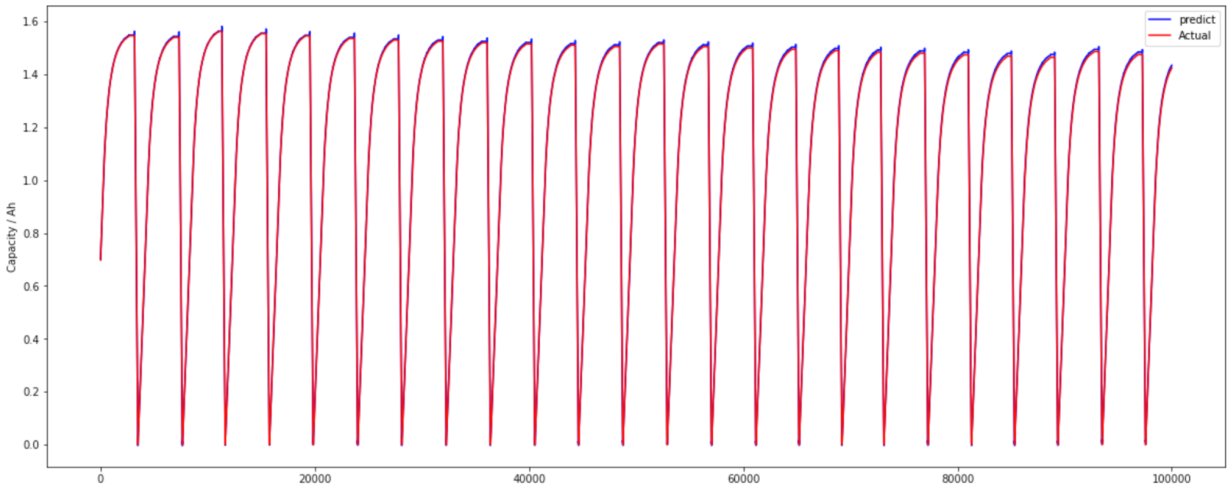


Table 4.13

LSTM model structure with 30 lag time steps and 1 time step ahead

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 50)                47200
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 47,251
Trainable params: 47,251
Non-trainable params: 0
```

Figure 4.14

Loss function converges for training and validation data with 30 lag time steps and 1 time step ahead

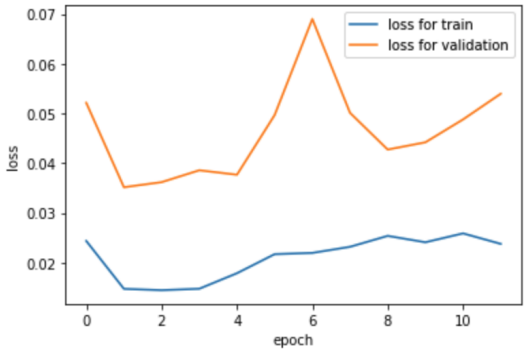


Figure 4.15

LSTM prediction of battery capacity with 1 lag timestep for test data

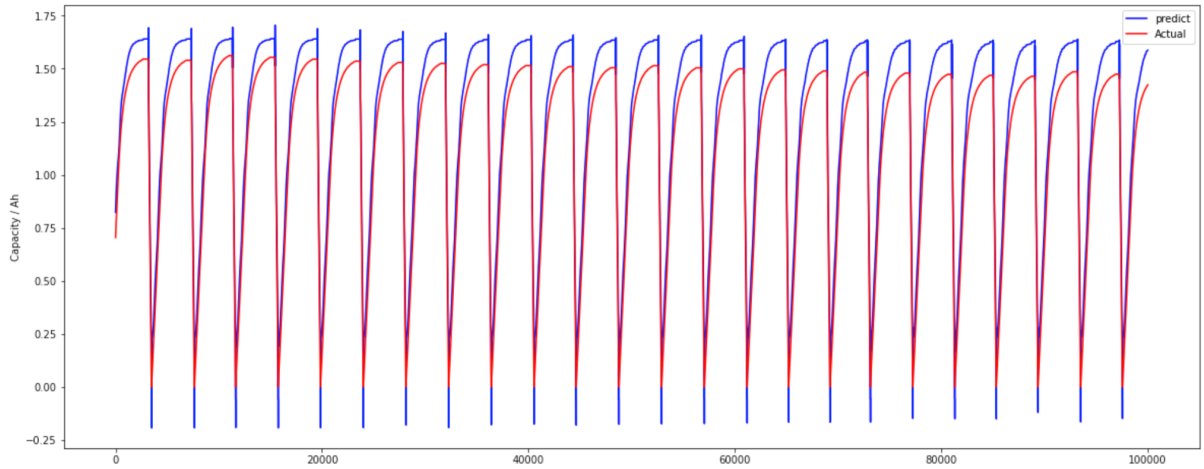
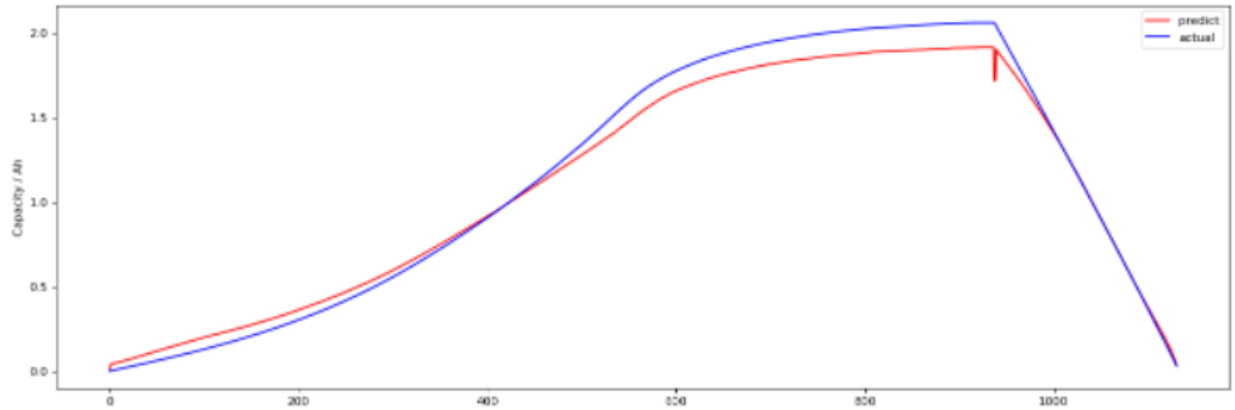


Figure 4.16

LSTM prediction of battery capacity with 1 lag timestep for one cycle of test data



4.5.5 MLP Results

The MLP model had a longer training time of 14.08 s/1000 records and shorter prediction time of 0.020 s/1000 records compared to the LSTM model but it also has less accuracy in the form of higher error metrics. The model consists of an input layer, two dense layers, two dropout layers and a flatten layer before the final output dense layer. The input layer has a shape of 30, 5 due to the reshaped input data that groups 30 records or approximately 90 second of data together to predict the capacity of the next time point and since there are 5 features. Of the activation functions available, Relu was chosen since it is fast while preventing the vanishing gradient problem and most importantly is used for predicting continuous target variables while sigmoid activation function can only be used for binary output variables and softmax activation function is only used for classification. Dropout layers were included after the first two dense layers to reduce overfitting so the model can make generalized predictions which will also speed up training and prediction time and use less memory. The flatten layer reshapes the output data before the output dense layer which doesn't use an activation function since the predicted capacity is a continuous value.

Table 4.14

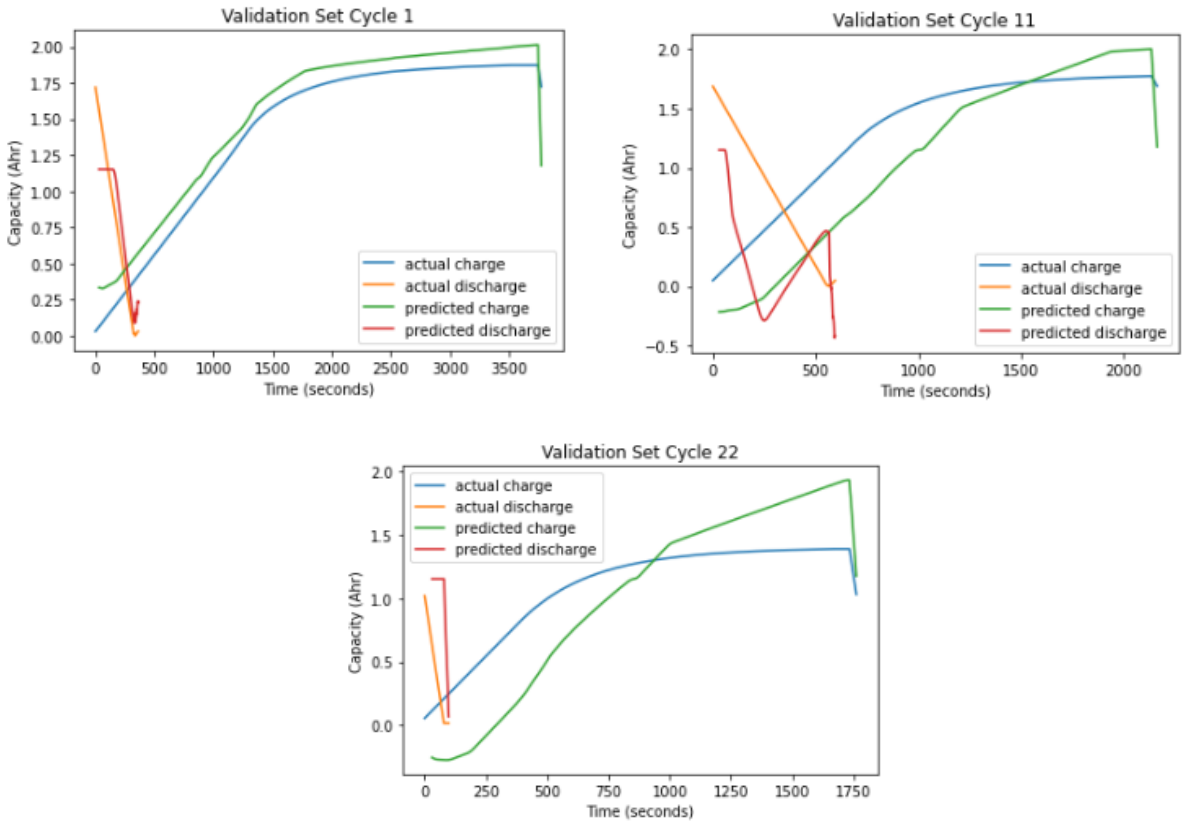
MLP model structure

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 30, 64)            384
_________________________________________________________________
dropout (Dropout)            (None, 30, 64)            0
_________________________________________________________________
dense_1 (Dense)              (None, 30, 32)            2080
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 32)            0
_________________________________________________________________
flatten (Flatten)            (None, 960)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 961
=================================================================
Total params: 3,425
Trainable params: 3,425
Non-trainable params: 0
```

The training time took 780 seconds for 10 epochs with a training dataset of 886269 rows and prediction time on the validation dataset took approximately 10 seconds for 295394 records. Training time was 14.08 s/1000 records and prediction time was 0.02 s/1000 records. The metrics generated from the validation dataset include a RMSE of 0.146, MSE of 0.021, MAE of 0.112 and a R2 score of 0.730.

Figure 4.17

Actual vs Predicted values on Validation set in select cycles



From the graphs comparing predicted values against actual values in select cycles in Figure 4.17, the predictions for earlier cycles or cycles with more records seems to be closer to actual values compared to later cycles or cycles with few records. The prediction of charge seems to be underestimated earlier in the cycle and then overestimated later in the cycle.

Figure 4.18

MLP prediction of battery capacity on validation data

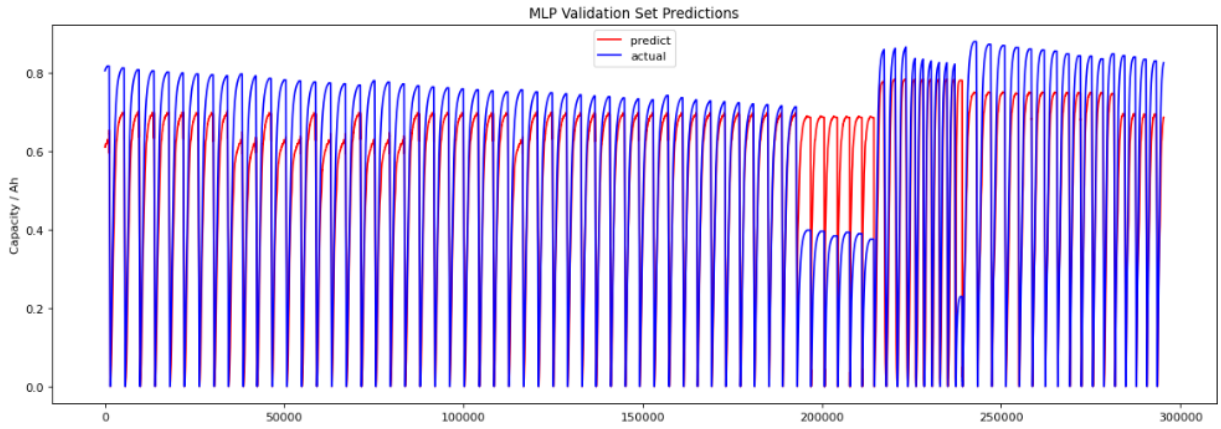
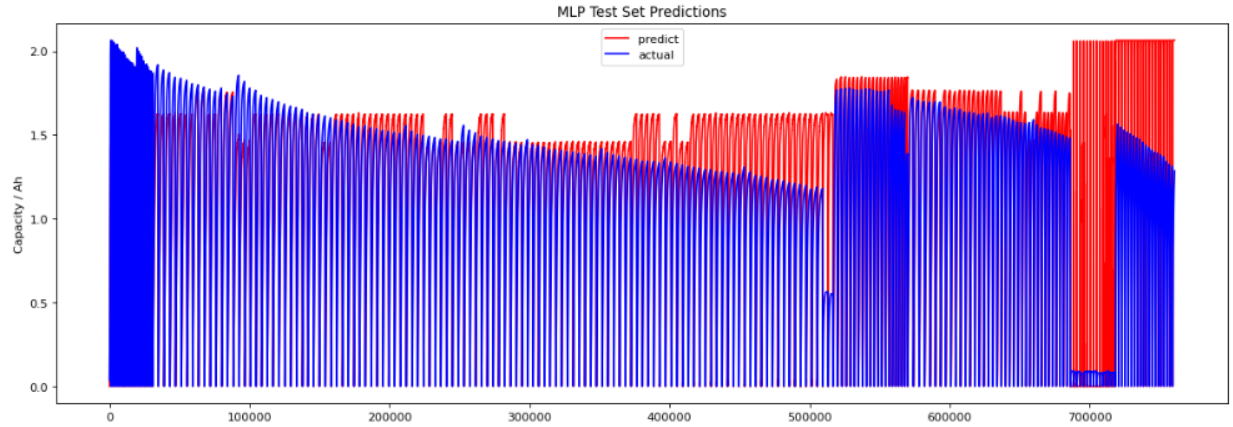


Figure 4.19

MLP prediction of battery capacity on test data

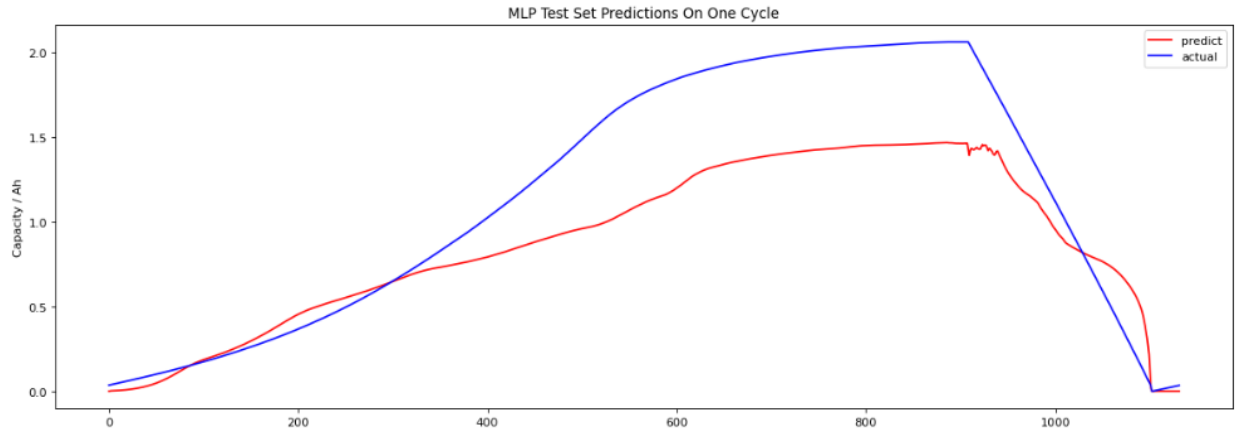


As seen in Figure 4.18, MLP underpredicts capacity for higher actual capacities in the validation set but overpredicts for lower actual capacities. This suggests that the MLP model is not as suitable for making predictions on cycles where capacity isn't fully charged. The predictions on the test set tend to underpredict actual capacity values for earlier cycles and then in later cycles the predicted capacity values are much higher than the actual capacity value as seen in Figure 4.19.

For one cycle of test data the predicted capacity value is slightly higher but still close to the actual capacity at the start of the cycle but in the middle of the cycle the predicted capacity is much lower than the actual capacity value. However, the shape of the prediction trend and highest capacity value prediction occurs close to the actual cycle with the highest capacity values as seen in Figure 4.20.

Figure 4.20

MLP prediction of battery capacity on once cycle of test data



4.5.6 SARIMAX Results

The data exploration has indicated cyclical behavior and the trend of decreasing capacity over time. So the decomposition of the data is critical for the time series forecasting. As seen in Figure 4.21 and Figure 4.22, the battery degrades with the decreasing trend over time no matter how the environment conditions change. There is no seasonality and the data is neat. The Dickey Fuller test results shows that this data is stationary because the test statistic p value is much smaller than the critical values at 1%.

With the input data analysis, the model is trained by iteration of different combinations of

parameters. The non-seasonal parameters and the seasonal parameters are in the range of 0 to 2 and the period changed due to the different charge and discharge speed. The optimal parameters for single environment is (0, 1, 0)×(1, 0, 1, 3) and (1 1, 1)×(0, 0, 0, 3)for varied environments. The short-timer prediction is to forecast one cycle ahead. The running time is 51 seconds.

The in-sample prediction is conducted using the entire training dataset as shown in Figure 4.23 with the metrics of RSME, MAE, and R2 score of 0.0002, 6.2001e-8 and 0.9998 for battery capacity. However, the performance of the out-of-sample prediction is too poor to evaluate. The result in Figure 4.24 shows the prediction of the 75th cycle while 74 cycles have been trained with the optimal parameter. The main reason would be that the data is stationary and cyclical with changed periods so the trend is not very predictable.

Figure 4.21

Time-series Decomposition for Room Temperature Data

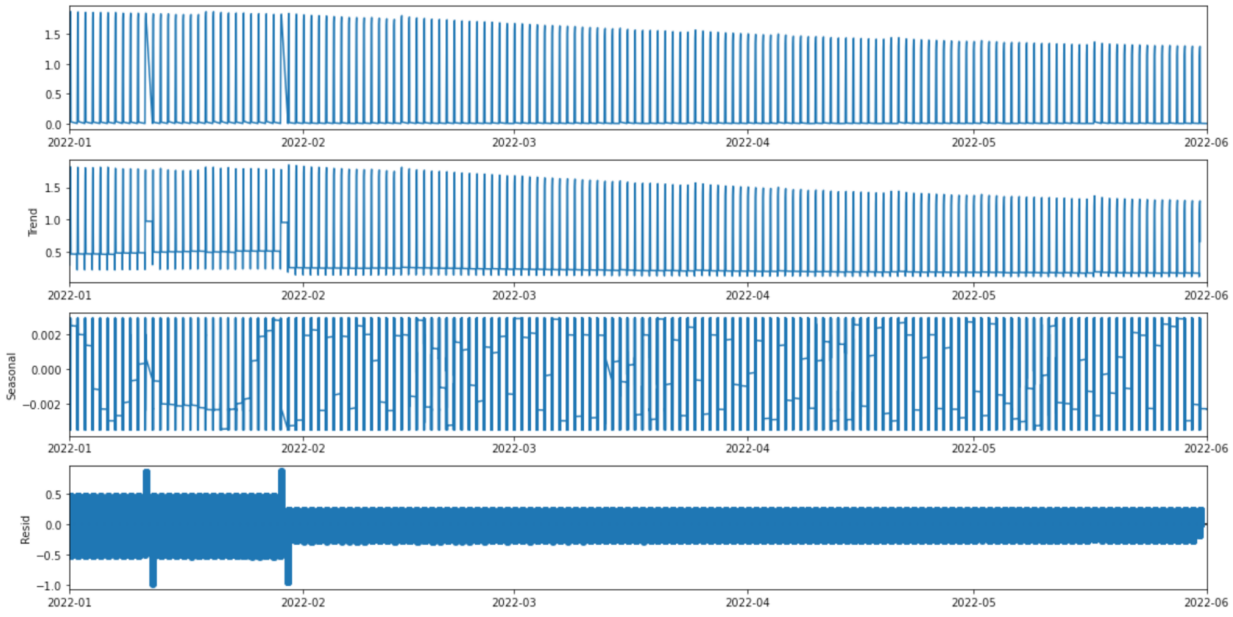


Figure 4.22

Time-series Decomposition for Data at Various Temperatures under Multi Loading Profiles

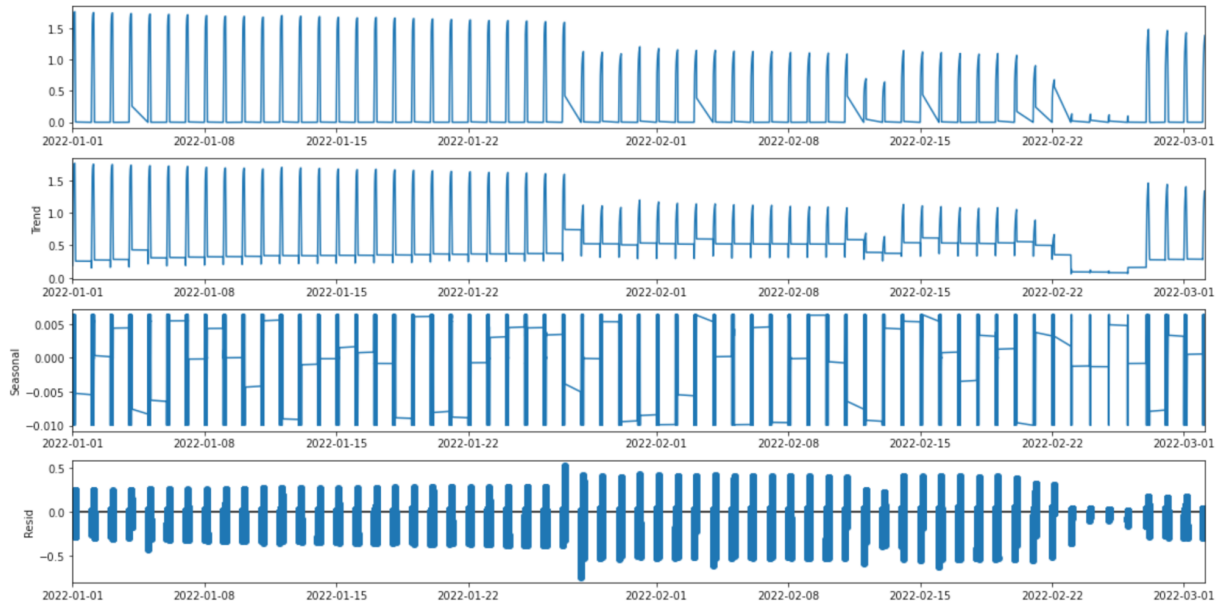


Figure 4.23

SARIMAX in-sample prediction of battery capacity at various temperatures under multi loading profiles

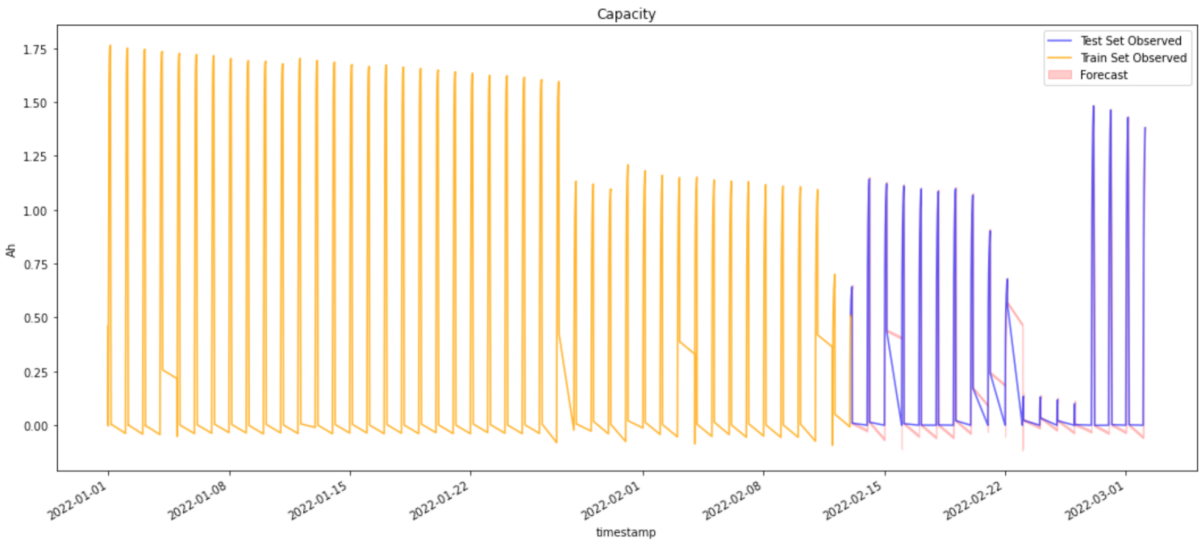
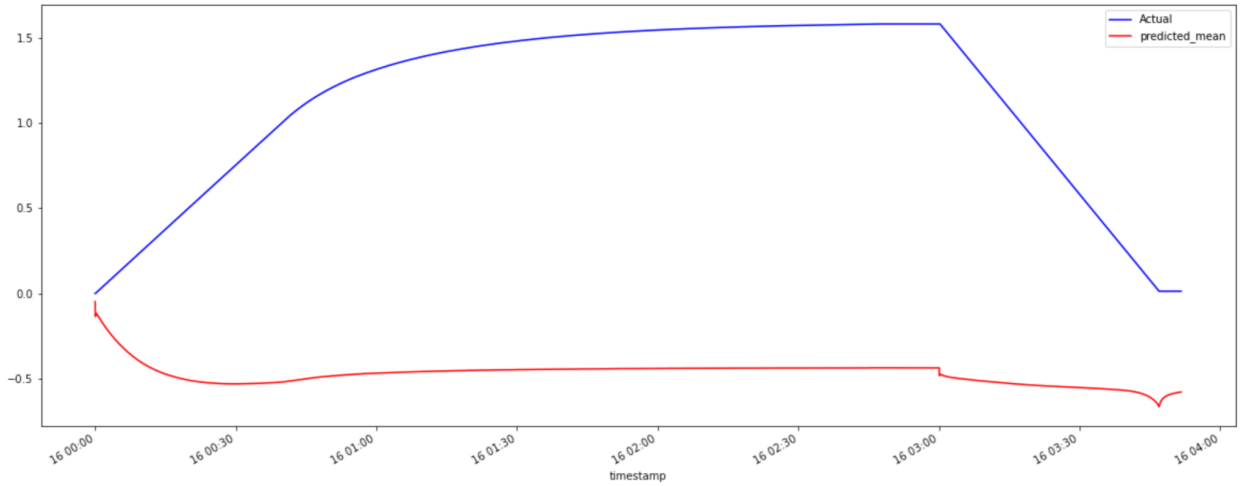


Figure 4.24

SARIMAX out-of-sample prediction of battery capacity on one cycle ahead



Since the project focus is on forecasting upcoming battery status which is out-of-sample, the machine learning and deep learning models significantly outperformed the statistical model.

5. Data Analytics System

5.1 System Requirements Analysis

5.1.1 Use Cases

There are a variety of scenarios depending on how the actors wish to interact with the system. Actors refers to drivers and the two main categories of actors are private EV owners and commercial users. Since each driving trip is unique with different conditions, drivers often have range anxiety or worry about if they can reach a charging station in time before the batteries run out. Private EV owners or potential owners would have range anxiety before travel or even before purchasing despite the advertising claims or promising user experiences. Commercial users such as Uber drivers might also have range anxiety as they cannot spend time charging during a passenger trip since passengers would be unsatisfied since they pay based on time and distance. Being able to manage and monitor trips allows businesses to plan for efficient usage such as only allowing potential passengers to book with drivers with enough battery. Figure 5.1 outlines the use cases and actors who interact with the system.

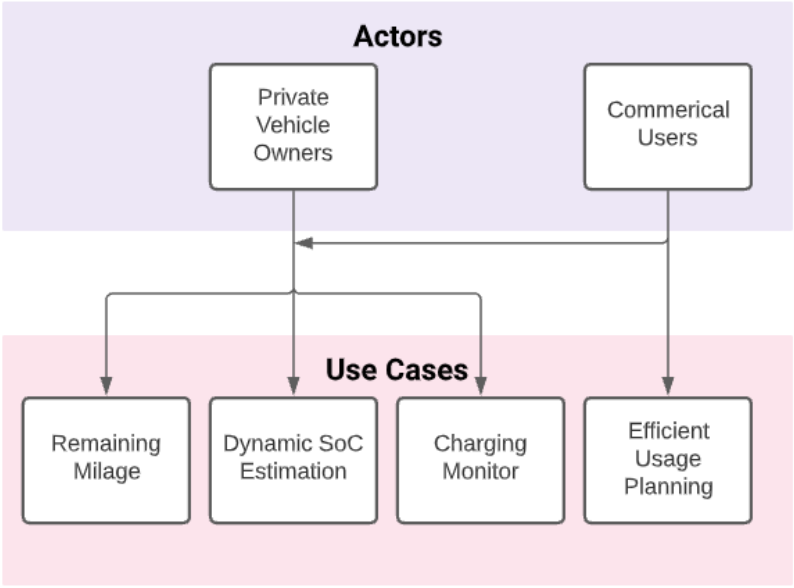
Based on the interactions between the users and system, the use cases are viewing real-time battery state changes while driving or monitoring charging in-person or potentially remotely. Drivers would be able to view accurate information on a touchscreen within their electrical vehicle while driving which would include mileage remaining and the battery percentage (SoC). This helps alleviate range anxiety and allows users to plan ahead on finding a charging station. The final web-based application could also be implemented into a smartphone app that would allow users to remotely view the battery percentage (SoC) and remaining charge

time which would be useful when using charging stations outside as it is possible that charging time to full battery might be different compared to using a charging station at home.

Regarding the interaction between the users and system, the two use cases are charging and driving. In the case of charging, there is no energy consumption involved. The charging status is displayed showing the battery percentage (SoC) and the plug-in duration. In the other case, the user is driving the car. The data are updated and passed to the battery management system. The battery status is predicted and updated in real-time. The percentage of battery, the mileage this car can travel for, and the plug-in duration are displayed.

Figure 5.1

Actors and Their Associated Use Cases



5.1.2 High Level Data Analytics Capabilities

The three planned modules are the input module, processing module, and the prediction module. The input module is responsible for allowing the user to upload, track, and update the input data. This data should include vehicle status, battery status, and the environment status.

The processing module cleans and preprocesses the input data to prepare it for the machine learning process. This module is able to detect and regulate abnormal data from hardware or software issues as well as filter the noise caused by irregular pulses in both voltages and currents. The processing module is able to detect the abnormal data produced by the hardware or software and to filter the signal noises including data with unusual pulses in the voltage and current data, the driving conditions data lagged behind due to the hysteresis of the battery, and the errors data from the unstable system.

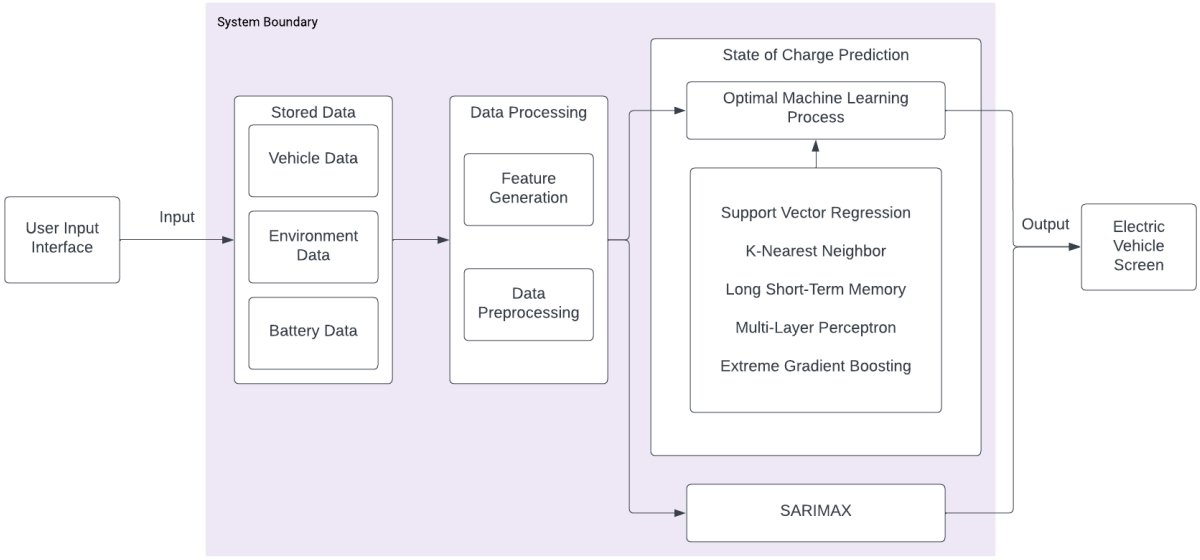
Finally the most comprehensive module, the prediction module. This module is responsible for utilizing the prepared data and making a prediction of battery capacity on that data. The data analytics for this section consists of a singular testing phase as well as a finalized implemented model. Primarily, multiple machine learning models are used to evaluate the performance of each by comparison. The evaluation metrics used for this are MAE, RMSE and R^2 coefficient. After comparing and determining the model best suited for estimation, a condensed version of the system will be made to utilize that model to downsize the data requirements and latency of the system.

The prediction modules include the use of a non-machine learning algorithm that will be utilized for comparison. If this standard and widely used approach more accurately estimates capacity than machine learning approaches, then it will be utilized in conjunction with a machine learning model. The final visualization output may be modified to show both estimations to satisfy machine learning requirements while also providing the most accurate estimation. The desired output is the final implementation of the product, a data visualization interface that can display all the necessary components for accurate capacity estimation.

Figure 5.2 shows the system boundary and output of the three modules. The SARIMAX lies beyond the machine learning processes as it is mathematically derived using previous values.

Figure 5.2

System Boundaries



5.2 System Design

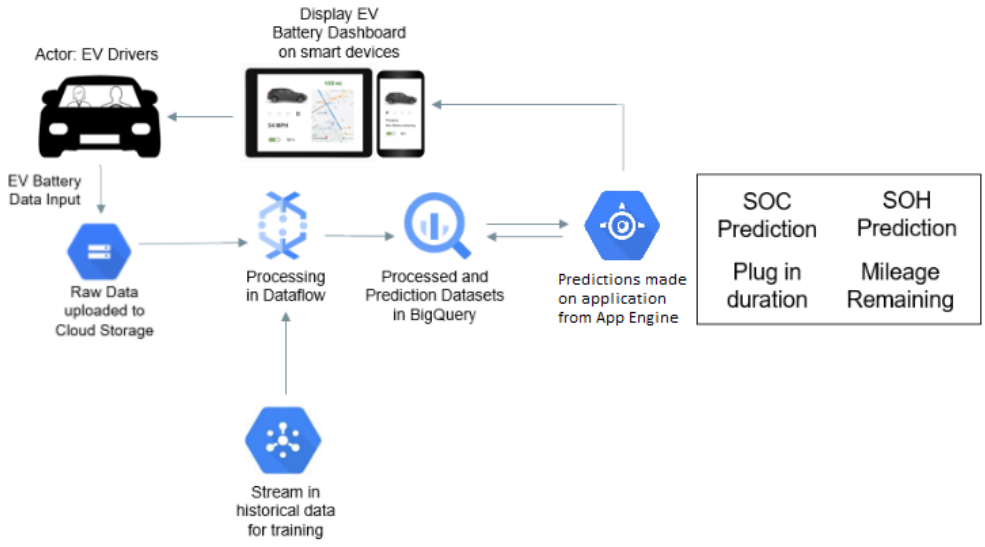
5.2.1 System Architecture Design

The system is designed to take in real-time battery data from electrical vehicles and user input while also maintaining historical data to train prediction models with the end goal of displaying metrics useful to drivers in a dashboard on smart devices that can be refreshed to view the most updated data. Data input from the vehicle battery is collected and automatically uploaded to cloud storage in the form of raw data without user interaction as users are either driving when discharging data is collected or away from the vehicle when charging data is collected.

After data processing, the processed datasets are stored in BigQuery for the web application to access when generating SOC, SOH, plug in duration and mileage remaining predictions which will be displayed on the web application. Models are trained with historical data offline before users drive their electric vehicles for the first time.

Figure 5.3

System Architecture



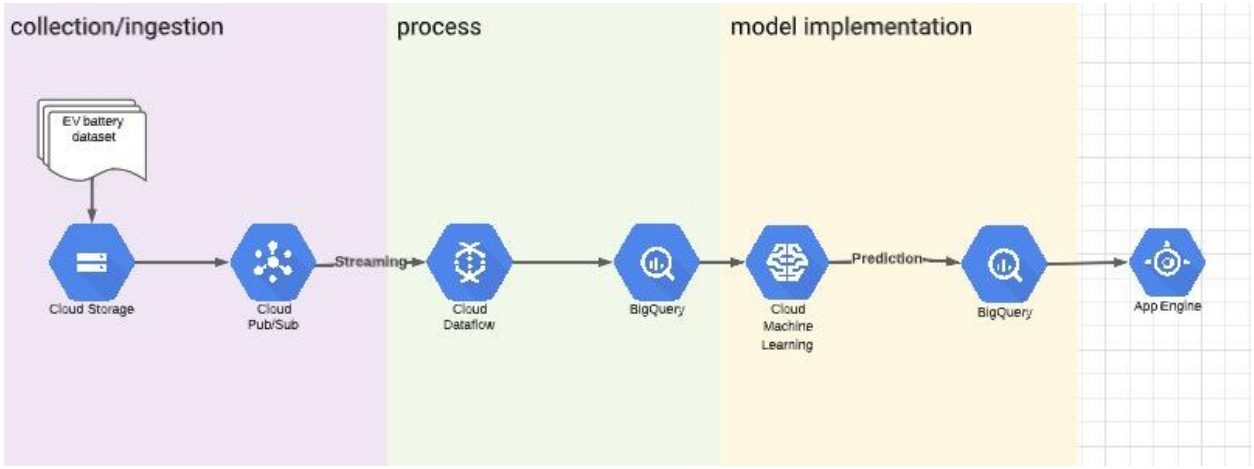
5.2.2 Supporting Platforms, Frameworks, and Cloud Environment

The end goal is a real-time SoC & plug-in duration & remaining mileage forecasting pipeline that takes in raw measurement data from the car, processes data for usage in a machine learning model and presents predicted metrics on a dashboard. This pipeline will be implemented on Google Cloud Platform which is able to provide security, compute power, storage and networking. On top of these are big data and machine learning products for users to easily carry out the analytics work (Qwiklabs, n.d.).

Generally, the workflow involves uploading the EV battery data to the cloud, then cloud computing, and then the prediction is sent to the user's dashboard. The project is created on the cloud with a virtual machine and Python environment set up to deploy the whole pipeline. Figure 5.4 shows the workflow on cloud. Cloud Storage is used to store raw data due to the high velocity of new incoming data; Pub/Sub publishes the record one by one to stream in EV battery data uploaded to cloud; Dataflow handles data processing; Bigquery is used as a data warehouse; App Engine is then connected to Bigquery as the data source for prediction in the web application before sending the results back to Bigquery for storage; the final web application displays the predicted results.

Figure 5.4

Cloud Environment



5.2.3 Data Management Solution and Database

Locally, the raw data is stored on PC and also on Google Drive as a backup. On the cloud, data can be managed within the Google Cloud Platform using a combination of BigQuery and Google Cloud Storage. Google Cloud Storage is an object based cloud storage that is easily integrated with the rest of the Google Cloud Platform. It allows for ingestion of uploaded data by users as well as intermediate storage between data processing steps or modules. This method is used to store large amounts of data for archival purposes as well as temporary storage. Google Cloud Storage will be used for storage of input data by the user, data created from an intermediate step between data processing and data analysis, and final analytical data.

Multiple tables are utilized in BigQuery. The fields in each table are the same including cycle, type, Time, Voltage_measured, Current_measured, Environment_tempreature, Temperature_meansured, discharge_speed, consumption_per_second, and capacity. The fact table and the dimension table can be seen in Table 5.1a and 5.1b.

Table 5.1a

Fact Table

Field	Key
File name	FK
Table location	

Table 5.1b

Battery Raw Data Table

Field	Type	Null	Key
file name	char	NO	FK
cycle	int	NO	FK
type	int	NO	
Time	float	NO	
Voltage_measured	float	NO	
Current_measured	float	NO	
Environment_tempreature	float	NO	FK
Temperature_meansured	float	NO	
discharge_speed	float	NO	
consumption_per_second	float	NO	
capacity	float	NO	

Table 5.1c

Nominal Capacity Table Schema

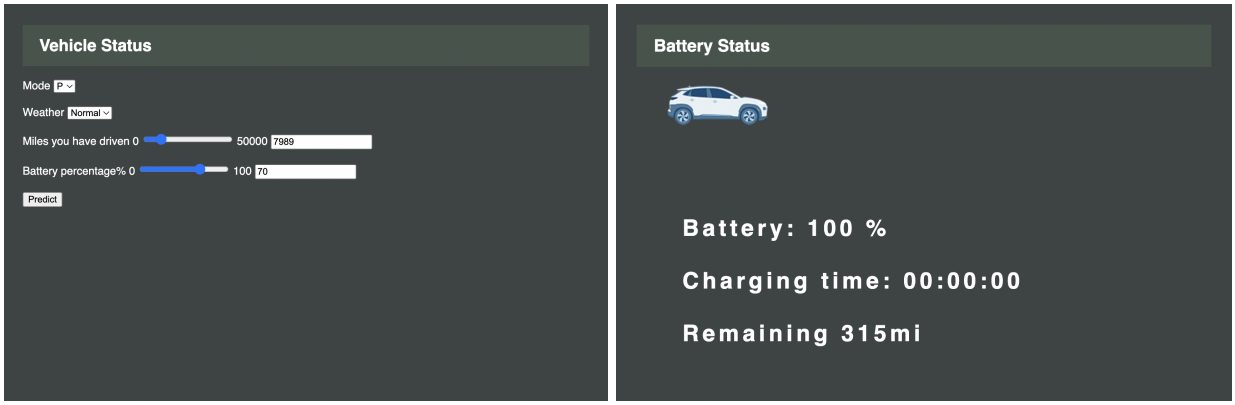
Field	Type	Null	Key
cycle	int	NO	FK
Environment_tempreature	float	NO	FK
Nominal capacity	float	NO	

5.2.4 User Interface and Data Visualization

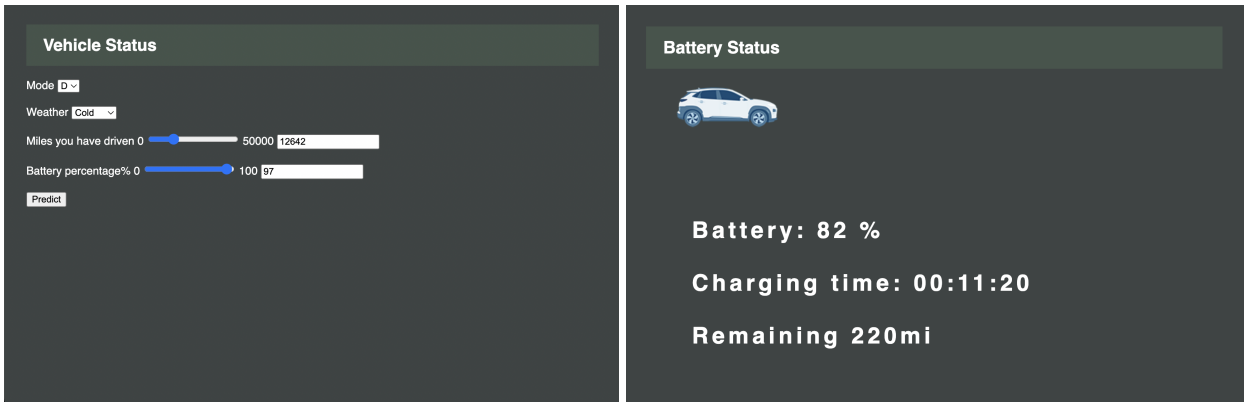
The user interface will be in the form of a web-based serverless platform that is accessible on smart devices either from the built-in screen in electrical vehicles or from a smartphone app. There are two views in the user interfaces depending on the car's driving state seen in figure 5.5. Figure 5.5 shows an example of the interface shown when the car is parking/charging in (a) and driving/discharging in (b). The users select the vehicle status, environment state, and battery status. This output view covers the real-time updating charging time, battery percentage, and remaining mileage. As the new data comes in, the prediction will be updated to show the new battery and vehicle status. The system can be adaptive to new features such as the driving behavior and complicated environment by adjusting the model. Figure 5.5 has an example of the interface shown when the user is driving and the EV battery is discharging on the right.

Figure 5.5

User Interface When Vehicle is Parked and Charging (a) or Driving and Discharging (b)



(a) User case: parking



(b) User case: driving

5.3 Intelligent Solution

5.3.1 Developed AI and Machine Learning Solutions

Although battery management systems in electric vehicles already exist, many use non-AI methods such as Kalman filtering or coulomb counting. This project focuses on using machine learning models that are compatible with non-linear data for time series prediction. The end goal of this project is to simulate the battery management system to monitor the battery status under various application environments. The system is designed to demonstrate the battery status and the vehicle information as well. In the user's cases of charging or driving, the system will predict plug-in duration, SoC, and remaining mileage as displayed in the touchscreen on the vehicles.

Since it is a time-series prediction, the data with comprehensive cycles are collected that can cover the battery status from empty to full. The data can represent the battery charging and discharging behavior. Battery degradation over time is also considered. In commercial electric vehicles, the batteries end their lives and retire when the state-of-health (SoH) has a 20% fade compared to the fresh cells. In the collected data, SoH is dropped to 30% as tested.
To reflect the various driving conditions, the data about the environment temperature, and discharging speed are also included.

The deep learning algorithms are applied including long short-term memory (LSTM) for time-series prediction and multilayer perceptron (MLP) which uses back propagation to adjust the model’s parameters. The support vector machine and the k-nearest neighbor regression approaches are used to predict the capacities by finding the nearest or closest datasets that would be an efficient way. Another machine learning model implemented to test if prediction accuracy

can keep up with reduced prediction time is XGBoost. Finally, the statistical model SARIMAX is used as a baseline as it is also suitable for time-series prediction.

The plug-in duration is determined by the current capacity and the capacity in the full charged state (nominal or rated capacity). The nominal capacity would fade gradually as the battery is aging. The nominal capacities over various cycling conditions are obtained using the linear regression model, so that the plug-in duration and SoC can be estimated.

To maximize the prediction accuracy, the k-fold cross-validation method is employed to resample the data as much as possible. The data resources contain the battery status data in order of sequential cycling under different driving conditions that are stored separately. For example, the batteries would discharge faster when the vehicle is accelerating, vice versa. It is reasonable to consider all the possible user scenarios.

5.3.2 Input and Outputs, Supporting System Contexts, and Solution APIs

Input datasets contain features related to vehicle state (parking, driving, remaining mileage), the battery state (percentage), and environment conditions (weather). When the charging is not completed, the battery is fast charged until 80% under the constant current, and then charged slowly for the rest 20% under the constant voltage. During charging there is no acceleration or deceleration and the environment temperature does not change. When the user is driving the car, the battery is consumed along with voltage decreasing and the current changes based on the user's driving behavior. All raw input datasets are uploaded to the cloud for storage before processing.

Expected output data include the current capacity of the battery, the plug-in duration in the charging that demonstrates how long the battery is charged to 100%, and SoC, remaining mileage based on remaining charge which is updated in real-time.

According to the whole workflow, this project includes two supporting systems: offline machine learning system that aims at finding the optimized model; online cloud system to implement the real time prediction. Offline system is based on the local PC environment. Anaconda Jupyter notebook and Python with necessary libraries are used to support the model training and evaluation. Online cloud system is based on Google Cloud Platform. Google Cloud Platform can provide a suite of data analytics tools from real time data ingestion to data visualization to meet the requirements of the real time prediction pipeline. Also, Google Cloud Platform's scalable storage and computing ability helps this project to have more practical significance.

In the offline model training, a necessary API is Keras. Keras is a high-level deep learning interface for users to easily build models (Aurélien Géron, 2019). On Google Cloud Platform, several APIs are used: Identity and Access Management (IAM) API is used to manage access control of the project; Cloud Storage API is used for data storage; Dataflow API is used for model performance comparison.

5.4 System Support Environment

5.4.1 Information and Features of the Offline System Support Environment

The offline system support requires a basic PC as hardware and OS environment as software. Development language Python 3.7 is run on Anaconda Jupyter Notebook. Several libraries and APIs are used for model development: Pandas and Numpy for data preparation and exploration, Matplotlib for data visualization and plotting, Scikit-learn for machine learning, Tensorflow and Keras for advanced deep learning. All project files are saved to Google Drive.

5.4.2 Information and Features of the Online System Support Environment

The online system support for hosting the web application that can be accessed by users will be done in Flask which can be deployed from Cloud App Engine. Trained machine learning models saved in pickle files or deep learning models saved in .json and .h5 format can be deployed in this application. The backend is designed to embed the machine learning prediction and send the streaming data to the frontend on the Flask service platform. The user interfaces consist of two pages: the homepage to let users input the vehicle state, battery state, and the environment, and the prediction page to show the real-time updating of battery SoC, plug-in duration, and remaining mileage. The frontend is designed via Javascript. The build settings will be set through app.yaml and requirement.txt to configure the virtual machine and launch the application.

All source code and finished documentation for web design and deployment are uploaded to this Github repository:

Repository URL: https://github.com/0d11d/Battery-Status-Estimation.git

6. System Evaluation and Visualization

6.1 Analysis of Model Execution and Evaluation Results

The prediction of battery capacity in electric vehicles is a regression analysis and the SoC estimation, plug-in duration in charging, and the remaining mileage during driving are estimated by the predicted capacity. It is also a time series forecasting as the battery has cyclic behavior and degrades over time. When we build models for battery status prediction, the most essential criteria is accuracy. The cost efficiency is also weighed such as short-time consumption and small-storage needs.

The data with seven features are fed into three machine learning models, two deep learning models, and one statistical model for battery capacity prediction, namely XGBoost, SVM, KNN, MLP, LSTM, and SARIMAX. The performances of the models are evaluated by the metrics of RMSE, MAE, and R2 and summarized in Table 6.1. The running times for model training, evaluation, and test are considered as the cost efficiency criteria.

Among three machine learning models, the XGBoost model runs faster than the rest on training and prediction and improves the accuracy by 75% compared to SVR. Considering the long-term prediction, the deep learning models show better learning skills than the machine learning model. Both the deep learning models MLP and LSTM have exhibited 70% and 96% accuracy improvement respectively through validation results compared to SVR. Both models have a shorter running time than machine learning models except XGBoost. In the meantime, the deep learning model can automatically optimize itself that matches our desired system. The capacity prediction accuracy of LSTM with time-series transformation is 4 times higher than that of MLP and XGBoost. The SARIMAX model offers the most accurate in-sample prediction but

is not capable of out-of-sample forecasting. Since the battery status prediction is out-of-sample forecasting, the benchmark model can be excluded.

Table 6.1
Prediction results from six models

	Model	**LSTM**	**MLP**	**XGBoost**	**SVR**	**KNN**	**SARIMAX**
Training	**Training Time (sec/1000 records)**	6.78	14.08	0.54	0.156	N/A	12.2
Validation	**Prediction Time (sec/1000 records)**	0.074	0.020	0.0034	0.234	2.7	14
	Capacity Prediction Accuracy (RMSE)	0.0156	0.1457	0.1219	0.489	N/A	0.0015
Test	**Prediction Time (sec/1000 records)**	0.88	0.023	0.44			
	MSE	0.0019	0.0274	0.0206			
	MAE	0.0331	0.1085	0.1275			
	RMSE	0.0432	0.1654	0.1434			
	R2	0.9860	0.5782	0.8462			

6.2 Achievements and Constraints

6.2.1 Solving Target Problems

Target Problem: How to convert predicted battery data into readable output data on the user interface

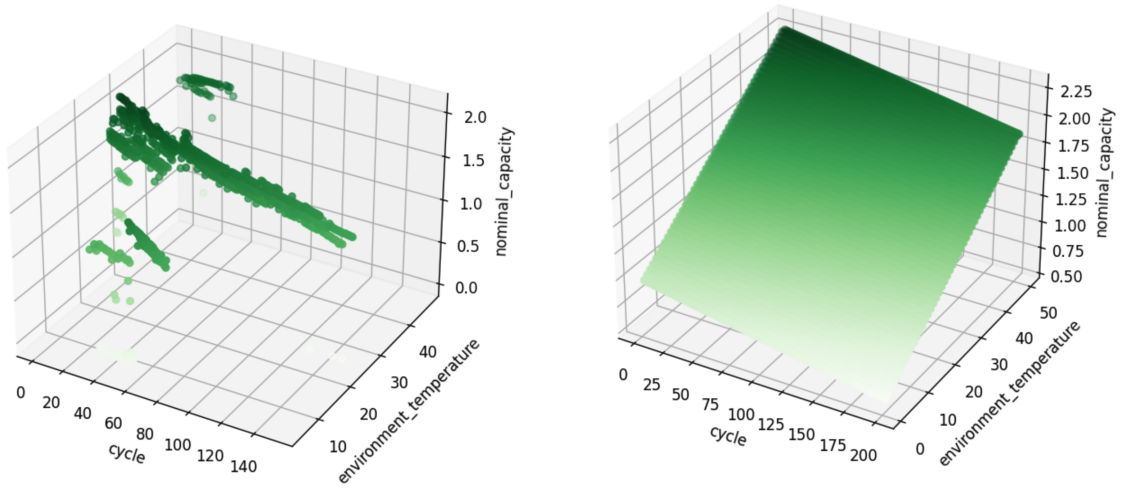
Solution: A mature interface for an EV should provide the information on the remaining percentage of battery (SOC), how long the car will be charged to full in the parking state or how much mileage remained in the driving. Those are not the features included in the datasets. Instead of simulating those parameters, we chose to predict the battery capacity in any environment conditions.

The SOC value is determined by the current capacity over the nominal capacity in a certain cycle. Our battery is a classic 18650 lithium-ion battery with a maximum capacity of 2Ah. We establish the nominal capacity lookup table for calculating SOC as seen in Figure 6.1. The plug-in duration is the time derived from the ratio of remaining capacity over the charging speed and the remaining mileage is derived from the nearly linear relationship between mileage and capacity.

Figure 6.1

(a) nominal capacity derived from raw data in different cycle at varied environment temperature;

(b) nominal capacity mesh by interpolation method



(a) (b)

Target Problem: How to make the battery state of health predictable.

Solution: The prediction of battery degradation over time depends on physical characteristics changing over time which includes the resistances, the open-circuit voltage, or the charges flowing through the battery. The current dataset is inadequate for accurately predicting the state of health. We chose to focus on the state of charge prediction models with capacity as the target variable which is then compared to lookup tables of nominal capacity to provide a state of charge estimation.

Target Problem: How to make the system simulate fleet analysis in the real time

Solution: The charging process is equivalent to the charging in the parking state, and the discharge process is to the driving. So the final interfaces will provide the options for the users of ‘parking’ or ‘driving’. The system can predict the battery status with a continuous output and the

prediction for the charge and the discharge are separately. The system can detect the battery status before prediction according to information on user behavior, the environment temperature, remaining battery, and remaining mileage.

6.2.2 Constraints Encountered

The main constraint encountered in this project comes from the data source since the only environmental feature is temperature but details on weather, location, road conditions, etc. were not given. Since collecting electrical vehicle data is extremely hard due to non-disclosure agreements in the battery and the EV manufacturers as well as the tools needed to record the status of vehicle battery packs in real life trips, we were not able to collect data ourselves and had to rely on existing datasets. While a dataset collected from a more recent electrical vehicle from driving trips outside of a lab setting with more environmental features would be more ideal, this is likely confidential to electrical vehicle companies.

Another constraint is the lack of a higher budget which limits the complexity of the platform. Due to budgetary reasons, the Google Cloud Platform was unable to be a complete part of the final implementation. Model training was performed offline to conserve costs, but the numerous online predictions, online data preparation tests, and final modeling tests made it impossible to move forward with this method. Unfortunately, this caused model retraining in real time impossible as running nodes of multiple high memory virtual machines is extremely expensive even for a short time period.

Finally, due to the limited capabilities and time learning a new webapp system, the web-based system can interact with users only. The Google Cloud Platform App Engine does not support HTTP stream.

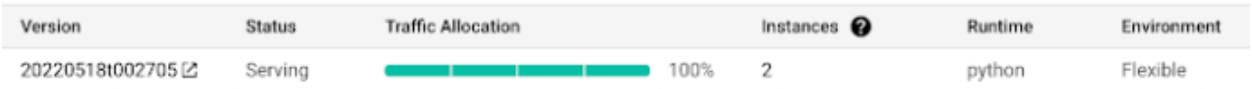
6.3 System Quality Evaluation of Model Functions and Performance

Latency is always an issue when dealing with online modeling and implementation. Users can be greatly affected by latency in this use case but not as much as other use cases. During a continuous running of a charging or discharging cycle, a latency of a minute would display information that is a minute old. Depending on the accuracy of the information, that minute wouldn't greatly affect the user in most situations. However, if the predictions become inaccurate, that minute of delayed information could easily keep the user misinformed for longer and lead to unexpected results. Furthermore, the user should expect a decently quick response when switching from charging to discharging modes and vice versa.

The initial setup configuration of the web application in App Engine takes around 4 minutes to complete and then deployment takes around 15 minutes. The deployed web application can be stopped if the application design or model needs to be updated and relaunching the application will take approximately 2-3 minutes.

Figure 6.2

Successful deployment of web application from App Engine



System testing was done for discharging and charging modes with different combinations of temperatures (normal, hot, cold), miles driven (100, 2500, 5000) and initial battery percentages (90%, 50%, 10%). The loading and prediction time is around 30 - 40 seconds as seen in both Table 6.2 and 6.3 with response time varying based on user input.

Table 6.2
System Testing on Park Mode for various charging conditions

Temperature	Miles Driven (mi)	Initial Battery Percentage (%)	Loading and prediction time (seconds)
Normal	100	90	31.50
		50	35.73
		10	36.83
	2500	90	33.03
		50	36.89
		10	36.68
	5000	90	30.35
		50	37.04
		10	38.45
Hot	100	90	33.66
		50	35.82
		10	36.52
	2500	90	36.13
		50	36.70
		10	37.54
	5000	90	36.20
		50	36.58
		10	37.49
Cold	100	90	34.72
		50	35.84
		10	36.17
	2500	90	35.35
		50	35.89
		10	36.58
	5000	90	35.74
		50	35.99
		10	36.33

Table 6.3
System Testing on Drive Mode for various discharging conditions

Temperature	Miles Driven (mi)	Initial Battery Percentage (%)	Loading and prediction time (seconds)
Normal	100	90	35.65
		50	34.97
		10	32.22
	2500	90	35.80
		50	34.74
		10	32.47
	5000	90	35.63
		50	34.31
		10	33.34
Hot	100	90	35.08
		50	34.25
		10	33.13
	2500	90	35.24
		50	34.23
		10	33.47
	5000	90	35.92
		50	34.33
		10	33.54
Cold	100	90	35.36
		50	34.09
		10	33.61
	2500	90	35.72
		50	34.96
		10	33.67
	5000	90	35.80
		50	34.29
		10	33.73

Loading and prediction time for both charging and discharging modes have longer wait times when more miles have been driven as this requires predicting on more records of historical data. Temperature does not seem to determine how long loading and prediction time will take for both charging and discharging modes. However, similar to miles driven, initial battery percentage does affect loading and prediction time as it takes longer when discharging from a higher initial battery percentage while driving as seen in Figure 6.3 and longer to charge from a lower initial battery percentage as seen in Figure 6.4.

Figure 6.3

Loading and Prediction time when discharging based on different input conditions

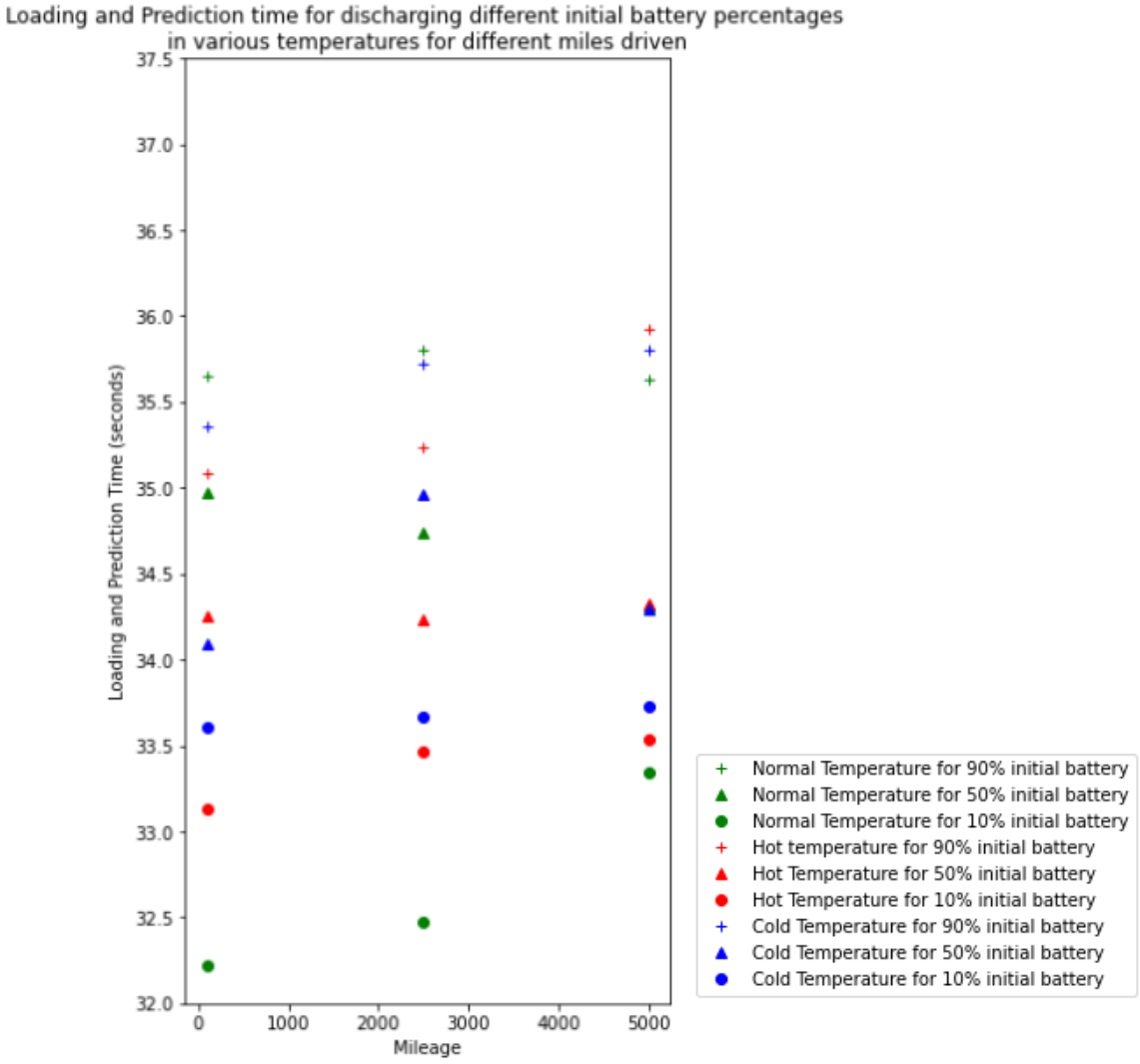
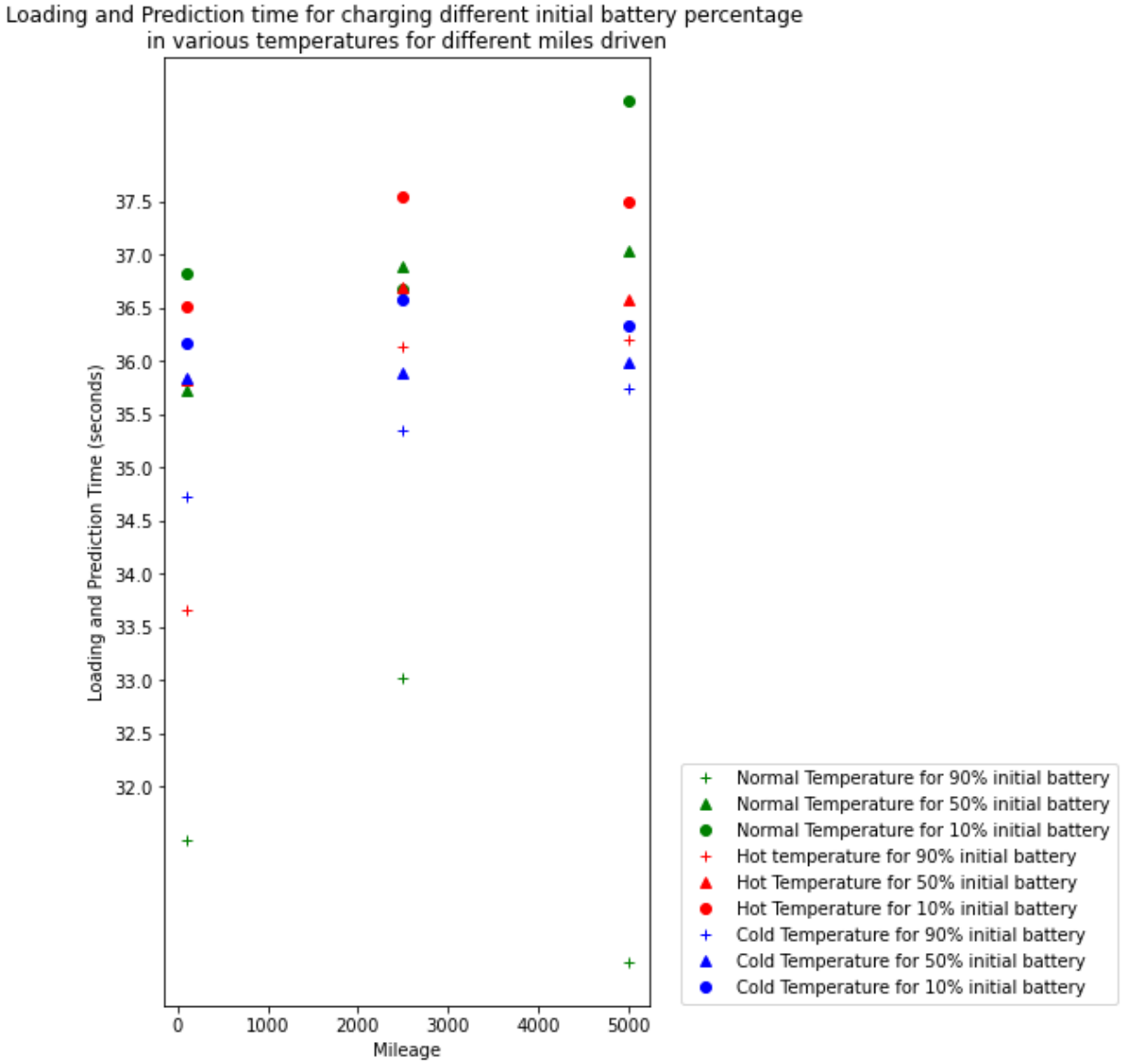


Figure 6.4

Loading and Prediction time when charging based on different input conditions



6.4 System Visualization

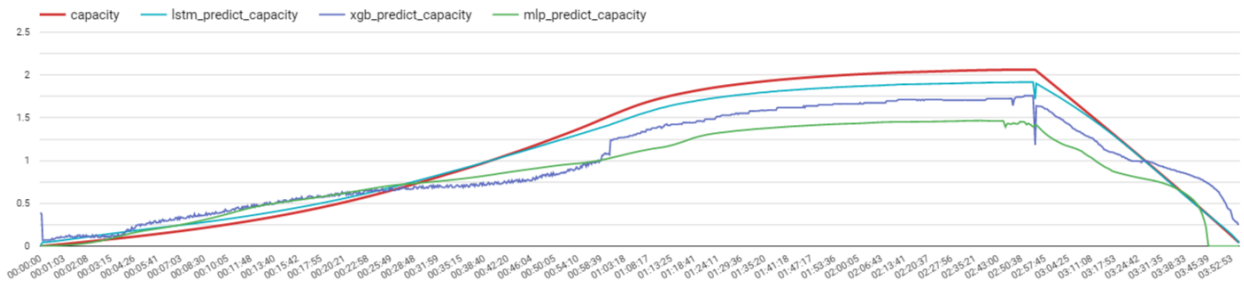
The model selected to predict the online environment was LSTM. The designed LSTM model can predict the sequential change of battery status and the status under specific conditions as well. For comparative purposes, the predictions of LSTM, XGBoost, and the MLP model were compared against the actual data. In a more developed system visualization, these charts would be included. In the end, only LSTM was implemented in the online environment due to its relatively low prediction times and extremely high accuracy. LSTM would be perfectly suited to adhere to the time constraints as well as produce high performing results.

Unfortunately, App Engine was limited in its capabilities and other tools for app deployment would both decrease run times and increase the amount of possibilities available for system visualizations. The following are examples of the possible graphs that could be included in an improved implementation.

The first graph is a comparison of the predicted and actual values of the target variable. After training the model on the training dataset, the final cycle was used as the test dataset to create prediction values for. Figure 6.5 outlines the comparison of predicted and actual values of capacity.

Figure 6.5

Predicted Capacity vs Actual Capacity Over Time



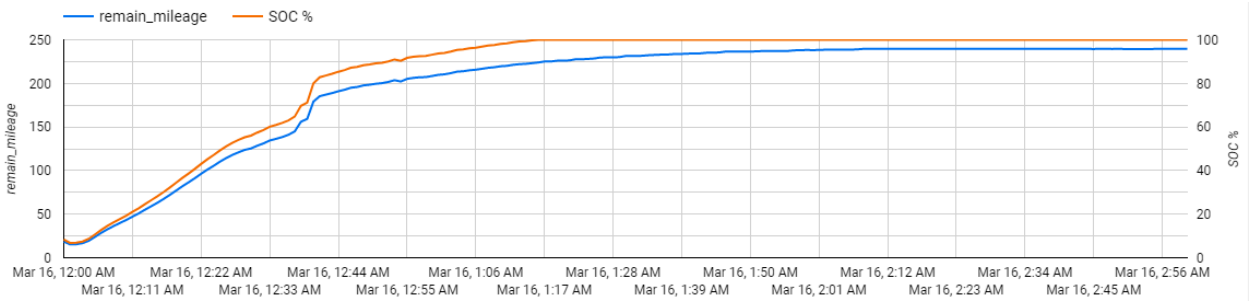
The red line represents the actual values of this cycle, the light blue line are the predictions made by the LSTM model, the dark blue line are predictions made by XGboost, and the green line are predictions made by the MLP model. From Figure 6.2, we can see that the LSTM predictions are very accurate until the capacity reaches the nominal capacity area. Meanwhile, the XGboost model has many areas where the predictions are unreliable and do not conform to the actual value's arching structure. Finally, MLP had generally the correct structure, but its predictions begin to fall short making it the worst performing model.

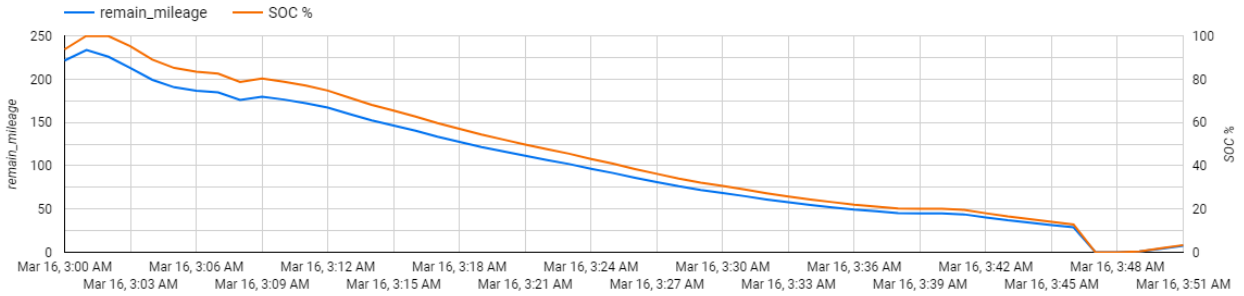
Another observation that could be made is that the accuracy of both MLP and XGboost decline greatly at the end of the cycle. This could be a problem with how the models were trained on multiple cycles instead of splitting up charging and discharging data. A way to prevent this could be to introduce two models, one for charging and one for discharging.

Values such as SoC, plug-in duration, and remaining mileage can be added to the final dashboard for informational purposes. These can be calculated using nominal capacity, ambient temperature, and other input features. Figure 6.6 is an example of the relationship between the estimated SoC values as well as the remaining mileage of an electric vehicle. The figure is split into two parts, charging and discharging.

Figure 6.6

SoC and Remaining Mileage



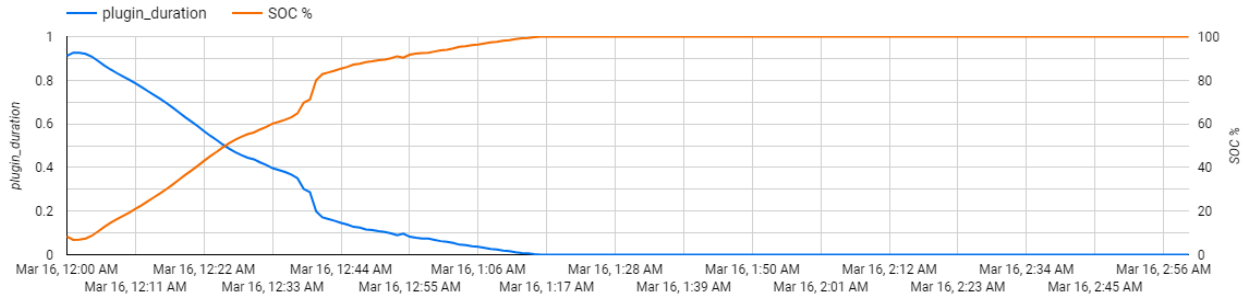


On this dual-axis chart, remaining mileage values are displayed on the left axis while SoC percentages are on the right axis. As expected, these calculated values are extremely similar in their behavior as they are highly correlated. Mileage is based on predicted capacity alone while SoC relies on both nominal capacity as well as predicted capacity. This information will be useful when designing a user interface that can give the driver real-time information on their approximate remaining driving times. On top of that, it's interesting to note the relationship between the two lines is not one to one. In fact, when the values are approaching zero they are much closer together and at higher values, they have the most distance apart. Also, when the vehicle is in a charging state, SoC can read 100% while the remaining mileage still increases.

The next figure, Figure 6.7, compares another calculated value, plug in duration, with SoC. These two values are inversely related as SoC decreases, the required plug in duration increases.

Figure 6.7

SOC and Plug in Duration while Charging

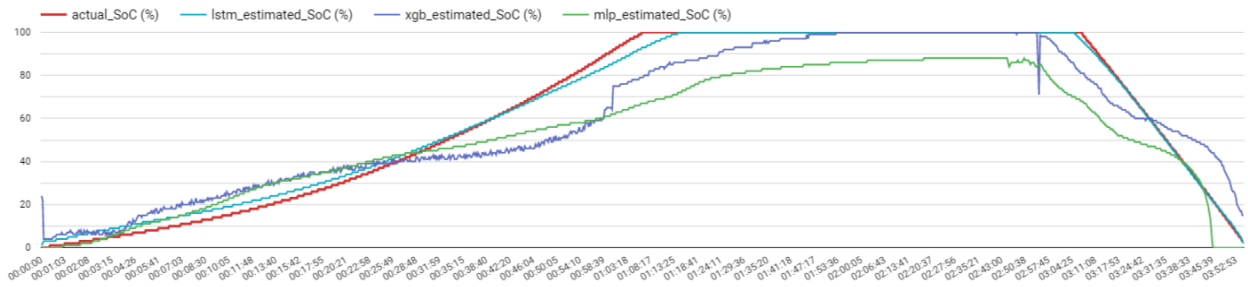


Plug in duration is very useful information in regards to the user who would be able to switch their usage mode from discharging to charging. The user would be able to estimate the amount of time required for a full charge. A problem occurs as the plug in duration estimation is not a very linear pattern. As this variable is concerned with time, it should not be wildly fluctuating. A reason for this is because the current predictions of SOC and capacity are also very non-linear. A way to fix this in the future is to create an estimated trend line that could replace the plug-in duration. Although there are fluctuations, the estimation is decently accurate. At the start of the cycle, it estimates around .9 hours or 54 minutes until a full charge is complete. And, at the 54 minute mark, the SOC is charged to about 90% capacity.

Returning to the subject of the accuracy of predictions, SoC, remaining mileage, and plug-in duration can also be calculated from the predictions and compared against the actual values. In Figure 6.8, SoC estimations are compared against actual values.

Figure 6.8

Predicted SoC Values



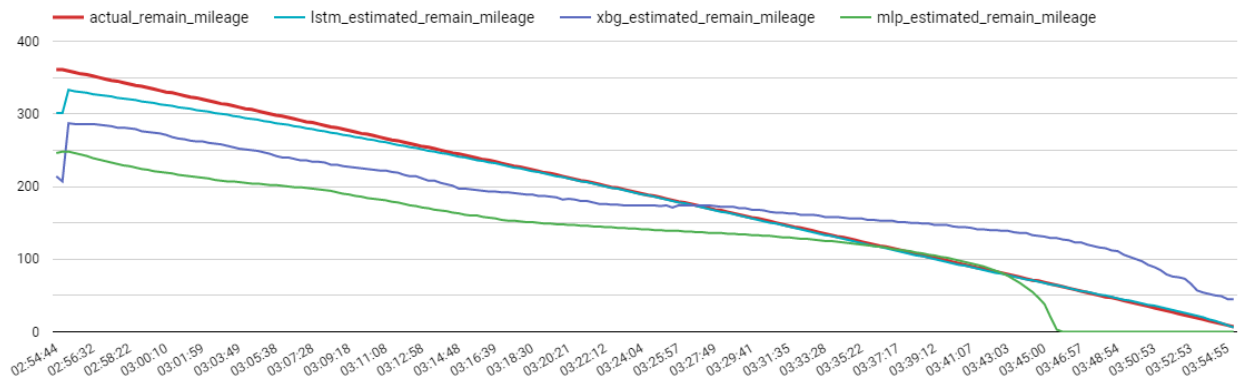
Much like Figure 6.5, LSTM outperforms each other model because of how well it predicted capacity. In fact, LSTM likely has less error in predicting SoC because as it reaches nominal capacity, the predictions are maximized to a full charge. This would reduce the amount of error created from when it is predicted capacity. There are still some observations that can be made regarding its real effectiveness in predicting SoC. For instance, after the actual SoC

achieves a 100% charge, LSTM will reach that conclusion eight minutes later. For charging, this seems like an acceptable amount of error in the model. Meanwhile, XGboost takes 46 minutes to reach that value.

Finally, Figure 6.9 shows the remaining mileage calculated using capacity and nominal capacity. This is a very important visual to display correctly as the user will heavily rely on it to get an accurate prediction on how many miles they could drive before needing to recharge.

Figure 6.9

Predicted Remaining Mileage Values



While LSTM has a low initial accuracy, it becomes extremely accurate as remaining mileage reaches zero. For this use case, that's when the remaining mileage estimates are most important. LSTM has the best accuracy overall, and in terms of accuracy during the final predictions, both MLP and XGBoost have mixed results. XGboost overestimates those final predictions by 60 miles or so while MLP underestimates it by about 60 miles.

7 Conclusion

7.1 Summary

This project is launched under the surging market of electric vehicles and the rising range anxiety from users. The systems for current electric vehicles are based on the physical and statistical models which highly relies on the experiments in the lab. We can aim to build a system to estimate the electric vehicle battery status accurately and promptly deliver a user-friendly application to adapt complicated data types and streaming data.

In this project, machine learning and deep learning models are designed to predict the battery status by using battery cell data. We are challenged to train the model by the small dataset with a limited number of features. The related features about driving behavior and weather conditions are extracted from raw data with the rich domain knowledge of Li-ion batteries. The data extraction, transformation, and loading pipeline is built on Google Cloud Platform for the purpose of potential new data in the future.

Five models are proposed: XGBoost, SVR, KNN, LSTM, and MLP after sufficient literature review so that we can evaluate the system on data type compatibility, training and prediction speed, and application flexibility. Each model is trained and evaluated on the same datasets to make a fair comparison. The optimization is deployed in each model respectively. The evaluation results on the validation dataset demonstrate XGBoost, LSTM, and MLP outperformed SVR, KNN, and benchmark statistical model. Among them, LSTM with time-series transformation shows the best performance with high prediction accuracy and ability to predict sequential data or single point.

The user inferences are designed with two user cases: parking and driving. In the parking mode, the user will park and charge the car. The battery percentage, charging time, and the

remaining mileage are updated until the batter is charged to full. In the driving mode, the user will see the battery percentage and remaining mileage dropping while the chagrin time increases to remind users how long to charge the battery to full. The interfaces are designed in the Flask platform and deployed on Google Cloud Platform App Engine for public use.

7.2 Benefits and Shortcoming

The benefit of the solution presented is that it provides a generic prediction of battery capacity for lithium ion batteries based on temperature which can be used for any electric vehicle using the similar batteries. This attempt differs greatly from many of the other researches that use single cell batteries for SoC predictions. By including the battery's ambient temperature, it allows for the model to be more dynamic for different temperature conditions. The predictions generated can be used to understand if an electric vehicle is still efficient in cold or hot temperatures.

Unfortunately, this solution only accounts for temperature as the only variable external to the battery as other variables such as weather, incline, added weight to vehicle, and more were not measured in the data. Because of the limitations of the data available, many of these other dynamic features would not be able to be included in this attempt. The available data for electric vehicles were very limited to a specific model which would also not be sufficient in predicting the SoC for all electric vehicles. The only way to circumvent this is for multiple models to either make their collected data public or if researchers collected it themselves.

Regardless of this shortage of available data, this method can still adequately derive SoC, plug-in duration, and remaining mileage using the nominal capacity of the cycle which is given by the NASA dataset. This approach shows what is possible in using machine learning models to accomplish predicting SoC or capacity. Not only is it possible, but it boasted low prediction

times and some low training times with high accuracy especially when compared to more common time-series predictors like SARIMAX.

7.3 Potential System and Model Applications

One of the potential commercial applications of our system is to help model the SoC of a battery given temperature and battery inputs. This designed system can be integrated in many current models of vehicles to provide real-time battery and vehicle status. This is due to the simple nature of the system, but of course it has room for improvement. As it stands, users can interact with the system and would be able to estimate SoC, remaining mileage, or plug-in duration given simple battery data.

Another application of this system is academic in nature. The use of an LSTM model with time-series transformations can be applied to any cyclical data. This model would likely excel even beyond battery predictions. Furthermore, this system can be studied and improved upon for future iterations. The more resources a researcher contributes to this methodology, the more advanced the results could become. For instance, with minimal modifications, the LSTM model could be applied to battery data collected in real life to create real-time predictions.

While it's likely not possible that our model is more accurate than private industries methods to estimate SoC, the prevalence of being able to incorporate machine learning models to predict these values could be useful in applying them in modern techniques. With further study and implementation on actual electric vehicles, machine learning models could prove to be extremely more accurate than conventional methods. Unfortunately, it’s even hard to tell how accurate the current methods are as there are minimal amounts of studies released to estimate it.

7.4 Experience and Lessons Learned

Our team previously used battery data from a BMW i3 electric vehicle during the first half of the project as it contained real trip data with more features related to the vehicle and environment but failed to consider that it was announced during Summer 2021 that this vehicle would be discontinued in 2022. Another issue was that predictions made from the BMW dataset would only apply to BMW models rather than electric vehicles in general. After reassessing this dataset we also realized that we would only be able to predict state of charge but not plugin duration or mileage remaining. Although the dataset we used to complete the project comes from lab data rather than real life driving trip data, 18650 Li-ion batteries are used in popular consumer electric vehicles such as the Tesla Model S so our results are more applicable for future and current electric vehicle drivers.

One major lesson learned is that high quality data from popular competitive research domains are extremely valuable but also limited. Even when contacting the original researcher for the dataset we used, they were not able to go into detail for some of our questions for confidentiality reasons.

7.5 Recommendations for Future Work

One possible method to greatly improve the quality of this project is to gather the real trip data with multiple environmental profiles from electric vehicle companies including the driving behaviors from the driver, current weather, altitude, and time of day. Since electric vehicles are a hot topic, many companies are aiming to produce electric vehicles with innovative technologies such as self-driving and battery swap. This makes data relating to electric vehicles very competitive which also means the most recent data is often confidential and not easily accessible. The more data available to fully capture dynamic features of an electric vehicle, the more

interesting the machine learning model would be in predicting SoC. The addition of these features would also allow comparisons between the different conditions to reveal important insights on which of these features have the heaviest impacts on SoC drain.

Due to lack of funding outside of Google Cloud Platform credits acquired from the department, the final web application was limited to tools on GCP such as App Engine. However, it is possible that using tools for app deployment from other cloud providers could result in a web application with a faster run time between capacity prediction to loading the output page.

While on the topic of funding, there are additional features that could be added to this project to make it more accessible, automated, and even address other shortcomings. For instance, in the future, direct driving trips could be collected directly from electric vehicles. This way, nominal capacity and mileage driven could also be recorded and taken in to account. Another recommendation of future work is being able to set up an active streaming system where users are able to use the SoC estimator through some sort of sensor system. This way, new information may be collected from the sensors while simultaneously achieving the project goal of SoC estimations.

7.6 Contributions and Impacts on Society

The benefits and limitations of electric vehicles are widely discussed as improvements are being made as new electric vehicles are added to the market along with both gas prices and global warming increasing making electric vehicles appealing for both economic and environmental reasons. Improving the accuracy of battery capacity predictions which are used to calculate battery state of charge prediction helps reduce doubts from current and future electric vehicle owners.

Gas prices have been rising due to the recent Russian-Ukraine war as Russia is a major source for gas as well as public transport becoming less favorable as coronavirus is extremely contagious resulting in more usage of personal vehicles. The difficulty in predicting how much gas prices will increase in the future may deter new car buyers from buying a vehicle that runs exclusively on gasoline and may instead encourage car buyers to look towards hybrid and electric cars which may have a higher initial price but may also provide more savings from reduced or no spending on gas.

References

Ali, M. U., Zafar, A., Nengroo, S. H., Hussain, S., Alvi, M. J., & Kim, H. J. (2019). Towards a smarter battery management system for electric vehicle applications: A critical review of lithium-ion battery state of charge estimation. In *Energies* (Vol. 12, Issue 3). https://doi.org/10.3390/en12030446

B. Saha and K. Goebel. (2007). *Battery Data Set*. NASA Ames Prognostics Data Repository. http://ti.arc.nasa.gov/project/prognostic-data-repository

Chai, T., & Draxler, R. R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature. Geoscientific Model Development, 7(3), 1247–1250. https://doi.org/10.5194/gmd-7-1247-2014

Chaka, K., Thanh, N. L., Flamary, R., & Belleudy, C. (2018). Performance Comparison of the KNN and SVM Classification Algorithms in the Emotion Detection System EMOTICA. International Journal of Sensor Networks and Data Communications, 07(01). https://doi.org/10.4172/2090-4886.1000153

Chandran, V., Patil, C. K., Karthick, A., Ganeshaperumal, D., Rahim, R., & Ghosh, A. (2021). State of charge estimation of lithium-ion battery for electric vehicles using machine learning algorithms. *World Electric Vehicle Journal*, *12*(1). https://doi.org/10.3390/wevj12010038

Charkhgard, M., & Farrokhi, M. (2010). State-of-Charge Estimation for Lithium-Ion Batteries Using Neural Networks and EKF. *IEEE Transactions on Industrial Electronics* (1982), 57(12), 4178–4187. https://doi.org/10.1109/TIE.2010.2043035

Chemali, E., Kollmeyer, P. J., Preindl, M., & Emadi, A. (2018). State-of-charge estimation of Li-ion batteries using deep neural networks: A machine learning approach. Journal of Power Sources, 400, 242–255. https://doi.org/10.1016/j.jpowsour.2018.06.104

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13-17-August-2016*. https://doi.org/10.1145/2939672.2939785

CRISP-DM (n.d.). Data Science Project Management. Retrieved March 12, 2021, from https://www.datascience-pm.com/crisp-dm-2/

Crowder, S. v. (1989). Design of Exponentially Weighted Moving Average Schemes. *Journal of Quality Technology*, *21*(3). https://doi.org/10.1080/00224065.1989.11979164

Densmore, A., & Hanif, M. (2015). Determining battery SoC using Electrochemical Impedance Spectroscopy and the Extreme Learning Machine. *2015 IEEE 2nd International Future Energy Electronics Conference, IFEEC 2015*. https://doi.org/10.1109/IFEEC.2015.7361603

Fuller, R. (2019, May 28). Panasonic NCR 18650 PF What are the Advantages for Lithium Cells. County Battery. https://www.countybattery.co.uk/the-battery-experts/panasonic-ncr-18650-pf-what-are-the-advantages-for-lithium-cells/

Gabbar, H. A., Othman, A. M., & Abdussami, M. R. (2021). Review of Battery Management Systems (BMS) Development and Industrial Standards. *Technologies*, *9*(2). https://doi.org/10.3390/technologies9020028

Géron, Aurélien. (2019). *Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc.

Gruosso, G., Storti Gajani, G., Ruiz, F., Valladolid, J. D., & Patino, D. (2020). A virtual sensor for electric vehicles' state of charge estimation. Electronics, 9(2), 278. https://doi.org/10.3390/electronics9020278

Hannan, M. A., Lipu, M. S. H., Hussain, A., Ker, P. J., Mahlia, T. M. I., Mansor, M., Ayob, A., Saad, M. H., & Dong, Z. Y. (2020). Toward Enhanced State of Charge Estimation of Lithium-ion Batteries Using Optimized Machine Learning Techniques. *Scientific Reports*, *10*(1). https://doi.org/10.1038/s41598-020-61464-7

Hannan, M. A., Lipu, M. S. H., Hussain, A., & Mohamed, A. (2017). A review of lithium-ion battery state of charge estimation and management system in electric vehicle applications: Challenges and recommendations. In *Renewable and Sustainable Energy Reviews* (Vol. 78). https://doi.org/10.1016/j.rser.2017.05.001

Hu, C., Youn, B. D., & Chung, J. (2012). A multiscale framework with extended Kalman filter for lithium-ion battery SOC and capacity estimation. *Applied Energy*, *92*. https://doi.org/10.1016/j.apenergy.2011.08.002

Hyndman, R., & Athanasopoulos, G. (2018). Forecasting: Principles and Practice. *Principles of Optimal Design*, *September*.

IEA. (2021). Global EV Outlook 2021 - Accelerating ambitions despite the pandemic. In *Global EV Outlook 2021*.

Kelleher, J. D., Mac Namee, B., & D'Arcy, A. (2015). Fundamentals of machine learning for predictive data analytics: algorithms. *Worked Examples, and Case Studies*.

Khalid, Sundararajan, A., Acharya, I., & Sarwat, A. I. (2019). Prediction of Li-Ion Battery State of Charge Using Multilayer Perceptron and Long Short-Term Memory Models. *ITEC 2019 - 2019 IEEE Transportation Electrification Conference and Expo*. https://doi.org/10.1109/ITEC.2019.8790533

Mastali, M., Vazquez-Arenas, J., Fraser, R., Fowler, M., Afshar, S., & Stevens, M. (2013). Battery state of the charge estimation using Kalman filtering. *Journal of Power Sources*, *239*. https://doi.org/10.1016/j.jpowsour.2013.03.131

Matthias Steinstraeter, Johannes Buberger, Dimitar Trifonov. (2020). Battery and Heating Data in Real Driving Cycles. IEEE Dataport. https://dx.doi.org/10.21227/6jr9-5235

Parashar, P. (2020, June 18). Support Vector Regression and its Mathematical Implementation. The Startup.

https://medium.com/swlh/support-vector-regression-and-its-mathematical-implementation-4800456e4878

Pevec, D., Babic, J., Carvalho, A., Ghiassi-Farrokhfal, Y., Ketter, W., & Podobnik, V. (2020). A survey-based assessment of how existing and potential electric vehicle owners perceive range anxiety. *Journal of Cleaner Production*, *276*. https://doi.org/10.1016/j.jclepro.2020.122779

Pontius, R. G., Thontteh, O., & Chen, H. (2007). Components of information for multiple resolution comparison between maps that share a real variable. Environmental and Ecological Statistics, 15(2), 111–142. https://doi.org/10.1007/s10651-007-0043-y

Qwiklabs. (n.d.). *Sign in | Google Cloud Skills Boost*. Qwiklabs. https://www.cloudskillsboost.google/course_sessions/820967/video/102222

Sahinoglu, G. O., Pajovic, M., Sahinoglu, Z., Wang, Y., Orlik, P. v., & Wada, T. (2018). Battery State-of-Charge Estimation Based on Regular/Recurrent Gaussian Process Regression. *IEEE Transactions on Industrial Electronics*, *65*(5). https://doi.org/10.1109/TIE.2017.2764869

Sautermeister, S., Falk, M., Baker, B., Gauterin, F., & Vaillant, M. (2018). Influence of measurement and prediction uncertainties on range estimation for electric vehicles. *IEEE Transactions on Intelligent Transportation Systems*, *19*(8). https://doi.org/10.1109/TITS.2017.2762829

Shrivastava, P., Soon, T. K., Idris, M. Y. I. Bin, & Mekhilef, S. (2019). Overview of model-based online state-of-charge estimation using Kalman filter family for lithium-ion batteries. In *Renewable and Sustainable Energy Reviews*(Vol. 113). https://doi.org/10.1016/j.rser.2019.06.040

Snihir, I., Rey, W., Verbitskiy, E., Belfadhel-Ayeb, A., & Notten, P. H. L. (2006). Battery open-circuit voltage estimation by a method of statistical analysis. *Journal of Power Sources*, *159*(2). https://doi.org/10.1016/j.jpowsour.2005.11.090

Steinstraeter, M., Lewke, M., Buberger, J., Hentrich, T., & Lienkamp, M. (2020). Range extension via electrothermal recuperation. World Electric Vehicle Journal, 11(2). https://doi.org/10.3390/WEVJ11020041

Varghese, D. (2019, May 10). Comparative study on Classic Machine learning Algorithms. Medium. https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222#:~:text=Logistic%20Regression%20vs%20KNN%20%3A

Vidal, C., Malysz, P., Kollmeyer, P., & Emadi, A. (2020). Machine Learning Applied to Electrified Vehicle Battery State of Charge and State of Health Estimation: State-of-the-Art. In *IEEE Access*. https://doi.org/10.1109/ACCESS.2020.2980961

Wikipedia Contributors. (2019, August 16). Mean absolute error. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Mean_absolute_error

Xu, C., Dai, Q., Gaines, L., Hu, M., Tukker, A., & Steubing, B. (2020). Future material demand for automotive lithium-based batteries. *Communications Materials*, *1*(1). https://doi.org/10.1038/s43246-020-00095-x

Zahid, Xu, K., Li, W., Li, C., & Li, H. (2018). State of charge estimation for electric vehicle power battery using advanced machine learning algorithm under diversified drive cycles. *Energy (Oxford)*, *162*, 871–882. https://doi.org/10.1016/j.energy.2018.08.071

Zhang, M., & Fan, X. (2020). Review on the state of charge estimation methods for electric vehicle battery. In *World Electric Vehicle Journal* (Vol. 11, Issue 1). https://doi.org/10.3390/WEVJ11010023

Zhihang Chen, Shiqi Qiu, Masrur, M. A., & Murphey, Y. L. (2011). Battery state of charge estimation based on a combined model of Extended Kalman Filter and neural networks. *The 2011 International Joint Conference on Neural Networks*, 2156–2163. https://doi.org/10.1109/IJCNN.2011.6033495

Appendices

Appendix A. System Testing

Figure A.1

GCP virtual machine

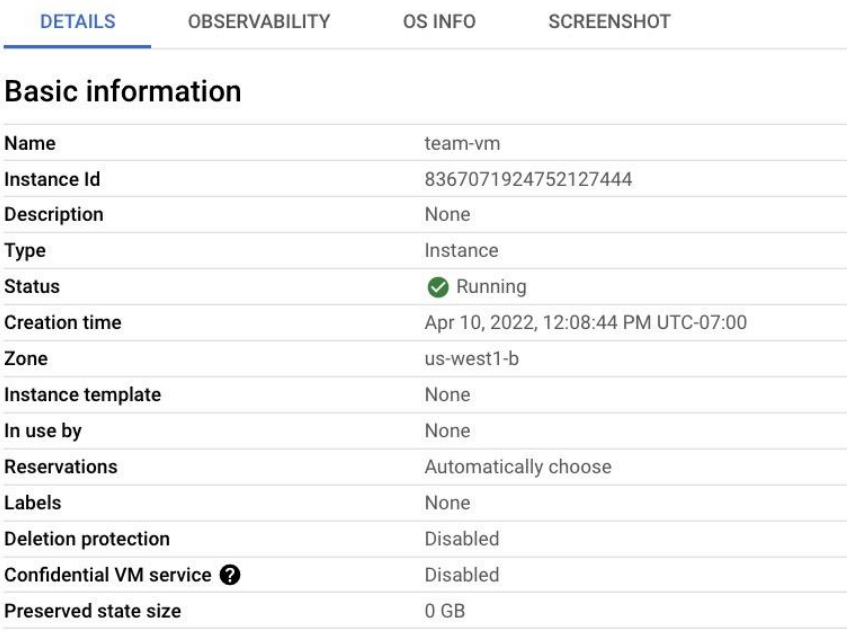


Figure A.2

GCP Pub/Sub to publish the real time data

```
(env) yt@team-vm:~/gcs_python$ ls
achieve  env  pub.py
(env) yt@team-vm:~/gcs_python$ python3 pub.py
       Unnamed: 0  cycle  ...  Temperature_measured  consumption_per_second
0               0    101  ...             26.736797            0.000000e+00
1           20529    101  ...             26.774045            0.000000e+00
2           40982    101  ...             26.697131            0.000000e+00
3           57357    101  ...             33.719042            0.000000e+00
4           57358    101  ...             33.643191            0.000000e+00
...           ...    ...  ...                   ...                    ...
62449       20525    141  ...             34.354656           -1.501043e-07
62450       40981    141  ...             32.545578           -3.744536e-07
62451       20526    141  ...             34.236572           -2.095817e-07
62452       20527    141  ...             34.111468           -9.119157e-07
62453       20528    141  ...             33.994672           -5.289352e-07

[62454 rows x 9 columns]
Publishing 101.0, 0.0, 0.0, 3.69137021699094, 1e-07, 27.0, 26.7367966968874, 0.0 to projects/knn-tea
m3-298/topics/sample_topic at 2022-04-13 05:55:23.831676 ...
Publishing 101.0, 0.0, 0.0, 3.69481074050992, 1e-07, 27.0, 26.7740451315864, 0.0 to projects/knn-tea
m3-298/topics/sample_topic at 2022-04-13 05:55:25.365028 ...
Publishing 101.0, 0.0, 0.0, 3.66651147601726, 1e-07, 26.0, 26.6971306989993, 0.0 to projects/knn-tea
m3-298/topics/sample_topic at 2022-04-13 05:55:26.199202 ...
```

Figure A.3

GCP Pub/Sub topic

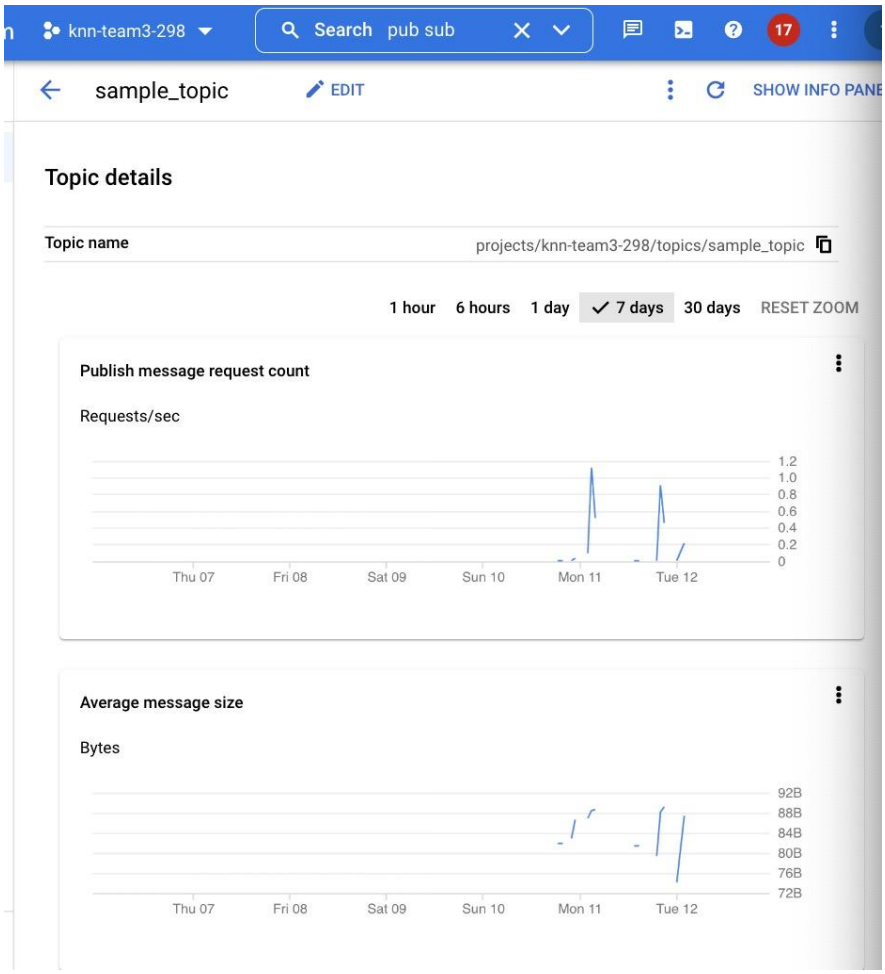


Figure A.4

GCP Dataflow to connect Pub/Sub stream to Bigquery

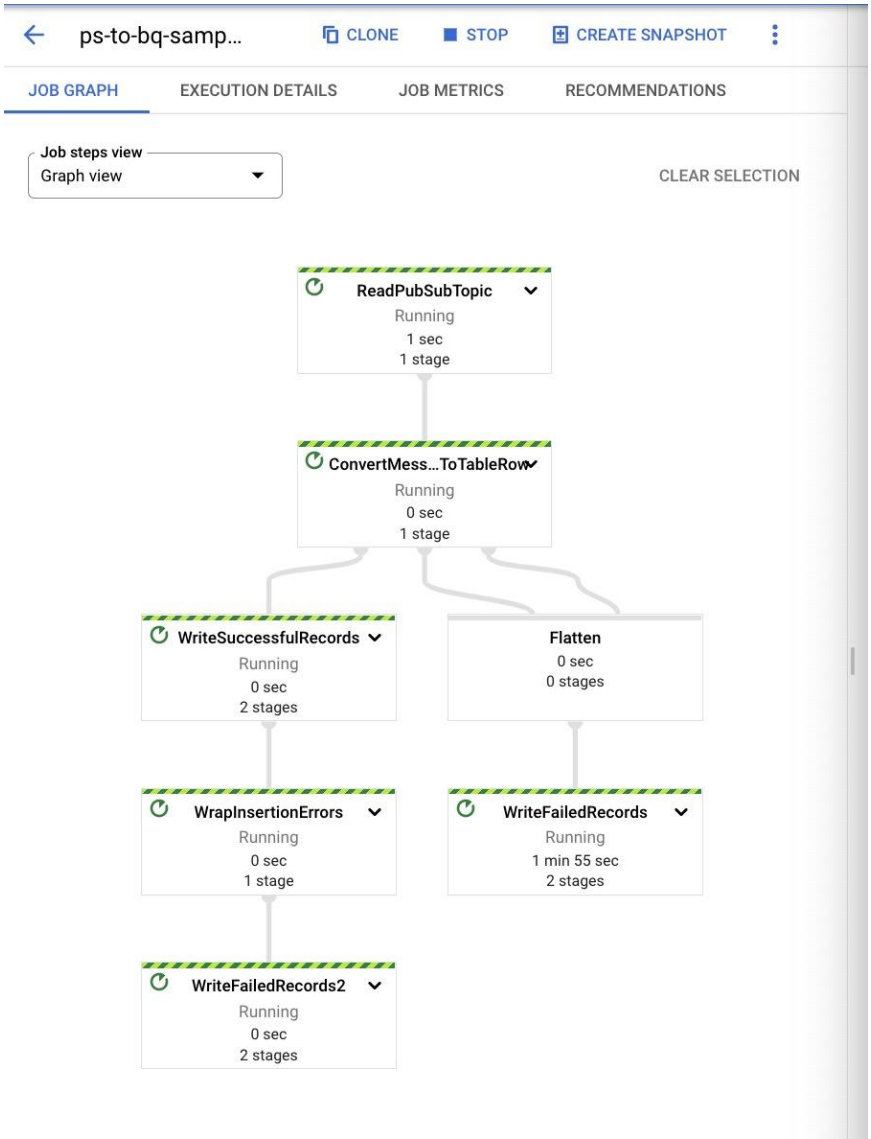


Figure A.5

GCP Bigquery table for storing streaming data

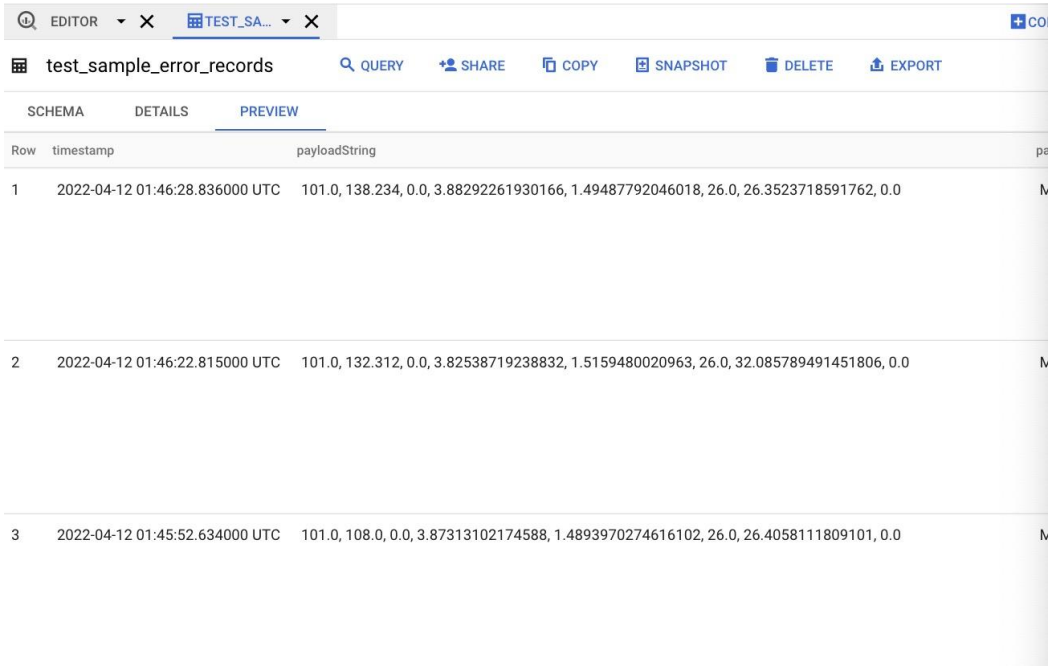


Figure A.6

GCP AI Platform to read real time data from Bigquery

Figure A.7

GCP AI Platform to write data with prediction back to Bigquery

Load DataFrame to Bigquery for visualization

```
[13]: from google.cloud import bigquery
      import datetime
      bigqueryClient = bigquery.Client()
      tableRef = bigqueryClient.dataset("sample").table("prediction")
      realtime['timestamp'] = datetime.datetime.now()
      bigqueryJob = bigqueryClient.load_table_from_dataframe(realtime, tableRef)
      bigqueryJob.result()

[13]: LoadJob<project=knn-team3-298, location=US, id=fee48880-ffe8-43ee-8aa1-5f19208a45b5>

[ ]:
```

Figure A.8

GCP Bigquery table for real time prediction

EDITOR 2 | PREDICT... | COMPOSE NEW QUERY

prediction | QUERY | SHARE | COPY | SNAPSHOT | DELETE | EXPORT

SCHEMA | DETAILS | PREVIEW

Row	cycle	Time	type	Voltage_measured	Current_measured	environment_temperature	Temperature_measured	consumption_per_second	capacity
1	101	462.578	0	4.03657165020561	1.5129972345091	27.0	27.1441139582478	0.0	0.19235996901879193
2	101	462.578	0	4.03657165020561	1.5129972345091	27.0	27.1441139582478	0.0	0.19235996901879193
3	101	462.578	0	4.03657165020561	1.5129972345091	27.0	27.1441139582478	0.0	0.19235996901879193

Figure A.9

Initial GUI Screen prototype

Vehicle status

Mode D

Weather Cold

Miles you have driven 1300

Battery percentage 87

Predict

Figure A.10

Example of Initial prototype GUI Screen After User Input with Prediction

Start!

99.0% 00:00:51 remaning 395.0mi

99.0% 00:00:41 remaning 396.0mi

99.0% 00:00:31 remaning 397.0mi

99.0% 00:00:21 remaning 398.0mi

100.0% 00:00:12 remaning 399.0mi

100.0% 00:00:02 remaning 400.0mi

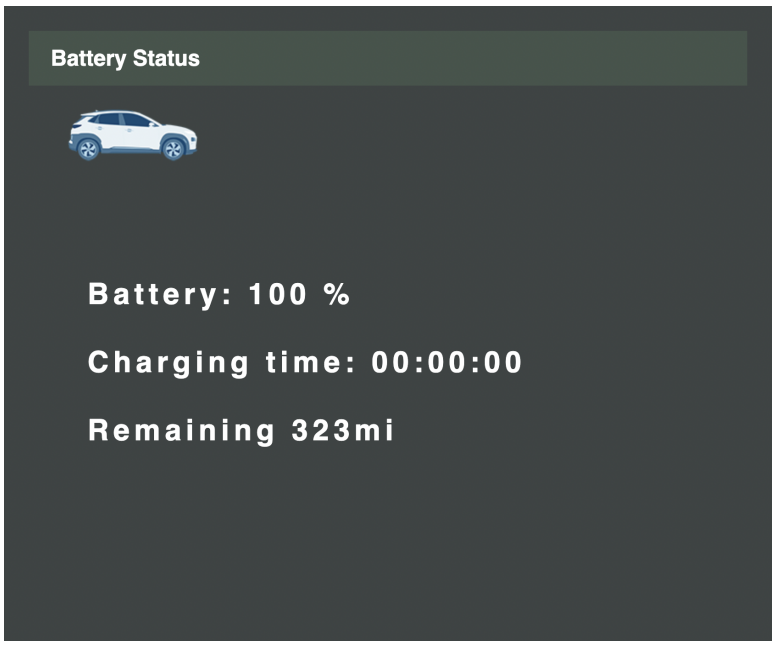
100.0% 00:00:00 remaning 401.0mi

Figure A.11

Updated GUI Screen in final web application

Figure A.12

Example of Updated GUI Screen After User Input with Prediction



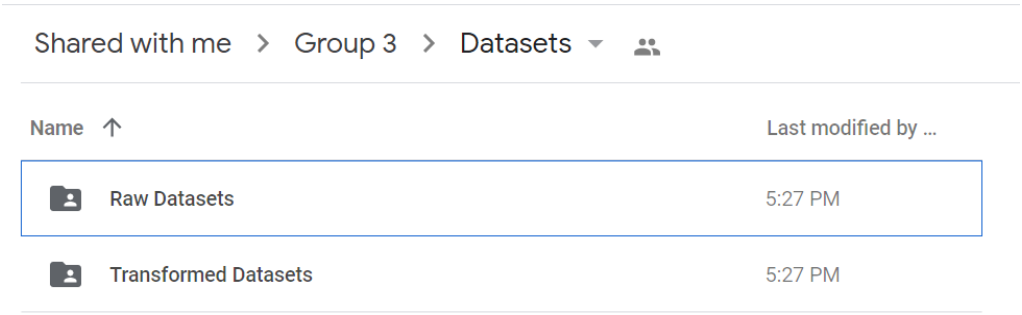
Appendix B. Project Data Source and Management Store

Local modeling:

The original data is stored in the Raw Datasets folder in the Datasets directory on Google Drive as excel files with a readme text file to clarify the details of the raw dataset features. The data can be downloaded for modeling purposes and serves as the backup source in case the source data is no longer available. Transformed datasets are also saved in the Datasets directory.

Figure B.1

Datasets Directory



Online modeling pipeline:

The data is pre-processed and uploaded to Google Cloud Storage. The data is then uploaded to BigQuery tables making it available for modeling purposes.

Figure B.2

Datasets Directory in Google Cloud Storage

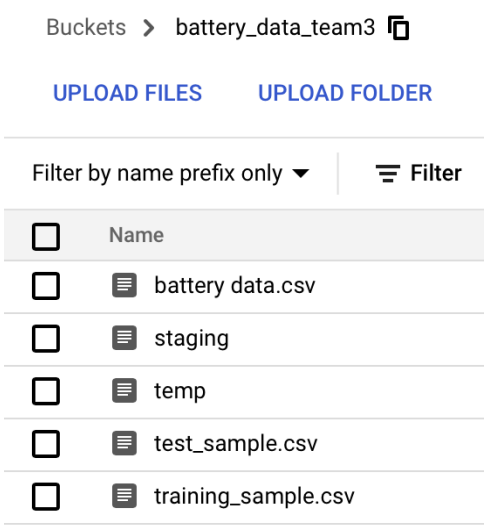
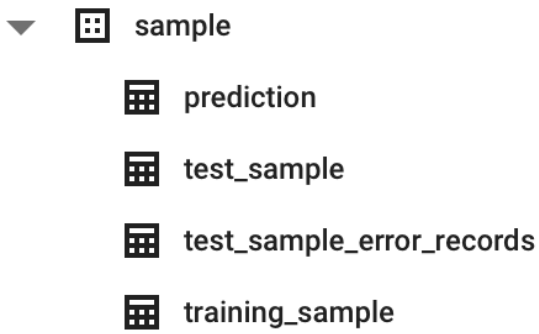


Figure B.3

Datasets Directory in Google Cloud Bigquery



System deployment:

The pre-embedded user data are prepared by different user cases and stored in cloud storage. The real time output data from prediction is transported to interfaces such as Flask for user input and visualization.

Appendix C. Project Program Source Library, Presentation, and Demonstration

Project program artifacts are uploaded to designated folders based on the file structure seen in Figure C.1.

Figure C.1

Project program artifacts