Applied Data Science, 16791

How long will my drone survive?: Modelling flight dynamics for prediction of remaining useful life

TEAM B:

Laura Simandl, Molly Strasser

Submission Date: May 9, 2020

1.0 Introduction

Lithium battery technology has improved dramatically over the last decade, increasing their use in high-tech technologies, such as in unmanned electric aerial vehicles (e-UAVs). E-UAVs can be used for a variety of civilian, military, commercial and humanitarian applications. For example, drones equipped with cameras and lidar have been used for reconnaissance by the military, Amazon is currently investigating the use of drones for parcel delivery, and NGOs and government organizations can use drones to deliver supplies and run reconnaissance trips over natural disaster zones [1] [2]. To guarantee that an e-UAV arrives at its target destination and returns to its origin on a single battery charge (loss or damage of equipment is undesirable), an understanding of their range is important.

The two primary objectives of this project are to:

1. Use battery performance data of fixed wing electric plane (E-540T) provided by NASA's Prognostics Research Center to explore various parametric and non-parametric algorithms for modelling battery performance (Figure 1).
2. Improve the accuracy the remaining flying time predictions of the vehicle.

The four models investigated in this report are: linear regression, polynomial regression, random survival forest, and Weibull accelerated failure time. The results of this research may also be extended to battery state predictions other applications, such as electric vehicles, cellular devices, and wearable devices.

Figure 1. e-UAV Edge 540-T sub-scale version. [3]

2.0 Literature Review of Previously Used Methods

A literature review was performed to gain familiarity with the topic of battery prognostics, and underlying physics governing battery health monitoring. The field of prognostics for lithium ion batteries is well-researched. Much of the literature reviewed uses physics-based modeling to predict the battery discharge for lithium ion e-UAVs.

While many methods have been investigated, the following stochastic filtering methods appear to be those most commonly used for estimations of probabilistic internal battery state (state of charge):

1) ***Extended Kalman Filtering (EKF):*** can be used to linearize a function around the mean and covariance error matrix. The model updates the parameters and estimates based on the previous step. Linearization errors (in the mean and covariance) are introduced in this model and provide a level of uncertainty when estimating for future battery discharge that must be considered.
2) ***Unscented Kauffman Filtering (UKF):*** remedies the problem of linearization error introduced when using EKF method by instead using a deterministic sampling approach [4]. This allows it to capture the posterior mean and covariance to the third order Taylor Series for any non-linearity. UKF assumes a Gaussian distribution of the data.
3) ***Particle Filtering (PF):*** approximates a state's probability density function using a set of Monte Carlo algorithms. 'Particles' are weighted samples used to represent the posterior state of a system, the particle weights in the next step are updated. The State of Charge estimates (SOC) used in our modeling use Unscented Kalman Filtering [3].

One academic paper that cites the NASA HIRF Battery Dataset is "Verification of a Remaining Flying Time Prediction System for Small Electric Aircraft" [3]. This paper uses the NASA dataset to present a verification testing procedure to build trust in predictions of remaining flying time prior to actual flight testing. A second paper, "Battery Remaining Useful Life Forecast Applied in Unmanned Aerial Vehicle" explores use of Extended Kalman Filtering to estimate the remaining useful life of the batteries [5]. The model produced reasonable results but requires more data for validation. Given the breadth of work that has already been done in the field of battery discharge prediction for Li-ion batteries, we decided to investigate the use of various models; the primary 'novel' contribution of this exercise is the use of survival model algorithms to predict remaining flying time.

3.0 Exploratory Data Analysis

An initial investigation of the data was performed to gain familiarity with the dataset, determine whether the objective was feasible, and to identify any anomalies present in the data files.

3.1 Description of Data and Experimental Set-up

The data used for this project was obtained from the NASA Prognostics Center Repository [6]. The data used was collected from ground-based tests on an Edge 540 e-UAV in a high intensity radiation field (HIRF) chamber located at NASA LaRC.

The aircraft was secured within the chamber and the motor and actuators were controlled by an operator in a nearby room outside of the chamber.

The aircraft itself has four batteries (Figure 2). For each battery, internal states including current, voltage, and temperature measurements were recorded for the duration of the experiment (Appendix I). Additionally, motor state data was also monitored (Appendix I).

In total, seven data sets were provided, consisting of a total of 53 test flights (one data file in MATLAB format was provided per flight).

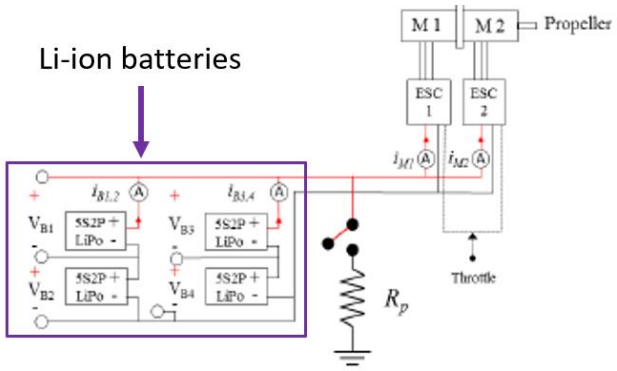


Figure 2. Schematic of Powertrain. Four batteries of interest in this project are indicated by the purple box. [3]

3.2 EDA Results

Results from the exploratory data analysis (EDA) revealed the trends in certain measured data features (current, voltage, temperature, motor speed), and also helped to identify which data files were not appropriate for use in modelling (Appendix II).

- Flight numbers 63, 64, 67, 68, 69 had corrupted battery health monitoring data. This meant that these files did not have state of charge (SOC) estimate data; one of the key feature parameters that was needed for modeling.
- For flights numbered 19, 31, 35, 74, 75, the batteries were never depleted to a 30% state of charge (SOC). Since a 30% SOC the threshold was selected as for this modeling task, it was required to omit the files. Note that we did not have the flexibility to lower the state of charge threshold, because the NASA experiment was
- Flight numbers 16 and 71 showed abnormal/outlier trends, which can be seen in the voltage plots for these datasets (Appendix II).

For each battery, twenty-six sets of time series estimates for the state of charge were provided. Plots of these time series revealed that there was very little variation between the sets of measurements. It was decided to only include one of these time series of state of charge estimates when producing each model. Redundant features sometimes will strongly affect performance of certain models; we wanted to eliminate this possibility.

4.0 Data Cleaning and Preparation

The files listed in the Section 3.2 that were either corrupted or did not meet requirements of this analysis were subsequently omitted for use in modelling. In addition, HIRF datasets 6 through 13 were used for experimental debugging, and did not have Battery Health Monitoring (BHM) data; therefore, they were also omitted. We consistently omitted the same files from each model.

Since the features provided had very different ranges, the data was standardized. This is particularly important for certain models because they have the underlying assumption that the data has a Gaussian distribution.

The time series data was converted into static features at different time horizons for direct use in model fitting and prediction.

a. Calculated static features for the 15%, 25%, 40%, 50%, 60%, 75%, 85% and 100% time horizons.
b. Computed the Mean, median and standard deviation for the values of the original features (Appendix I) over the time horizons; these were the features used as inputs for our model fitting/testing.
c. A subset of the experiments run by NASA in the HIRF chamber also had an additional electronic or 'parasitic' load added to simulate other battery draining activities besides flight. To represent the flights with these loads, a binary data feature was added, called 'Parasitic Load' where the encoding was: 0 – no parasitic load, 1 – parasitic load.
d. Each model was fit twice: once with the state of charge (SOC) included in the features and once without the SOC included in the features. This was done as it is believed that an SOC reading could be incorporated in real-time, though it is not known if that reading is currently available to the pilots in real-time.

5.0 Modelling Methodology

The following section introduces the models that we applied for this problem. Five model types were investigated: Linear Regression, Polynomial Regression (degrees 2 and 3), Random Survival Forest, and Weibull Accelerated Failure Time.

Models were developed using features at various time horizons (refer to Section 4.0 for explanation of static features). We also investigated the effect of including the state of charge estimates as features in our model training.

For all model types, leave-one-subject-out cross validation (LOSOCV) was used for training the models. This strategy ensures that the model generalizes well on the test data. Due to the limited size of the dataset, running a k-fold cross validation, or hold-out method was not appropriate. LOSOCV was computationally feasible and ensured an efficient use of the data.

5.1 Linear Regression

Linear regression was used as a baseline model, as it is one of the simplest machine learning methods and is straightforward to implement. This model was not expected to perform well, as the relationship between the features and the SOC is non-linear.

Package: sklearn LinearRegression [7]

5.2 Polynomial Regression

Polynomial regression was the second parametric model used to fit the data. The introduction of polynomials to allow for a non-linear modeling of the features and SOC was expected to improve the model over the linear regression. Using this model was a logical progression in our investigation, as it added another layer of complexity while still maintaining simplicity and transparency to the process used. Second degree and third degree polynomial models were

tested. Higher degrees were not investigated, as these would have been likely overfit the training data, and not generalize well to new data.

Package: sklearn LinearRegression with PolynomialFeatures and Pipeline [8]

5.3 Random Survival Forest

Random survival forests (RSF) is a non-parametric model. Our literature review revealed that the use of random survival forest models to predict battery discharge has not been explored. Our prediction was that this model would perform better than the regression models.

Package used: sksurv RandomSurvivalForest [9]

5.4 Weibull Accelerated Failure Time

The Weibull accelerated failure time (WAFT) model is a parametric model commonly used in engineering reliability research. This model was chosen since it is a special case of the deep learning model that we would like to implement in future work.

To run the Weibull AFT package in lifelines the number of features was required to be fewer than the number of samples. This required the team to perform Principal Component Analysis (PCA) to reduce the number of features. We retained as many features as can be allowed to successfully run the package.

Package used: Lifelines WeibullAFTFitter [10]

6.0 Modelling and Evaluation Strategy

Performance of models were assessed using mean squared error (MSE), mean absolute error (MAE), and brier score (BS). Each metric was recorded for each iteration of the leave-one-subject-out cross validation (LOSOCV) for each time horizon. At completion, the metrics were averaged to provide an overall metric for the model and time horizon pairing.

1) **Mean Squared Error**
   The mean squared error (MSE) measures the squared difference between the predicted and actual value. In this case, we evaluated the difference between the predicted survival times and the actual survival times. The measure was taken about the mean and the median for the two survival models (RSF and WAFT). In the case of the regression models the MSE about the mean and median are equal.

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2$$

2) **Mean Absolute Error**
   The mean absolute error (MAE) measures the absolute errors of the pairs of predicted and actual values, again we evaluated the predicted and actual survival times. This was done about the mean and median for all models. The survival models were also predicted about the median.

$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i|$$

3) Brier Score

The Brier Score is a common metric used to evaluate survival models. It measures the mean squared difference between the predicted probability assigned to the outcome and the actual outcome. The outcome is a binary variable, in this case survival or not, encoded as 0 for survived and 1 as 'censored', or not survived. The Brier Score will always be a value between 0 and 1, with a lower score being better. In our applications, we calculated the Brier Score for each quartile.

$$BS = \frac{1}{N} \sum_{t=1}^{N} (f_t - o_t)^2$$

where f_t is the predicted probability, o_t is the actual outcome and N is the number of prediction instances.

7.0 Results

Five model types were tried: Linear Regression, Polynomial Regression (degrees 2 and 3), Random Survival Forest, and Weibull Accelerated Failure Time. Each model type was run for seven different time horizons, with two different feature sets. One of the feature sets 'with SOC' contained the State of Charge estimates provided in the HIRF dataset, and the other set 'without SOC' did not contain the State of Charge estimates. Time was not included in the feature set. This was done because the goal of the modeling was to see if survival time could be modeled accurately as a function of battery and parasitic load. Inclusion of time created models that too heavily weighted the time feature.

The predicted survival times versus actual survival times were plotted for each of these combinations and can be found in the Appendix. A comparison was then done between the models using the evaluation metrics. After analysis of the results, the 15% time horizon was deemed to be a poor predictor which led to outliers in the evaluation metrics. Due to this and the effect it had on the scales of the plots the evaluation metrics for this time horizon were not included in these plots.

The confidence intervals of the evaluation metrics for each model were found and can be viewed on the boxplots in Appendix IX. The confidence intervals for the results were important due to the nature of the consequences of incorrect predictions for the imagined application. These consequences consist of things like loss of drone, drone crashing in to public or private property, loss of package on drone which are not insignificant. This means we are less tolerant to prediction errors and we are looking for models with a tighter confidence interval.

The best time horizon for each model and dataset pairing (with SOC, without SOC) was then chosen for each evaluation metric. A subset of the results are shown in Figure 3, A-C. The results for each quantile of the Brier Score can be found in the Appendix IX. The results suggest that linear regression without SOC was the best performing model given our data set. This could be due to the small training set, leading to a preference of lower complexity models. It could be due to an overfitting of the model. As discussed in the 'Future Work' section, a feature analysis to identify the most important features would add benefit to the modeling exercise.

In evaluating the models we found that the Random Survival Forest model was the most consistently best performer, however there exists one case in which Linear Regression performs better than all other models. This case happened at the 50% time horizon. Weibull AFT has large overall errors, due to performance evaluating certain flights that appeared to indicate outlier behavior. However, the behavior of these flights did not seem to affect the other models. Polynomial Regression performed best for degree 2 over degree 3. It is likely that at degree 3 the model began to overfit the data, aligning with our initial thoughts that over degree 3 would not lead to any performance improvements.

Another outcome of the experiments is that time horizons were identified that add have the best predictive results. Initially, the time horizons were arbitrarily picked. We were unable to find any research on the best time horizons for li-ion battery health analysis. On the conclusion of our experiments we are able to say that the time horizons between 40% - 85% do the best in terms of evaluation metrics. In further experimentation, this allows researchers to focus on these time horizons and spend less time working with less well performing time horizons.

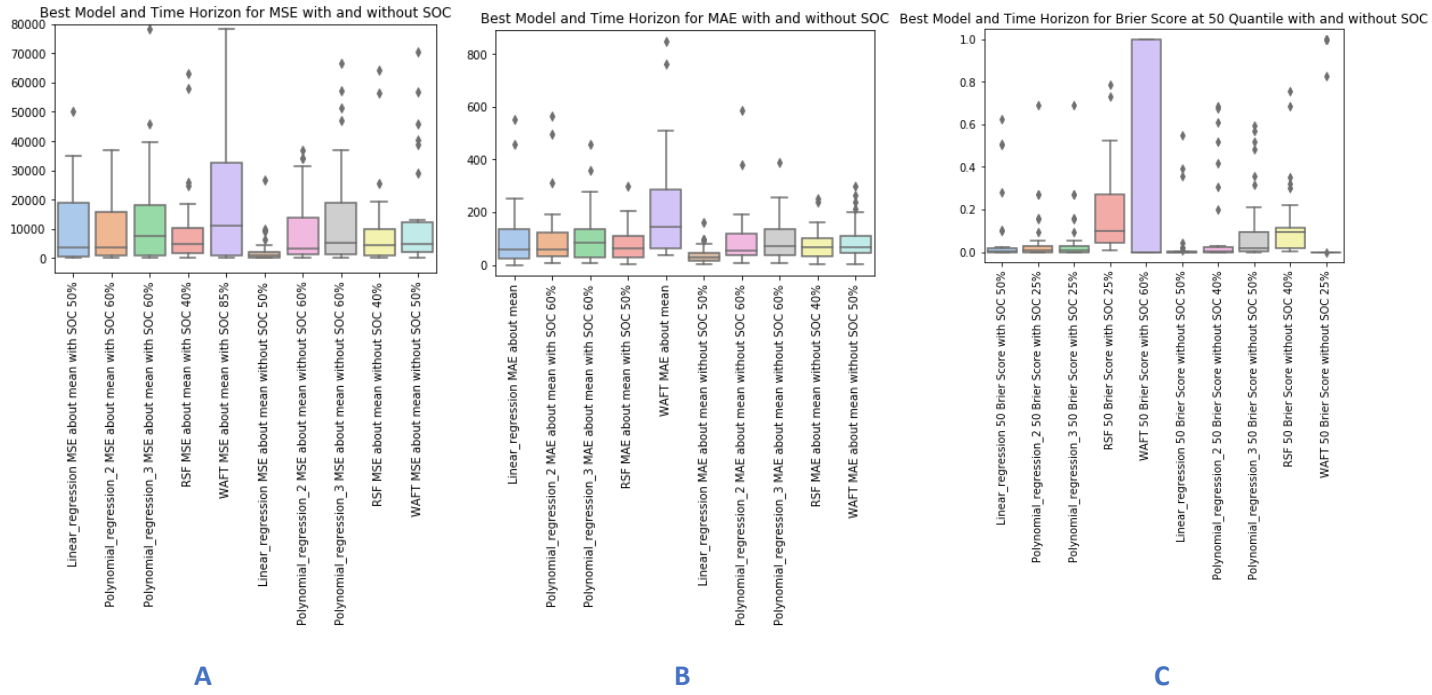


Figure 3. Model comparison using various metrics. A) 'Best' performing time horizon for each model type on basis of MSE. B) 'Best' performing time horizon for each model type on basis of MAE. C) 'Best' performing time horizon for each model type on basis of Brier Score at 50% quantile.

As discussed above, after an exploratory data analysis of the HIRF battery dataset and observations made when modeling the flights, flights were dropped as outliers. Due to this limited dataset, the results might not scale to a larger test, for example the lower complexity models that we see working well

currently may not perform as well when scaled up. More data would be needed to make confident recommendations and observations.

8.0 Future work

We would like to investigate the use of a deep learning survival model on this dataset to see if it enhances the predictive performance. The results would also be benefitted by utilization of data expanding techniques such as bootstrapping to grow the size of the dataset.

9.0 Conclusion

9.1 On the Results

The initial hypothesis was that the survival models (Random Survival Forests, WAFT) would outperform simpler modeling techniques such as linear and polynomial regression. Wht we found was that random survival forest models performed consistently well aligning with our hypothesis. A surprising outcome was that linear regression also performed well and under certain conditions (50% time horizon), the linear regression model performed better than the more complex models. Given these results and the consistency with which RSF performed well, random survival forests is the model we would suggest using out of the models looked at in this evaluation.

9.2 On the Execution of the Project

This project revealed to us how much more time-consuming data cleaning and preparation can be for data science projects, compared to the actual model training and fitting time. It was interesting to see how quickly a dataset was quickly reduced after the exploratory data analysis, data cleaning and preparation stages (by over 25%). This begs the question of the quality of the results of the many papers that were reviewed as part of the literature review portion of the project.

10.0 References

[1] "Amazon Prime Air," [Online]. Available: (https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011. [Accessed 02 05 2020].

[2] "Drones," Unicef, [Online]. Available: https://www.unicef.org/innovation/drones. [Accessed 01 05 2020].

[3] E. Hogge, B. Bole, S. Vazquez, J. Celaya, T. Strom, B. Hill, K. Smalling and C. Quach, "Verification of a Remaining Flying Time Prediction System for Small Electric Aircraft," in *Annual Conference of the Prognostics and Health Management Society 2015*, 2015.

[4] E. Wan and R. van der Merwe, "The Unscented Kalman Filter for Nonlinear Estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*, Lake Louise, AB, Canada, 2000.

[5] D. Souza, J. Torres, L. Albuquerque, A. Brito and V. Pinto, "Battery Remaining Useful Life Forecast Applied in Unmanned Aerial Vehicle," *International Journal of Engineering Research and Applications,* vol. 6, no. 12, pp. 5-9, 2016.

[6] C. Kulkarni, E. Hogge, C. Quach and K. Goebel, "HIRF Battery Data Set," NASA Ames Prognostics Data Repository, NASA Ames Research Center, Moffett Field, CA, [Online]. Available: http://ti.arc.nasa.gov/project/prognostic-data-repository. [Accessed 01 03 2020].

[7] "sklearn.linear_model.LinearRegression," scikit learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linear%20regression#sklearn.linear_model.LinearRegression. [Accessed 01 04 2020].

[8] "sklearn.linear_model.LinearRegression," scikit learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linear%20regression#sklearn.linear_model.LinearRegression. [Accessed 04 04 2020].

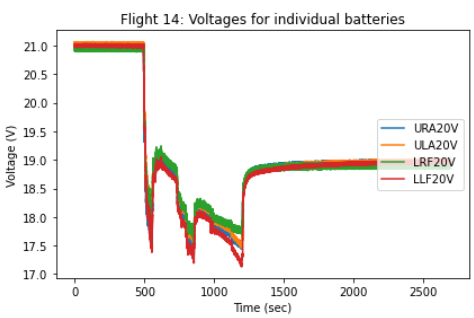
[9] "sksurv.ensemble.RandomSurvivalForest," scikit-survival, [Online]. Available: sksurv.ensemble.RandomSurvivalForest. [Accessed 04 04 2020].

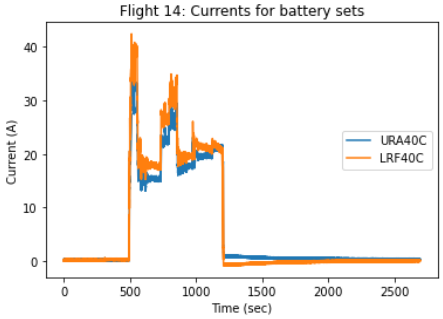
[10] "WeibullAFTFitter," lifelines, [Online]. Available: https://lifelines.readthedocs.io/en/latest/fitters/regression/WeibullAFTFitter.html. [Accessed 15 04 2020].

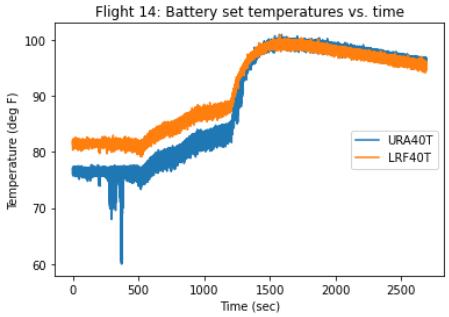
APPENDIX I. NASA HIRF Dataset Features

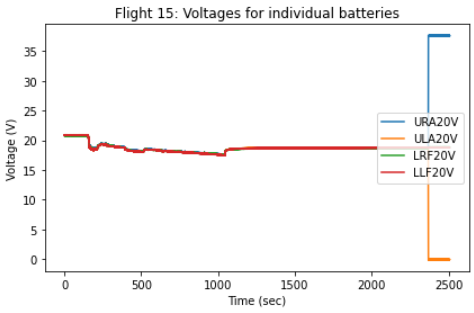
General	
t	*Time*
Motor State	
RPM	*Revolutions per minute*
Voltage	
LLF 20V	*Lower Left Forward – Battery Voltage*
ULA 20V	*Upper Left After – Battery Voltage*
LRF 40V	*Lower Right Forward – 2 Battery Set Voltage*
URA 40V	*Upper Right After – Battery Set Voltage*
LRF 20V	*Lower Right Forward – Battery Voltage*
URA 20V	*Upper Right After – Battery Voltage*
Current	
LLF 20C	*Lower Left Forward – Battery Current*
LRF 40C	*Lower Right Forward – Battery Set Current*
ULA 20C	*Upper Left After – Battery Current*
URA 40C	*Upper Right After – Battery Set Current*
LESCIM2	*Left ESC current sensor*
RESCIM1	*Right ESC current sensor.*
Temperature	
LLF 20T	*Lower Left Forward – Battery Temperature*
ULA 20T	*Upper Left After – Battery Temperature*
LRF 40T	*Lower Left Forward - Battery Set Temperature*
URA 40T	*Upper Right After – Battery Set Temperature*
Battery Health Monitoring Data	
LLF SOC	*Lower Left Forward – State of Charge*
LRF SOC	*Upper Left After – State of Charge*
ULA SOC	*Lower Left Forward – State of Charge*
URA SOC	*Upper Right After – State of Charge*

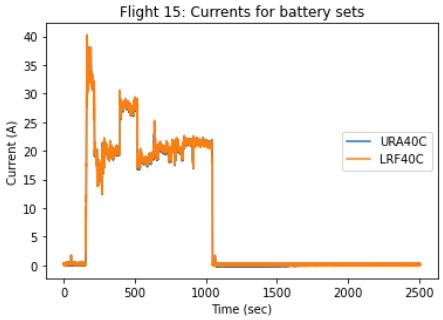
APPENDIX II. Plots of Measured Experimental Values

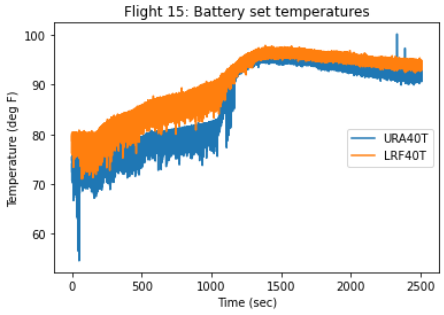


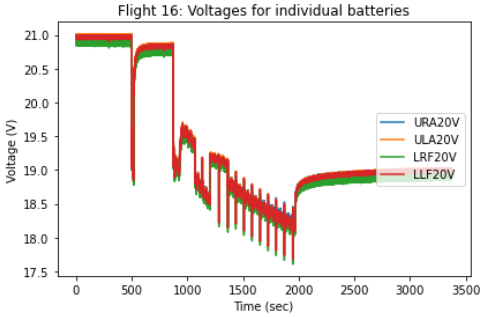


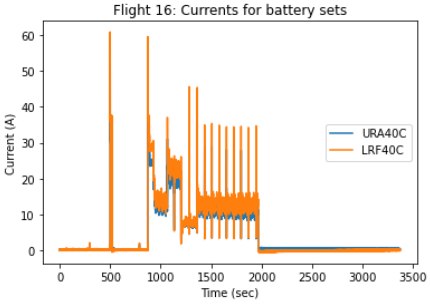


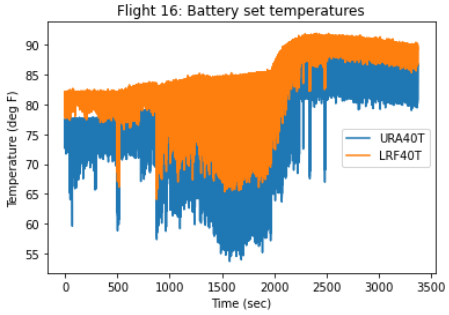


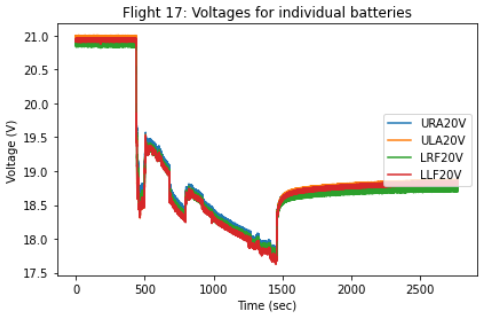


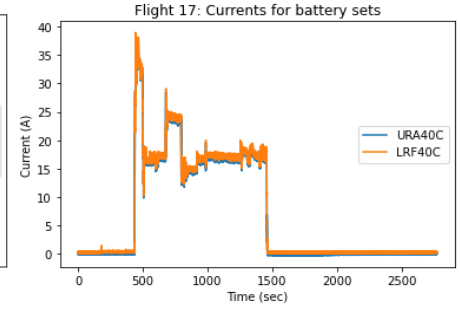


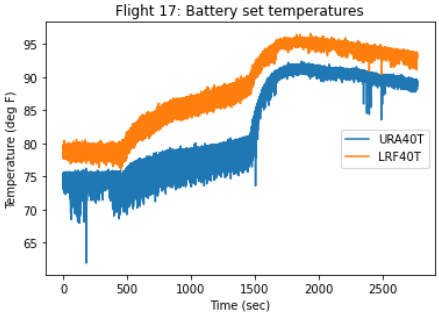


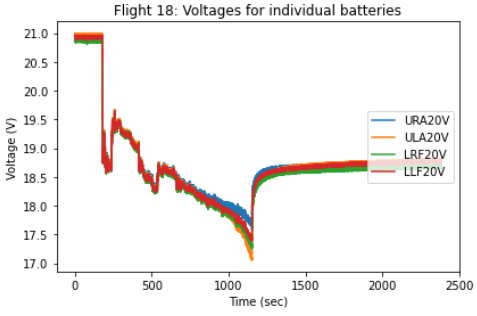


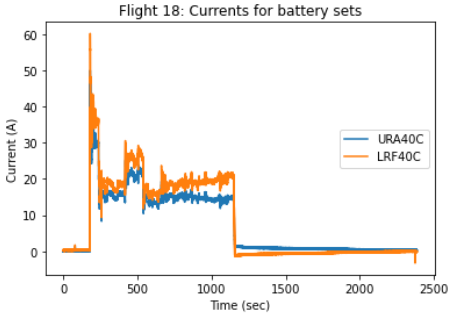


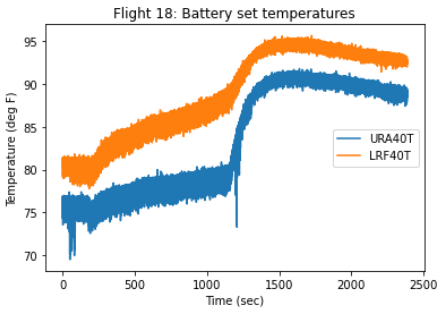


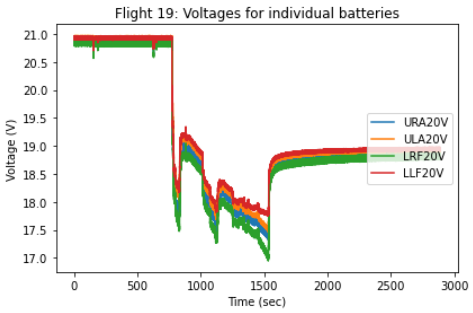


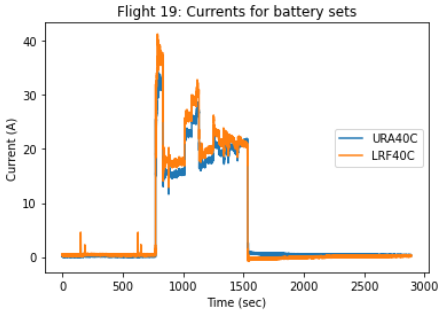


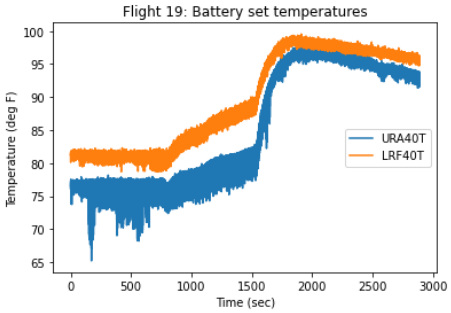


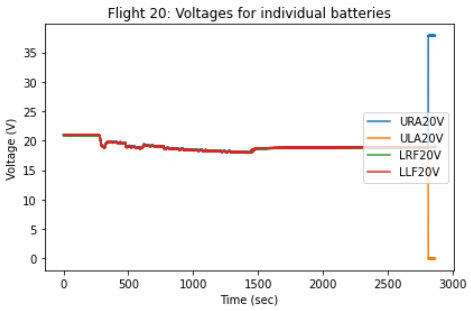


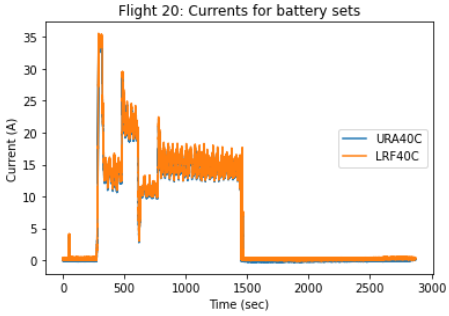


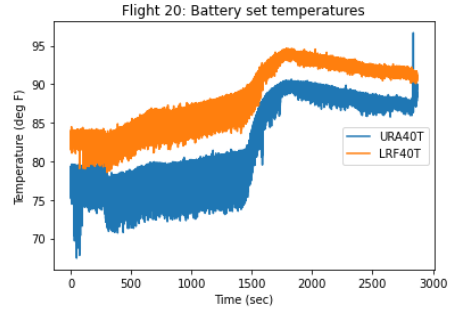

Flight 19: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

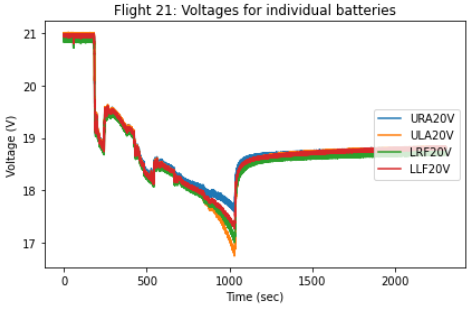

Flight 19: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

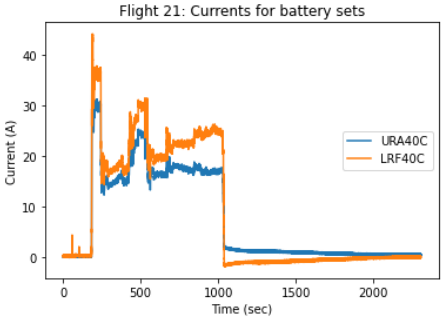

Flight 19: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

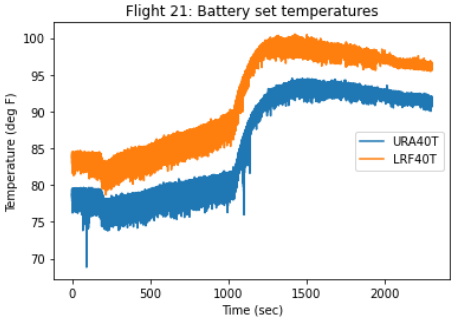

Flight 20: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

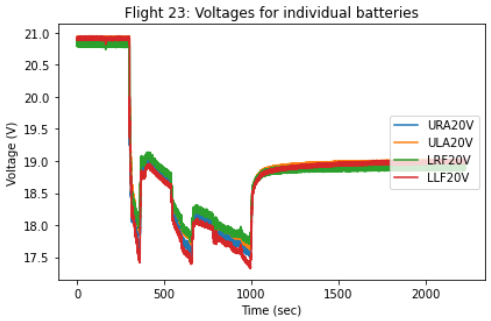

Flight 20: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

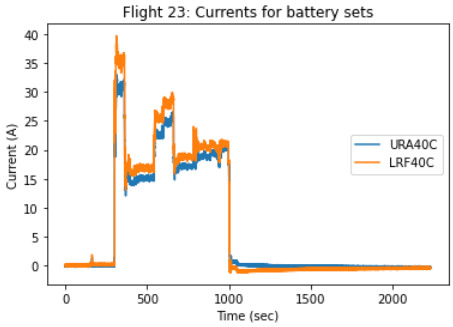

Flight 20: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

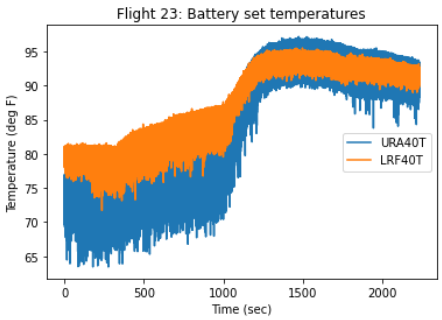

Flight 21: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

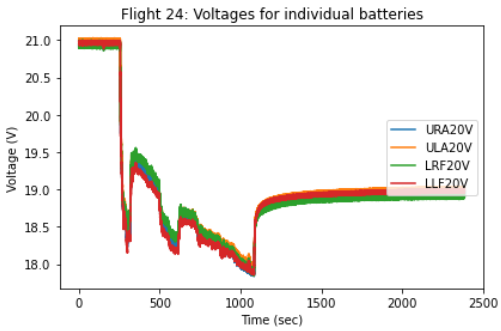

Flight 21: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

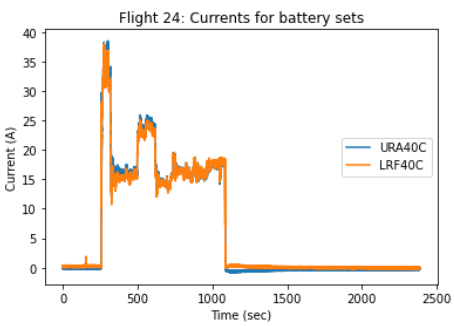

Flight 21: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

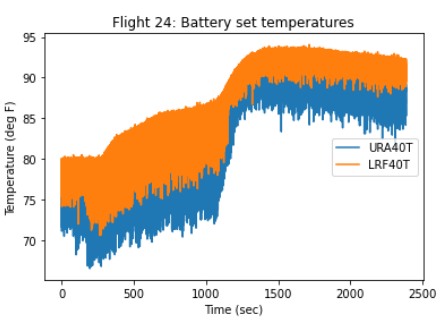

Flight 23: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

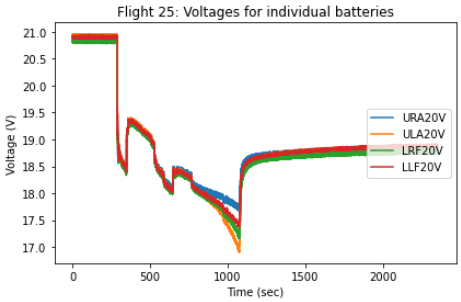

Flight 23: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

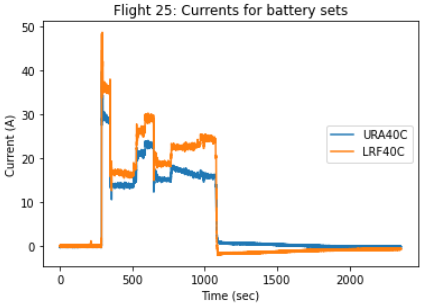

Flight 23: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

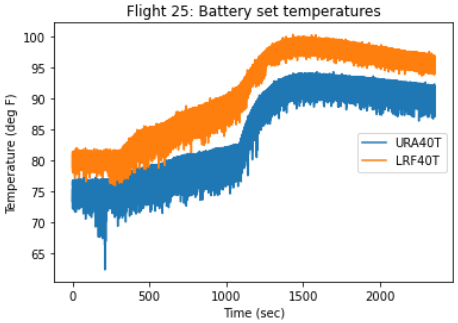

Flight 24: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

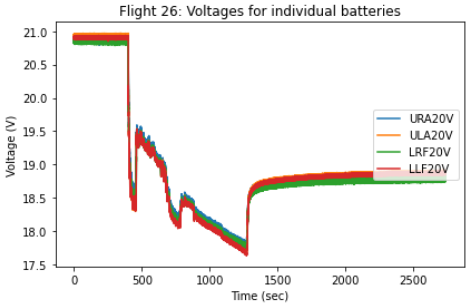

Flight 24: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

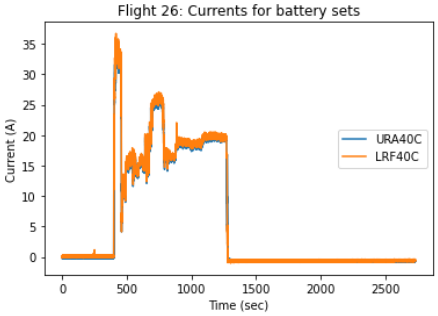

Flight 24: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

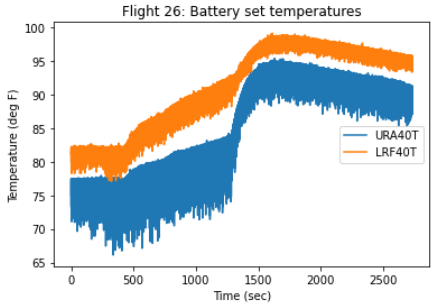

Flight 25: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

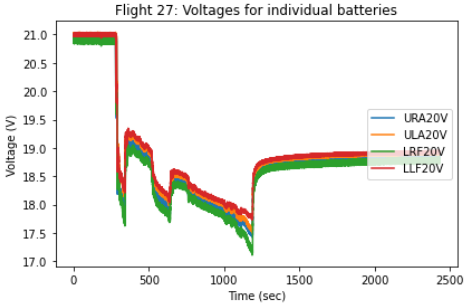

Flight 25: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

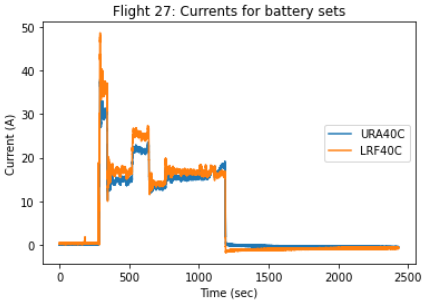

Flight 25: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

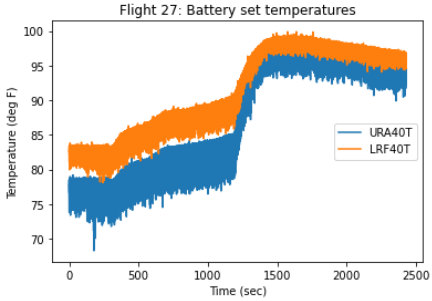

Flight 26: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

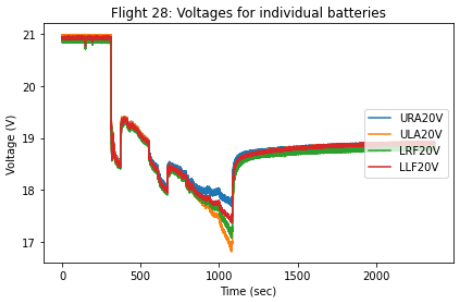

Flight 26: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

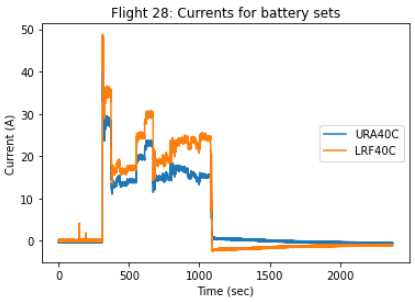

Flight 26: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

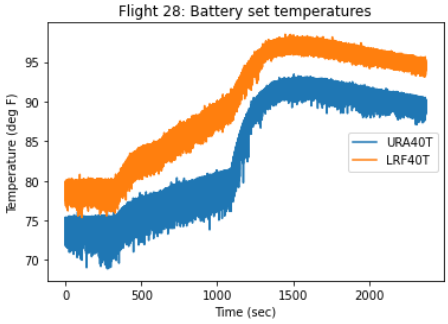

Flight 27: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

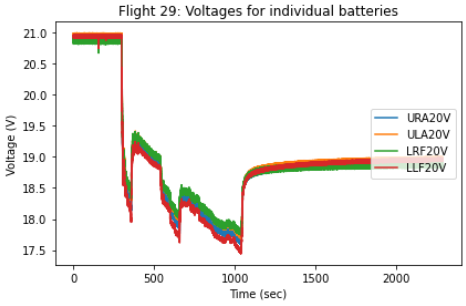

Flight 27: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

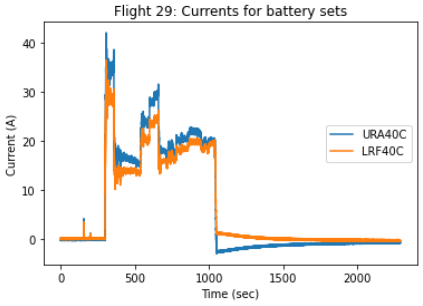

Flight 27: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

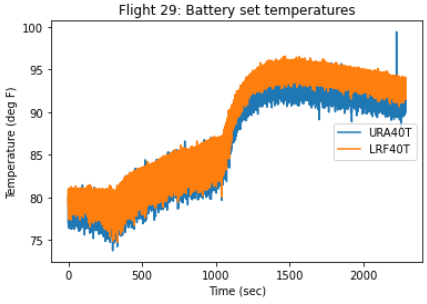

Flight 28: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

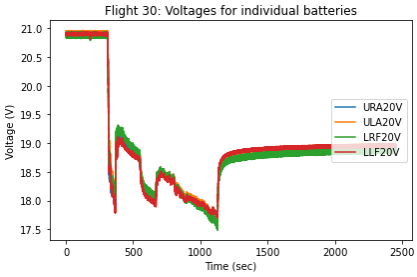

Flight 28: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

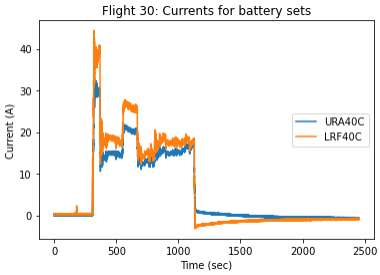

Flight 28: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

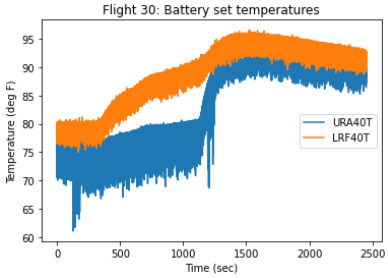

Flight 29: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

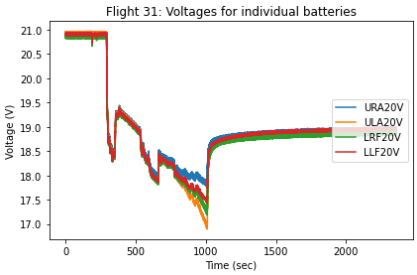

Flight 29: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

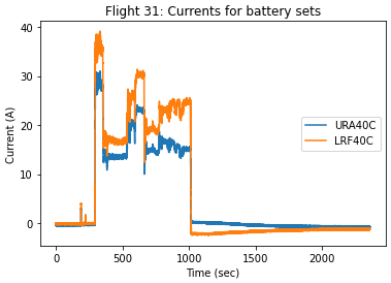

Flight 29: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

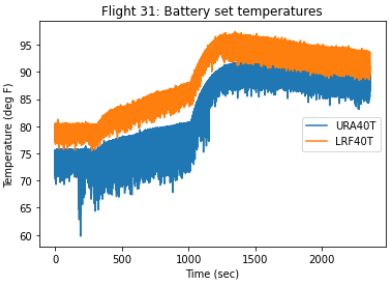

Flight 30: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

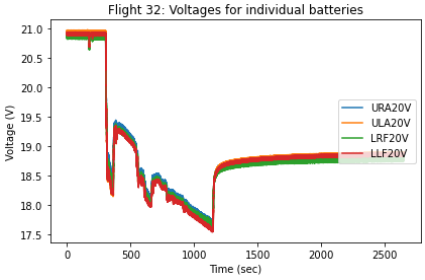

Flight 30: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

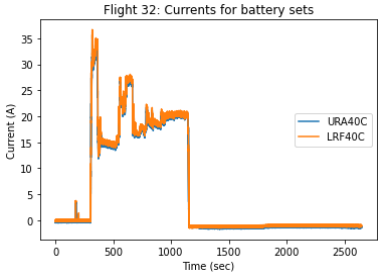

Flight 30: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

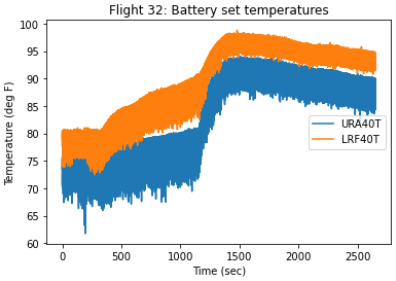

Flight 31: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

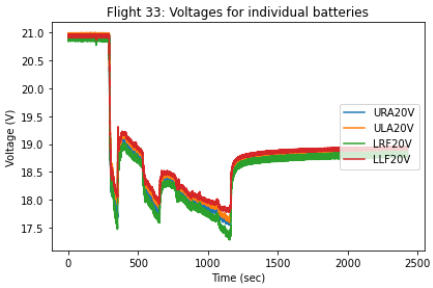

Flight 31: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

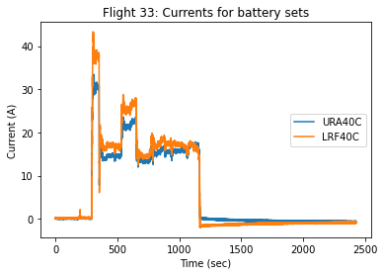

Flight 31: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

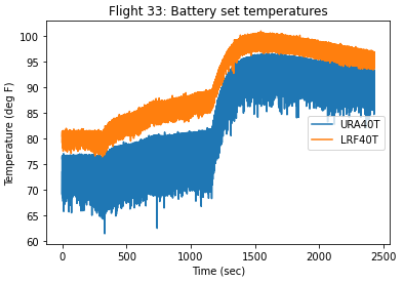

Flight 32: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

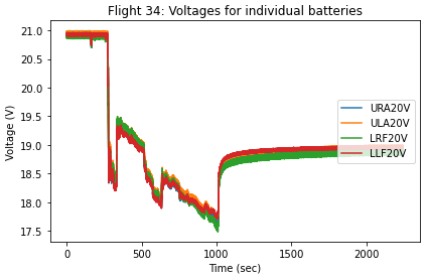

Flight 32: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

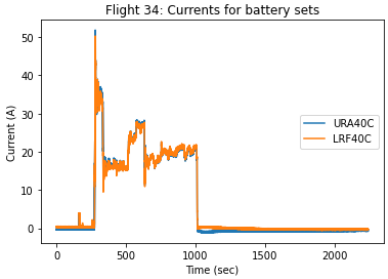

Flight 32: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

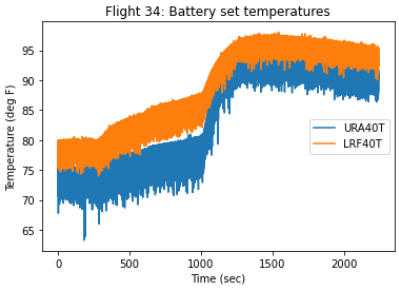

Flight 33: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

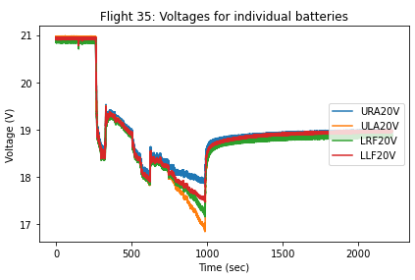

Flight 33: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

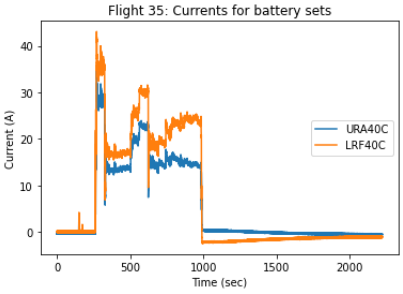

Flight 33: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

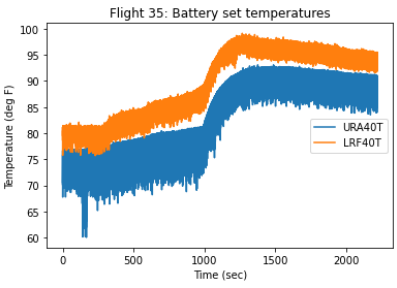

Flight 34: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

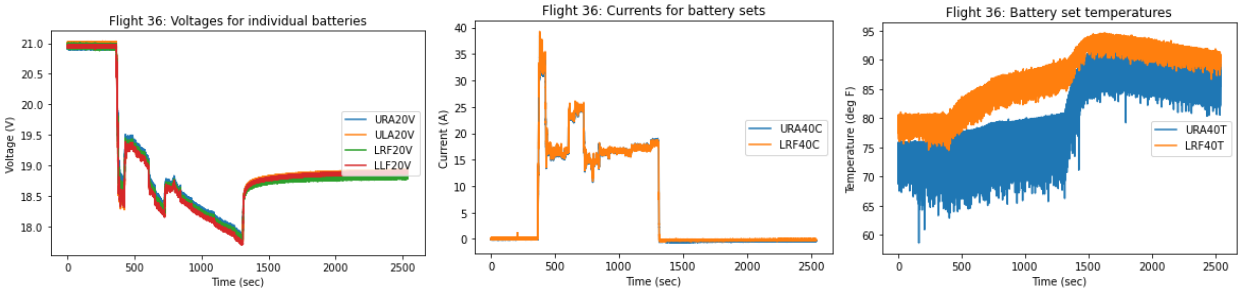

Flight 34: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

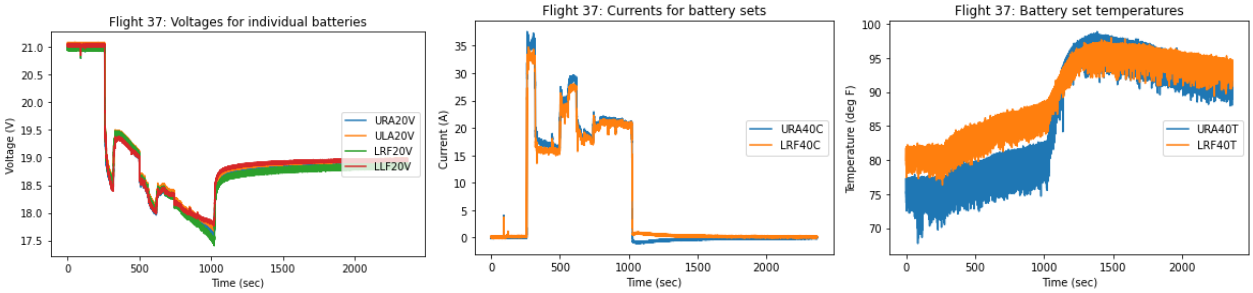

Flight 34: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

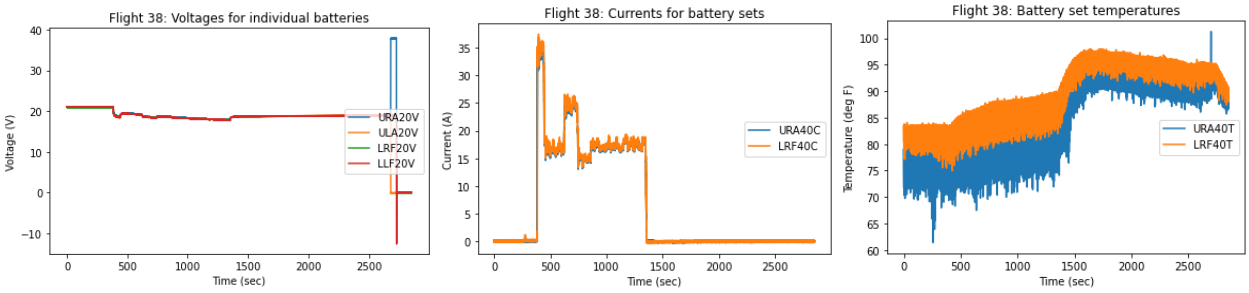

Flight 35: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

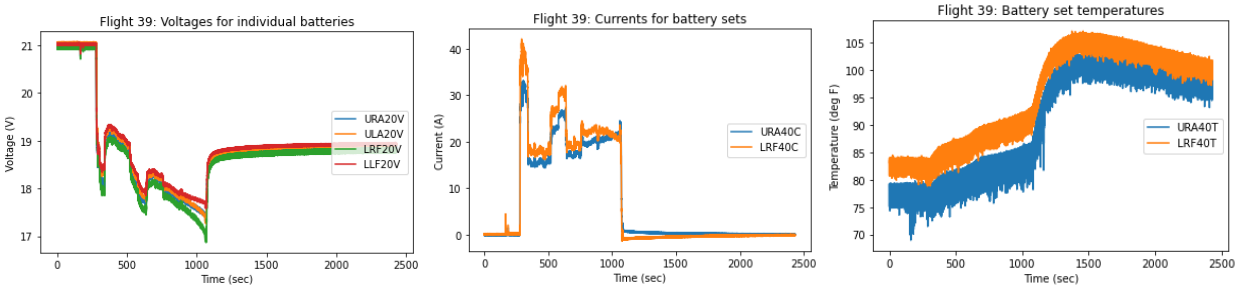

Flight 35: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

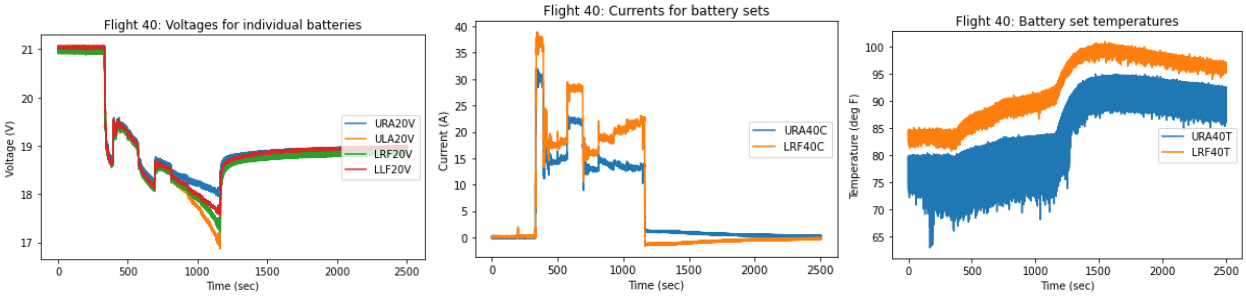

Flight 35: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

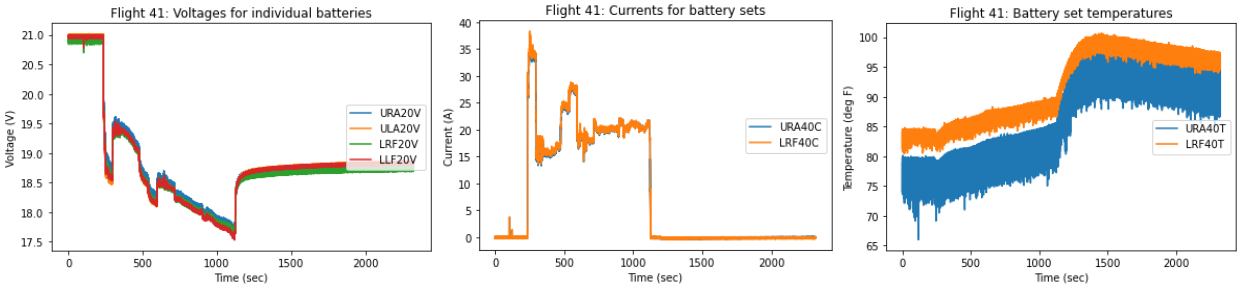

Flight 36: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V
Flight 36: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C
Flight 36: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

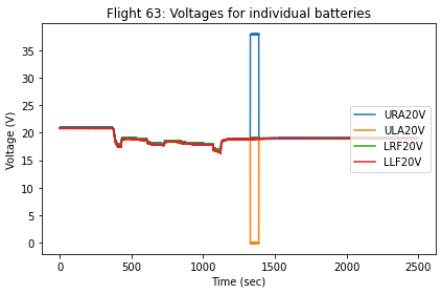

Flight 37: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V
Flight 37: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C
Flight 37: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

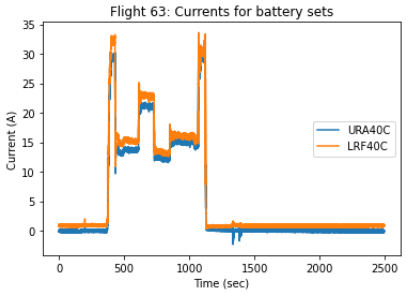

Flight 38: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V
Flight 38: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C
Flight 38: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T


Flight 39: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V
Flight 39: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C
Flight 39: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

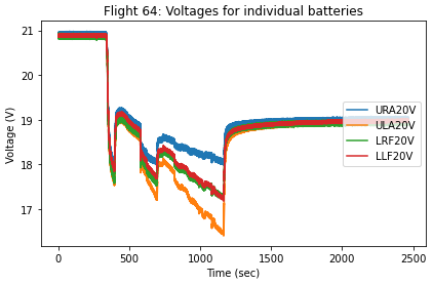

Flight 40: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V
Flight 40: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C
Flight 40: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

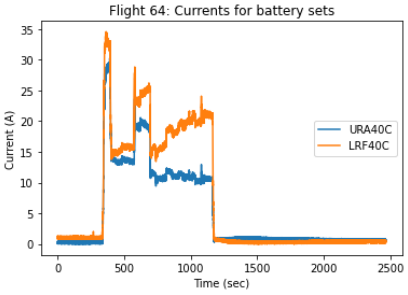

Flight 41: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V
Flight 41: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C
Flight 41: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T



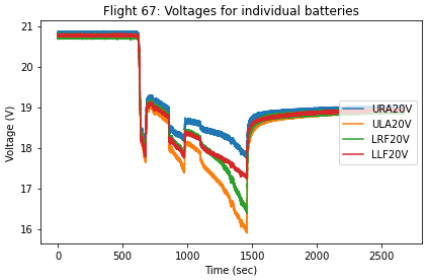


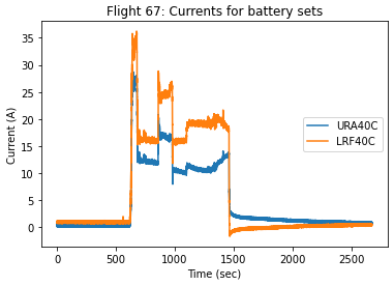
Temperature file corrupted

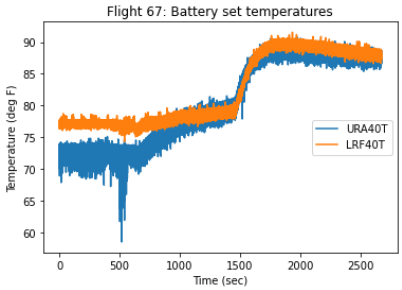


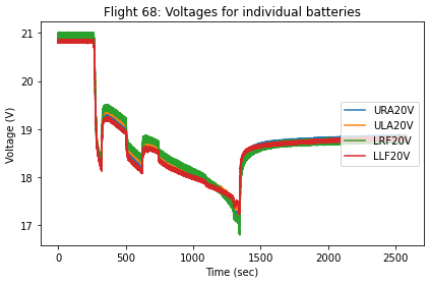


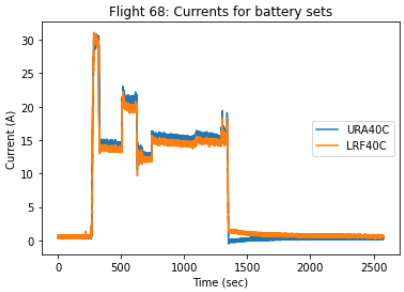
Temperature file corrupted

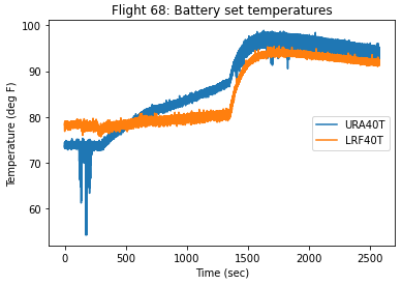


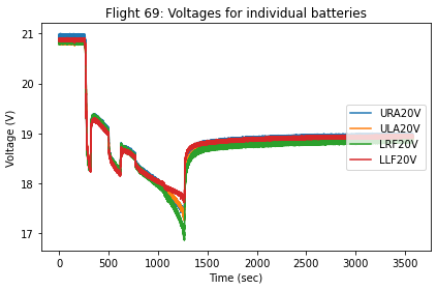


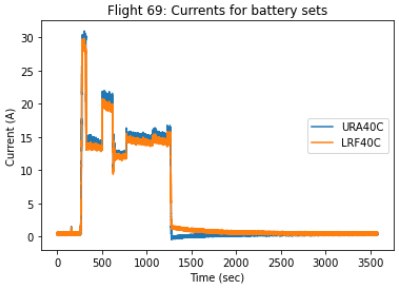


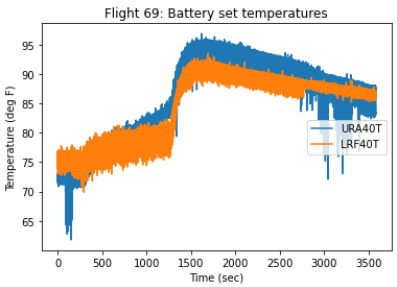


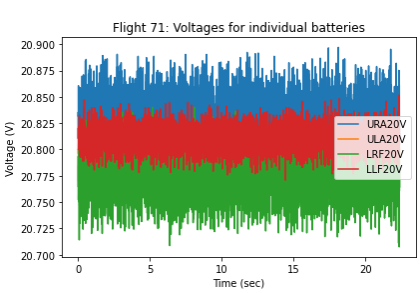


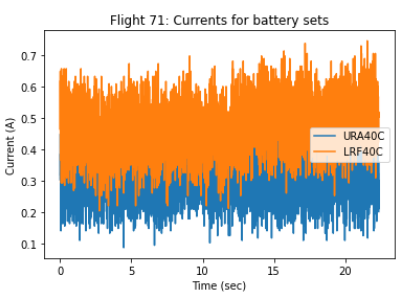


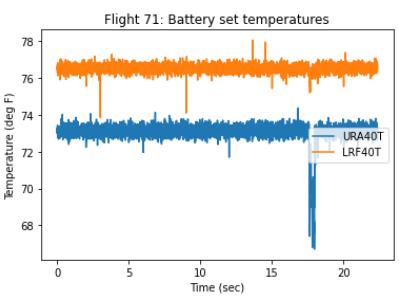


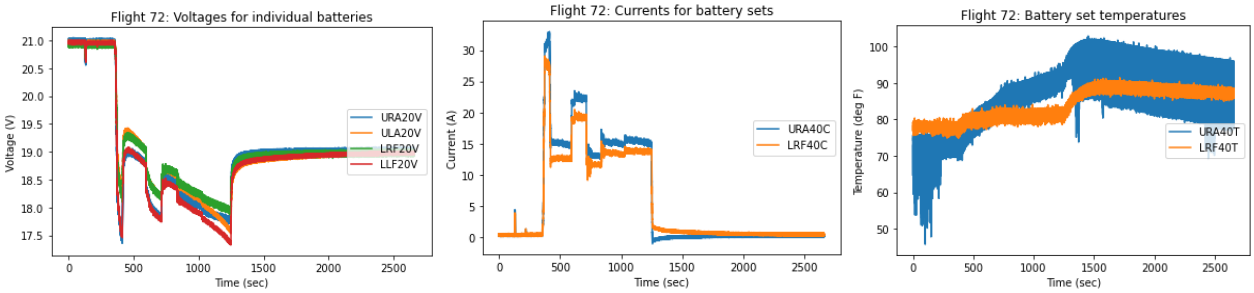


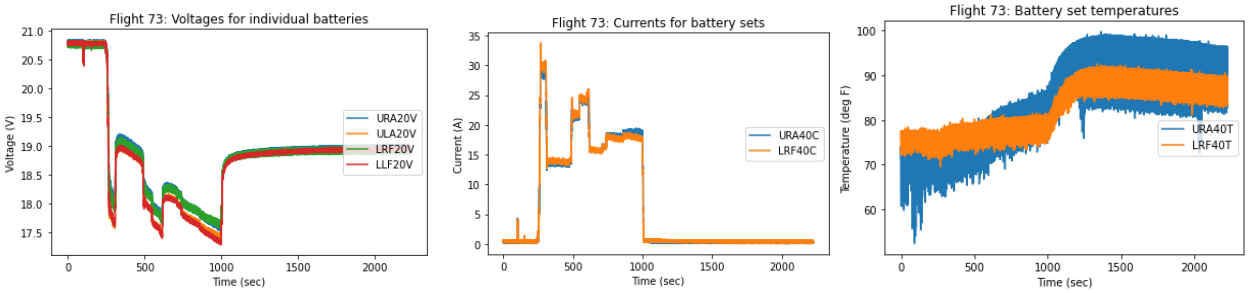


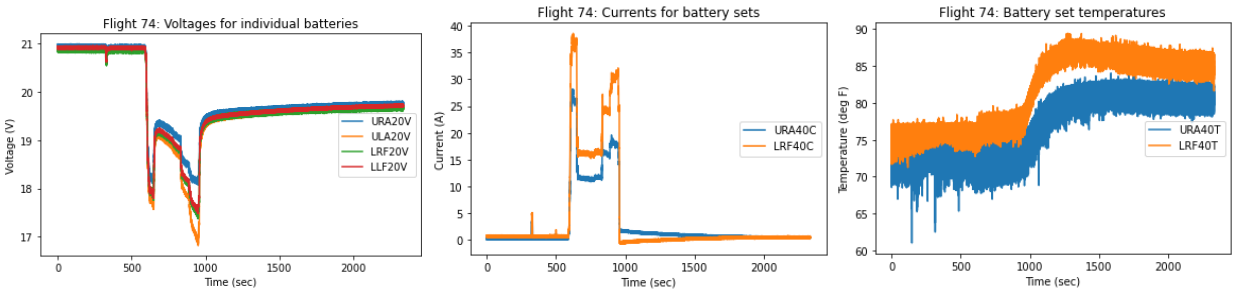


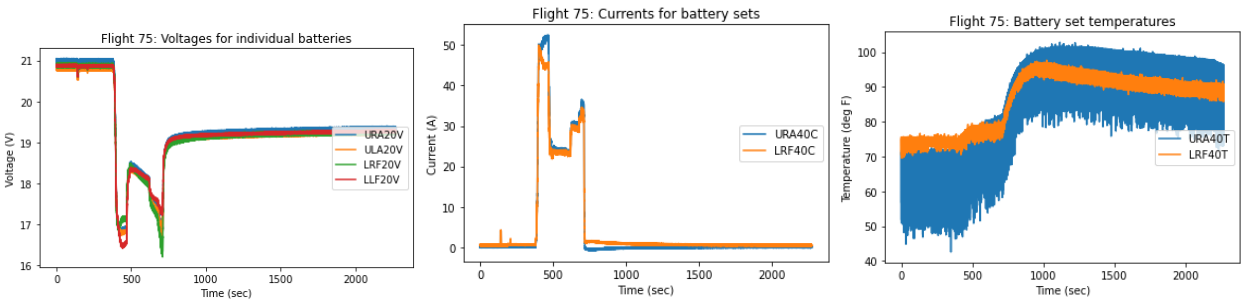


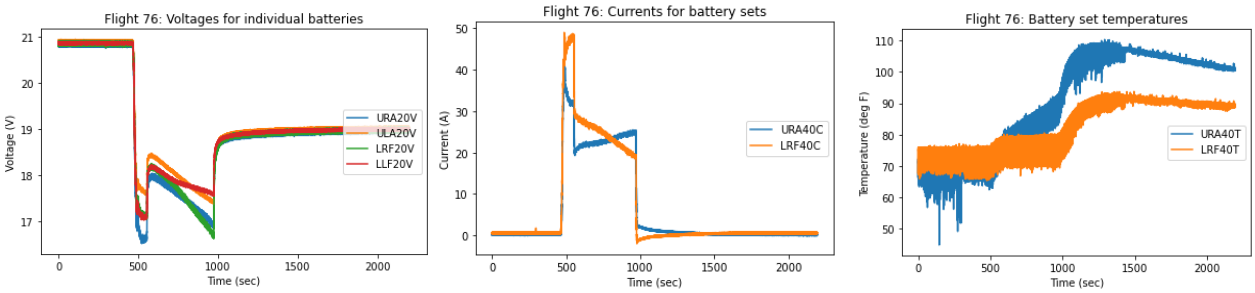


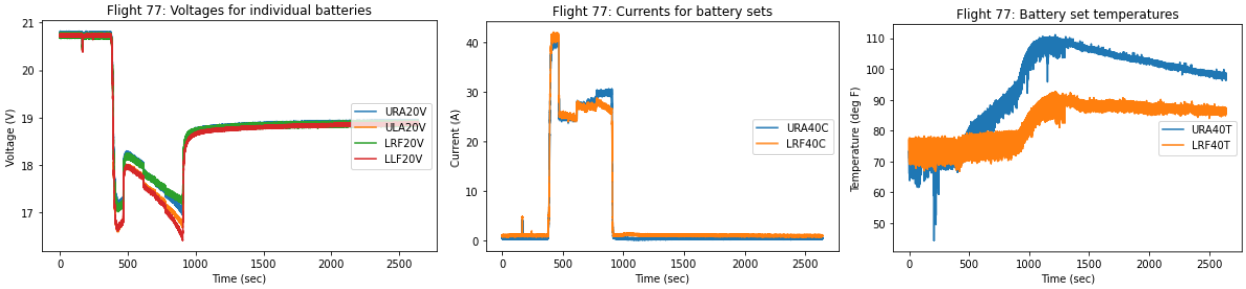

Flight 72: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V
Flight 72: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C
Flight 72: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

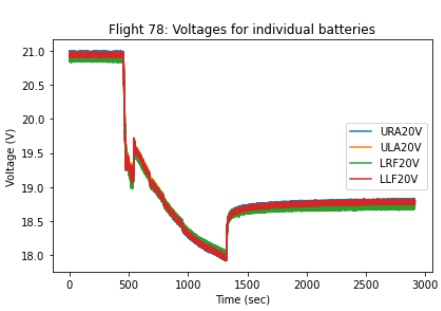

Flight 73: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V
Flight 73: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C
Flight 73: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

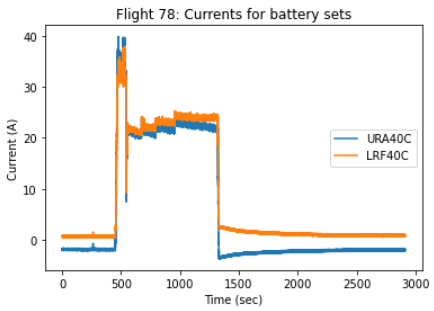

Flight 74: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V
Flight 74: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C
Flight 74: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

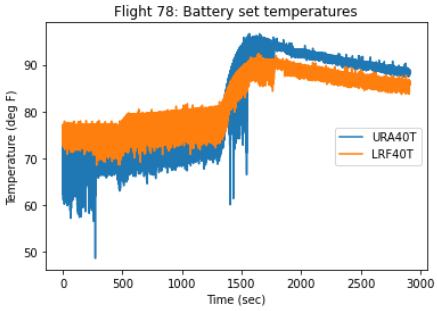

Flight 75: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V
Flight 75: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C
Flight 75: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

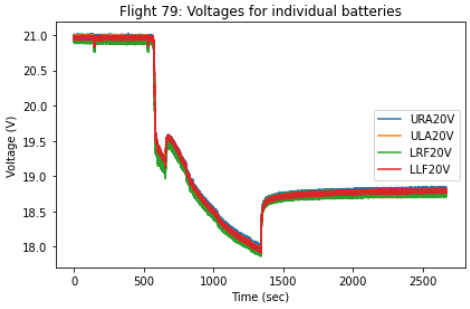

Flight 76: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V
Flight 76: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C
Flight 76: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

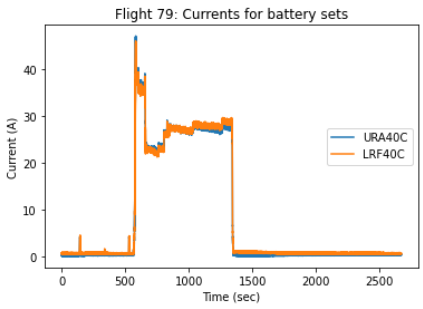

Flight 77: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V
Flight 77: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C
Flight 77: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

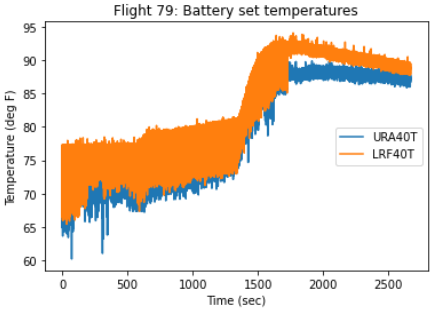

Flight 78: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

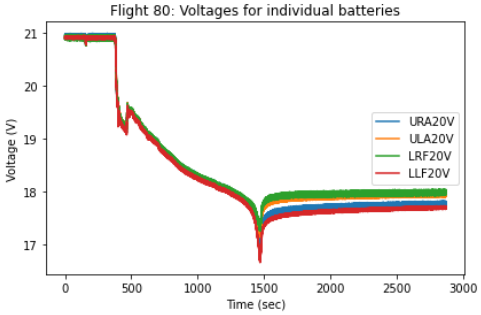

Flight 78: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

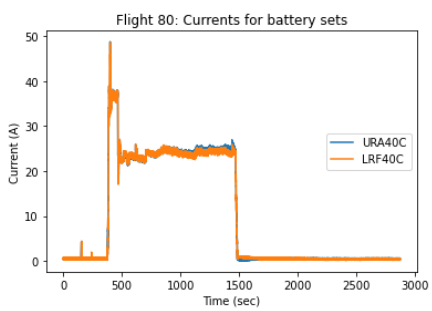

Flight 78: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

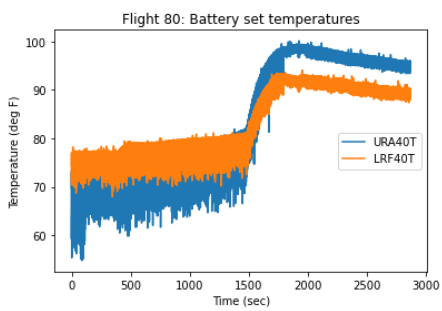

Flight 79: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

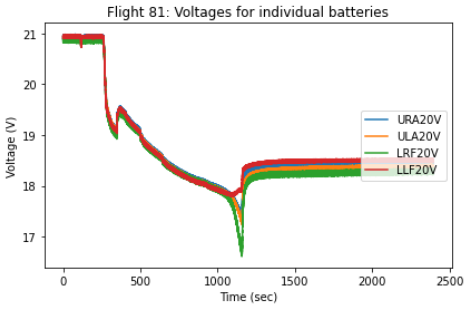

Flight 79: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

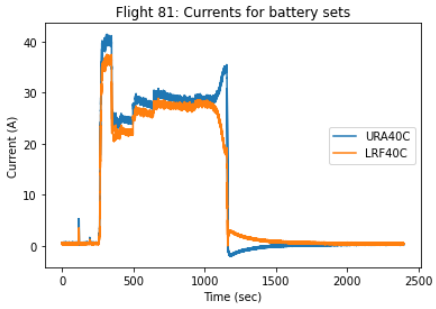

Flight 79: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

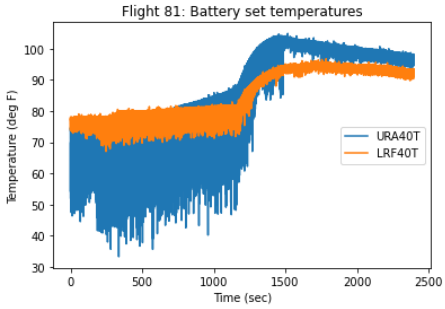

Flight 80: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

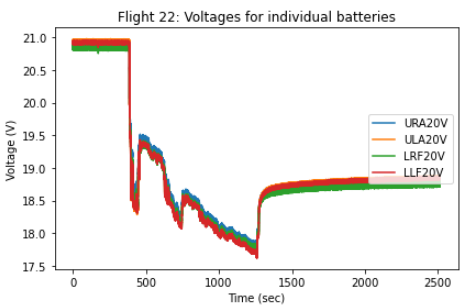

Flight 80: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

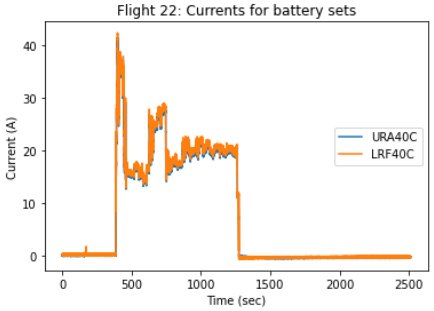

Flight 80: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

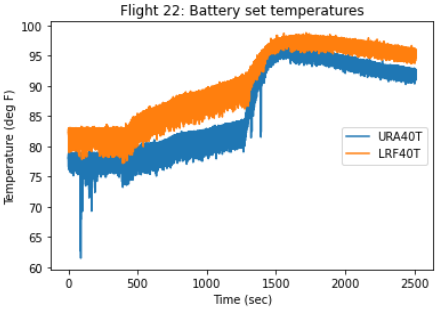

Flight 81: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V


Flight 81: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C


Flight 81: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T


Flight 22: Voltages for individual batteries
Voltage (V)
Time (sec)
URA20V
ULA20V
LRF20V
LLF20V

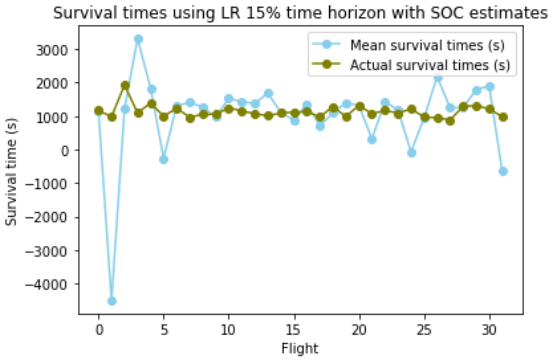

Flight 22: Currents for battery sets
Current (A)
Time (sec)
URA40C
LRF40C

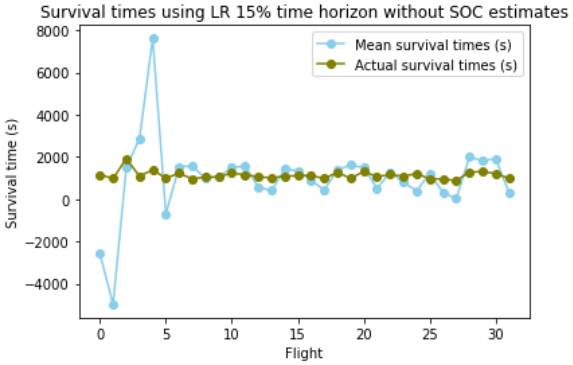

Flight 22: Battery set temperatures
Temperature (deg F)
Time (sec)
URA40T
LRF40T

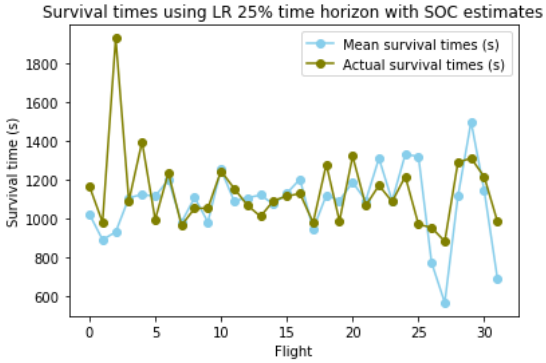
APPENDIX III. Survival Time Estimates using Linear Regression Model

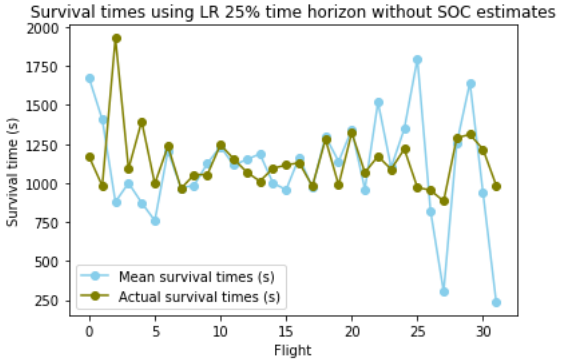
Time horizons: 0 – 50%

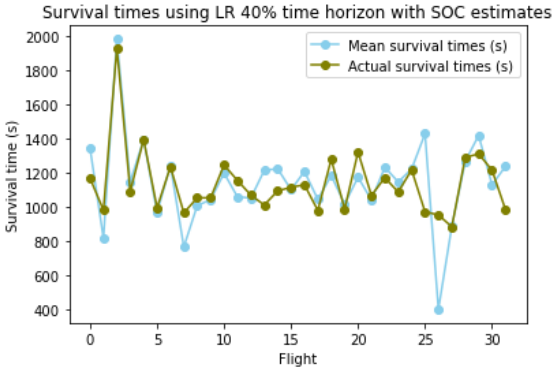
Figures on the left of the page included state of charge (SOC) estimates as features in the model (figures on the right did not).

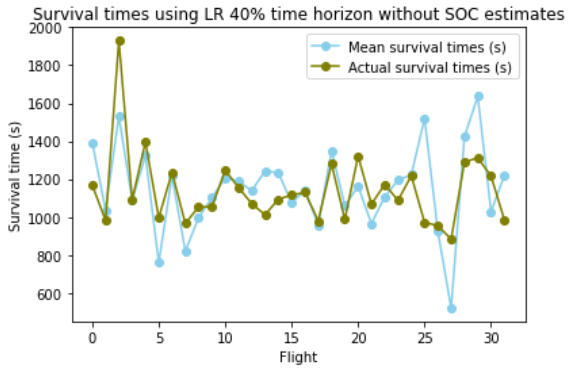


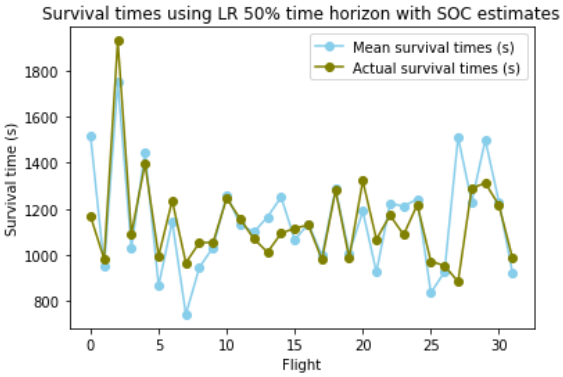


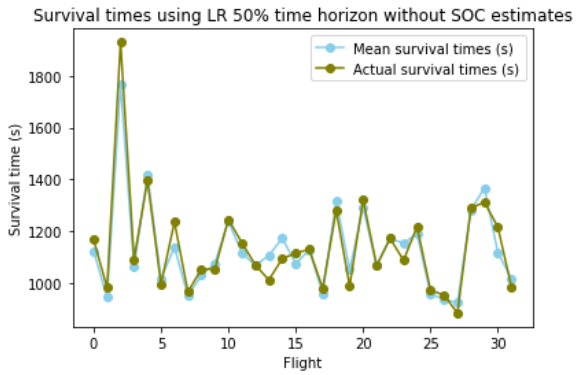








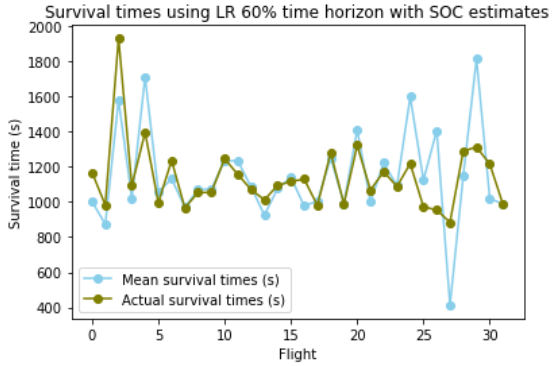


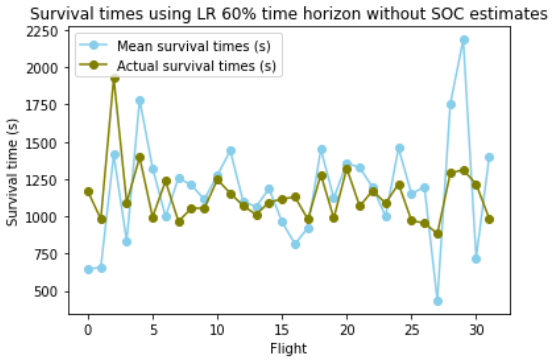


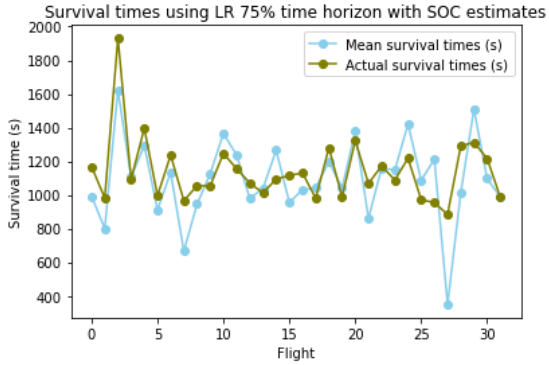
APPENDIX III continued. Survival Time Estimates using Linear Regression Model

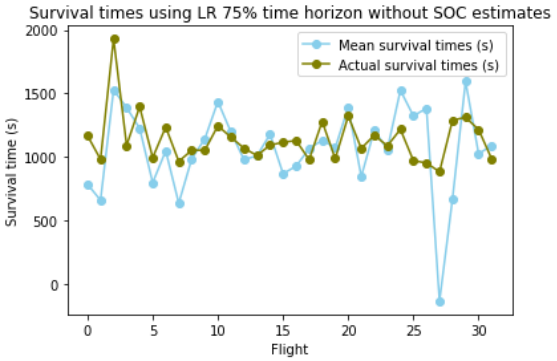
Time horizons: 60-100 %

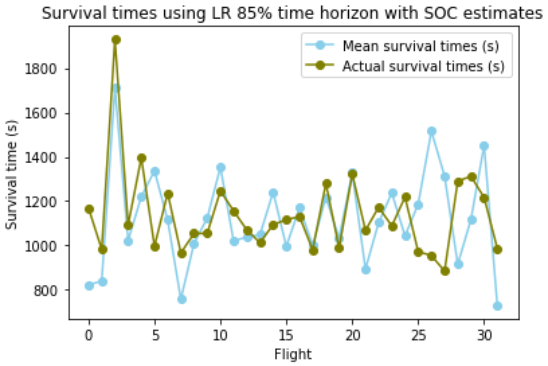
Figures on the left of the page included state of charge (SOC) estimates as features in the model (figures on the right did not).

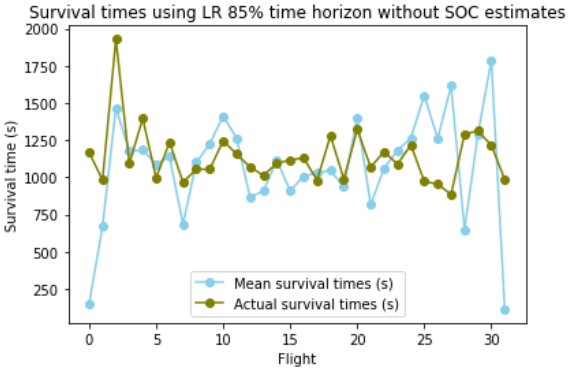


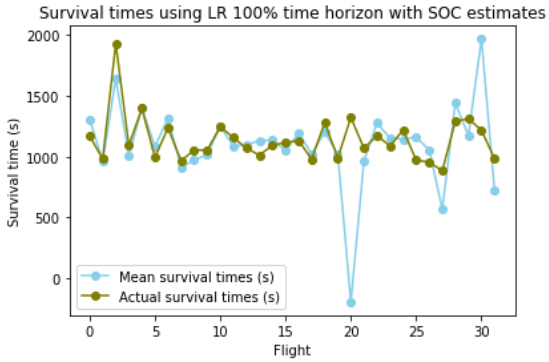


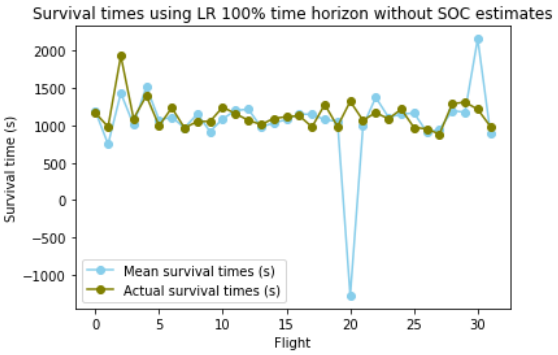






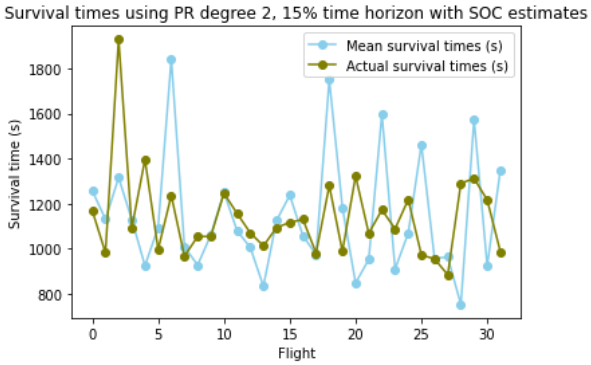


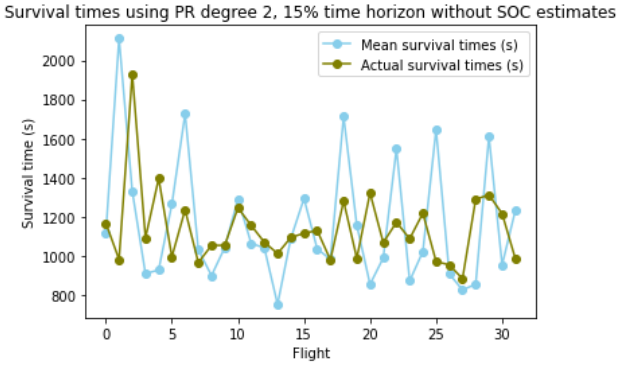


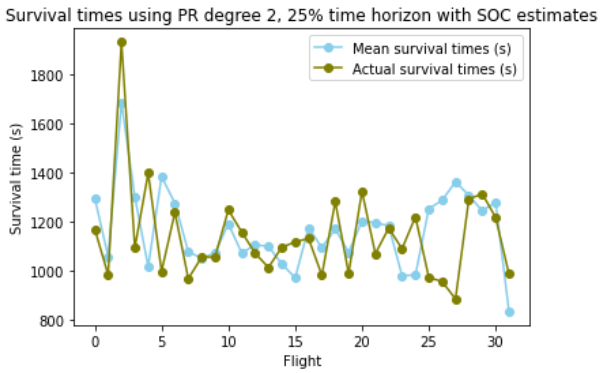


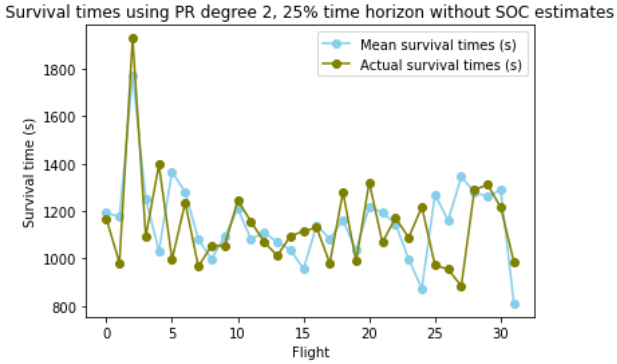
APPENDIX IV. Survival Time Estimates using Polynomial Regression Model, Degree 2

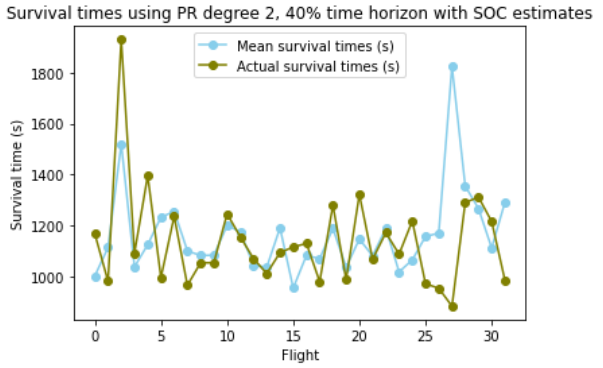
Time horizons: 0 - 50%

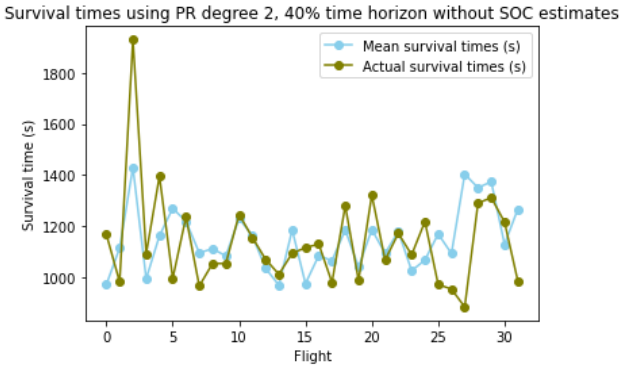


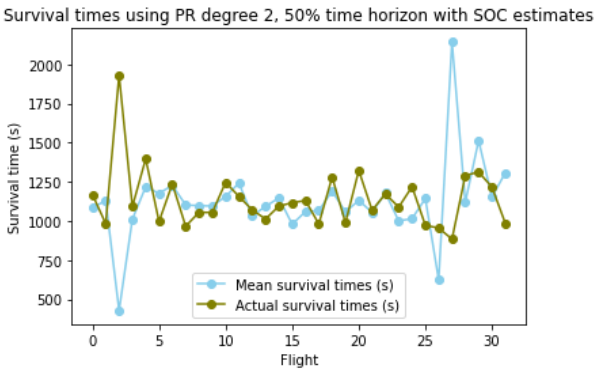


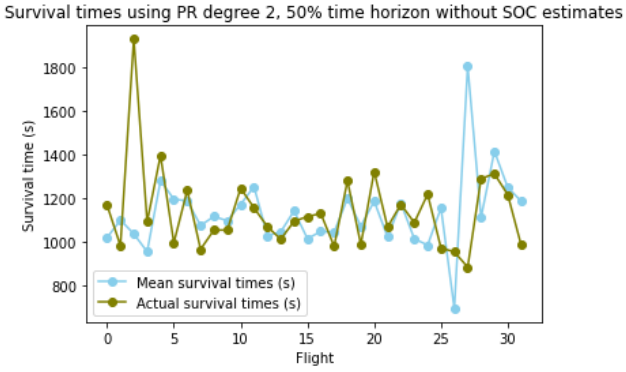






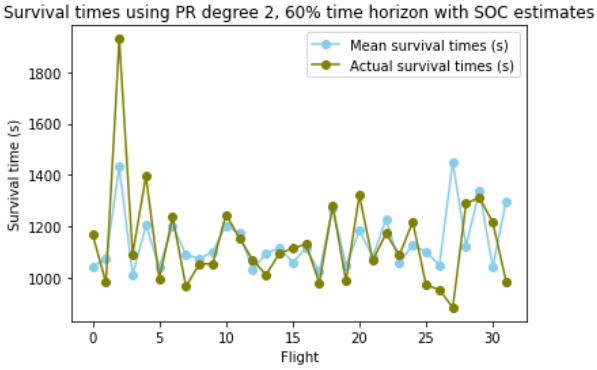


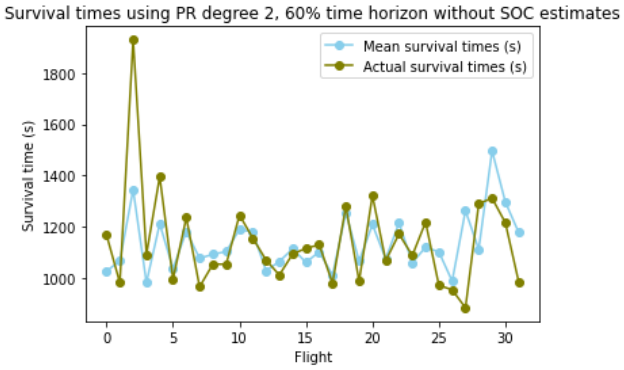


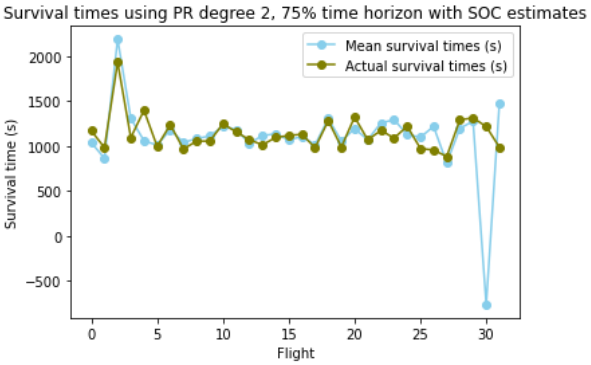


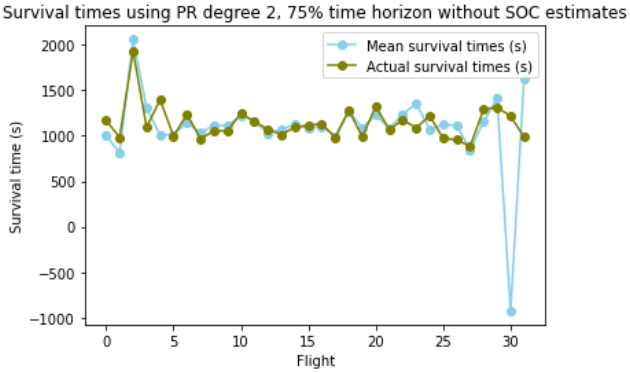
APPENDIX IV. cont'd. Survival Time Estimates using PR Model, Degree 2

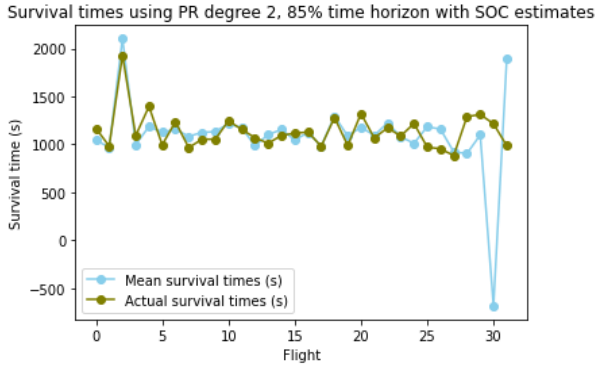
Time Horizons: 60-100 %

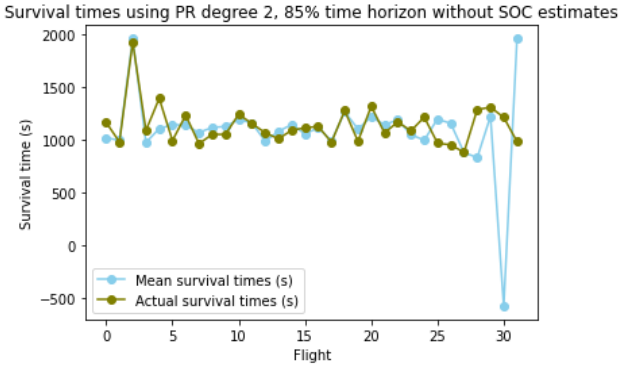


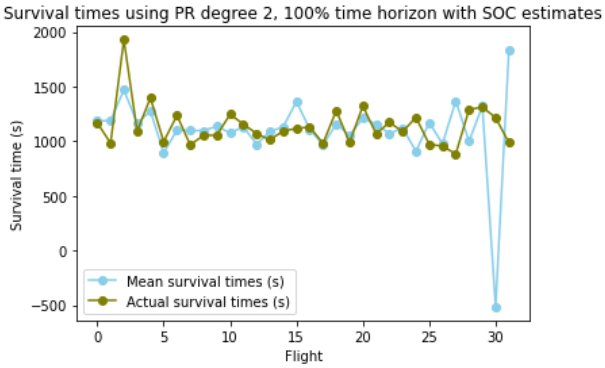


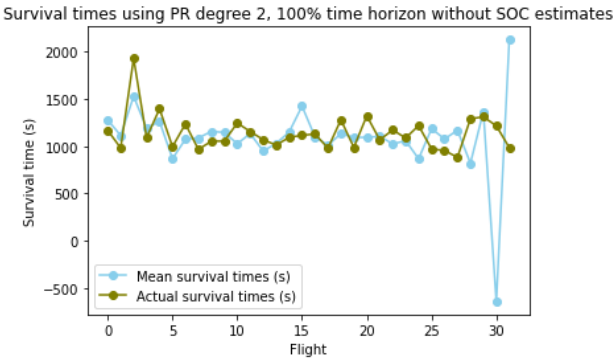






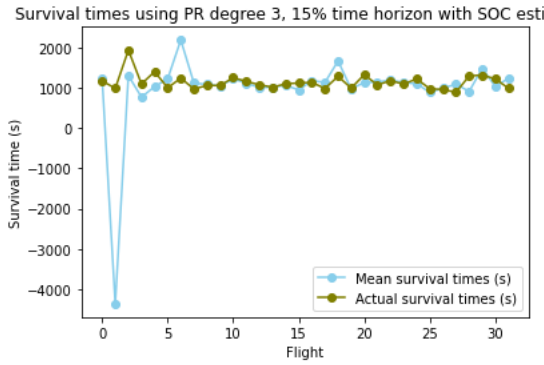


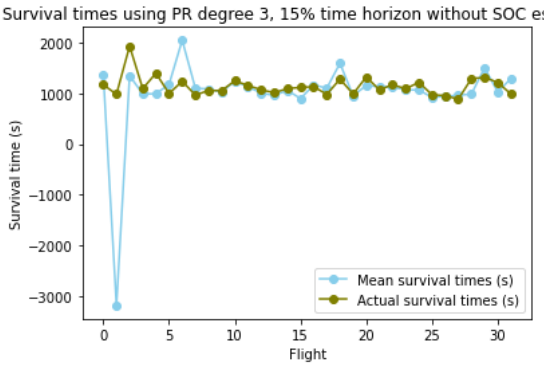


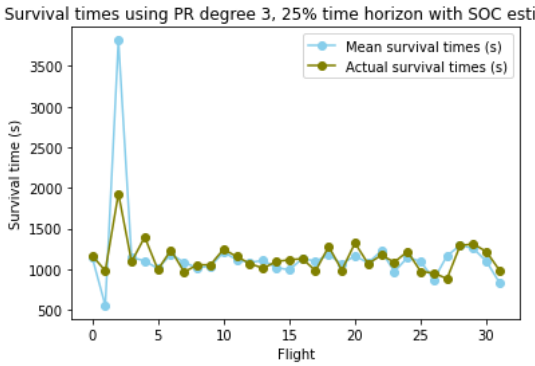


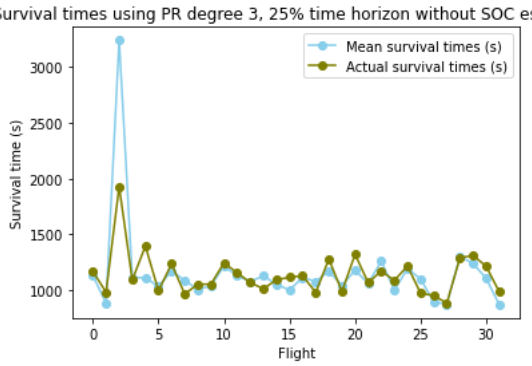
APPENDIX V. Survival Time Estimates using Polynomial Regression Model, Degree 3

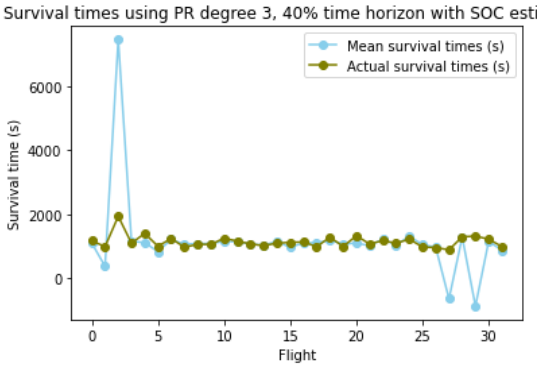
Time Horizons: 0-50 %

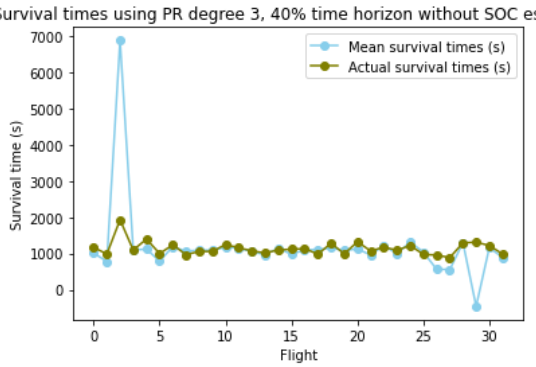


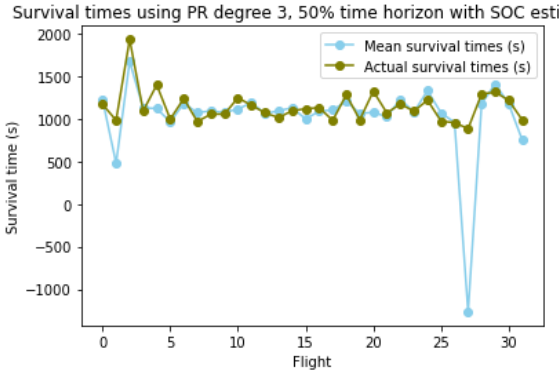


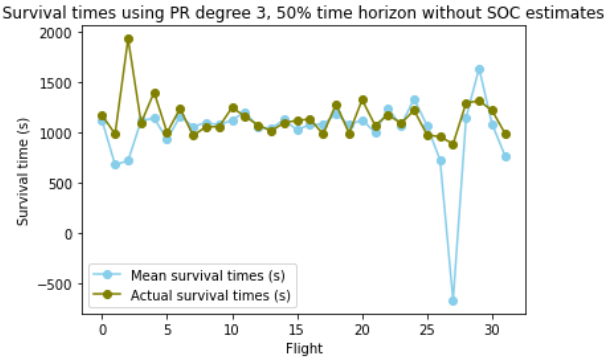






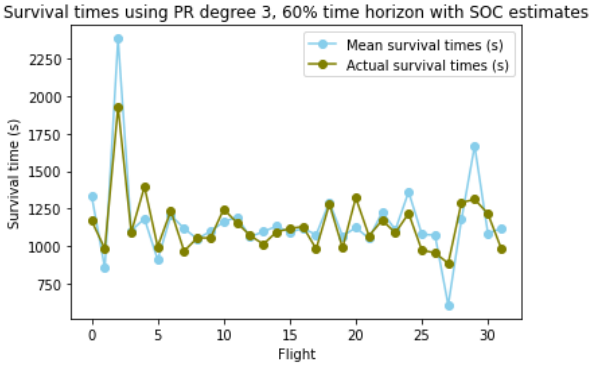


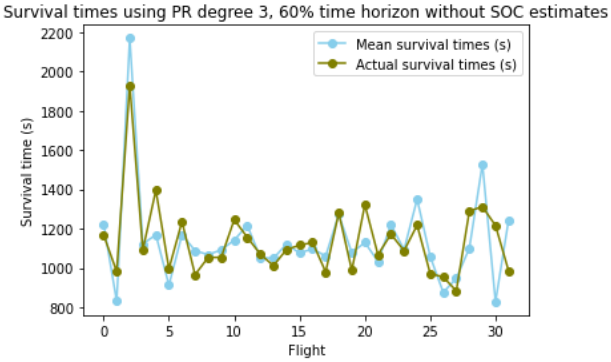


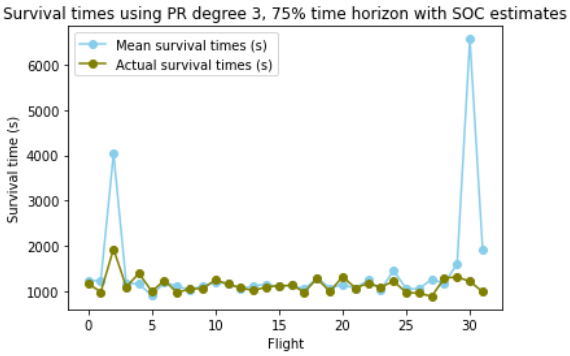


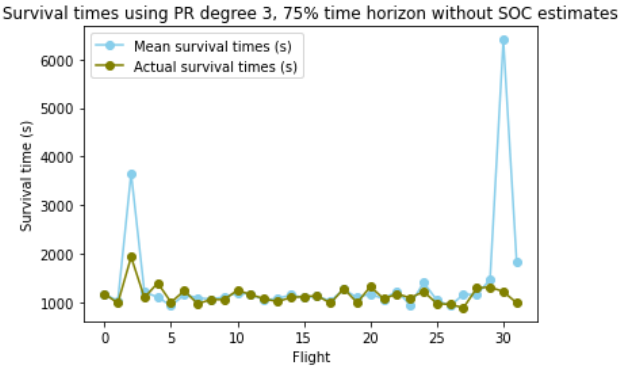
APPENDIX V cont'd. Survival Time Estimates using PR Model, Degree 3

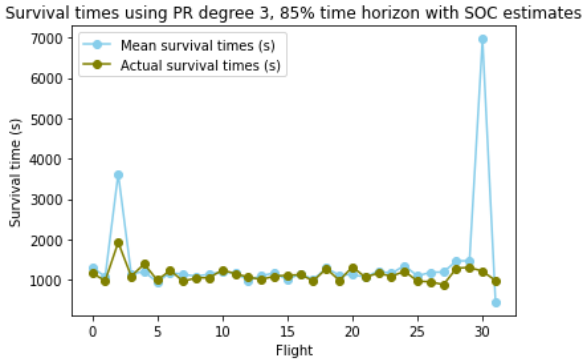
Time Horizons: 60-100 %

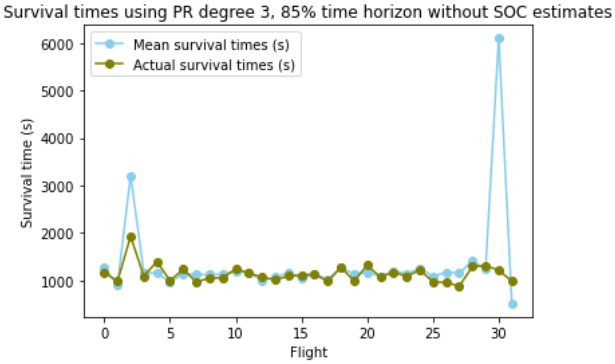


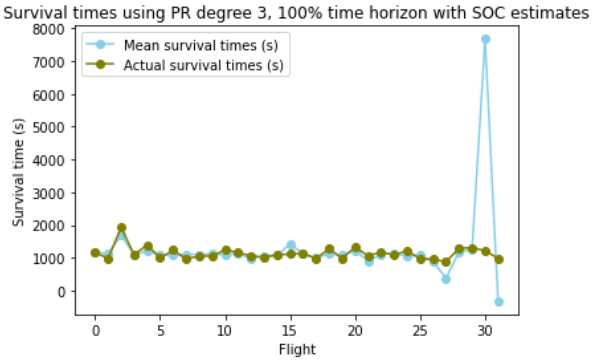


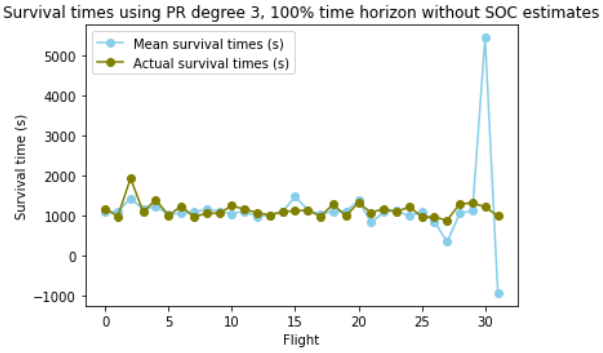






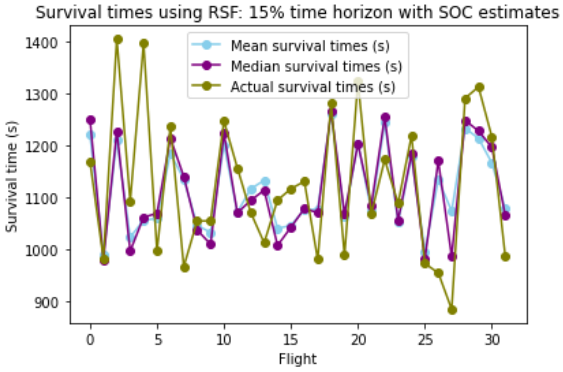


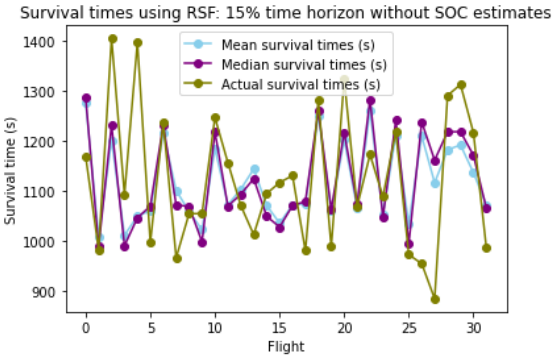


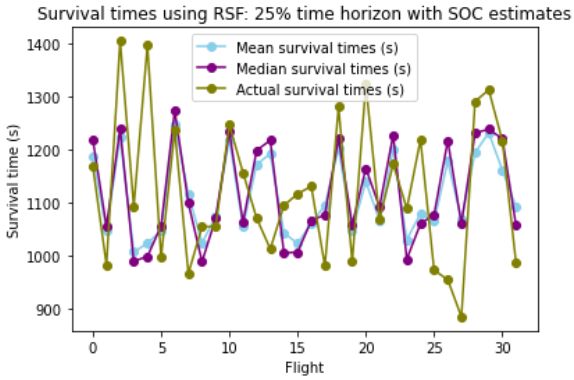


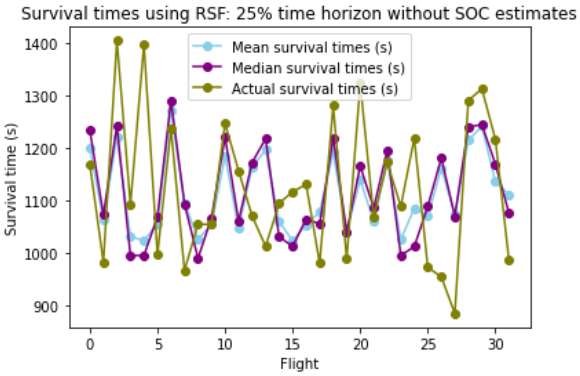
APPENDIX VI. Survival Time Estimates using a Random Survival Forest Model

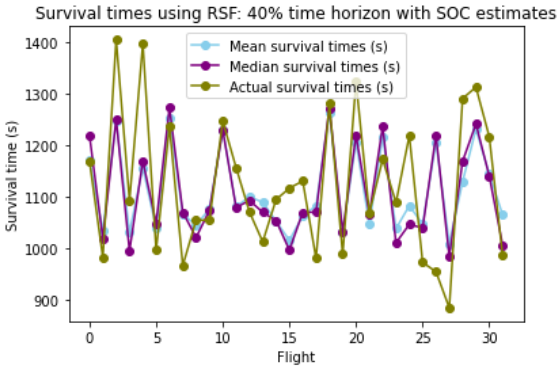
Time Horizon: 0-50 %

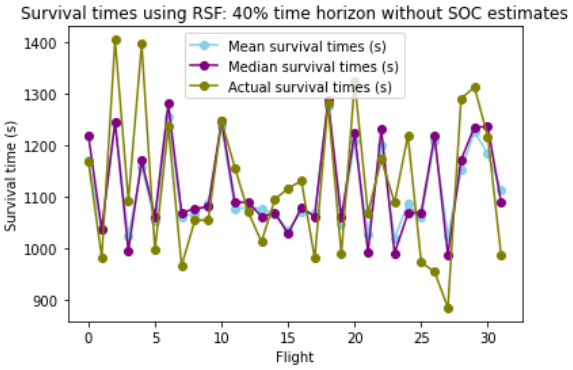


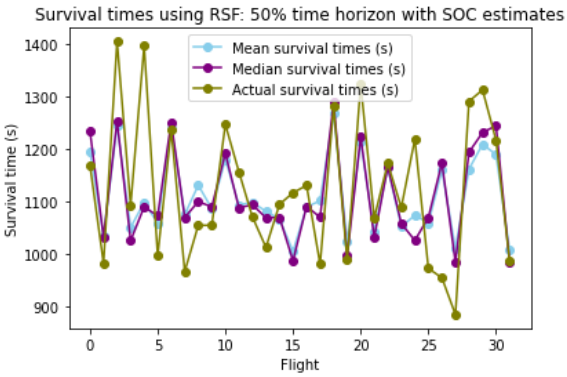


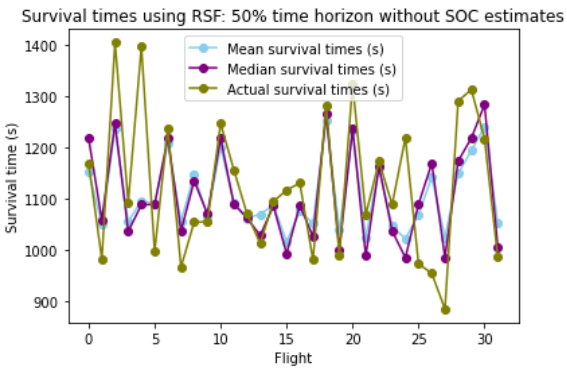






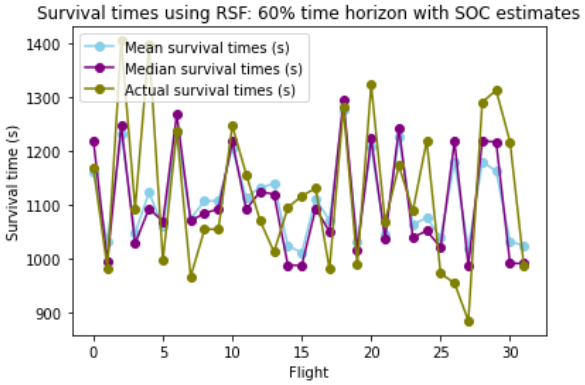


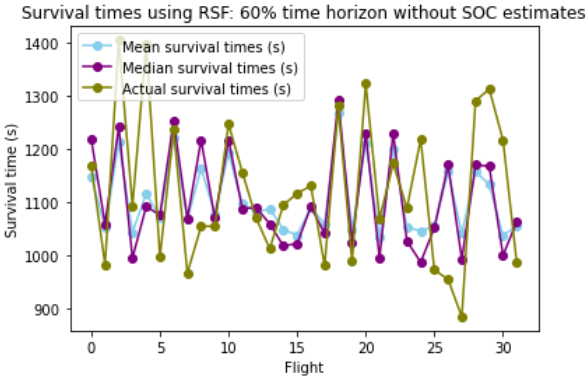


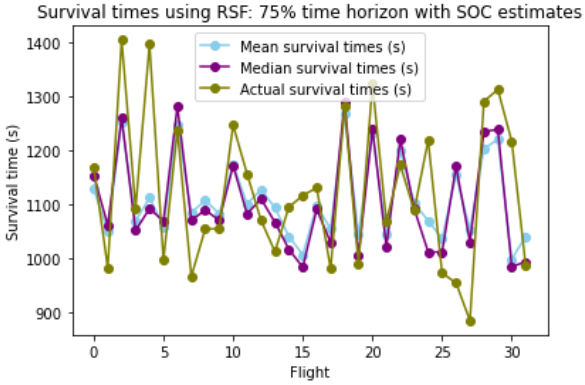


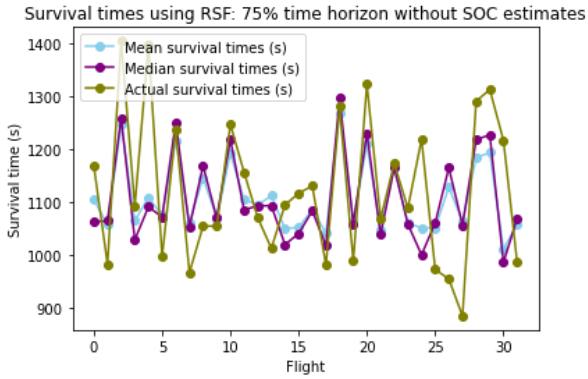
APPENDIX VI cont'd. Survival Time Estimates using a Random Survival Forest Model

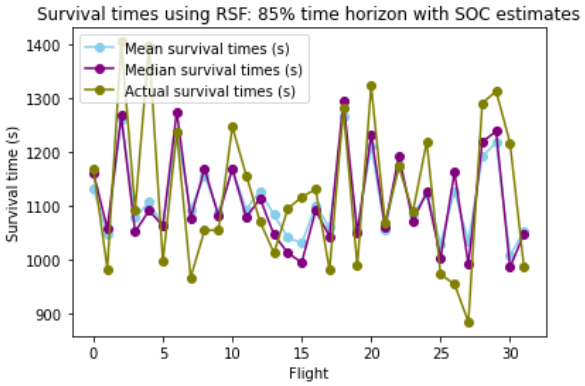
Time Horizon: 60-100 %

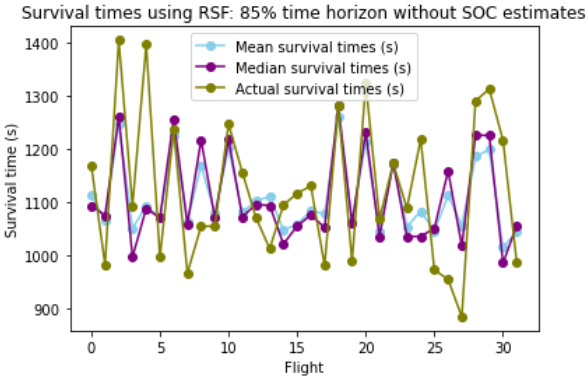


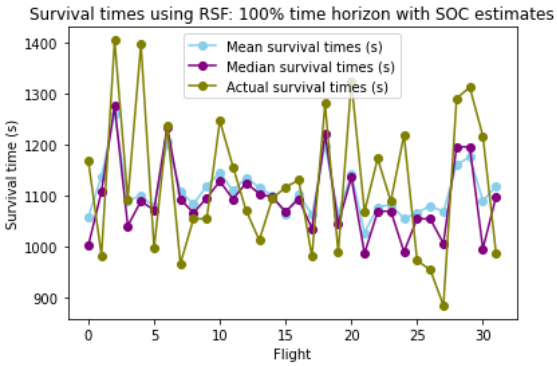


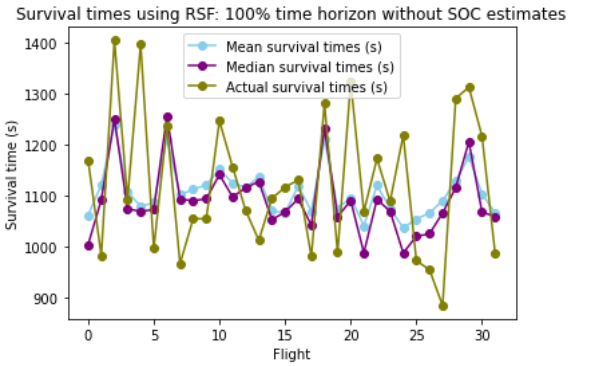






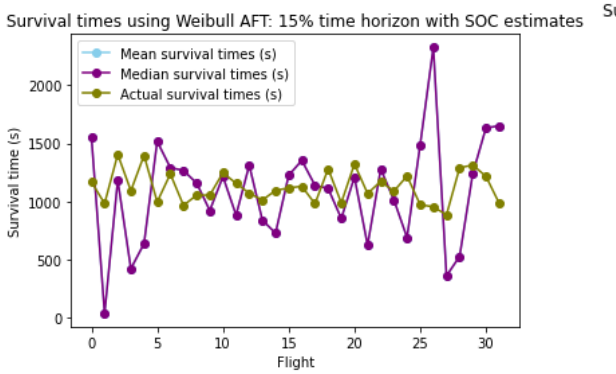


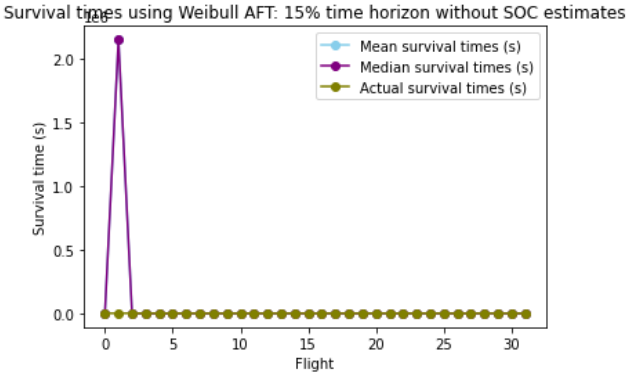


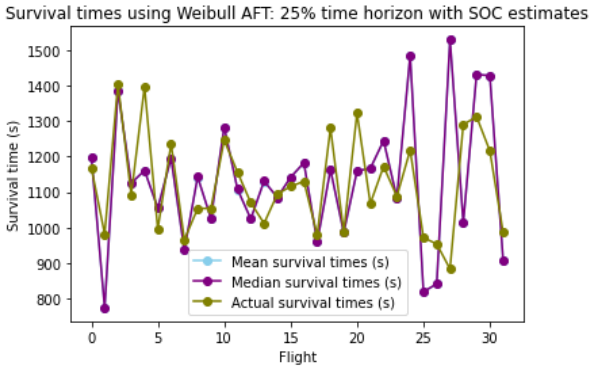


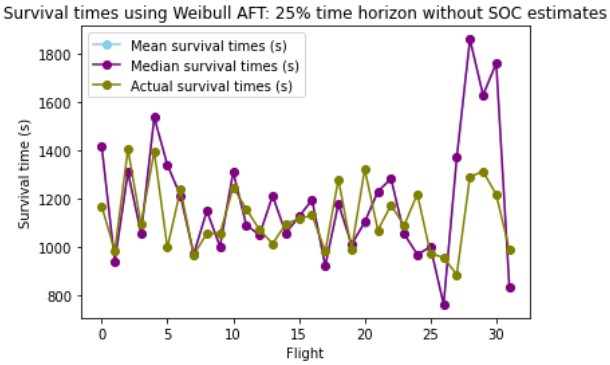
APPENDIX VII. Survival Time Estimates using a Weibull AFT Model

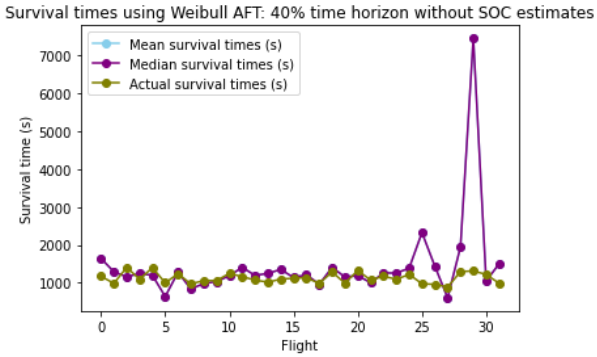
Time Horizon: 0-50 %

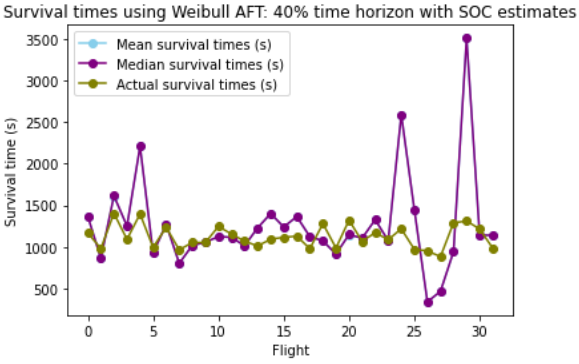


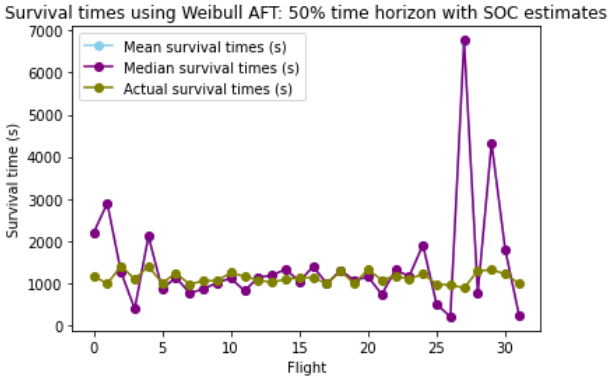


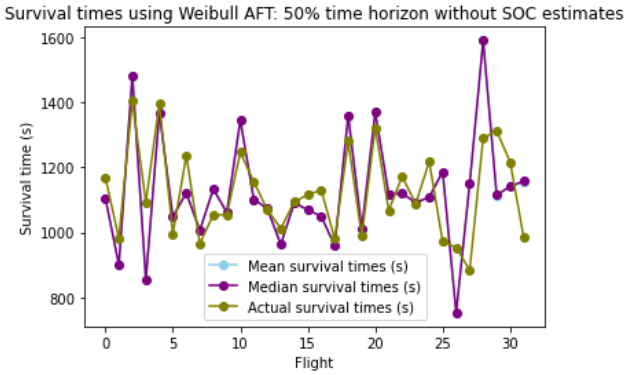






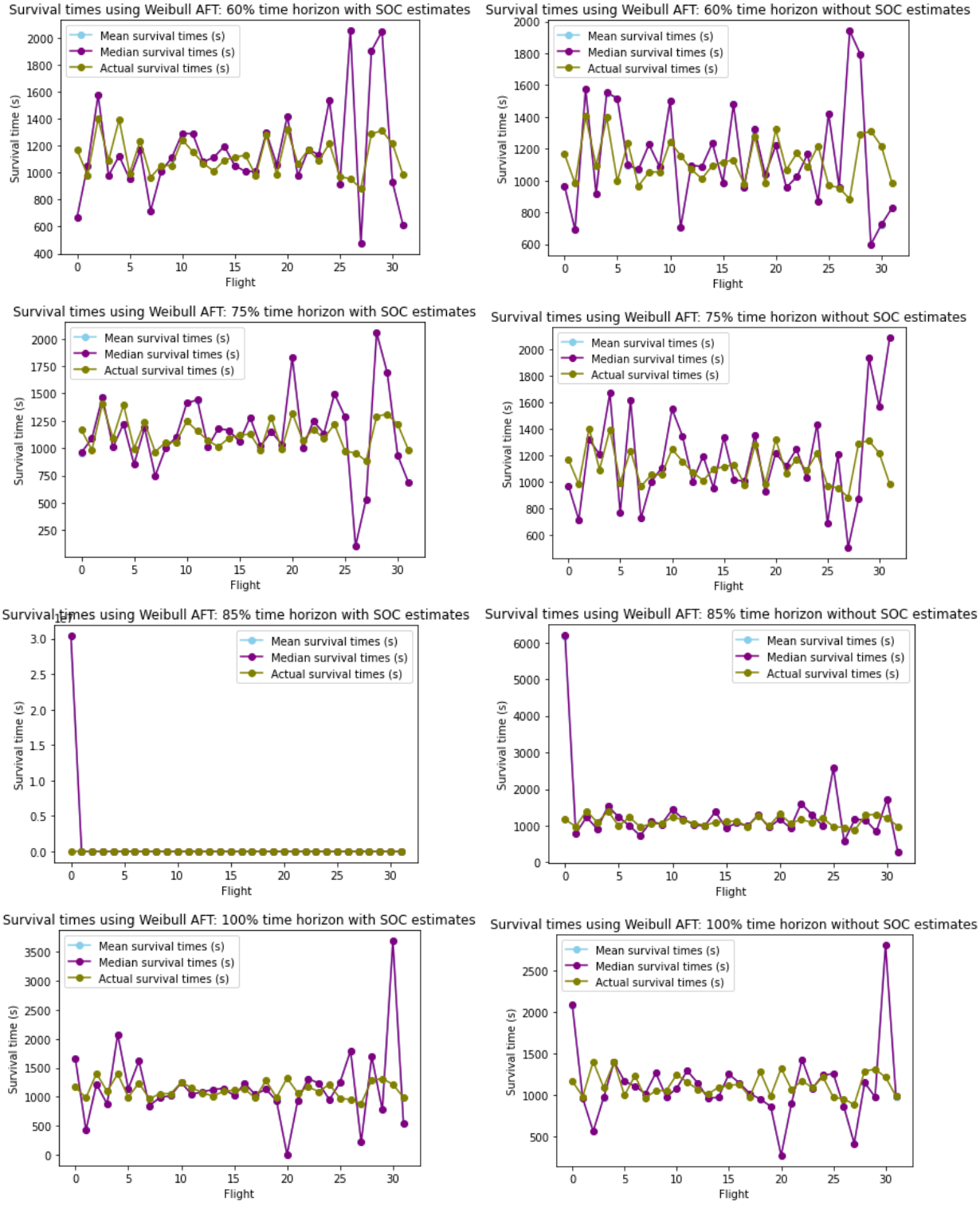




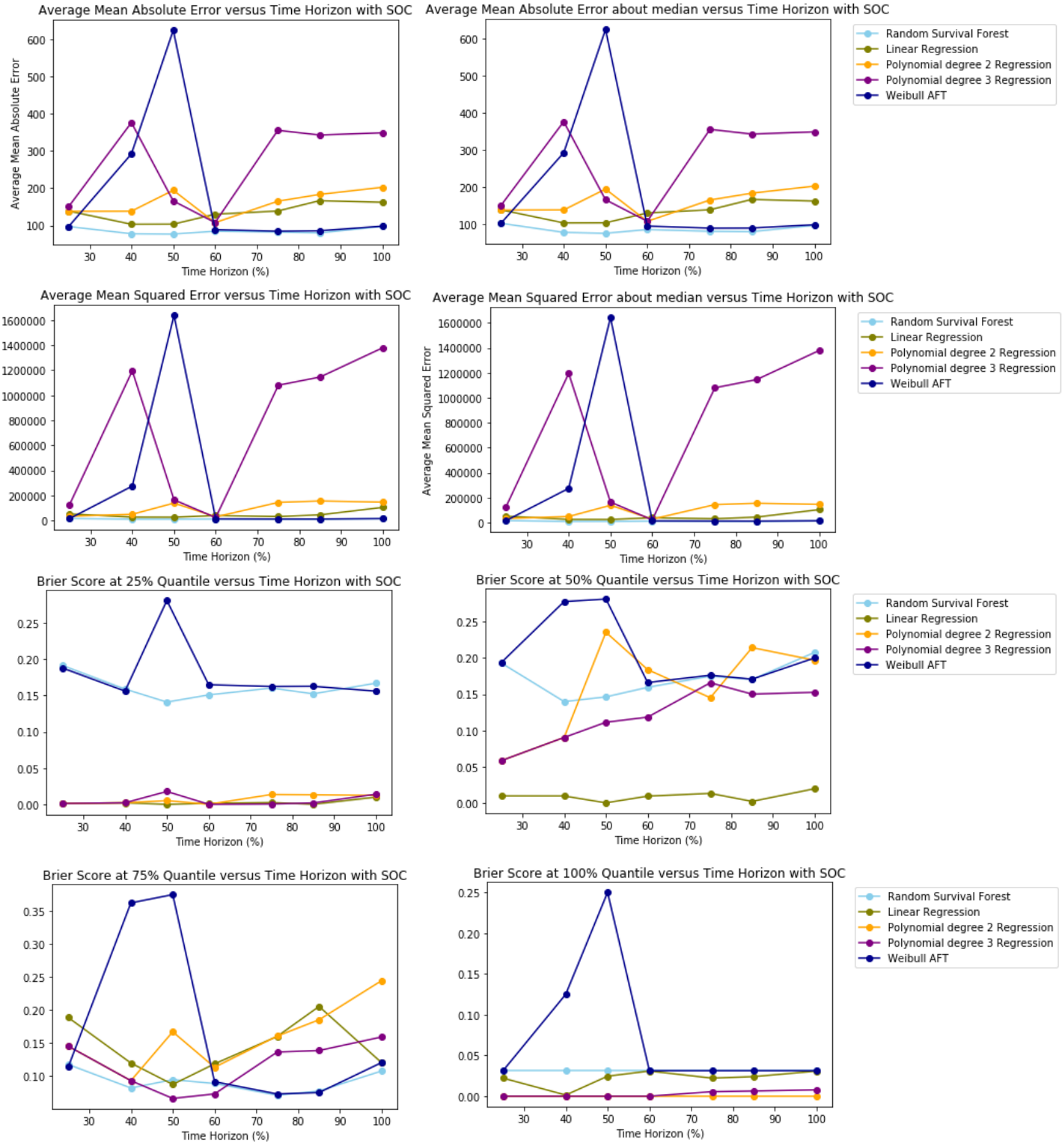


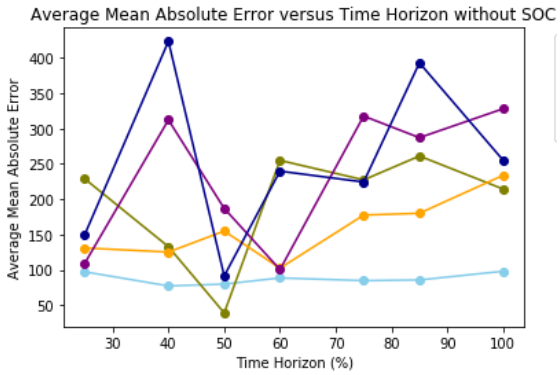
APPENDIX VII continued. Survival Time Estimates using a Weibull AFT Model

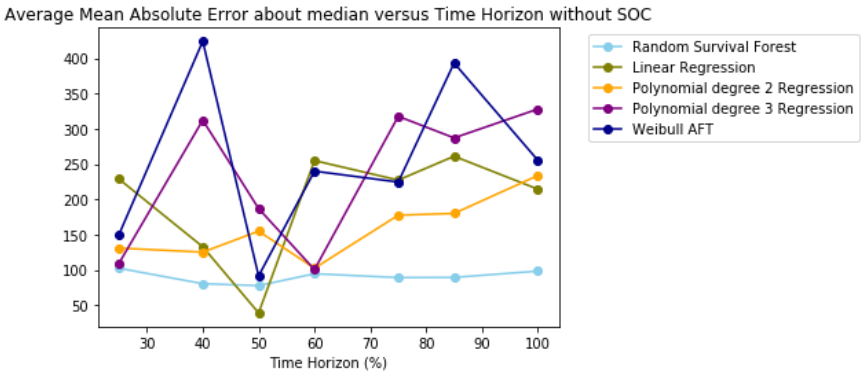
Time Horizon: 60-100 %

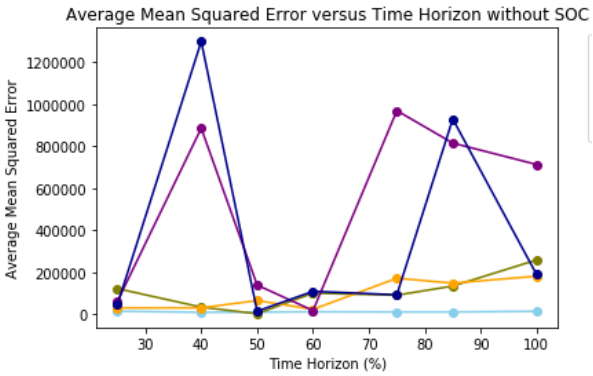


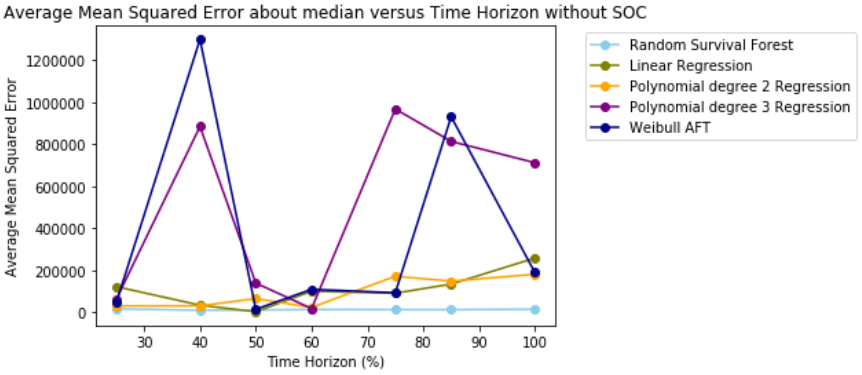
APPENDIX VIII. Model Comparison using Evaluation Metrics per Time Horizon

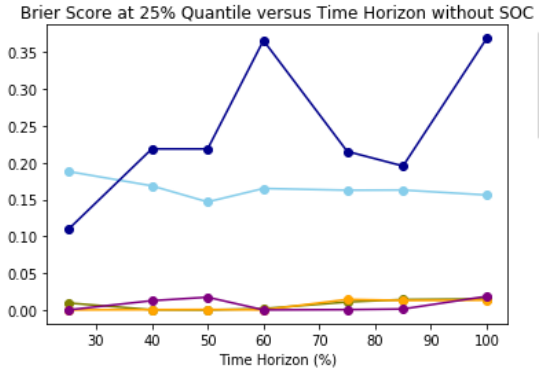


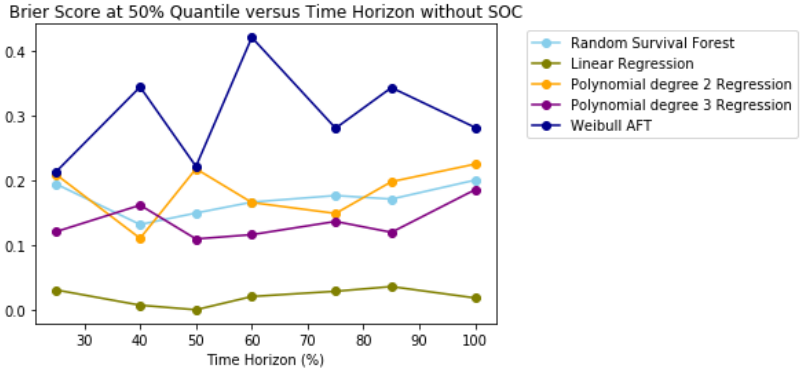

Average Mean Absolute Error versus Time Horizon without SOC
Average Mean Absolute Error
Time Horizon (%)

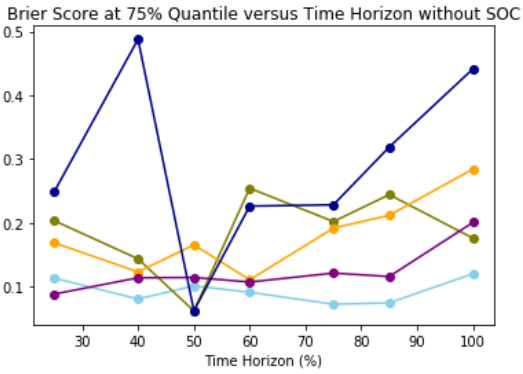

Average Mean Absolute Error about median versus Time Horizon without SOC
Time Horizon (%)
Random Survival Forest
Linear Regression
Polynomial degree 2 Regression
Polynomial degree 3 Regression
Weibull AFT

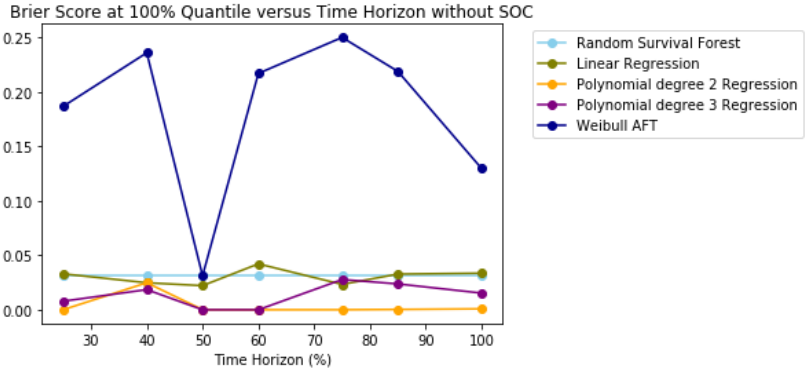

Average Mean Squared Error versus Time Horizon without SOC
Average Mean Squared Error
Time Horizon (%)


Average Mean Squared Error about median versus Time Horizon without SOC
Average Mean Squared Error
Time Horizon (%)
Random Survival Forest
Linear Regression
Polynomial degree 2 Regression
Polynomial degree 3 Regression
Weibull AFT

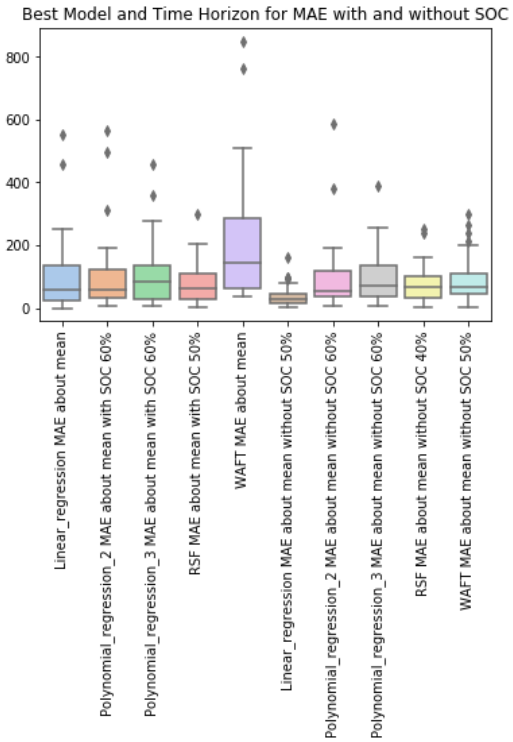

Brier Score at 25% Quantile versus Time Horizon without SOC
Time Horizon (%)

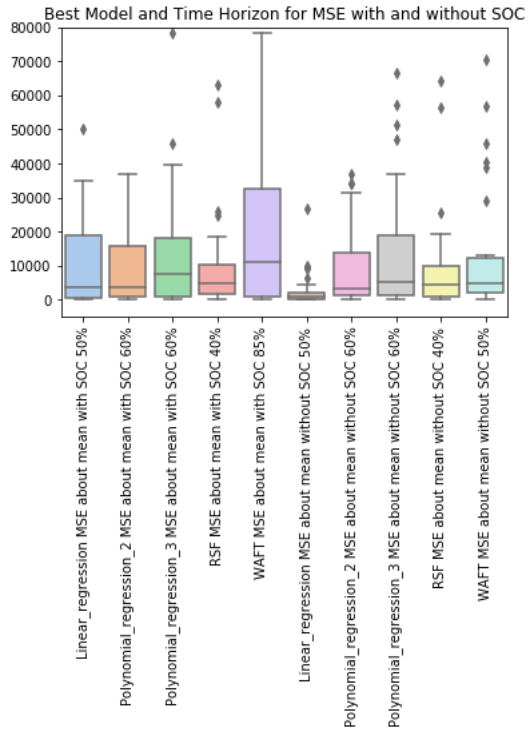

Brier Score at 50% Quantile versus Time Horizon without SOC
Time Horizon (%)
Random Survival Forest
Linear Regression
Polynomial degree 2 Regression
Polynomial degree 3 Regression
Weibull AFT

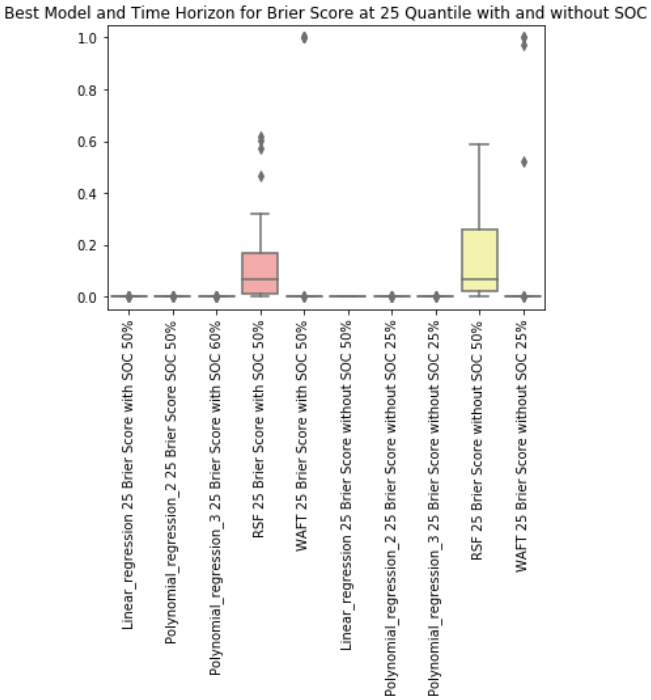

Brier Score at 75% Quantile versus Time Horizon without SOC
Time Horizon (%)

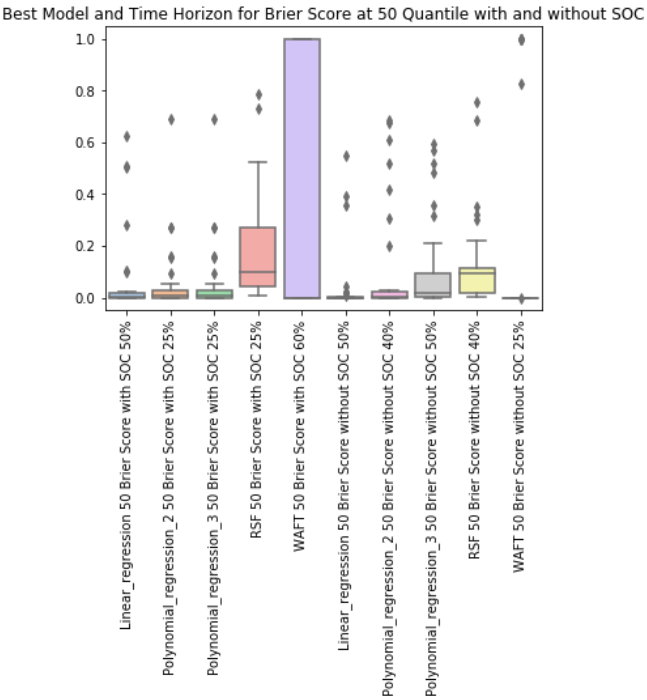

Brier Score at 100% Quantile versus Time Horizon without SOC
Time Horizon (%)
Random Survival Forest
Linear Regression
Polynomial degree 2 Regression
Polynomial degree 3 Regression
Weibull AFT

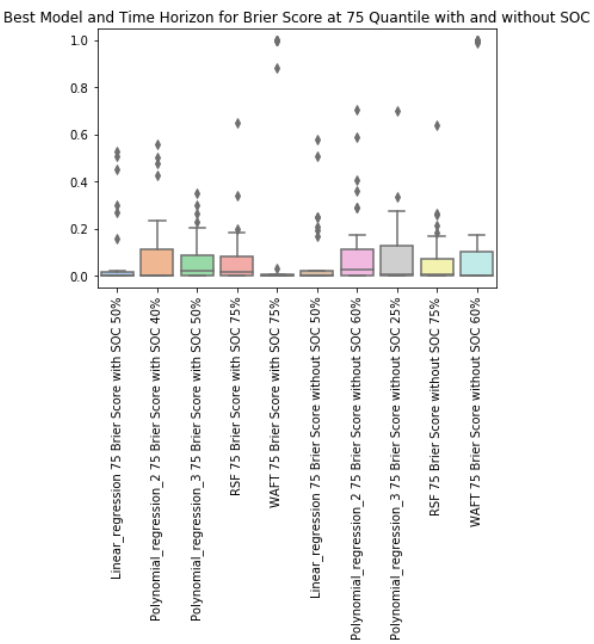
APPENDIX IX. Best Time Horizon by Model using Evaluation Metrics

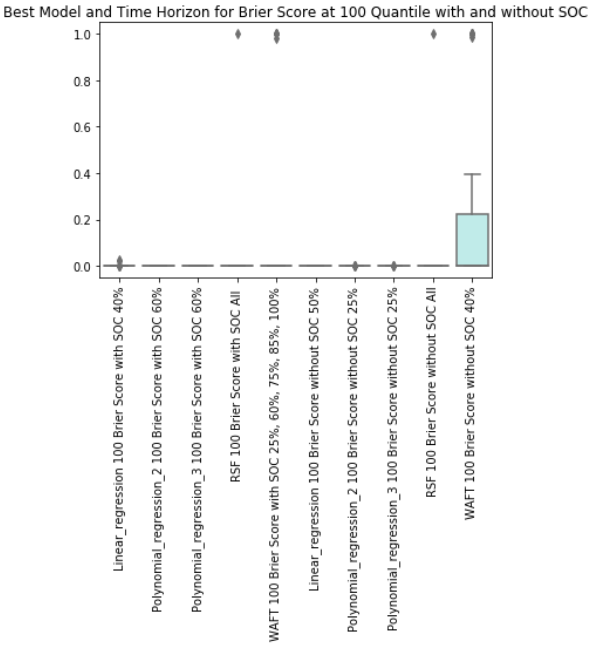









Best Model and Time Horizon for Brier Score at 75 Quantile with and without SOC
1.0
0.8
0.6
0.4
0.2
0.0
Linear_regression 75 Brier Score with SOC 50%
Polynomial_regression_2 75 Brier Score with SOC 40%
Polynomial_regression_3 75 Brier Score with SOC 50%
RSF 75 Brier Score with SOC 75%
WAFT 75 Brier Score with SOC 75%
Linear_regression 75 Brier Score without SOC 50%
Polynomial_regression_2 75 Brier Score without SOC 60%
Polynomial_regression_3 75 Brier Score without SOC 25%
RSF 75 Brier Score without SOC 75%
WAFT 75 Brier Score without SOC 60%


Best Model and Time Horizon for Brier Score at 100 Quantile with and without SOC
1.0
0.8
0.6
0.4
0.2
0.0
Linear_regression 100 Brier Score with SOC 40%
Polynomial_regression_2 100 Brier Score with SOC 60%
Polynomial_regression_3 100 Brier Score with SOC 60%
RSF 100 Brier Score with SOC All
WAFT 100 Brier Score with SOC 25%, 60%, 75%, 85%, 100%
Linear_regression 100 Brier Score without SOC 50%
Polynomial_regression_2 100 Brier Score without SOC 25%
Polynomial_regression_3 100 Brier Score without SOC 25%
RSF 100 Brier Score without SOC All
WAFT 100 Brier Score without SOC 40%

APPENDIX X. Python Scripts

EDA: Reading estimated state of charge values for each flight (from battery health monitoring data)

Authors: Laura Simandl, Molly Strasser

Course: 16791 Applied Data Science

In [ ]:
```
import pandas as pd
from matplotlib import pyplot as plt
from scipy import signal
import numpy as np
```

In [ ]:
```
original = [14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36
,37,38,39,40,41,63,64,67,68,69,71,72,73,74,75,76,77,78,79,80,81]

# parasitic_load_files = [14,15,19,20,21,22,23,25,26,28,29,31,32,34,35,37,39,41]
# no_load_files = [16,17,18,24,27,30,33,36,38,40,71,72,73,74,75,76,77,78,79,80,8
1]
# files_selected = [14,15,17,18,20,21,22,23,24,25,26,27,28,29,30,32,33,34,36,37,
38,39,40,41,72,73,76,77,78,79,80,81]

plot_num =1
plt.figure(figsize=(30,40), facecolor='white')

for num in original:

    # read file
    bhm_file = "HIRF data sets/BHM estimate files/HIRF"+str(num)+"_BHM.csv"
    df = pd.read_csv(bhm_file)
    labels = df.columns

    # add labels to all columns in file
    new_labels = ['t', 'LLF_EOD']

    last_label = labels[1]
    index = 0
    for col in labels[2:len(labels)]:
        curr = col

        if 'Unnamed' in curr:
            new = last_label
            index += 1
        else:
            new = curr
            last_label = new
            index = 0
        new = new+str(index)
        new_labels.append(new)

    df.columns = new_labels
    if 'URA_V_est1' in df.columns:
        df = df.drop('URA_V_est1', axis=1)

    ############ PLOT SOC ############

    time = df['t']
    LLF_SOC_df = time
    ULA_SOC_df = time
    LRF_SOC_df = time
    URA_SOC_df = time

    for col in df.columns:
        if 'LLF_SOC' in col:
            LLF_SOC_df = pd.concat([LLF_SOC_df, df[col]], axis = 1)
        elif 'ULA_SOC' in col:
            ULA_SOC_df = pd.concat([ULA_SOC_df, df[col]], axis = 1)
        elif 'LRF_SOC' in col:
            LRF_SOC_df = pd.concat([LRF_SOC_df, df[col]], axis = 1)
        elif 'URA_SOC' in col:
            URA_SOC_df = pd.concat([URA_SOC_df, df[col]], axis = 1)
```

```
    plt.figure(plot_num)

    # Plot LLF SOC
    ax1 = plt.subplot(9,5,plot_num)
    ax1.plot(time, LLF_SOC_df[col])
    title = 'Flight '+str(num)+': LLF SOC vs. time'
    ax1.set(title = title, xlabel = 'Time (sec)', ylabel = 'State of Charge (%)'
)

    # Plot ULA SOC
    ax2 = plt.subplot(9,5,plot_num)
    for col in ULA_SOC_df.columns[1]:
        ax2.plot(time, ULA_SOC_df[col])
    title = 'Flight '+str(num)+': ULA SOC vs. time'
    # ax2.set(title = title, xlabel = 'Time (sec)', ylabel = 'State of Charge
 (%)')

    # Plot LRF SOC
    ax3 = plt.subplot(9,5,plot_num)
    for col in LRF_SOC_df.columns[1]:
        ax3.plot(time, LRF_SOC_df[col])
    title = 'Flight '+str(num)+': LRF SOC vs. time'
    # ax3.set(title = title, xlabel = 'Time (sec)', ylabel = 'State of Charge
 (%)')

    # Plot URA SOC
    ax4 = plt.subplot(9,5,plot_num)
    for col in URA_SOC_df.columns[1]:
        ax4.plot(time, URA_SOC_df[col], label = )
    title = 'Flight '+str(num)+': URA SOC vs. time'
    # ax3.set(title = title, xlabel = 'Time (sec)', ylabel = 'State of Charge
 (%)')

    plt.tight_layout()

    plot_num += 1

plt.savefig('Results/EDA/estimated_plot_original_files')
```

Reading measured instrument values for each flight

Authors: Laura Simandl, Molly Strasser

Course: 16791 Applied Data Science

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
```

In [ ]:
```
def smoothSav(df):
    return df
```

In [ ]:
```
# Produce plots for all flights except those in Set 1 (5-13).

original = [14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36
,37,38,39,40,41,63,64,67,68,69,71,72,73,74,75,76,77,78,79,80,81]
files_selected = original

plot_num = 1
for num in files_selected:

    hirf_file = "HIRF data sets/Measured data files/HIRF"+str(num)+".csv"
    df = pd.read_csv(hirf_file)
    time = df['t']

    # Motor state
    rpm = smoothSav(df['RPM'])

    # Voltages of batteries
    v_low_left = smoothSav(df['LLF20V'])
    v_up_left = smoothSav(df['ULA20V'])
    v_low_right = smoothSav(df['LRF20V'])
    v_up_right = smoothSav(df['URA20V'])

    # Battery set voltages
    v_up_right_set = smoothSav(df['URA40V'])
    v_low_right_set = smoothSav(df['LRF40V'])

    # Current of batteries (individual)
    c_low_left = smoothSav(df['LLF20C'])
    c_up_left = smoothSav(df['ULA20C'])

    # Battery set currents
    c_low_right_set = smoothSav(df['LRF40C'])
    c_up_right_set = smoothSav(df['URA40C'])

    # Temperatures of Battery Sets
    temp_upper_right = smoothSav(df['URA40T'])
    temp_lower_right = smoothSav(df['LRF40T'])

    ######## PLOTTING #########

    # Plot of voltages (individual batteries)
    plt.figure(plot_num)
    plt.plot(time, v_up_right)
    plt.plot(time, v_up_left)
    plt.plot(time, v_low_right)
    plt.plot(time, v_low_left)

    plt.legend(['URA20V', 'ULA20V', 'LRF20V', 'LLF20V'], loc = 'right')
    title = 'Flight '+str(num)+': Voltages for individual batteries'
    plt.title(title)
    plt.xlabel('Time (sec)')
    plt.ylabel('Voltage (V)')
    plt.tight_layout()
    plt.savefig('Results/EDA/measured/flight'+str(num)+'_1voltage', bbox_inches=
'tight')

    # Plot of current (battery set)
    plot_num+=1
```

```
    plt.figure(plot_num)
    plt.plot(time, c_up_right_set)
    plt.plot(time, c_low_right_set)

    plt.title('Flight '+str(num)+': Currents for battery sets')
    plt.legend(['URA40C', 'LRF40C'], loc = 'right')
    plt.xlabel('Time (sec)')
    plt.ylabel('Current (A)')
    plt.savefig('Results/EDA/measured/flight'+str(num)+'_2current', bbox_inches=
'tight')

    # Plot of temperature of battery sets
    plot_num+=1
    plt.figure(plot_num)
    plt.plot(time, temp_upper_right)
    plt.plot(time, temp_lower_right)

    plt.legend(['URA40T', 'LRF40T'], loc = 'right')
    plt.title('Flight '+str(num)+': Battery set temperatures')
    plt.xlabel('Time (sec)')
    plt.ylabel('Temperature (deg F)')
    plt.savefig('Results/EDA/measured/flight'+str(num)+'_3temp', bbox_inches='ti
ght')

#     # Plot RPM
#     plot_num+=1
#     plt.figure(plot_num)
#     plt.plot(time, rpm)
#     plt.title('Flight '+str(num)+': Motor RPM')
#     plt.xlabel('Time (sec)')
#     plt.ylabel('Revolutions per minute (RPM)')
#     plt.savefig('Results/EDA/measured/flight'+str(num)+'_rpm', bbox_inches='ti
ght')

    plot_num+=1
```

In [ ]:
```
# # Produce plots of all the files selected for use in modeling.

# files_selected = [14,15,17,18,20,21,22,23,24,25,26,27,28,29,30,32,33,34,36,37,
38,39,40,41,72,73,76,77,78,79,80,81]

# plot_num = 1

# plt.figure(figsize=(60,260), facecolor='white')
# plt.rc('font', size=20)
# plt.rc('axes', titlesize=20)
# plt.rc('axes', labelsize=20)
# plt.rc('xtick', labelsize=20)
# plt.rc('ytick', labelsize=20)
# plt.rc('legend', fontsize=20)
# plt.rc('figure', titlesize=20)

# for num in files_selected:

#     hirf_file = "HIRF data sets/Measured data files/HIRF"+str(num)+".csv"
#     df = pd.read_csv(hirf_file)
#     time = df['t']

#     # Motor state
#     rpm = smoothSav(df['RPM'])

#     # Voltages of batteries
#     v_low_left = smoothSav(df['LLF20V'])
#     v_up_left = smoothSav(df['ULA20V'])
#     v_low_right = smoothSav(df['LRF20V'])
#     v_up_right = smoothSav(df['URA20V'])

#     # Battery set voltages
#     v_up_right_set = smoothSav(df['URA40V'])
#     v_low_right_set = smoothSav(df['LRF40V'])

#     # Current of batteries (individual)
#     c_low_left = smoothSav(df['LLF20C'])
#     c_up_left = smoothSav(df['ULA20C'])

#     # Battery set currents
#     c_low_right_set = smoothSav(df['LRF40C'])
#     c_up_right_set = smoothSav(df['URA40C'])

#     # Temperatures of Battery Sets
#     temp_upper_right = smoothSav(df['URA40T'])
#     temp_lower_right = smoothSav(df['LRF40T'])


#     ######## PLOTTING #########

#     # Plot of voltages (individual batteries)
#     ax1 = plt.subplot(len(files_selected),4,plot_num)
#     ax1.plot(time, v_up_right)
#     ax1.plot(time, v_up_left)
#     ax1.plot(time, v_low_right)
#     ax1.plot(time, v_low_left)
#     ax1.legend(['URA20V', 'ULA20V', 'LRF20V', 'LLF20V'], loc = 'right')
#     title = 'Flight '+str(num)+': Voltages for individual batteries'
```

```
#     ax1.set(title = title, xlabel = 'Time (sec)', ylabel = 'Voltage (V)')

# #     # Plot of voltages (battery set)
# #     plot_num+=1
# #     ax2 = plt.subplot(len(files_selected),4,plot_num)
# #     ax2.plot(time, v_up_right_set)
# #     ax2.plot(time, v_low_right_set)
# #     ax2.legend(['URA40V', 'LRF40V'], loc = 'right')
# #     ax2.set(title = 'Flight '+str(num)+': Battery set voltages vs. time',
# #            xlabel = 'Time (sec)',
# #            ylabel = 'Voltage (V)')

#     # Plot of current (battery set)
#     plot_num+=1
#     ax3 = plt.subplot(1,4,plot_num)
#     ax3.plot(time, c_up_right_set)
#     ax3.plot(time, c_low_right_set)
#     ax3.legend(['URA40C', 'LRF40C'], loc = 'right')
#     ax3.set(title = 'Flight '+str(num)+': Battery set currents vs. time',
#            xlabel = 'Time (sec)',
#            ylabel = 'Current (A)')

#     # Plot of temperature of battery sets
#     plot_num+=1
#     ax4 = plt.subplot(1,4,plot_num)
#     ax4.plot(time, temp_upper_right)
#     ax4.plot(time, temp_lower_right)
#     ax4.legend(['URA40T', 'LRF40T'], loc = 'right')
#     ax4.set(title = 'Flight '+str(num)+': Battery set temperatures vs. time',
#            xlabel = 'Time (sec)',
#            ylabel = 'Temperature (deg F))')

#     # Plot RPM
#     plot_num+=1
#     ax5 = plt.subplot(1,4,plot_num)
#     ax5.plot(time, rpm)
#     ax5.set(title = 'Flight '+str(num)+': Motor RPM',
#            xlabel = 'Time (sec)',
#            ylabel = 'Revolutions per minute (RPM)')

#     plt.savefig('Results/EDA/measured_plot_selected_files')

# #     # Plot Motor Controller Sensor data
# #     plot_num+=1
# #     ax6 = plt.subplot(len(files_selected),5,plot_num)
# #     ax6.plot(time, amc)
# #     ax6.plot(time, fmc)
# #     ax6.legend(['AMC', 'FMC'], loc = 'right')
# #     ax6.set(title = 'Flight '+str(num)+': Motor Controller Sensors',
# #            xlabel = 'Time (sec)',
# #            ylabel = '')

#     plt.tight_layout()
#     plot_num+=1

# # plt.savefig('Results/EDA/measured_plot'+str(num)+'_selected_files')
```

In [ ]:

Random Survival Forest Model

Authors: Laura Simandl, Molly Strasser

Course: 16791 Applied Data Science

The following code implements a random survival forest model for the Battery Prognostics Project.

In [ ]:
```
# Import modules and packages

import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sksurv.ensemble import RandomSurvivalForest
from sklearn import metrics
from sklearn.model_selection import LeaveOneOut
from sklearn.preprocessing import StandardScaler

np.random.seed(1000)
```

In [ ]:
```
def create_status_ind(df_bhm):
# Create 'status' column
# where '0' indicates that e-UAV is 'alive', and '1' indicates it is 'dead'
# (i.e. one of the four batteries reached a state of charge < 30%)

    status_col = []

    for i in range(len(df_bhm)):
        df_soc_row = np.array(df_bhm.iloc[i,217:325])
        indicator = 0
        for value in df_soc_row:
            if value < 30:
                indicator = 1  # dead
        status_col.append(indicator)

    return status_col
```

In [ ]:
```
def calc_input_feat(X, num):
# Calculate static features
# Calculate mean, median, standard deviation of each column
# Indicate if file has parametric load (boolean 1: yes, 0: no)

    input_param = []
    labels = []

    for col_name in X.columns:

        col = X[col_name]
        mean = round(col.mean(),3)
        median = round(col.median(),3)
        std = round(col.std(),3)
        input_param.append(mean)
        input_param.append(median)
        input_param.append(std)
        labels.append(col_name+"_mean")
        labels.append(col_name+"_median")
        labels.append(col_name+"_std")

    if num in parasitic_load_files:
        input_param.append(1)
    else:
        input_param.append(0)
    labels.append('file type')

    return input_param, labels
```

In [ ]:
```
def plot_surv_fcn(rsf, iteration, surv):
# Plot survival functions

    plt.figure(0)

    for i, s in enumerate(surv):
        plt.step(rsf.event_times_, s, where="post", label='fold '+str(iteration
))
    plt.ylabel("Survival probability")
    plt.xlabel("Time in seconds")
    plt.title("Survival functions on test data")
    plt.grid(True)

    plt.savefig('Results/RSF/plots/rsf_surv_fcn_TH'+str(percent)+save, bbox_inch
es='tight')
```

In [ ]:
```
def compute_mean_survival_t(surv, x_time, start):
# Compute mean predicted survival time; calculate auc.

    lst = []
    for i, s in enumerate(surv):
        y_prob = s  # array of probabilities for events in one fold
        area = metrics.auc(x_time,y_prob)
        mean_surv_t = area+start

    print('Predicted mean survival time (s):', round(mean_surv_t,2))

    return mean_surv_t
```

In [ ]:
```
def find_median_survival_t(summary):
# Finds median predicted survival time from a dataframe of survival times and as
sociated probabilites

    summary = summary.sort_values(by=['Probabilities'], ascending=True)

    p = summary['Probabilities'].array
    t = summary['Survival time (s)'].array

    median_survival_t = np.interp(0.5, p, t)  # median survival time when p=0.5

    print('Predicted median survival time (s):', round(median_survival_t,2))

    return median_survival_t
```

In [ ]:
```
def compute_metrics_about_mean(surv, x_time, actual_surv_t, mse_lst, mae_lst, me
an_surv_t_lst):
# Compute metrics about mean survival time for given fold

    start = min(x_time)

    # Compute mean predicted survival time
    mean_surv_t = compute_mean_survival_t(surv, x_time, start)

    MSE = (actual_surv_t - mean_surv_t)**2
    print('MSE for fold about mean:', round(MSE,2))
    mse_lst.append(MSE)

    MAE = abs(actual_surv_t - mean_surv_t)
    print('MAE for fold about mean:', round(MAE,2), '\n')
    mae_lst.append(MAE)

    mean_surv_t_lst.append(mean_surv_t)

    return mse_lst, mae_lst, mean_surv_t_lst
```

In [ ]:
```
def compute_metrics_about_median(summary, actual_surv_t, mse_lst, mae_lst, surv_
t_lst):
# Compute metrics about median survival time for given fold

    # Compute median predicted survival time
    median = find_median_survival_t(summary)

    MSE = (actual_surv_t - median)**2
    print('MSE for fold about median:', round(MSE,2))
    mse_lst.append(MSE)

    MAE = abs(actual_surv_t - median)
    print('MAE for fold about median:', round(MAE,2), '\n')
    mae_lst.append(MAE)

    surv_t_lst.append(median)

    return mse_lst, mae_lst, surv_t_lst
```

In [ ]:
```
def comp_sets(fold_num, train_index, test_index, target, X):
# Select test and train sets

    print("Iteration:", fold_num, '\n')
    print("TRAIN indeces:", train_index, "\nTEST index:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = target[train_index], target[test_index]

    return X_train, X_test, y_train, y_test
```

In [ ]:
```
def find_ci(score_lst, score_type, ci):
# Calculate confidence intervals on metrics

    z_scores = {90: 1.645, 95: 1.96, 98: 2.326, 99: 2.576}
    z_score = z_scores.get(ci)

    std_dev = np.std(score_lst)
    mean = np.mean(score_lst)
    me = z_score*std_dev/math.sqrt(len(score_lst))
    low_bound = mean - me
    up_bound = mean + me
    print('Lower bound on ', score_type ,'for', str(ci),'% confidence interval:'
, round(low_bound,4))
    print('Upper bound on ', score_type ,'for', str(ci),'% confidence interval:'
, round(up_bound,4), '\n')

    return me
```

In [ ]:
```
def find_avg_metric(score_lst, score_type):
# Calculate average of metric (MSE or MAE) from all the folds

    avg_metric = sum(score_lst)/len(score_lst)
    print('Average', score_type, 'for all iterations:', round(avg_metric,2))
    return avg_metric
```

In [ ]:
```
def create_summary(x_time, p_surv):
# Output data frame of surival times and probabilities

    t = pd.DataFrame(x_time, columns=['Survival time (s)'])
    p = pd.DataFrame(p_surv[0], columns=['Probabilities'])
    df = pd.concat([t,p], axis = 1)

    return df
```

In [ ]:
```
def comparison_plot(means, medians, actual, flights, percent, soc_included):

    if soc_included == True:
        use_soc = 'with'
        save = '_w_SOC'
    else:
        use_soc = 'without'
        save = '_wo_SOC'

    x_labels = str(flights)

    # data frame include calculation for mean surival times for each flight
    data = {'Flight number': flights, 'Mean survival times (s)': means, 'Median
 survival times (s)': medians, 'Actual survival times (s)': actual}
    df = pd.DataFrame(data, columns = ['Flight number', 'Mean survival times
 (s)', 'Median survival times (s)', 'Actual survival times (s)'])
    display(df)

    # Plot actual, median and mean survival time predictions
    plt.figure(1)
    plt.plot(means, marker = 'o',color = 'skyblue', label='Mean survival times
 (s)')
    plt.plot(medians, marker = 'o',color = 'purple', label='Median survival time
s (s)')
    plt.plot(actual, marker = 'o', color = 'olive', label ='Actual survival time
s (s)')

    title = 'Survival times using RSF: '+ str(percent) +'% time horizon '+ use_s
oc +" SOC estimates"
    plt.title(title)
    plt.ylabel('Survival time (s)')
    plt.xlabel('Flight')
    plt.legend()
    plt.show

    plt.savefig('Results/RSF/plots/rsf_surv_time_TH'+str(percent)+save, bbox_inc
hes='tight')
```

In [ ]:
```
def compute_brier(event_times, quantile_time, prob_quant, actual_time):
# Compute Brier score (between 0 and 1, where 0 is the best possible value)

    # note that here 0 is assigned to 'dead' and 1 is assigned to 'survival'; si
nce this is what the metric requires.
    y_true = [1 if quantile_time <= actual_time else 0]

    bs = metrics.brier_score_loss(y_true, prob_quant)

    return bs
```

In [ ]:
```
def box_plot(metric_about_mean, metric_about_median, score_type, ci):
# Note that in final output we will instead be comparing the mse's from all the
 models on one plot,
# the mae's for all the models on one plot

    bars = [metric_about_mean[0], metric_about_median[0]]
    yer1 = [metric_about_mean[1], metric_about_median[1]]
    y_pos = np.arange(len(bars))

    labels = ['About mean', 'About median']
    plt.bar(y_pos, bars, width = 0.3, yerr=yer1)
    plt.xticks(y_pos, labels)
    plt.title('Average ' + str(score_type) + ' about mean and median survival ti
mes with ' + str(ci) + '% C.I.')

    plt.ylabel(str(score_type) + ' score')
    plt.show()
```

Flight numbers selected for analysis

Not all the data files downloaded from the NASA Prognostics Data Repository were suitable for analysis.

Data set citation: C. Kulkarni, E. Hogge, C. Quach and K. Goebel "HIRF Battery Data Set", NASA Ames Prognostics Data Repository (http://ti.arc.nasa.gov/project/prognostic-data-repository (http://ti.arc.nasa.gov/project/prognostic-data-repository)), NASA Ames Research Center, Moffett Field, CA

The following flights were omitted:

- Set 1 (flight numbers 6 through 13) were used for experimental debugging, and did not have BHM data.
- Flight numbers 63, 64, 67, 68, 69 had corrupted BHM data.
- Flight numbers 19, 31, 35, 74, 75 never reached a 30% state of charge.
- Flights 16, 71 showed outlier behaviour (potentially corrupted files).

In [ ]:
```
# Flight numbers

parasitic_load_files = [14,15,19,20,21,22,23,25,26,28,29,31,32,34,35,37,39,41]
no_load_files = [16,17,18,24,27,30,33,36,38,40,71,72,73,74,75,76,77,78,79,80,81]
files_selected = [14,15,17,18,20,21,22,23,24,25,26,27,28,29,30,32,33,34,36,37,38
,39,40,41,72,73,76,77,78,79,80,81]

file_numbers = files_selected
```

In [ ]:
```
# Enter boolean to develop model including or discluding the data state of charg
e estimates.
SOC_included = False

# Select time horizon until which to run the analysis (15%, 25%, 40%, 50%, 60%,
 75%, 85%, 100%).
percent = 100
time_horizons = [15, 25, 40, 50, 60, 75, 85, 100]
```

Perform data cleaning and calculation of static features

In [ ]:
```
all_input, survival_time, censor = [], [], []

# Loop through data files
for num in file_numbers:

    # Read csv files
    bhm_filename = "HIRF data sets/BHM estimate files/HIRF"+str(num)+"_BHM.csv"
    hirf_filename = "HIRF data sets/Measured data files/HIRF"+str(num)+".csv"
    df = pd.read_csv(hirf_filename)
    df_bhm = pd.read_csv(bhm_filename)
    df_soc_original = df_bhm.iloc[:,217:325] # disregard all columns except thos
e for SOC

    # Remove redundant SOC columns; select 1st column of SOC measurements for ea
ch battery
    df_soc = pd.concat([df_soc_original.iloc[:,0:1],df_soc_original.iloc[:,27:28
],df_soc_original.iloc[:,54:55],df_soc_original.iloc[:,81:82]],axis=1)

    # Indicator of whether plane is alive or dead
    status_indicator = create_status_ind(df_bhm)

    # Time measurements
    time = df_bhm.iloc[:,0]

    # Concatenate data frame with SOC columns with time array
    df_soc = pd.concat([time, df_soc], axis=1)

    # Merge SOC and measured data (on common time readings)
    merged = df.merge(df_soc, on='t', how = 'inner')

    # Accounts for case when SOC is not included as a model feature
    if SOC_included == False:
        m,n = df.shape
        drop_index = merged.iloc[:, n:]
        merged.drop(drop_index.columns,axis=1,inplace=True)

    merged['Status'] = status_indicator

    # Calculate survival time, append to list
    y = merged.loc[:,['t','Status']]
    for j in range(len(y)):
        if y.iloc[j, 1] == 1:   # dead
            survival_t = y.iloc[j,0]
            break
    survival_time.append(survival_t)

    # Append censor value to list; always equal to 1.
    censor.append(1)

    # Create parameter array
    x = merged.copy()
    x.drop('Status', axis=1, inplace=True) #drop the status column from the X da
taframe

    # Cleaning: remove extra empty columns, and time (not including as a featur
e)
    if 'BHM' in x.columns:
        x.drop('BHM', axis=1, inplace=True)
```

```
    if 'Unnamed: 19' in x.columns:
        x.drop('Unnamed: 19', axis=1, inplace=True)
    if 't' in x.columns:
        x.drop('t', axis=1, inplace=True)

    # Select data for percent of file included
    fraction = int(percent/100*len(x))
    X = x.iloc[:fraction,:]

    input_param, labels = calc_input_feat(X, num)
    all_input.append(input_param)
```

In [ ]:
```
data = {'Censor': bool(censor), 'Survival time (s)': survival_time}
target = pd.DataFrame(data)
display (target)

# convert above data frame to structured array for input into RSF function.
y = target.to_records(index=False)
```

In [ ]:
```
x = pd.DataFrame(all_input, columns=labels)
display(x)

# Standardization
columns = x.columns
scaler = StandardScaler()
df = scaler.fit_transform(x)
x = pd.DataFrame(df, columns=columns)
display(x)

# convert above data frame to structured array for input into RSF function.
X = x.to_numpy()
```

Perform leave-one-subject-out cross validation and plot survival functions.

For each iteration (fold) of LOOCV:

- Fit Random Survival Forest (RSF) model to each **training set**.
- Use model on **test set**.
  - Determine survival probabilities for each event time.
  - Compute median and mean survival times.
  - Calculate MSE around mean and median predicted survival times.
  - Calculate MAE around mean and median predicted survival times.
  - Calculate Brier Score.

Module used for random survival forest: **scikit-surival**

NOTE: Plot of survival functions at bottom of output.

In [ ]:
```
# Cross validation (leave one data set out)
cv = LeaveOneOut()
mean_surv_t_lst, mse_lst, mae_lst = [], [], []
med_surv_t_lst, mse_med_lst, mae_med_lst = [], [], []
act_surv_t_lst = []
bs_lst = []

# Loop over different flights
for train_index, test_index in cv.split(X):

    iteration = test_index[0] + 1     # iteration number of cross validation
    X_train, X_test, y_train, y_test = comp_sets(iteration, train_index, test_in
dex, y, X)

    # Fit RSF model to training set
    rsf = RandomSurvivalForest()
    rsf.fit(X_train, y_train)

    # Calculate probabilities at event times
    p_surv = rsf.predict_survival_function(X_test)
    plot_surv_fcn(rsf, iteration, p_surv)

    # Find event times
    event_times = rsf.event_times_

    # Display table of probabilities
    summary_table = create_summary(event_times, p_surv)

    # Actual survival time
    actual_surv_t = y_test['Survival time (s)'][0]
    print('Actual survival time (s):', actual_surv_t, '\n')
    act_surv_t_lst.append(actual_surv_t)

    # Calculate MSE and MAE around mean survival time for each fold
    mse_lst, mae_lst, mean_surv_t_lst = compute_metrics_about_mean(p_surv, event
_times,
                                        actual_surv_t, mse_lst, mae_lst, mean_su
rv_t_lst)

    # Calculte MSE and MAE around median survival time for each fold
    mse_med_lst, mae_med_lst, med_surv_t_lst = compute_metrics_about_median(summ
ary_table,
                                        actual_surv_t, mse_med_lst, mae_med_lst,
med_surv_t_lst)

    # Compute brier score
    # compute for all quantiles (loop)
    array_prob = p_surv[0]

    quantiles = np.quantile(event_times, [0.25, 0.5, 0.75, 1])
    bs = []
    for quantile in quantiles:
        idx = (np.abs(event_times - quantile)).argmin()
        prob_at_quant = [array_prob[idx]]
        bs_ = compute_brier(event_times, quantile, prob_at_quant, actual_surv_t)
        bs.append(bs_)
    print('Brier scores for 25%, 50%, 75%, 100% quantiles:', [round(val, 7) for
val in bs])
```

```
    print('--------')
    bs_lst.append(bs)
```

Print metrics summary (MSE, MAE, Brier Score)

Calculate mean and confidence intervals for MSE, MAE, and Brier Score.

In [ ]:
```
# Set confidence interval for MSE and MAE
ci = 90

# Metrics about mean
print("1) Metrics about mean predicted survival time")
mse_1 = find_avg_metric(mse_lst, 'MSE')    # Average MSE for all folds
mse_me1 = find_ci(mse_lst, 'MSE', ci)         # Confidence bounds on MSE
mae_1 = find_avg_metric(mae_lst, 'MAE')    # Average MAE for all folds
mae_me1 = find_ci(mae_lst, 'MAE', ci)         # Confidence bounds on MAE
print('--------------')

# Metrics about median
print("2) Metrics about median predicted survival time")
mse_2 = find_avg_metric(mse_med_lst, 'MSE')    # Average MSE for all folds
mse_me2 = find_ci(mse_med_lst, 'MSE', ci)         # Confidence bounds on MSE
mae_2 = find_avg_metric(mae_med_lst, 'MAE')    # Average MAE for all folds
mae_me2 = find_ci(mae_med_lst, 'MAE', ci)         # Confidence bounds on MAE
print('--------------')

# Brier score
print("3) Brier Score")
bs_25_lst = [subset[0] for subset in bs_lst]
bs_50_lst = [subset[1] for subset in bs_lst]
bs_75_lst = [subset[2] for subset in bs_lst]
bs_100_lst = [subset[3] for subset in bs_lst]

bs_25 = np.mean(bs_25_lst)
bs_50 = np.mean(bs_50_lst)
bs_75 = np.mean(bs_75_lst)
bs_100 = np.mean(bs_100_lst)
print('Brier scores for 25%, 50%, 75%, 100% quantiles:', bs_25, bs_50, bs_75, bs
_100)

bs_ci_25 = find_ci(bs_25_lst, 'BS', ci)
bs_ci_50 = find_ci(bs_50_lst, 'BS', ci)
bs_ci_75 = find_ci(bs_75_lst, 'BS', ci)
bs_ci_100 = find_ci(bs_100_lst, 'BS', ci)


# Output metrics in format
lst = {'MSE_mean': [mse_1], 'MSE m.e._mean': [mse_me1], 'MAE_mean': [mae_1], 'MA
E m.e._mean': [mae_me1], 'MSE_median': [mse_2], 'MSE m.e._median': [mse_me2], 'M
AE_median': [mae_2], 'MAE m.e._median': [mae_me2], 'BS 25%': [bs_25], 'BS 25% m.
e.': [bs_ci_25], 'BS 50%': [bs_50], 'BS 50% m.e.': [bs_ci_50], 'BS 75%': [bs_75
], 'BS 75% m.e.': [bs_ci_75], 'BS 100%': [bs_100], 'BS 100% m.e.': [bs_ci_100]}
columns = ['MSE_mean', 'MSE m.e._mean', 'MAE_mean', 'MAE m.e._mean', 'MSE_media
n', 'MSE m.e._median', 'MAE_median', 'MAE m.e._median', 'BS 25%', 'BS 25% m.e.',
'BS 50%', 'BS 50% m.e.', 'BS 75%', 'BS 75% m.e.','BS 100%', 'BS 100% m.e.']

df = pd.DataFrame(lst, columns=columns)
df.index = [str(percent)]
display(df)

if SOC_included == True:
    soc_cond = '_T.H._w_SOC'
else:
    soc_cond = '_T.H._wo_SOC'
```

```
df.to_csv(r'Results/RSF/rsf_'+str(percent)+soc_cond+'.csv', index = False)

# Plot
box_plot([mse_1, mse_me1], [mse_2, mse_me2], 'MSE', ci)
box_plot([mae_1, mae_me1], [mae_2, mae_me2], 'MAE', ci)

# Metrics table for box plots
data = {'RSF MSE about mean': mse_lst, 'RSF MAE about mean': mae_lst, 'RSF MSE a
bout median': mse_med_lst, 'RSF MAE about median': mae_med_lst, 'RSF 25% Brier S
core': bs_25_lst, 'RSF 50% Brier Score': bs_50_lst, 'RSF 75% Brier Score': bs_75
_lst, 'RSF 100% Brier Score': bs_100_lst}
df_for_boxplot = pd.DataFrame(data)
df_for_boxplot.to_csv(r'Results/RSF/boxplot_data/rsf_'+str(percent)+soc_cond+'.c
sv', index = False)
```

Summary of Predicted Survival Times (Mean and Median) and Actual Survival Times

Note that for the graph, x axis is flight numbers

In [ ]:
```
# Plot mean survival time, median survival time, actual survival time
comparison_plot(mean_surv_t_lst, med_surv_t_lst, act_surv_t_lst, file_numbers, s
tr(percent), SOC_included)
```

In [ ]:
```
```

Weibull Accelerated Failure Time Model

Authors: Laura Simandl, Molly Strasser

Course: 16791 Applied Data Science

The following code implements an **Weibull accelerated failure time** model for the Battery Prognostics Project.

In [ ]:
```
# Import modules and packages

import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import LeaveOneOut
from lifelines import WeibullAFTFitter
from collections import OrderedDict
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

np.random.seed(1000)
```

In [ ]:
```
def create_status_ind(df_bhm):
# Create 'status' column
# where '0' indicates that e-UAV is 'alive', and '1' indicates it is 'dead'
# (i.e. one of the four batteries reached a state of charge < 30%)

    status_col = []

    for i in range(len(df_bhm)):
        df_soc_row = np.array(df_bhm.iloc[i,217:325])
        indicator = 0
        for value in df_soc_row:
            if value < 30:
                indicator = 1  # dead
        status_col.append(indicator)

    return status_col
```

In [ ]:
```
def calc_input_feat(X, num):
# Calculate static features
# Calculate mean, median, standard deviation of each column
# Indicate if file has parametric load (boolean 1: yes, 0: no)

    input_param = []
    labels = []

    for col_name in X.columns:

        col = X[col_name]
        mean = round(col.mean(),3)
        median = round(col.median(),3)
        std = round(col.std(),3)
        input_param.append(mean)
        input_param.append(median)
        input_param.append(std)
        labels.append(col_name+"_mean")
        labels.append(col_name+"_median")
        labels.append(col_name+"_std")

    if num in parasitic_load_files:
        input_param.append(1)
    else:
        input_param.append(0)
    labels.append('file type')

    return input_param, labels
```

In [ ]:
```
def compute_mean_survival_t(surv, x_time, start):
# Compute mean predicted survival time; calculate auc.

    lst = []
    for i, s in enumerate(surv):
        y_prob = s  # array of probabilities for events in one fold
        area = metrics.auc(x_time,y_prob)
        mean_surv_t = round(area+start, 2)

    print('Predicted mean survival time (s):', mean_surv_t)

    return mean_surv_t
```

In [ ]:
```
def find_median_survival_t(summary):
# Finds median predicted survival time from a dataframe of survival times and as
sociated probabilites

    summary = summary.sort_values(by=['Probabilities'], ascending=True)

    p = summary['Probabilities'].array
    t = summary['Survival time (s)'].array

    median_survival_t = np.interp(0.5, p, t)  # median survival time when p=0.5

    print('Predicted median survival time (s):', round(median_survival_t,2))

    return median_survival_t
```

In [ ]:
```
def find_ci(score_lst, score_type, ci):
# Calculate confidence intervals on metrics

    z_scores = {90: 1.645, 95: 1.96, 98: 2.326, 99: 2.576}
    z_score = z_scores.get(ci)

    std_dev = np.std(score_lst)
    mean = np.mean(score_lst)
    me = z_score*std_dev/math.sqrt(len(score_lst))
    low_bound = mean - me
    up_bound = mean + me
    print('Lower bound on ', score_type ,'for', str(ci),'% confidence interval:'
, round(low_bound,4))
    print('Upper bound on ', score_type ,'for', str(ci),'% confidence interval:'
, round(up_bound,4), '\n')

    return me
```

In [ ]:
```
def find_avg_metric(score_lst, score_type):
# Calculate average of metric (MSE or MAE) from all the folds

    avg_metric = sum(score_lst)/len(score_lst)
    print('Average', score_type, 'for all iterations:', round(avg_metric,2))
    return avg_metric
```

In [ ]:
```
def create_summary(x_time, p_surv):
# Output data frame of surival times and probabilities

    t = pd.DataFrame(x_time, columns=['Survival time (s)'])
    p = pd.DataFrame(p_surv, columns=['Probabilities'])
    df = pd.concat([t,p], axis = 1)

    display(df.head())

    return df
```

In [ ]:
```
def comparison_plot(means, medians, actual, flights, percent, soc_included):

    if soc_included == True:
        use_soc = 'with'
        save = '_w_SOC'
    else:
        use_soc = 'without'
        save = '_wo_SOC'

    x_labels = str(flights)

    # data frame include calculation for mean surival times for each flight
    data = {'Flight number': flights, 'Mean survival times (s)': means, 'Median
 survival times (s)': medians, 'Actual survival times (s)': actual}
    df = pd.DataFrame(data, columns = ['Flight number', 'Mean survival times
 (s)', 'Median survival times (s)', 'Actual survival times (s)'])
    display(df)

    # Plot actual, median and mean survival time predictions
    plt.figure(1)
    plt.plot(means, marker = 'o',color = 'skyblue', label='Mean survival times
 (s)')
    plt.plot(medians, marker = 'o',color = 'purple', label='Median survival time
s (s)')
    plt.plot(actual, marker = 'o', color = 'olive', label ='Actual survival time
s (s)')

    title = 'Survival times using Weibull AFT: '+ str(percent) +'% time horizon
 '+ use_soc +" SOC estimates"
    plt.title(title)
    plt.ylabel('Survival time (s)')
    plt.xlabel('Flight')
    plt.legend()
    plt.show

    plt.savefig('Results/WAFT/plots/waft_surv_time_TH'+str(percent)+save, bbox_i
nches='tight')
```

In [ ]:
```
def feature_importance(rsf, X_test, y_test, labels):

    perm = PermutationImportance(rsf, n_iter=15)
    perm.fit(X_test, y_test)
    eli5.show_weights(perm, feature_names=labels)
```

In [ ]:
```
def box_plot(metric_about_mean, metric_about_median, score_type, ci):
# Note that in final output we will instead be comparing the mse's from all the
 models on one plot,
# the mae's for all the models on one plot

    bars = [metric_about_mean[0], metric_about_median[0]]
    yer1 = [metric_about_mean[1], metric_about_median[1]]
    y_pos = np.arange(len(bars))

    labels = ['About mean', 'About median']
    plt.bar(y_pos, bars, width = 0.3, yerr=yer1)
    plt.xticks(y_pos, labels)
    plt.title('Average ' + str(score_type) + ' about mean and median survival ti
mes with ' + str(ci) + '% C.I.')

    plt.ylabel(str(score_type) + ' score')
    plt.show()
```

In [ ]:
```
def comp_sets(fold_num, train_index, test_index, y, X):
# Select test and train sets

    print("Iteration:", fold_num, '\n')
    print("TRAIN indeces:", train_index, "\nTEST index:", test_index, '\n')

    X_test = X.iloc[test_index, :]


    X_train = X.drop(df.index[test_index])

    y_test = y.iloc[test_index, :]

    y_train = y.drop(df.index[test_index])

    return X_train, X_test, y_train, y_test
```

In [ ]:
```
def compute_metrics_about_mean(actual_surv_t, mse_lst, mae_lst, mean_surv_t):
# Compute metrics about mean survival time for given fold

    MSE = (actual_surv_t - mean_surv_t)**2
    print('MSE for fold about mean:', round(MSE,2))
    mse_lst.append(MSE)

    MAE = abs(actual_surv_t - mean_surv_t)
    print('MAE for fold about mean:', round(MAE,2), '\n')
    mae_lst.append(MAE)

    return mse_lst, mae_lst
```

In [ ]:
```
def compute_metrics_about_median(actual_surv_t, mse_lst, mae_lst, med_surv_t):
# Compute metrics about median survival time for given fold

    MSE = (actual_surv_t - med_surv_t)**2
    print('MSE for fold about median:', round(MSE,2))
    mse_lst.append(MSE)

    MAE = abs(actual_surv_t - med_surv_t)
    print('MAE for fold about median:', round(MAE,2), '\n')
    mae_lst.append(MAE)

    return mse_lst, mae_lst
```

In [ ]:
```
def compute_brier(event_times, quantile_time, prob_quant, actual_time):
# Compute Brier score (between 0 and 1, where 0 is the best possible value)

    # note that here 0 is assigned to 'dead' and 1 is assigned to 'survival'; si
nce this is what the metric requires.
    y_true = [1 if quantile_time <= actual_time else 0]

    bs = metrics.brier_score_loss(y_true, prob_quant)

    return bs
```

In [ ]:
```
def plot_surv_fcn(model, iteration, surv, event_times):
# Plot survival functions

    plt.figure(0)

    plt.plot(event_times, surv)
    plt.ylabel("Survival probability")
    plt.xlabel("Time in seconds")
    plt.title("Survival functions on test data")
    plt.grid(True)
```

In [ ]:
```
def retain_percent(pca, percent):
# Number of components needed to retain X% of total variance can be calculated o
r inferred from previous graph

    total_var = 0
    number_comp = 0
    for value in pca.explained_variance_ratio_:
        if total_var < percent/100:
            total_var += value
            number_comp += 1

    print('Number of principal components that must be retained to capture '+str
(percent)+'% of total variance:', number_comp)
    print('\n')
```

In [ ]:
```
def scree_plot(pca):
#Scree plot for distribution of variance contained in principal components (sort
ed by eigenvalues)

    fig1 = plt.figure()
    ax = fig1.add_subplot(111)
    ax.plot(pca.explained_variance_)
    ax.set(title='Scree plot for variance contained in principal components', xl
abel='Principal Component Index',
           ylabel='Explained Variance')
    plt.show()
```

Flight numbers selected for analysis

Not all the data files downloaded from the NASA Prognostics Data Repository were suitable for analysis.

Data set citation: C. Kulkarni, E. Hogge, C. Quach and K. Goebel "HIRF Battery Data Set", NASA Ames Prognostics Data Repository (http://ti.arc.nasa.gov/project/prognostic-data-repository (http://ti.arc.nasa.gov/project/prognostic-data-repository)), NASA Ames Research Center, Moffett Field, CA

The following flights were omitted:

- Set 1 (flight numbers 6 through 13) were used for experimental debugging, and did not have BHM data.
- Flight numbers 63, 64, 67, 68, 69 had corrupted BHM data.
- Flight numbers 19, 31, 35, 74, 75 never reached a 30% state of charge.
- Flights 16, 71 showed outlier behaviour.

In [ ]:
```
# Flight numbers

original = [14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36
,37,38,39,40,41,63,64,67,68,69,71,72,73,74,75,76,77,78,79,80,81]
files_selected = [14,15,17,18,20,21,22,23,24,25,26,27,28,29,30,32,33,34,36,37,38
,39,40,41,72,73,76,77,78,79,80,81]

parasitic_load_files = [14,15,19,20,21,22,23,25,26,28,29,31,32,34,35,37,39,41]
no_load_files = [16,17,18,24,27,30,33,36,38,40,71,72,73,74,75,76,77,78,79,80,81]

file_numbers = files_selected
```

In [ ]:
```
# Enter boolean to develop model including or discluding the data state of charg
e estimates.
SOC_included = True

# Select time horizon until which to run the analysis (15%, 25%, 40%, 50%, 60%,
 75%, 85%, 100%).
percent = 85
time_horizons = [15, 25, 40, 50, 60, 75, 85, 100]
```

Perform data cleaning and calculation of static features

In [ ]:
```
all_input, survival_time, censor = [], [], []

# Loop through data files
for num in file_numbers:

    # Read csv files
    bhm_filename = "HIRF data sets/BHM estimate files/HIRF"+str(num)+"_BHM.csv"
    hirf_filename = "HIRF data sets/Measured data files/HIRF"+str(num)+".csv"
    df = pd.read_csv(hirf_filename)
    df_bhm = pd.read_csv(bhm_filename)
    df_soc_original = df_bhm.iloc[:,217:325] # disregard all columns except thos
e for SOC

    # Remove redundant SOC columns; select 1st column of SOC measurements for ea
ch battery
    df_soc = pd.concat([df_soc_original.iloc[:,0:1],df_soc_original.iloc[:,27:28
],df_soc_original.iloc[:,54:55],df_soc_original.iloc[:,81:82]],axis=1)

    # Indicator of whether plane is alive or dead
    status_indicator = create_status_ind(df_bhm)

    # Time measurements
    time = df_bhm.iloc[:,0]

    # Concatenate data frame with SOC columns with time array
    df_soc = pd.concat([time, df_soc], axis=1) # data fram of soc columns with t
ime

    # Merge SOC and measured data (on common time readings)
    merged = df.merge(df_soc, on='t', how = 'inner')

    # Accounts for case when SOC is not included as a model feature
    if SOC_included == False:
        m,n = df.shape
        drop_index = merged.iloc[:, n:]
        merged.drop(drop_index.columns,axis=1,inplace=True)

    merged['Status'] = status_indicator

    # Calculate survival time, append to list
    y = merged.loc[:,['t','Status']]
    for j in range(len(y)):
        if y.iloc[j, 1] == 1:   # dead
            survival_t = y.iloc[j,0]
            break
    survival_time.append(survival_t)

    # Append censor value to list; always equal to 1.
    censor.append(1)

    # Create parameter array
    x = merged.copy()
    x.drop('Status', axis=1, inplace=True) #drop the status column from the X da
taframe

    # Cleaning: remove empty columns
    if 'BHM' in x.columns:
        x.drop('BHM', axis=1, inplace=True)
```

```
    if 'Unnamed: 19' in x.columns:
        x.drop('Unnamed: 19', axis=1, inplace=True)
    if 't' in x.columns:
        x.drop('t', axis=1, inplace=True)

    # Select data for percent of file included
    fraction = int(percent/100*len(x))
    X = x.iloc[:fraction,:]

    input_param, labels = calc_input_feat(X, num)
    all_input.append(input_param)
```

In [ ]:
```
# Data frame of target
data = {'Censor': bool(censor), 'Survival time (s)': survival_time}
y = pd.DataFrame(data)
display (y)

# miny = min(y['Survival time (s)'])
# maxy = max(y['Survival time (s)'])
# y_std = (y['Survival time (s)']-miny)/(maxy - miny)
# y_std += 0.001  # model doesn't run if there's a value of zero; add a constant
# y_std = pd.DataFrame(y_std)

# y = pd.concat([y['Censor'],y_std], axis = 1)
# display(y)
```

In [ ]:
```
# Data frame with all static features for all flights
X = pd.DataFrame(all_input, columns=labels)
display(X)
```

In [ ]:
```
# Standardization of features
Xstd = StandardScaler().fit_transform(X)

# Perform PCA to produce scree plot and for model
pca = PCA(n_components=28)
pca_components = pca.fit_transform(Xstd)
scree_plot(pca)

pca_components = pd.DataFrame(pca_components)
display(pca_components)
```

Perform leave-one-subject-out cross validation and plot survival functions.

For each iteration (fold) of LOOCV:

- Fit model to each **training set**.
- Use model on **test set**.
  - Determine survival probabilities for each event time.
  - Compute median and mean survival times.
  - Calculate MSE around mean and median predicted survival times.
  - Calculate MAE around mean and median predicted survival times.
  - Calculate Brier Score.

Module used for model: **lifelines**

NOTE: Plot of survival functions at bottom of output.

In [ ]:
```
# Cross validation (leave one data set out)
cv = LeaveOneOut()
mean_surv_t_lst, mse_lst, mae_lst = [], [], []
med_surv_t_lst, mse_med_lst, mae_med_lst = [], [], []
act_surv_t_lst = []
bs_lst = []

# Loop over different flights
for train_index, test_index in cv.split(pca_components):

    iteration = test_index[0] + 1     # iteration number of cross validation
    X_train, X_test, y_train, y_test = comp_sets(iteration, train_index, test_in
dex, y, pca_components)

    # Combine X_train, y_train into one dataframe
    dataset = pd.concat([X_train, y_train], axis = 1)
    display(dataset)

    # Fit AFT model to training set
    aft = WeibullAFTFitter(penalizer = 0.01)
    aft.fit(dataset, duration_col = 'Survival time (s)', event_col = 'Censor')

    # Event times (training)
    event_times = sorted(y_train['Survival time (s)'])

    # Plots and table
    aft.print_summary(3)

    # Predict the survival function for individuals.
    p_surv = aft.predict_survival_function(X_test)

    # Predict mean lifetime for test individual
    mean_test_flight = aft.predict_expectation(X_test)
    mean = mean_test_flight[iteration-1]
    mean_surv_t_lst.append(mean)
    print('Mean predicted survival time (s):', round(mean,3), '\n')

    # Find median lifetime for test individual
    median_test_flight = aft.predict_median(X_test)
    median = median_test_flight[iteration-1]
    med_surv_t_lst.append(median)
    print('Median predicted survival time (s):', round(median,3), '\n')

    # Actual survival time
    actual_surv_t = y_test.loc[:, 'Survival time (s)'][iteration-1]
    act_surv_t_lst.append(actual_surv_t)
    print('Actual survival time (s):', actual_surv_t, '\n')

    # Calculate MSE and MAE around mean survival time for each fold
    mse_lst, mae_lst = compute_metrics_about_mean(actual_surv_t, mse_lst, mae_ls
t, mean)

    # Calculte MSE and MAE around median survival time for each fold
    mse_med_lst, mae_med_lst = compute_metrics_about_median(actual_surv_t, mse_m
ed_lst, mae_med_lst, median)

    print('----------------')
```

```
    from collections import OrderedDict
    event_times = list(OrderedDict.fromkeys(event_times))

    # Plot survival functions (of test sets)
    plot_surv_fcn(aft, iteration, p_surv, event_times)
    use_soc = 'with' if SOC_included == True else 'without'
    title = 'Survival times: '+ str(percent) +'% time horizon '+ use_soc +" SOC
 estimates"
    plt.title(title)

    # Compute brier score
    quantiles = np.quantile(event_times, [0.25, 0.5, 0.75, 1])

    bs = []
    for quantile in quantiles:
        idx = (np.abs(event_times - quantile)).argmin()
        prob_at_quant = p_surv.iloc[idx]
        bs_ = compute_brier(event_times, quantile, prob_at_quant, actual_surv_t)
        bs.append(bs_)
    print('Brier scores for 25%, 50%, 75%, 100% quantiles:', [round(val, 7) for
val in bs])
    print('--------')
    bs_lst.append(bs)
```

Print metrics summary (MSE, MAE, Brier Score)

Calculate mean and confidence intervals for MSE, MAE, and Brier Score.

In [ ]:
```
# Set confidence interval for MSE and MAE
ci = 90

# Metrics about mean
print("1) Metrics about mean predicted survival time")
mse_1 = find_avg_metric(mse_lst, 'MSE')    # Average MSE for all folds
mse_me1 = find_ci(mse_lst, 'MSE', ci)         # Confidence bounds on MSE
mae_1 = find_avg_metric(mae_lst, 'MAE')    # Average MAE for all folds
mae_me1 = find_ci(mae_lst, 'MAE', ci)         # Confidence bounds on MAE
print('--------------')

# Metrics about median
print("2) Metrics about median predicted survival time")
mse_2 = find_avg_metric(mse_med_lst, 'MSE')    # Average MSE for all folds
mse_me2 = find_ci(mse_med_lst, 'MSE', ci)         # Confidence bounds on MSE
mae_2 = find_avg_metric(mae_med_lst, 'MAE')    # Average MAE for all folds
mae_me2 = find_ci(mae_med_lst, 'MAE', ci)         # Confidence bounds on MAE
print('--------------')

# Brier score
bs_25_lst = [subset[0] for subset in bs_lst]
bs_50_lst = [subset[1] for subset in bs_lst]
bs_75_lst = [subset[2] for subset in bs_lst]
bs_100_lst = [subset[3] for subset in bs_lst]

bs_25 = np.mean(bs_25_lst)
bs_50 = np.mean(bs_50_lst)
bs_75 = np.mean(bs_75_lst)
bs_100 = np.mean(bs_100_lst)
print('Brier scores for 25%, 50%, 75%, 100% quantiles:', bs_25, bs_50, bs_75, bs
_100)

bs_ci_25 = find_ci(bs_25_lst, 'BS', ci)
bs_ci_50 = find_ci(bs_50_lst, 'BS', ci)
bs_ci_75 = find_ci(bs_75_lst, 'BS', ci)
bs_ci_100 = find_ci(bs_100_lst, 'BS', ci)

# Output avg metrics in format
lst = {'MSE_mean': [mse_1], 'MSE m.e._mean': [mse_me1], 'MAE_mean': [mae_1], 'MA
E m.e._mean': [mae_me1], 'MSE_median': [mse_2], 'MSE m.e._median': [mse_me2], 'M
AE_median': [mae_2], 'MAE m.e._median': [mae_me2], 'BS 25%': [bs_25], 'BS 25% m.
e.': [bs_ci_25], 'BS 50%': [bs_50], 'BS 50% m.e.': [bs_ci_50], 'BS 75%': [bs_75
], 'BS 75% m.e.': [bs_ci_75], 'BS 100%': [bs_100], 'BS 100% m.e.': [bs_ci_100]}
columns = ['MSE_mean', 'MSE m.e._mean', 'MAE_mean', 'MAE m.e._mean', 'MSE_media
n', 'MSE m.e._median', 'MAE_median', 'MAE m.e._median', 'BS 25%', 'BS 25% m.e.',
'BS 50%', 'BS 50% m.e.', 'BS 75%', 'BS 75% m.e.','BS 100%', 'BS 100% m.e.']

df = pd.DataFrame(lst, columns=columns)
df.index = [str(percent)]
display(df)

if SOC_included == True:
    soc_cond = '_T.H._w_SOC'
else:
    soc_cond = '_T.H._wo_SOC'

df.to_csv(r'Results/WAFT/waft_'+str(percent)+soc_cond+'.csv', index = False)
```

```
# Plot
box_plot([mse_1, mse_me1], [mse_2, mse_me2], 'MSE', ci)
box_plot([mae_1, mae_me1], [mae_2, mae_me2], 'MAE', ci)

# Metrics table for box plots
data = {'WAFT MSE about mean': mse_lst, 'WAFT MAE about mean': mae_lst, 'WAFT MS
E about median': mse_med_lst, 'WAFT MAE about median': mae_med_lst, 'WAFT 25% Br
ier Score': bs_25_lst, 'WAFT 50% Brier Score': bs_50_lst, 'WAFT 75% Brier Score'
: bs_75_lst, 'WAFT 100% Brier Score': bs_100_lst}
df_for_boxplot = pd.DataFrame(data)
df_for_boxplot.to_csv(r'Results/WAFT/boxplot_data/waft_'+str(percent)+soc_cond+
'.csv', index = False)
```

Summary of Predicted Survival Times (Mean and Median) and Actual Survival Times

Note that for the graph, x axis is flight numbers

In [ ]:
```
# Plot mean survival time, median survival time, actual survival time
comparison_plot(mean_surv_t_lst, med_surv_t_lst, act_surv_t_lst, files_selected,
str(percent), SOC_included)
```

Linear Regression and Polynomial Regression Models

Authors: Laura Simandl, Molly Strasser

Course: 16791 Applied Data Science

The following code implements a linear regression model and polynomial models (d=2,3) for the Battery Prognostics Project.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
all_output=[]
all_y=[]
labels=[]
time_of_death_col=[]
flight_number = []

def find_static_features_with_soc(j, quar, find_labels=False, parasitic = False
):
        bhm = "HIRF"+str(j)+"_BHM.csv"
        hirf = "HIRF"+str(j)+".csv"
        df = pd.read_csv(hirf)
        df2 = pd.read_csv(bhm)

        found = False
        for i in range(len(df2)):
            df2_row = np.array(df2.iloc[i,217:325])
            for p in range(0, len(df2_row)):
                if df2_row[p] < 30:
                    time_of_death_col.append(df2.iloc[i,0])
                    found = True
                    break
            if (found==True):
                break
        if(found == False):
            print(j)

        df2.drop(df2.iloc[:, 1:217], axis = 1, inplace=True)

        # remove redundant SOC columns; select 1st column
        df_soc = pd.concat([df2.iloc[:,0:1],df2.iloc[:,27:28],df2.iloc[:,54:55],
df2.iloc[:,81:82]],axis=1)
        # the following line of code is omitted when run " without SOC"
        merged_inner = df.merge(df_soc, on='t', how = 'inner') # get the rows wh
ere BHM has been recorded
        timeline = merged_inner.filter(['t'], axis=1)
        quartiles = round(len(timeline)*quar)
        q_25 = timeline[0:quartiles]


        if 'Unnamed: 329' in merged_inner.columns:
            merged_inner.drop('Unnamed: 329', axis=1, inplace=True)
        if 'BHM' in merged_inner.columns:
            merged_inner.drop('BHM', axis=1, inplace=True)
        if 'Unnamed: 19' in merged_inner.columns:
            merged_inner.drop('Unnamed: 19', axis=1, inplace=True)
        if 'ULA_V_est' in merged_inner.columns:
            merged_inner.drop('ULA_V_est', axis=1, inplace=True)
        if 'LRF_V_est' in merged_inner.columns:
            merged_inner.drop('LRF_V_est', axis=1, inplace=True)
        if 'URA_V_est' in merged_inner.columns:
            merged_inner.drop('URA_V_est', axis=1, inplace=True)

        X = merged_inner.merge(q_25, on='t', how = 'inner')
        X.drop(['t'], axis = 1, inplace=True)


        output = []
        labels=[]
```

```
        for col in X.columns:
            mean = X[col].mean()
            median = X[col].median()
            std = X[col].std()
            output.append(mean)
            output.append(median)
            output.append(std)
            if(find_labels == True):
                labels.append(col+"_mean")
                labels.append(col+"_median")
                labels.append(col+"_std")
        if (parasitic == True):
            output.append(1)
        if (parasitic == False):
            output.append(0)
        return(output, labels)

# identify all the non-parastic load files
non_parasitic_loads = [18,24,27,30,33,36,38,40,72,73,76,77,78,79,80,81]

# identify all the parastic load files
parasitic_loads = [14,15,20,21,22,23,25,26,28,29,32,34,37,39,41]

# create a list to store all the data output to use to create dataframe
all_output=[]
# create a list to store the labels to use for the dataframe
labels=[]

# read in the first file and set the labels to be used for all the files
# set the parameter for quartile
x_val, labels = find_static_features_with_soc(16, .15, True, True)
all_output.append(x_val) # add data to the output list
labels.append("parasitic_load") # add the column name for parastic load

# cycle through all the files and read them in
for j in non_parasitic_loads:
    x_val, l = find_static_features_with_soc(j, .15, False, True)
    all_output.append(x_val)

# cycle through all the files and read them in
for j in parasitic_loads:
    x_val, l = find_static_features_with_soc(j, .15, False, False)
    all_output.append(x_val)

# create a dataframe to store the data for the 40% time horizon
df_15 = pd.DataFrame(data=all_output, columns = labels)
scaler = StandardScaler()
np_standardized = scaler.fit_transform(df_15)
X_15 = pd.DataFrame(np_standardized, columns = labels)

# create a list to store all the data output to use to create dataframe
all_output=[]
labels=[]

x_val, labels = find_static_features_with_soc(16, .25, True, True)
all_output.append(x_val)
labels.append("parasitic_load")
```

```
for j in non_parasitic_loads:
    x_val, l = find_static_features_with_soc(j, .25, False, True)
    all_output.append(x_val)

for j in parasitic_loads:
    x_val, l = find_static_features_with_soc(j, .25, False, False)
    all_output.append(x_val)

df_25 = pd.DataFrame(data=all_output, columns = labels)
np_standardized = scaler.fit_transform(df_25)
X_25 = pd.DataFrame(np_standardized, columns = labels)

# create a list to store all the data output to use to create dataframe
all_output=[]
# create a list to store the labels to use for the dataframe
labels=[]

# read in the first file and set the labels to be used for all the files
# set the parameter for quartile
x_val, labels = find_static_features_with_soc(16, .4, True, True)
all_output.append(x_val)
labels.append("parasitic_load")

for j in non_parasitic_loads:
    x_val, l = find_static_features_with_soc(j, .4,False, True)
    all_output.append(x_val)

for j in parasitic_loads:
    x_val, l = find_static_features_with_soc(j, .4, False, False)
    all_output.append(x_val)

df_40 = pd.DataFrame(data=all_output, columns = labels)
np_standardized = scaler.fit_transform(df_40)
X_40 = pd.DataFrame(np_standardized, columns = labels)

all_output=[]
labels=[]

x_val, labels = find_static_features_with_soc(16, .5, True, True)
all_output.append(x_val)
labels.append("parasitic_load")

for j in non_parasitic_loads:
    x_val, l = find_static_features_with_soc(j, .5,False, True)
    all_output.append(x_val)


for j in parasitic_loads:
    x_val, l = find_static_features_with_soc(j, .5, False, False)
    all_output.append(x_val)

df_50 = pd.DataFrame(data=all_output, columns = labels)
np_standardized = scaler.fit_transform(df_50)
X_50 = pd.DataFrame(np_standardized, columns = labels)

all_output=[]
labels=[]
```

```
x_val, labels = find_static_features_with_soc(16, .60, True, True)
all_output.append(x_val)
labels.append("parasitic_load")

for j in non_parasitic_loads:
    x_val, l = find_static_features_with_soc(j, .60, False, True)
    all_output.append(x_val)

for j in parasitic_loads:
    x_val, l = find_static_features_with_soc(j,.60, False, False)
    all_output.append(x_val)

df_60 = pd.DataFrame(data=all_output, columns = labels)
np_standardized = scaler.fit_transform(df_60)
X_60 = pd.DataFrame(np_standardized, columns = labels)

all_output=[]
labels=[]

x_val, labels = find_static_features_with_soc(16, .75, True, True)
all_output.append(x_val)
labels.append("parasitic_load")

for j in non_parasitic_loads:
    x_val, l = find_static_features_with_soc(j, .75, False, True)
    all_output.append(x_val)

for j in parasitic_loads:
    x_val, l = find_static_features_with_soc(j,.75, False, False)
    all_output.append(x_val)

df_75 = pd.DataFrame(data=all_output, columns = labels)
np_standardized = scaler.fit_transform(df_75)
X_75 = pd.DataFrame(np_standardized, columns = labels)

all_output=[]
labels=[]

x_val, labels = find_static_features_with_soc(16,.85, True, True)
all_output.append(x_val)
labels.append("parasitic_load")

for j in non_parasitic_loads:
    x_val, l = find_static_features_with_soc(j, .85,False, True)
    all_output.append(x_val)

for j in parasitic_loads:
    x_val, l = find_static_features_with_soc(j,.85, False, False)
    all_output.append(x_val)

df_85 = pd.DataFrame(data=all_output, columns = labels)
np_standardized = scaler.fit_transform(df_85)
X_85 = pd.DataFrame(np_standardized, columns = labels)

all_output=[]
labels=[]
```

```
x_val, labels = find_static_features_with_soc(16, True, True)
all_output.append(x_val)
labels.append("parasitic_load")

for j in non_parasitic_loads:
    x_val, l = find_static_features_with_soc(j, 1.0,False, True)
    all_output.append(x_val)

for j in parasitic_loads:
    x_val, l = find_static_features_with_soc(j,1.0, False, False)
    all_output.append(x_val)

df_100 = pd.DataFrame(data=all_output, columns = labels)
np_standardized = scaler.fit_transform(df_100)
X_100 = pd.DataFrame(np_standardized, columns = labels)
```

In [ ]:
```
time_of_death=[]
def find_death(j):
        bhm = "HIRF"+str(j)+"_BHM.csv"
        df2 = pd.read_csv(bhm)

        found = False
        for i in range(len(df2)):
            df2_row = np.array(df2.iloc[i,217:325])
            for p in range(0, len(df2_row)):
                if df2_row[p] < 30:
                    time_of_death.append(df2.iloc[i,0])
                    found = True
                    break
            if (found==True):
                break
        if(found == False):
            time_of_death.append(df2.iloc[len(df2)-1,0])
# 19 35 31 74 75
flight = [16,18,24,27,30,33,36,38,40,72,73,76,77,78,79,80,81,14,15,20,21,22,23,2
5,26,28,29,32,34,37,39,41]
for f in flight:
    find_death(f)
```

In [ ]:
```
df_ToD = pd.DataFrame({'Flight No': flight, 'Time Of Death': time_of_death})
```

In [ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import math
import statistics
from sklearn.metrics import brier_score_loss
import scipy.stats
event_times = df_ToD.sort_values(by = 'Time Of Death').to_numpy()

# Confidence bounds
def findCI(score_list, score_type):
    if (score_type == 'Brier'):
        se = []
        bs_25 = np.array([i[0] for i in score_list])
        bs_50 = np.array([i[1] for i in score_list])
        bs_75 = np.array([i[2] for i in score_list])
        bs_100 = np.array([i[3] for i in score_list])
        se_25 = round(1.645 * np.std(bs_25) / math.sqrt(len(score_list)),3)
        se_50 = round(1.645 * np.std(bs_50) / math.sqrt(len(score_list)),3)
        se_75 = round(1.645 * np.std(bs_75) / math.sqrt(len(score_list)),3)
        se_100 = round(1.645 * np.std(bs_100) / math.sqrt(len(score_list)),3)
        se.append((se_25,se_50,se_75,se_100))
        print("The standard error for each bs quantile is: ", se)
    else:
        std_dev = np.std(score_list)
        mean = np.mean(score_list)
        se = round(1.645 * std_dev / math.sqrt(len(score_list)),1)
        low_bound = round(mean - se, 1)
        up_bound = round(mean + se, 1)
        print('Lower bound on ', score_type , ' (90%):', low_bound)
        print('Upper bound on ', score_type ,' (90%):', up_bound)
    return (se)

def lin_reg_cv(df, output):
    X = df.to_numpy()
    output = output[['Time Of Death']].to_numpy()
    bs_lst =[]
    mean_mse = np.inf
    best_mse = np.inf
    num_folds = len(X)
    y_pred = []
    k_fold = KFold(num_folds,shuffle=False, random_state=0)

    indices = range(len(X))
    classifier = LinearRegression()
    mse_scores = []
    mae_scores = []

    for train_indices, val_indices in k_fold.split(indices):
        classifier.fit(X[train_indices],
                       output[train_indices])
        predicted_val = classifier.predict(X[val_indices])
        y_pred.append(predicted_val[0,0])
        mse = mean_squared_error(output[val_indices], predicted_val)
        mae = mean_absolute_error(output[val_indices], predicted_val)
        mse_scores.append(mse)
        mae_scores.append(mae)
```

```
        quants = [0.25, 0.5, 0.75, 1]

        sigma = math.sqrt(mse)

        actual_surv_t = output[val_indices][0,0]

        quantiles = np.quantile(event_times, quants)
        bs = []

        for quantile in quantiles:
            norm = statistics.NormalDist(predicted_val[0,0], sigma)
            cdf = norm.cdf(quantile)

            array_prob = [1-cdf]

            bs_ = compute_brier(quantile, actual_surv_t,array_prob)

            bs.append(bs_)
        bs_lst.append(bs)

    mean_mse = np.mean(mse_scores)
    mean_mae = np.mean(mae_scores)
    print('Mean squared error:', mean_mse)
    print('Mean absolute error:', mean_mae)
    bs_25 = np.array([i[0] for i in bs_lst])
    bs_50 = np.array([i[1] for i in bs_lst])
    bs_75 = np.array([i[2] for i in bs_lst])
    bs_100 = np.array([i[3] for i in bs_lst])
    print('Mean brier score 25%:', np.mean(bs_25))
    print('Mean brier score 50%:', np.mean(bs_50))
    print('Mean brier score 75%:', np.mean(bs_75))
    print('Mean brier score 100%:', np.mean(bs_100))
    return mean_mse, mean_mae, y_pred, mse_scores, mae_scores, bs_lst

def print_plot(y_q, quartile, use_soc, model, degree=1):
    df_ToD['Predicted Time of Death'] = y_q
    df_sorted = df_ToD.sort_values(by=['Flight No'])
    flight = list(range(0,df_sorted.shape[0]))
    predicted_y = df_sorted[['Predicted Time of Death']]
    actual_y = df_sorted[['Time Of Death']]
    pred, = plt.plot(flight,predicted_y, marker = 'o',color = 'skyblue', label=
'Mean survival times (s)')
    act, = plt.plot(flight,actual_y, marker = 'o', color = 'olive', label ='Actu
al survival times (s)')
    plt.legend(handles=[pred, act])
    if (model == 'LinReg'):
        title = 'Survival times using LR ' + quartile + "% time horizon with SOC
estimates"
    else:
        title = 'Survival times using PR degree '+ str(degree) +", " + quartile
+ "% time horizon with SOC estimates"
    plt.title(title)
    plt.xlabel('Flight')
    plt.ylabel('Survival time (s)')

    plt.show
    plt.savefig('/Users/Molly/Documents/Battery/Plots/'+ model+ "_with_SOC_horiz
```

```
on" + quartile + "_" + str(degree),
                bbox_inches='tight')

def send_to_excel(mse, mae, time_horizon, model, bs_lst,
                  mse_se, mae_se, bs_se, mse_scores, mae_scores,
                  degree = "zero"):
    bs_25 = np.array([i[0] for i in bs_lst])
    bs_50 = np.array([i[1] for i in bs_lst])
    bs_75 = np.array([i[2] for i in bs_lst])
    bs_100 = np.array([i[3] for i in bs_lst])
    data = {'MSE_mean':mse,'MSE m.e._mean': mse_se,'MAE_mean':mae, 'MAE m.e._mea
n': mae_se,
            "BS 25%": np.mean(bs_25), "BS 25% m.e.": bs_se[0][0], "BS 50%" : np.
mean(bs_50), "BS 50% m.e.": bs_se[0][1],
            "BS 75%" : np.mean(bs_75), "BS 75% m.e.": bs_se[0][2], "BS 100%" : n
p.mean(bs_100), "BS 100% m.e.": bs_se[0][3]}
    df = pd.DataFrame(data, index=[0])
    df.to_csv(model + "_" + degree + "_" + time_horizon + "_avg.csv")

    mse_col = model + ' MSE about mean'
    mae_col = model + ' MAE about mean'
    bs_col1 = model + ' 25% Brier Score'
    bs_col2 = model + ' 50% Brier Score'
    bs_col3 = model + ' 75% Brier Score'
    bs_col4 = model + ' 100% Brier Score'
    arr = {mse_col: mse_scores, mae_col: mae_scores, bs_col1: bs_25, bs_col2:bs_
50, bs_col3: bs_75, bs_col4: bs_100}
    df = pd.DataFrame(data = arr)
    df.to_csv(model + "_w_SOC" + degree + "_" + time_horizon + "_all.csv")


def compute_brier(event_quantile, actual_surv_t, array_prob):
# Compute Brier score (between 0 and 1, where 0 is the best possible value)
    y_true = [1 if event_quantile <= actual_surv_t else 0 for i in range(len(arr
ay_prob))]
    # note that here 0 is assigned to 'dead' and 1 is assigned to 'survival'; si
nce this is what the metric requires
    bs = brier_score_loss(y_true, array_prob)
    return bs

mean_mse_q15, mean_mae_q15, y_pred_q15, mse_scores_q15, mae_scores_q15, bs_lst_q
15 = lin_reg_cv(X_15,df_ToD)
brier_me_q15 = findCI(bs_lst_q15, 'Brier')
mse_ci_q15 = findCI(mse_scores_q15, 'MSE')
mae_ci_q15 = findCI(mae_scores_q15, 'MAE')
print_plot(y_pred_q15, "15", " using ", "LinReg")
```

In [ ]:

```
mean_mse_q25, mean_mae_q25, y_pred_q25, mse_scores_q25, mae_scores_q25, bs_lst_q
25 = lin_reg_cv(X_25,df_ToD)
brier_me_q25 = findCI(bs_lst_q25, 'Brier')
mse_ci_q25 = findCI(mse_scores_q25, 'MSE')
mae_ci_q25 = findCI(mae_scores_q25, 'MAE')
print_plot(y_pred_q25, "25", " using ", "LinReg")
```

In [ ]:
```
mean_mse_q40, mean_mae_q40, y_pred_q40, mse_scores_q40, mae_scores_q40, bs_lst_q
40 = lin_reg_cv(X_40,df_ToD)
brier_me_q40 = findCI(bs_lst_q40, 'Brier')
mse_ci_q40 = findCI(mse_scores_q40, 'MSE')
mae_ci_q40 = findCI(mae_scores_q40, 'MAE')
print_plot(y_pred_q40, "40", " using ", "LinReg")
```

In [ ]:
```
mean_mse_q50, mean_mae_q50, y_pred_q50, mse_scores_q50, mae_scores_q50, bs_lst_q
50 = lin_reg_cv(X_50,df_ToD)
brier_me_q50 = findCI(bs_lst_q50, 'Brier')
mse_ci_q50 = findCI(mse_scores_q50, 'MSE')
mae_ci_q50 = findCI(mae_scores_q50, 'MAE')
print_plot(y_pred_q50, "50", " using ", "LinReg")
```

In [ ]:
```
mean_mse_q60, mean_mae_q60, y_pred_q60, mse_scores_q60, mae_scores_q60, bs_lst_q
60 = lin_reg_cv(X_60,df_ToD)
brier_me_q60 = findCI(bs_lst_q60, 'Brier')
mse_ci_q60 = findCI(mse_scores_q60, 'MSE')
mae_ci_q60 = findCI(mae_scores_q60, 'MAE')
print_plot(y_pred_q60, "60", " using ", "LinReg")
```

In [ ]:
```
mean_mse_q75, mean_mae_q75, y_pred_q75, mse_scores_q75, mae_scores_q75, bs_lst_q
75 = lin_reg_cv(X_75,df_ToD)
brier_me_q75 = findCI(bs_lst_q75, 'Brier')
mse_ci_q75 = findCI(mse_scores_q75, 'MSE')
mae_ci_q75 = findCI(mae_scores_q75, 'MAE')
print_plot(y_pred_q75, "75", " using ", "LinReg")
```

In [ ]:
```
mean_mse_q85, mean_mae_q85, y_pred_q85, mse_scores_q85, mae_scores_q85, bs_lst_q
85 = lin_reg_cv(X_85,df_ToD)
brier_me_q85 = findCI(bs_lst_q85, 'Brier')
mse_ci_q85 = findCI(mse_scores_q85, 'MSE')
mae_ci_q85 = findCI(mae_scores_q85, 'MAE')
print_plot(y_pred_q85, "85", " using ", "LinReg")
```

In [ ]:
```
mean_mse_q100, mean_mae_q100, y_pred_q100, mse_scores_q100, mae_scores_q100, bs_
lst_q100 = lin_reg_cv(X_100,df_ToD)
brier_me_q100 = findCI(bs_lst_q100, 'Brier')
mse_ci_q100 = findCI(mse_scores_q100, 'MSE')
mae_ci_q100 = findCI(mae_scores_q100, 'MAE')
print_plot(y_pred_q100, "100", " using ", "LinReg")
```

In [ ]:
```
send_to_excel(mean_mse_q15, mean_mae_q15,  '15', 'Linear_regression', bs_lst_q15
,
              mse_ci_q15, mae_ci_q15, brier_me_q15, mse_scores_q15, mae_scores_q
15)

send_to_excel(mean_mse_q25, mean_mae_q25,  '25', 'Linear_regression', bs_lst_q25
,
              mse_ci_q25, mae_ci_q25, brier_me_q25, mse_scores_q25, mae_scores_q
25)

send_to_excel(mean_mse_q40, mean_mae_q40, '40', "Linear_regression",bs_lst_q40,
              mse_ci_q40, mae_ci_q40, brier_me_q40, mse_scores_q40, mae_scores_q
40)

send_to_excel(mean_mse_q50, mean_mae_q50, '50', "Linear_regression",bs_lst_q50,
              mse_ci_q50, mae_ci_q50, brier_me_q50, mse_scores_q50, mae_scores_q
50)

send_to_excel(mean_mse_q60, mean_mae_q60, '60', "Linear_regression",bs_lst_q60,
              mse_ci_q60, mae_ci_q60, brier_me_q60, mse_scores_q60, mae_scores_q
60)

send_to_excel(mean_mse_q75, mean_mae_q75, '75', "Linear_regression",bs_lst_q75,
              mse_ci_q75, mae_ci_q75, brier_me_q75, mse_scores_q75, mae_scores_q
75)

send_to_excel(mean_mse_q85, mean_mae_q85, '85', "Linear_regression",bs_lst_q85,
              mse_ci_q85, mae_ci_q85, brier_me_q85, mse_scores_q85, mae_scores_q
85)

send_to_excel(mean_mse_q100, mean_mae_q100, '100', 'Linear_regression', bs_lst_q
100,
              mse_ci_q100, mae_ci_q100, brier_me_q100, mse_scores_q100, mae_scor
es_q100)
```

In [ ]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

def poly_reg(X,y, d):
    np.random.seed(0)
    output = y[['Time Of Death']].to_numpy()
    X = X.to_numpy()
    y = y.to_numpy()
    num_folds = len(X)
    y_pred = []
    indices = range(len(X))
    classifier = LinearRegression()
    mse_scores = []
    mae_scores = []
    bs_lst = []

    k_fold = KFold(num_folds,shuffle=False, random_state=0)

    for train_indices, val_indices in k_fold.split(indices):

        polynomial_features = PolynomialFeatures(degree=d,
                                                 include_bias=False)
        pipeline = Pipeline([("polynomial_features", polynomial_features),
                             ("linear_regression", classifier)])

        pipeline.fit(X[train_indices], output[train_indices])
        predicted_val = pipeline.predict(X[val_indices])
        y_pred.append(predicted_val[0,0])

        mse = mean_squared_error(output[val_indices], predicted_val)
        mae = mean_absolute_error(output[val_indices], predicted_val)
        mse_scores.append(mse)
        mae_scores.append(mae)
        quants = [0.25, 0.5, 0.75, 1]
        sigma = math.sqrt(mse)
        actual_surv_t = output[val_indices][0,0]
        quantiles = np.quantile(event_times, quants)
        bs = []
        for quantile in quantiles:
            norm = statistics.NormalDist(predicted_val[0,0], sigma)
            cdf = norm.cdf(quantile)
            array_prob = [1-cdf]
            bs_ = compute_brier(quantile, actual_surv_t,array_prob)
            bs.append(bs_)
        bs_lst.append(bs)
    mean_mse = np.mean(mse_scores)
    mean_mae = np.mean(mae_scores)
    return mean_mse, mean_mae, y_pred, mse_scores, mae_scores, bs_lst

mean_mse_q15, mean_mae_q15, y_pred_q15, mse_scores_q15, mae_scores_q15, bs_lst_1
5 = poly_reg(X_15,df_ToD,2)
brier_me_q15 = findCI(bs_lst_q15, 'Brier')
mse_ci_q15 = findCI(mse_scores_q15, 'MSE')
mae_ci_q15 = findCI(mae_scores_q15, 'MAE')
print_plot(y_pred_q15, "15", " using ", "PolyReg",2)
send_to_excel(mean_mse_q15, mean_mae_q15,  '15', 'Polynomial_regression_2', bs_l
st_q15,
```

```
                mse_ci_q15, mae_ci_q15, brier_me_q15, mse_scores_q15, mae_scores_q
15, degree="2")
```

In [ ]:

In [ ]:
```
mean_mse_q15, mean_mae_q15, y_pred_q15, mse_scores_q15, mae_scores_q15, bs_lst_1
5 = poly_reg(X_15,df_ToD,3)
brier_me_q15 = findCI(bs_lst_q15, 'Brier')
mse_ci_q15 = findCI(mse_scores_q15, 'MSE')
mae_ci_q15 = findCI(mae_scores_q15, 'MAE')
print_plot(y_pred_q15, "15", " using ", "PolyReg",3)
send_to_excel(mean_mse_q15, mean_mae_q15,  '15', 'Polynomial_regression_3', bs_l
st_q15,
                mse_ci_q15, mae_ci_q15, brier_me_q15, mse_scores_q15, mae_scores_q
15, degree="3")
```

In [ ]:
```
mean_mse_q25, mean_mae_q25, y_pred_q25, mse_scores_q25, mae_scores_q25, bs_lst_2
5 = poly_reg(X_25,df_ToD,2)
brier_me_q25 = findCI(bs_lst_q25, 'Brier')
mse_ci_q25 = findCI(mse_scores_q25, 'MSE')
mae_ci_q25 = findCI(mae_scores_q25, 'MAE')
print_plot(y_pred_q25, "25", " using ", "PolyReg",2)
send_to_excel(mean_mse_q25, mean_mae_q25,  '25', 'Polynomial_regression_2', bs_l
st_q25,
                mse_ci_q25, mae_ci_q25, brier_me_q25, mse_scores_q25, mae_scores_q
25, degree="2")
```

In [ ]:
```
mean_mse_q25, mean_mae_q25, y_pred_q25, mse_scores_q25, mae_scores_q25, bs_lst_2
5 = poly_reg(X_25,df_ToD,3)
brier_me_q25 = findCI(bs_lst_q25, 'Brier')
mse_ci_q25 = findCI(mse_scores_q25, 'MSE')
mae_ci_q25 = findCI(mae_scores_q25, 'MAE')
print_plot(y_pred_q25, "25", " using ", "PolyReg",3)
send_to_excel(mean_mse_q25, mean_mae_q25,  '25', 'Polynomial_regression_3', bs_l
st_q25,
                mse_ci_q25, mae_ci_q25, brier_me_q25, mse_scores_q25, mae_scores_q
25, degree="3")
```

In [ ]:
```
mean_mse_q40, mean_mae_q40, y_pred_q40, mse_scores_q40, mae_scores_q40, bs_lst_4
0 = poly_reg(X_40,df_ToD,2)
brier_me_q40 = findCI(bs_lst_q40, 'Brier')
mse_ci_q40 = findCI(mse_scores_q40, 'MSE')
mae_ci_q40 = findCI(mae_scores_q40, 'MAE')
print_plot(y_pred_q40, "40", " using ", "PolyReg",2)
send_to_excel(mean_mse_q40, mean_mae_q40,  '40', 'Polynomial_regression_2', bs_l
st_q40,
                mse_ci_q40, mae_ci_q40, brier_me_q40, mse_scores_q40, mae_scores_q
40, degree="2")
```

In [ ]:
```
mean_mse_q40, mean_mae_q40, y_pred_q40, mse_scores_q40, mae_scores_q40, bs_lst_4
0 = poly_reg(X_40,df_ToD,3)
brier_me_q40 = findCI(bs_lst_q40, 'Brier')
mse_ci_q40 = findCI(mse_scores_q40, 'MSE')
mae_ci_q40 = findCI(mae_scores_q40, 'MAE')
print_plot(y_pred_q40, "40", " using ", "PolyReg",3)
send_to_excel(mean_mse_q40, mean_mae_q40,  '40', 'Polynomial_regression_3', bs_l
st_q40,
              mse_ci_q40, mae_ci_q40, brier_me_q40, mse_scores_q40, mae_scores_q
40, degree="3")
```

In [ ]:
```
mean_mse_q50, mean_mae_q50, y_pred_q50, mse_scores_q50, mae_scores_q50, bs_lst_q
50 = poly_reg(X_50,df_ToD,2)
brier_me_q50 = findCI(bs_lst_q50, 'Brier')
mse_ci_q50 = findCI(mse_scores_q50, 'MSE')
mae_ci_q50 = findCI(mae_scores_q50, 'MAE')
print_plot(y_pred_q50, "50", " using ", "PolyReg",2)
send_to_excel(mean_mse_q50, mean_mae_q50,  '50', 'Polynomial_regression_2', bs_l
st_q50,
              mse_ci_q50, mae_ci_q50, brier_me_q50, mse_scores_q50, mae_scores_q
50, degree="2")
```

In [ ]:
```
mean_mse_q50, mean_mae_q50, y_pred_q50, mse_scores_q50, mae_scores_q50, bs_lst_q
50 = poly_reg(X_50,df_ToD,3)
brier_me_q50 = findCI(bs_lst_q50, 'Brier')
mse_ci_q50 = findCI(mse_scores_q50, 'MSE')
mae_ci_q50 = findCI(mae_scores_q50, 'MAE')
print_plot(y_pred_q50, "50", " using ", "PolyReg",3)
send_to_excel(mean_mse_q50, mean_mae_q50,  '50', 'Polynomial_regression_3', bs_l
st_q50,
              mse_ci_q50, mae_ci_q50, brier_me_q50, mse_scores_q50, mae_scores_q
50, degree="3")
```

In [ ]:
```
mean_mse_q60, mean_mae_q60, y_pred_q60, mse_scores_q60, mae_scores_q60, bs_lst_q
60 = poly_reg(X_60,df_ToD,2)
brier_me_q60 = findCI(bs_lst_q60, 'Brier')
mse_ci_q60 = findCI(mse_scores_q60, 'MSE')
mae_ci_q60 = findCI(mae_scores_q60, 'MAE')
print_plot(y_pred_q60, "60", " using ", "PolyReg",2)
send_to_excel(mean_mse_q60, mean_mae_q60,  '60', 'Polynomial_regression_2', bs_l
st_q60,
              mse_ci_q60, mae_ci_q60, brier_me_q60, mse_scores_q60, mae_scores_q
60, degree="2")
```

In [ ]:
```
mean_mse_q60, mean_mae_q60, y_pred_q60, mse_scores_q60, mae_scores_q60, bs_lst_q
60 = poly_reg(X_60,df_ToD,3)
brier_me_q60 = findCI(bs_lst_q60, 'Brier')
mse_ci_q60 = findCI(mse_scores_q60, 'MSE')
mae_ci_q60 = findCI(mae_scores_q60, 'MAE')
print_plot(y_pred_q60, "60", " using ", "PolyReg",3)
send_to_excel(mean_mse_q60, mean_mae_q60,  '60', 'Polynomial_regression_3', bs_l
st_q60,
              mse_ci_q60, mae_ci_q60, brier_me_q60, mse_scores_q60, mae_scores_q
60, degree="3")
```

In [ ]:
```
mean_mse_q75, mean_mae_q75, y_pred_q75, mse_scores_q75, mae_scores_q75, bs_lst_q
75 = poly_reg(X_75,df_ToD,2)
brier_me_q75 = findCI(bs_lst_q75, 'Brier')
mse_ci_q75 = findCI(mse_scores_q75, 'MSE')
mae_ci_q75 = findCI(mae_scores_q75, 'MAE')
print_plot(y_pred_q75, "75", " using ", "PolyReg",2)
send_to_excel(mean_mse_q75, mean_mae_q75,  '75', 'Polynomial_regression_2', bs_l
st_q75,
              mse_ci_q75, mae_ci_q75, brier_me_q75, mse_scores_q75, mae_scores_q
75, degree="2")
```

In [ ]:
```
mean_mse_q75, mean_mae_q75, y_pred_q75, mse_scores_q75, mae_scores_q75, bs_lst_q
75 = poly_reg(X_75,df_ToD,3)
brier_me_q75 = findCI(bs_lst_q75, 'Brier')
mse_ci_q75 = findCI(mse_scores_q75, 'MSE')
mae_ci_q75 = findCI(mae_scores_q75, 'MAE')
print_plot(y_pred_q75, "75", " using ", "PolyReg",3)
send_to_excel(mean_mse_q75, mean_mae_q75,  '75', 'Polynomial_regression_3', bs_l
st_q75,
              mse_ci_q75, mae_ci_q75, brier_me_q75, mse_scores_q75, mae_scores_q
75, degree="3")
```

In [ ]:
```
mean_mse_q85, mean_mae_q85, y_pred_q85, mse_scores_q85, mae_scores_q85, bs_lst_q
85 = poly_reg(X_85,df_ToD,2)
brier_me_q85 = findCI(bs_lst_q85, 'Brier')
mse_ci_q85 = findCI(mse_scores_q85, 'MSE')
mae_ci_q85 = findCI(mae_scores_q85, 'MAE')
print_plot(y_pred_q85, "85", " using ", "PolyReg",2)
send_to_excel(mean_mse_q85, mean_mae_q85,  '85', 'Polynomial_regression_2', bs_l
st_q85,
              mse_ci_q85, mae_ci_q85, brier_me_q85, mse_scores_q85, mae_scores_q
85, degree="2")
```

In [ ]:
```
mean_mse_q85, mean_mae_q85, y_pred_q85, mse_scores_q85, mae_scores_q85, bs_lst_q
85 = poly_reg(X_85,df_ToD,3)
brier_me_q85 = findCI(bs_lst_q85, 'Brier')
mse_ci_q85 = findCI(mse_scores_q85, 'MSE')
mae_ci_q85 = findCI(mae_scores_q85, 'MAE')
print_plot(y_pred_q85, "85", " using ", "PolyReg",3)
send_to_excel(mean_mse_q85, mean_mae_q85,  '85', 'Polynomial_regression_3', bs_l
st_q85,
              mse_ci_q85, mae_ci_q85, brier_me_q85, mse_scores_q85, mae_scores_q
85, degree="3")
```

In [ ]:
```
mean_mse_q100, mean_mae_q100, y_pred_q100, mse_scores_q100, mae_scores_q100, bs_
lst_q100 = poly_reg(X_100,df_ToD,2)
brier_me_q100 = findCI(bs_lst_q100, 'Brier')
mse_ci_q100 = findCI(mse_scores_q100, 'MSE')
mae_ci_q100 = findCI(mae_scores_q100, 'MAE')
print_plot(y_pred_q100, "100", " using ", "PolyReg",2)
send_to_excel(mean_mse_q100, mean_mae_q100,  '100', 'Polynomial_regression_2', b
s_lst_q100,
              mse_ci_q100, mae_ci_q100, brier_me_q100, mse_scores_q100, mae_scor
es_q100, degree="2")
```

In [ ]:
```
mean_mse_q100, mean_mae_q100, y_pred_q100, mse_scores_q100, mae_scores_q100, bs_
lst_q100 = poly_reg(X_100,df_ToD,3)
brier_me_q100 = findCI(bs_lst_q100, 'Brier')
mse_ci_q100 = findCI(mse_scores_q100, 'MSE')
mae_ci_q100 = findCI(mae_scores_q100, 'MAE')
print_plot(y_pred_q100, "100", " using ", "PolyReg",3)
send_to_excel(mean_mse_q100, mean_mae_q100,  '100', 'Polynomial_regression_3', b
s_lst_q100,
              mse_ci_q100, mae_ci_q100, brier_me_q100, mse_scores_q100, mae_scor
es_q100, degree="3")
```

In [ ]:

In [ ]:

Model Results Graphing

Authors: Laura Simandl, Molly Strasser

Course: 16791 Applied Data Science

The following code graphs the results from battery prognostic modelling.

In [ ]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
linear_reg_df = pd.read_csv('/Users/Molly/Documents/Battery/Averages/linear_regression_zero_avg.csv')
linear_reg_wo_soc_df = pd.read_csv('/Users/Molly/Documents/Battery/Averages/linear_regression_without_SOC_one_avg.csv')
rsf_df = pd.read_csv('/Users/Molly/Documents/Battery/Averages/rsf_w_SOC.csv')
rsf_wo_soc_df = pd.read_csv('/Users/Molly/Documents/Battery/Averages/rsf_wo_SOC.csv')
poly2_df = pd.read_csv('/Users/Molly/Documents/Battery/Averages/Polynomial_regression_2_avg.csv')
poly2_wo_soc_df = pd.read_csv('/Users/Molly/Documents/Battery/Averages/Polynomial_regression_without_SOC_2_avg.csv')
poly3_df = pd.read_csv('/Users/Molly/Documents/Battery/Averages/Polynomial_regression_3_avg.csv')
poly3_wo_soc_df = pd.read_csv('/Users/Molly/Documents/Battery/Averages/Polynomial_regression_without_SOC_3_avg.csv')
waft_df = pd.read_csv('/Users/Molly/Documents/Battery/Averages/waft_w_SOC.csv')
waft_wo_soc_df = pd.read_csv('/Users/Molly/Documents/Battery/Averages/waft_wo_SOC.csv')
```

In [ ]:
```
df1 = linear_reg_df[linear_reg_df['Time Horizon'] != 15]
df2 = linear_reg_wo_soc_df[linear_reg_wo_soc_df['Time Horizon'] != 15]
df3 = rsf_df[rsf_df['Time Horizon'] != 15]
df4 = rsf_wo_soc_df[rsf_wo_soc_df['Time Horizon'] != 15]
df5 = poly2_df[poly2_df['Time Horizon'] != 15]
df6 = poly2_wo_soc_df[poly2_wo_soc_df['Time Horizon'] != 15]
df7 = poly3_df[poly3_df['Time Horizon'] != 15]
df8 = poly3_wo_soc_df[poly3_wo_soc_df['Time Horizon'] != 15]
df9 = waft_df[waft_df['Time Horizon'] != 15]
df10 = waft_wo_soc_df[waft_wo_soc_df['Time Horizon'] != 15]
```

In [ ]:
```
def print_plot(lin_reg_df, rsf_df, poly2_df, poly3_df, waft_df, col, soc=True):
    x = rsf_df['Time Horizon']
    if (col == 'MSE_median'):
        col_rsf = rsf_df[col]
        col_waft = waft_df[col]
        col_lr = lin_reg_df['MSE_mean']
        col_p2 = poly2_df['MSE_mean']
        col_p3 = poly3_df['MSE_mean']
    if (col == 'MAE_median'):
        col_rsf = rsf_df[col]
        col_waft = waft_df[col]
        col_lr = lin_reg_df['MAE_mean']
        col_p2 = poly2_df['MAE_mean']
        col_p3 = poly3_df['MAE_mean']
    elif (col != 'MSE_median' and col != 'MAE_median'):
        col_rsf = rsf_df[col]
        col_lr = lin_reg_df[col]
        col_p2 = poly2_df[col]
        col_p3 = poly3_df[col]
        col_waft = waft_df[col]
    fig, ax = plt.subplots()
    rsf, = ax.plot(x, col_rsf, marker = 'o',color = 'skyblue', label='Random Sur
vival Forest')
    lr, = ax.plot(x, col_lr, marker = 'o', color = 'olive', label ='Linear Regre
ssion')
    poly2, = ax.plot(x, col_p2, marker = 'o', color = 'orange', label ='Polynomi
al degree 2 Regression')
    poly3, = ax.plot(x, col_p3, marker = 'o', color = 'purple', label ='Polynomi
al degree 3 Regression')
    waft, = ax.plot(x, col_waft, marker = 'o', color = 'darkblue', label ='Weibu
ll AFT')
    ax.legend(handles=[rsf, lr, poly2, poly3,waft])
    ax.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
    if (col == 'BS 25'):
        title = "Brier Score at 25% Quantile versus Time Horizon"
    if (col == 'BS 50'):
        title = "Brier Score at 50% Quantile versus Time Horizon"
    if (col == 'BS 75'):
        title = "Brier Score at 75% Quantile versus Time Horizon"
    if (col == 'BS 100'):
        title = "Brier Score at 100% Quantile versus Time Horizon"
    if (col == 'MSE_mean'):
        title = "Average Mean Squared Error versus Time Horizon"
        ax.set(ylabel='Average Mean Squared Error')
    if (col == 'MAE_mean'):
        title = "Average Mean Absolute Error versus Time Horizon"
        ax.set(ylabel='Average Mean Absolute Error')
    if (col == 'MSE_median'):
        title = "Average Mean Squared Error about median versus Time Horizon"
        ax.set(ylabel='Average Mean Squared Error')
    if (col == 'MAE_median'):
        title = "Average Mean Absolute Error about median versus Time Horizon"
    if (soc):
        title = title + " with SOC"
        ax.set_title(title)
        ax.set(xlabel='Time Horizon (%)')
        ax.figure.savefig('/Users/Molly/Documents/Battery/Plots/withSOC/'+ title
, bbox_inches='tight')
```

```
    else:
        title = title + " without SOC"
        ax.set_title(title)
        ax.set(xlabel='Time Horizon (%)')
        ax.figure.savefig('/Users/Molly/Documents/Battery/Plots/withoutSOC/'+ ti
tle, bbox_inches='tight')
```

In [ ]:
```
columns_graph = ['MSE_mean', 'MAE_mean', 'MAE_median','MSE_median','BS 25', 'BS
 50', 'BS 75', 'BS 100']
for col in columns_graph:
    print_plot(df1, df3, df5, df7, df9, col)
for col in columns_graph:
    print_plot(df2, df4, df6, df8, df10, col, False)
```

In [ ]:
```
import glob

df_15=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withoutSOC/15/*.csv'):
    tmp_df = pd.read_csv(f)
    df_15 = pd.concat([df_15,tmp_df], axis=1)

df_25=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withoutSOC/25/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_25 = pd.concat([df_25,tmp_df], axis=1)

df_40=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withoutSOC/40/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_40 = pd.concat([df_40,tmp_df], axis=1)

df_50=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withoutSOC/50/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_50 = pd.concat([df_50,tmp_df], axis=1)

df_60=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withoutSOC/60/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_60 = pd.concat([df_60,tmp_df], axis=1)

df_75=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withoutSOC/75/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_75 = pd.concat([df_75,tmp_df], axis=1)

df_85 =pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withoutSOC/85/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_85 = pd.concat([df_85,tmp_df], axis=1)

df_100=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withoutSOC/100/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_100 = pd.concat([df_100,tmp_df], axis=1)

dfs = [df_15, df_25, df_40, df_40, df_50, df_60, df_75, df_85, df_100]
```

In [ ]:
```
time_horiz = ['15%', '25%', '40%', '50%', '60%', '75%', '85%', '100%']
columns_graph = ['MSE about mean', 'MAE about mean', 'MAE about median', 'MAE ab
out median', '25% Brier Score','50% Brier Score', '75% Brier Score', '100% Brier
Score']
for i in range(len(time_horiz)):
    for key_word in columns_graph:
        fig, ax = plt.subplots()
        df = dfs[i]
        time = time_horiz[i]
        df_col = [col for col in df.columns if key_word in col]
        tmp_df = df[df_col]
        ax = sns.boxplot(data=tmp_df, palette = 'pastel')
        if (key_word == 'MSE about mean'):
            ax.set_ylim([-5000, 250000])
        ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
        title = time + ' time horizon ' + key_word + ' without SOC'
        ax.set_title(title)
        ax.figure.savefig('/Users/Molly/Documents/Battery/BoxPlots/withoutSOC/'+
title, bbox_inches='tight')
```

In [ ]:
```
import glob

df_15=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withSOC/15/*.csv'):
    tmp_df = pd.read_csv(f)
    df_15 = pd.concat([df_15,tmp_df], axis=1)

df_25=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withSOC/25/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_25 = pd.concat([df_25,tmp_df], axis=1)

df_40=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withSOC/40/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_40 = pd.concat([df_40,tmp_df], axis=1)

df_50=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withSOC/50/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_50 = pd.concat([df_50,tmp_df], axis=1)

df_60=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withSOC/60/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_60 = pd.concat([df_60,tmp_df], axis=1)

df_75=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withSOC/75/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_75 = pd.concat([df_75,tmp_df], axis=1)

df_85 =pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withSOC/85/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_85 = pd.concat([df_85,tmp_df], axis=1)

df_100=pd.DataFrame()
for f in glob.glob('/Users/Molly/Documents/Battery/All/withSOC/100/*.csv'):
    tmp_df = pd.read_csv(f, index_col=False)
    df_100 = pd.concat([df_100,tmp_df], axis=1)

dfs = [df_15, df_25, df_40, df_40, df_50, df_60, df_75, df_85, df_100]
```

In [ ]:
```
time_horiz = ['15%', '25%', '40%', '50%', '60%', '75%', '85%', '100%']
columns_graph = ['MSE about mean', 'MAE about mean', 'MAE about median', 'MAE ab
out median', '25% Brier Score','50% Brier Score', '75% Brier Score', '100% Brier
Score']

for i in range(len(time_horiz)):
    for key_word in columns_graph:
        fig, ax = plt.subplots()
        df = dfs[i]
        time = time_horiz[i]
        df_col = [col for col in df.columns if key_word in col]
        tmp_df = df[df_col]
        ax = sns.boxplot(data=tmp_df, palette = 'pastel')
        if (key_word == 'MSE about mean'):
            ax.set_ylim([-5000, 250000])
        ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
        title = time + ' time horizon ' + key_word + ' with SOC'
        ax.set_title(title)
        ax.figure.savefig('/Users/Molly/Documents/Battery/BoxPlots/withSOC/'+ ti
tle, bbox_inches='tight')
```

In [ ]:
```
mse_best_df = pd.read_csv('/Users/Molly/Documents/Battery/All/Min_MSE_Best_Perfo
rmance.csv')
#print(mse_best_df)
fig, ax = plt.subplots()
df = mse_best_df
ax = sns.boxplot(data=df, palette = 'pastel')
ax.set_ylim([-5000, 80000])
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
title =  'Best Model and Time Horizon for MSE with and without SOC'
ax.set_title(title)
ax.figure.savefig('/Users/Molly/Documents/Battery/BoxPlots/Best/'+ title, bbox_i
nches='tight')
```

In [ ]:
```
mse_best_df = pd.read_csv('/Users/Molly/Documents/Battery/All/Best_Model_MAE.cs
v')
fig, ax = plt.subplots()
df = mse_best_df
ax = sns.boxplot(data=df, palette = 'pastel')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
title =  'Best Model and Time Horizon for MAE with and without SOC'
ax.set_title(title)
ax.figure.savefig('/Users/Molly/Documents/Battery/BoxPlots/'+ title, bbox_inches
='tight')
```

In [ ]:
```
mse_best_df = pd.read_csv('/Users/Molly/Documents/Battery/All/best_model_bs25.csv')
fig, ax = plt.subplots()
df = mse_best_df
ax = sns.boxplot(data=df, palette = 'pastel')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
title =  'Best Model and Time Horizon for Brier Score at 25 Quantile with and without SOC'
ax.set_title(title)
ax.figure.savefig('/Users/Molly/Documents/Battery/BoxPlots/'+ title, bbox_inches='tight')
```

In [ ]:
```
mse_best_df = pd.read_csv('/Users/Molly/Documents/Battery/All/best_model_bs50.csv')
fig, ax = plt.subplots()
df = mse_best_df
ax = sns.boxplot(data=df, palette = 'pastel')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
title =  'Best Model and Time Horizon for Brier Score at 50 Quantile with and without SOC'
ax.set_title(title)
ax.figure.savefig('/Users/Molly/Documents/Battery/BoxPlots/'+ title, bbox_inches='tight')
```

In [ ]:
```
mse_best_df = pd.read_csv('/Users/Molly/Documents/Battery/All/best_model_bs75.csv')
fig, ax = plt.subplots()
df = mse_best_df
ax = sns.boxplot(data=df, palette = 'pastel')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
title =  'Best Model and Time Horizon for Brier Score at 75 Quantile with and without SOC'
ax.set_title(title)
ax.figure.savefig('/Users/Molly/Documents/Battery/BoxPlots/'+ title, bbox_inches='tight')
```

In [ ]:
```
mse_best_df = pd.read_csv('/Users/Molly/Documents/Battery/All/best_model_bs100.csv')
fig, ax = plt.subplots()
df = mse_best_df
ax = sns.boxplot(data=df, palette = 'pastel')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
title =  'Best Model and Time Horizon for Brier Score at 100 Quantile with and without SOC'
ax.set_title(title)
ax.figure.savefig('/Users/Molly/Documents/Battery/BoxPlots/'+ title, bbox_inches='tight')
```

In [ ]:
```

```