NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

PREDICTIVE MAINTENANCE USING MACHINE LEARNING AND EXISTING DATA SOURCES

by

William J. Frazier

September 2022

Thesis Advisor: Neil C. Rowe
Co-Advisor: Ying Zhao

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC, 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 2022	3. REPORT TYPE AND DATES COVERED Master's thesis

4. TITLE AND SUBTITLE PREDICTIVE MAINTENANCE USING MACHINE LEARNING AND EXISTING DATA SOURCES	5. FUNDING NUMBERS
6. AUTHOR(S) William J. Frazier	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	12b. DISTRIBUTION CODE A

13. ABSTRACT (maximum 200 words)

The United States Marine Corps must address material-readiness challenges with emerging technologies at minimum cost. Predictive maintenance using machine learning is a growing field that can be applied using free or commercial-off-the-shelf software. Naval aviation organizations already maintain a network of data repositories that collect and store current and historical data on repairable flight-critical components. Many components fail before their expected structural life as published their manufacturers, which results in costly unscheduled maintenance. The ability to predict component failures and plan for their replacement or repair can significantly increase operational readiness. This thesis develops and analyzes machine-learning models to predict the conditional probability of failure of various MV-22B flight-critical components using data from existing Naval aviation repositories. Data preprocessing, model training, and predictions use commercial-off-the-shelf software. This work can help improve material readiness and acclimatize military-aviation personnel to emerging technologies in decision making.

14. SUBJECT TERMS machine learning, predictive maintenance, conditional probability of failure, naval aviation	15. NUMBER OF PAGES 159
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

PREDICTIVE MAINTENANCE USING MACHINE LEARNING AND EXISTING DATA SOURCES

William J. Frazier
Captain, United States Marine Corps
BS, United States Naval Academy, 2013

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2022

Approved by: Neil C. Rowe
Advisor

Ying Zhao
Co-Advisor

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The United States Marine Corps must address material-readiness challenges with emerging technologies at minimum cost. Predictive maintenance using machine learning is a growing field that can be applied using free or commercial-off-the-shelf software. Naval aviation organizations already maintain a network of data repositories that collect and store current and historical data on repairable flight-critical components. Many components fail before their expected structural life as published their manufacturers, which results in costly unscheduled maintenance. The ability to predict component failures and plan for their replacement or repair can significantly increase operational readiness. This thesis develops and analyzes machine-learning models to predict the conditional probability of failure of various MV-22B flight-critical components using data from existing Naval aviation repositories. Data preprocessing, model training, and predictions use commercial-off-the-shelf software. This work can help improve material readiness and acclimatize military-aviation personnel to emerging technologies in decision making.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION....1
- A. BACKGROUND ON MAINTENANCE....1
- B. RESEARCH QUESTIONS....5
- C. SUMMARY....5

II. MACHINE LEARNING AND FAILURE ANALYSIS....7
- A. POSSIBLE MACHINE-LEARNING METHODS FOR PREDICTIVE ANALYSIS....8
  - 1. Linear Regression....8
  - 2. Artificial Neural Networks....9
- B. SURVIVAL ANALYSIS....13

III. FAILURES OF MILITARY AIRCRAFT COMPONENTS....17
- A. AIRCRAFT AND COMPONENT USE RATES....17
- B. NAVAL AVIATION DATA....20
  - 1. Existing Repositories....20
  - 2. Data Integrity....21
- C. CONDITION BASED MAINTENANCE PLUS....22
- D. RELIABILITY ANALYSIS....23
  - 1. Censored Data....23
  - 2. Applying the Weibull Distribution....23
  - 3. CPH....24
  - 4. Artificial Neural Networks....24

IV. METHODOLOGY....27
- A. RESEARCH AND DESIGN STRATEGY....27
- B. SCOPE AND LIMITATIONS....27
- C. DATA GENERATION....28
  - 1. Component Queries....30
  - 2. Component Imputations and Assumptions....33
  - 3. Flight Data....35
- D. TRAINING AND MODEL SELECTION....37
- E. FINAL TESTING....39

V. RESULTS AND ANALYSIS....41
- A. WEIBULL ANALYSIS COMPARISON....41
- B. WEIBULL MODELS WITH DECKPLATE DATA....44

C. CPH MODELS 45

VI. CONCLUSIONS AND FUTURE WORK 49
A. CONCLUSIONS 49
B. FUTURE WORK 50

APPENDIX A. WEIBULL MODEL COMPARISONS 51

APPENDIX B. WEIBULL AND CPH PERFORMANCE 59

APPENDIX C. COX PROPORTIONAL HAZARD PLOTS 75

APPENDIX D. MATLAB SOURCE CODE FOR REMAINING USEFUL LIFE PREDICTIONS USING LSTM NETWORK 81

APPENDIX E. MATLAB SOURCE CODE FOR REMAINING USEFUL LIFE PREDICTIONS USING CNN 87

APPENDIX F. PYTHON SOURCE CODE FOR GENERATING FLIGHT HOUR DATA 97

APPENDIX G. PYTHON SOURCE CODE FOR GENERATING SERIAL NUMBER HISTORIES 105

APPENDIX H. PYTHON SOURCE CODE FOR GENERATING SERIAL NUMBER REMOVAL DATA 117

APPENDIX I. PYTHON SOURCE CODE FOR WEIBULL AND CPH MODELS 123

LIST OF REFERENCES 133

INITIAL DISTRIBUTION LIST 137

LIST OF FIGURES

Figure 1. Scheduled versus Unscheduled Maintenance Man Hours. Source: NAVAIR Readiness Analysis Reports (2022).....2

Figure 2. Weibull Model for Pylon Conversion Actuators. Source: FRC East V22 FST Maintenance Optimization (2022). 4

Figure 3. Forecasted PCA Failure Rates in Flight Hours. Source: FRC East V22 FST Maintenance Optimization (2022). 4

Figure 4. NASA Turbofan Jet Dataset 8

Figure 5. NASA Engine Time Series Input Vectors for LSTM Training 10

Figure 6. LSTM RUL Predictions for NASA Engines 11

Figure 7. Deep CNN Architecture from Matlab Network Analyzer 12

Figure 8. Deep CNN Training for RUL in Matlab.....13

Figure 9. Weibull PDF with Various Shape Parameters, β. Source: McCool (2012, p. 75).....14

Figure 10. Proposed Approach of Failure Prediction using Weibull Analysis 15

Figure 11. MV-22B Use Rates from March 2021 to February 2022. Source: NAVAIR Vector Aircraft Readiness Dashboard (2022). 18

Figure 12. Histogram of MV-22B Use Rates from 2021 to 2022. Adapted from NAVAIR DECKPLATE Query of Flight Records (2022).....18

Figure 13. Histogram of MV-22B Use Rates from 2017 to 2022. Adapted from NAVAIR DECKPLATE Query of Flight Records (2022).....19

Figure 14. MV-22B PCA Flight Data Generation Flow 29

Figure 15. AME Query for MV-22B 275020 and 275021 WUCs. Source: Report from DECKPLATE AME Query Studio (2022).....31

Figure 16. Serial Number 161 PCA Removals and Installations. Adapted from Report run in DECKPLATE AME Query with SernoHistory.py (2022).....32

Figure 17. Serial Number 161 PCA Removals and Installations. Adapted from Report run in DECKPLATE DP-0025 Report with SernoHistory.py (2022). 33

Figure 18. Histogram of PCA Removals by Malfunction Code 34

Figure 19. Histogram of Filtered PCA Removals by Malfunction Code 34

Figure 20. Flight History of BuNo 168217 from 4/1/2018 to 4/13/2018. Adapted from DECKPLATE NAVFLIR records with FlightHours.py (2022)/ 37

Figure 21. PCA Serial Number 77 Removal Data from RemovalHistory.py 37

Figure 22. V22 FST Weibull PDF of PCA Mode 15: Cross-Over Jam. Source: FRC East V22 FST Maintenance Optimization (2022). 43

Figure 23. Thesis Weibull PDF of PCA Mode 15: Cross-Over Jam. Source: Models.py Python Script (2022). 43

Figure 24. Scaled Schoenfeld Residuals of Ship Landings for Malfunction Code 20 - Worn, Stripped, Chaffed, or Frayed – Not Wiring. Source: Models.py Python Script (2022). 46

Figure 25. CPH Predicted Survival for Malfunction Code 70 Test Data. Source: Models.py Python Script (2022). 47

Figure 26. Weibull PDF of PCA Mode 1: Internal Wear Before Scheduled Greasing. Source: FRC East V22 FST Maintenance Optimization (2022). 51

Figure 27. Weibull PDF of PCA Mode 1: Internal Wear Before Scheduled Greasing. Source: Models.py Python Script (2022). 51

Figure 28. V22 FST Weibull PDF of PCA Mode 2: Soft Stop. Source: FRC East V22 FST Maintenance Optimization (2022). 52

Figure 29. Thesis Weibull PDF of PCA Mode 2: Soft Stop. Source: Models.py Python Script (2022). 52

Figure 30. V22 FST Weibull PDF of PCA Mode 14: Internal Wear Post Scheduled Greasing. Source: FRC East V22 FST Maintenance Optimization (2022). 53

Figure 31. Thesis Weibull PDF of PCA Mode 14: Internal Wear Post Scheduled Greasing. Source: Models.py Python Script (2022). 53

Figure 32. V22 FST Weibull PDF of PCA Mode 15: Cross-Over Jam. Source: FRC East V22 FST Maintenance Optimization (2022). 54

Figure 33. Thesis Weibull PDF of PCA Mode 15: Cross-Over Jam. Source: Models.py Python Script (2022). 54

Figure 34. V22 FST Weibull PDF of PCA Mode 16: Insufficient Ballscrew. Source: FRC East V22 FST Maintenance Optimization (2022).....55

Figure 35. Thesis Weibull PDF of PCA Mode 16: Insufficient Ballscrew. Source: Models.py Python Script (2022). 55

Figure 36. V22 FST Weibull PDF of PCA Mode 16: Ratcheting. Source: FRC East V22 FST Maintenance Optimization (2022). 56

Figure 37. Thesis Weibull PDF of PCA Mode 16: Ratcheting. Source: Models.py Python Script (2022). 56

Figure 38. V22 FST Weibull PDF of PCA Mode 21: Seal Damage. Source: FRC East V22 FST Maintenance Optimization (2022). 57

Figure 39. Thesis Weibull PDF of PCA Mode 21: Seal Damage. Source: Models.py Python Script (2022). 57

Figure 40. Scaled Schoenfeld Residuals of Austere Landings for Malfunction Code 70 – Broken, Burst, Ruptured, Punctured, Torn, or Cut. Source: Models.py Python Script (2022). 75

Figure 41. Scaled Schoenfeld Residuals of TMR_7 Hours for Blank – Technical Directive Inspection Failure. Source: Models.py Python Script (2022). 75

Figure 42. CPH Predicted Survival for Malfunction Code 20 Test Data. Source: Models.py Python Script (2022). 76

Figure 43. CPH Predicted Survival for Malfunction Code 70 Test Data. Source: Models.py Python Script (2022). 76

Figure 44. CPH Predicted Survival for Malfunction Code 135 Test Data. Source: Models.py Python Script (2022). 77

Figure 45. CPH Predicted Survival for Malfunction Code 150 Test Data. Source: Models.py Python Script (2022). 77

Figure 46. CPH Predicted Survival for Malfunction Code 290 Test Data. Source: Models.py Python Script (2022). 78

Figure 47. CPH Predicted Survival for Malfunction Code 295 Test Data. Source: Models.py Python Script (2022). 78

Figure 48. CPH Predicted Survival for Malfunction Code 374 Test Data. Source: Models.py Python Script (2022). 79

Figure 49. CPH Predicted Survival for Failed Technical Directive Inspection Test Data. Source: Models.py Python Script (2022). 79

LIST OF TABLES

Table 1. NAVFLIR TMR General Purpose Codes 35

Table 2. NAVFLIR Landing Groupings 36

Table 3. Malfunction Codes Analyzed 38

Table 4. Training and Test Data Split by Malfunction Code.....38

Table 5. V22 Fleet Support Team Weibull Model Results.....42

Table 6. Best Performing Weibull PDF by Malfunction Code.....44

Table 7. Best Performing CPH Model by Malfunction Code.....45

Table 8. CPH Assumption Model Comparison 47

Table 9. Weibull and CPH Performance Comparison 48

Table 10. Weibull Model Performance by Failure Mode and Hyperparameter 59

Table 11. CPH Model Performance by Failure Mode and Hyperparameter 65

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ACC	Aircraft Controlling Custodian
AIC	Akaike Information Criteria
ALM	Aircraft Life Management
AME	Automated Maintenance Environment
AMSRR	Aviation Maintenance Supply Readiness Report
ANN	Artificial Neural Network
CBM	Conditions-based Maintenance
CBM+	Conditions-based Maintenance Plus
CDF	Cumulative Distribution Function
CFR	Code of Federal Regulations
CI	Confidence Interval
CNAF	Commander, Naval Air Forces
CNN	Convolutional Neural Network
COMFRC	Commander, Fleet Readiness Centers
CPH	Cox Proportional-Hazards
DECKPLATE	Decision Knowledge Programming for Logistics Analysis and Technical Evaluation
DOD	Department of Defense
FAA	Federal Aviation Administration
FCLP	Field Carrier Landing Practice
FDLP	Field Deck Landing Practice
FMC	Fully Mission-capable
FOUO	For Official Use Only
FRC	Fleet Readiness Center
FST	Fleet Support Team
GAO	Government Accountability Office
HPDU	Hydraulic Powered Drive Unit
JAIC	Joint Artificial Intelligence Center
LSTM	Long Short-term Memory
MAE	Mean Absolute Error

MAF	Maintenance Action Form
MDS	Maintenance Data System
MLE	Mean Likelihood Estimation
MMH	Maintenance Man-hours
MSE	Mean-squared Error
MTTF	Mean Time to Failure
MEU	Marine Expeditionary Unit
NAE	Naval Aviation Enterprise
NALCOMIS	Naval Aviation Logistics Command Management Information System
NAMP	Naval Aviation Maintenance Program
NASA	National Aeronautics and Space Administration
NAVAIR	Naval Air Systems Command
NAVFLIR	Naval Flight Record
NFO	Naval Flight Officer
NVD	Night Vision Device
OEM	Original Equipment Manufacturer
PCA	Pylon Conversion Actuator
PDF	Probability Density Function; Portable Document Format
PMIC	Periodic Maintenance Information Card
PRR	Proportional Reporting Ratio
RAST	Recovery Assist, Secure and Traverse
RBA	Ready Basic Aircraft
RCM	Reliability-centered Maintenance
RFI	Request for Information
R/I	Removal/Installation
RIP/TOA	Relief in Place/Transfer of Authority
RMSE	Root Mean-squared Error
RNN	Recurrent Neural Network
RRX	Rank Regression on x
RRY	Rank Regression on y
RUL	Remaining Useful Life

TEC	Type Equipment Code
TMR	Total Mission Requirement
TMS	Type Model Series
TTF	Time to Failure
TTMA	Time to Maintenance Action
V22	Bell Boeing V-22 Osprey
V/STOL	Vertical and/or Short Take-Off and Landing
WUC	Work Unit Code

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First, I would like to express my appreciation for the entire V-22 Fleet Support Team at Fleet Readiness Center East, Marine Corps Air Station Cherry Point for their hard work in reliability analysis. I would like to individually thank Mr. Ryan Wolcott and Mr. Gregory Clayson for providing a valuable objective for this thesis, their previous work, and their subject matter expertise. Next, I would like to thank the Marine Aircraft Group 26 V-22 Class Desk, Mr. Donald Lozano, and the Material Control Officer for Marine Light Attack Helicopter Training Squadron 303, Captain Patrick Whitehurst, United States Marine Corps. Their contributions towards gathering data were vital to the research. Lastly, I would like to thank my thesis advisors Dr. Neil Rowe and Dr. Ying Zhao for the valuable comments, professional oversight, and direction.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

This thesis examines the potential for machine-learning algorithms to improve reliability-centered maintenance (RCM) and conditions-based maintenance (CBM) to improve aircraft reliability in naval aviation.

A. BACKGROUND ON MAINTENANCE

The Commandant of the Marine Corps has said that much data collected and retained by the service is not exploited enough by emerging technologies (United States and Berger, 2019). Aircraft platforms in the Department of Defense (DOD) consistently struggle to meet annual readiness goals, despite the large budgets allocated to their programs (Crusher, 2020). Predictive maintenance using emerging technologies can use these large quantities of data and offer a cost-effective approach to improving aviation readiness. The Commandant also emphasized that solutions must use existing military data repositories due to the limited funding available. With over 40 billion records uploaded monthly, the Naval Air Systems Command (NAVAIR) data repository, Decision Knowledge Programming for Logistics Analysis and Technical Evaluation (DECKPLATE), could be a good source for machine-learning applications.

While significant efforts are being made in addressing the root causes of aircraft-readiness shortfalls, commanders require partial solutions now to accomplish their missions. For several years, commanders have resorted to high cannibalization (taking parts from one aircraft and putting them in another) rates and the transfer of fully mission-capable (FMC) aircraft from squadrons returning from deployment to those preparing for deployment. Although an "acceptable management choice only when necessary to meet operational objectives" (Department of the Navy, 2021), cannibalizations and squadron transfers have become the norm. Between 2011 and 2017, the Marine Corps transferred over 650 MV-22B Ospreys between squadrons to meet flight-hour and operational requirements due to the lack of ready basic aircraft (RBA) (Eckstein, 2017). Meanwhile, demands have increased. These temporary solutions hurt future readiness due to overuse

or underuse of individual aircraft and additional man-hours spent transferring and accepting aircraft.

Maintenance actions are categorized as scheduled or unscheduled (Susto et al., 2015). Scheduled maintenance is proactive and done before a component degrades or runs to failure. The frequency of scheduled maintenance on a component is usually based on the vendor or original equipment manufacturer (OEM) published structural life limits and recommended maintenance schedule. Unscheduled maintenance is done when a component degrades or fails. Figure 1 compares the number of maintenance man-hours (MMH) spent on scheduled versus unscheduled maintenance for the MV-22B aircraft in the Marine Corps. Between March 2021 and February 2022, unscheduled maintenance was five to six times more frequent than scheduled maintenance. This ratio indicates a significant unreliability of aircraft components as well as a difficulty in predicting unscheduled maintenance.

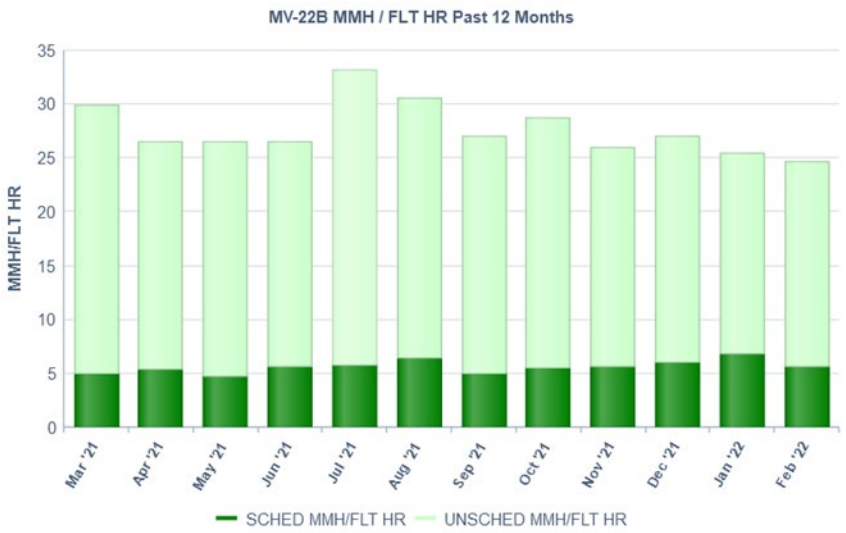

Figure 1. Scheduled versus Unscheduled Maintenance Man Hours.
Source: NAVAIR Readiness Analysis Reports (2022).

Scheduled maintenance is preventative or designed to continuously inspect and maintain components so they reach their service life. For a "type model series" (TMS) in

United States naval aviation, component inspection and removal schedules are published in the associated Periodic Maintenance Information Cards (PMIC) of the Inspection Requirements Manual (Commander, Naval Air Forces, 2021). All mandatory inspection, removal, or replacement events are included in this manual, which specifies scheduled-maintenance plans. Intervals are determined by the vendor or engineering reliability and maintainability analyses, along with the RCM program failure management strategies (Department of Defense, 2011, Department of Defense, 2020a). PMIC cards produced by them mandate scheduled maintenance for a fleet of aircraft or components. A problem with this is that maintenance intervals are identical for each aircraft or component. These intervals fail to consider the use, service history, or historical data for a unique component or aircraft.

Naval Aviation could benefit from innovative practices in maintenance based on evidence of need or a prediction for individual components. In recent years, RCM has adopted the Condition Based Maintenance Plus (CBM+) strategy to improve reliability. Part of the CBM+ strategy is using machine learning to predict when a component will fail based on historical evidence. Because Marine Corps aviation falls under the umbrella of Naval Aviation, any MV-22B RCM or CBM+ activities fall under the Commander of the Fleet Readiness Centers (COMFRC). The V22 Fleet Support Team (FST) at Fleet Readiness Center (FRC) East has been working on improving aircraft and component reliability through many initiatives.

One of these initiatives is estimating the probability of failure for MV-22B components using statistical models. Using historical maintenance records from existing Naval Aviation Enterprise (NAE) data repositories, the time before a component experiences a specific failure mode was estimated using the Weibull probability density function (PDF). Figure 2 is an example time-to-failure (TTF) graph showing the percentage of MV-22B pylon conversion actuators (PCA) that failed due to seal damage. For the PCA model, 70 percent of the fleet's inventory were forecast to require removal due to seal damage by 3,326 flight hours while 80 percent were forecast to fail by 3,696 flight hours. Model accuracy is discussed in Chapter V, but this approach takes steps towards improving preventative maintenance policy using relevant failure data.

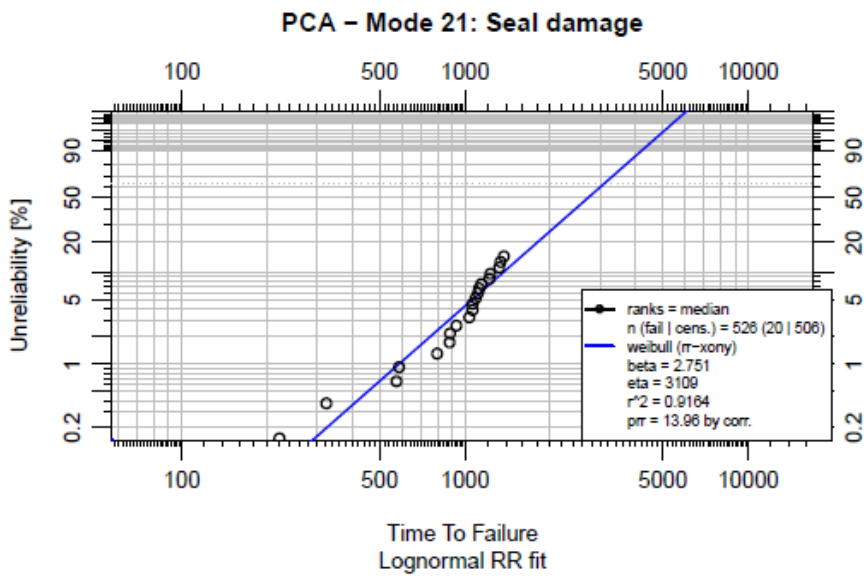

Figure 2. Weibull Model for Pylon Conversion Actuators.
Source: FRC East V22 FST Maintenance Optimization (2022).

This statistical model calculates reliability based on the true service life of a fleet of components before a defined failure mode. Figure 3 shows the results of the Weibull analysis for all PCA failure modes considered by the FRC East V22 FST.

Failure Modes	70th Percentile (FH)	80th Percentile (FH)
PCA - Mode 01: Int Wear Before Scheduled Greasing	2587.52	2993
PCA - Mode 02: Soft Stop	3963.89	4743.05
PCA - Mode 14: Int Wear Post Scheduled Greasing	4376.5	5453.95
PCA - Mode 15: Cross Over Jam	4993.21	6067.8
PCA - Mode 16: Inefficient Ballscrew	2900.72	3415.07
PCA - Mode 16: Ratcheting	3920.05	4810.93
PCA - Mode 21: Seal damage	3326.1	3696.16

Figure 3. Forecasted PCA Failure Rates in Flight Hours.
Source: FRC East V22 FST Maintenance Optimization (2022).

When all failure modes are considered, the estimated reliability of a component can determine a better scheduled maintenance interval. On average, 70 percent of the fleet's inventory are forecast to require removal by about 3,700 flight hours while 80 percent are forecast to fail by about 4,500 flight hours. Program leadership can make decisions based

on a confidence interval to replace the current scheduled maintenance intervals published in the PMIC cards. This interval would apply to the fleet of components and improve the likelihood of replacing the component before any of the failure modes evaluated occurred. The scheduled maintenance is improved with true service data and can be easily recalculated as data continues to be collected.

Unfortunately, this approach only provides a cumulative probability of failure for components. A better approach would be to estimate the conditional probability of failure, also known as the hazard rate, for a component as a function of time. Machine-learning models such as the Cox proportional-hazards (CPH) model and artificial neural networks (ANN) may be useful because they have recently been used in medical research for predicting mortality rates (Spooner et al., 2020). Similar work could use the data maintained in DECKPLATE.

B. RESEARCH QUESTIONS

This thesis will focus on the following research questions:

Primary Question: What machine learning algorithms produce the best survival models for preventative maintenance of aircraft components?

Secondary Questions: What features in DECKPLATE and other repositories can be exploited in predicting component survival? How much does the mean time to failure (MTTF) differ between published PMIC requirements and survival models? For appropriate data, do classic distributions such as Weibull fit the data well to estimate future failures?

C. SUMMARY

Chapter II goes over basic concepts in machine learning and reliability analysis, and investigates previous attempts to use machine-learning for predictive maintenance. Chapter III more precisely describes the problem this thesis aims to solve, and the general approach used. Chapter IV describes the methodology used in this thesis and the justification for its structure. Chapters V and VI discusses results of this thesis and conclusions drawn.

THIS PAGE INTENTIONALLY LEFT BLANK

II. MACHINE LEARNING AND FAILURE ANALYSIS

In May of 2020, the Joint Artificial Intelligence Center (JAIC) published a request for information (RFI) on an artificial intelligence-based predictive maintenance initiative for the H-60 helicopter platform's General Electric T700 turboshaft engine (Department of Defense, 2020b). Programs across the DOD have expressed interest in predictive maintenance, and are looking both internally and to industry to advance those capabilities within their CBM programs. This chapter first explores previous machine-learning applications towards predictive maintenance and identifies key features and practices that apply to this thesis. Next, the chapter explores a previous attempt to estimate the conditional probability of failure for the MV-22B Osprey flight-critical components using historical maintenance records.

A popular method for predictive maintenance is prediction of the remaining useful life (RUL) of a component. The remaining-life estimates the time that a component has left to operate until some defined failure state or level of degradation. Many industries measure and record machinery operating conditions using sensors. The true remaining-life of a component is unknown until the component fails. Therefore, supervised training of a machine-learning model depends on the amount of historical failure data. For data of components that have failed, the life can be calculated and added as the target feature for training a machine-learning model.

For predictive maintenance in the aviation industry, research has been published on a public dataset (Saxena & Goebel, 2008). The National Aeronautics and Space Administration (NASA) turbofan jet engine data have been used to compare machine-learning algorithms, architectures, and methodologies (Mathew et al., 2017). The repository records 26 numerical features for engines at the end of every cycle of operation, which is anywhere between a few minutes and a few hours. These features include the engine unit number, cycle number, operating modes, and 21 sensors. Figure 4 is a snapshot of one training data set imported using the Orange data mining platform. The data gives a chronological record of sensor readings at the end of each cycle of operation. The three operational settings have a significant impact on engine performance and the sensors are

for health diagnostics. Failure occurs at the last cycle recorded for an engine unit number. Therefore, the remaining life is the time difference between the last record and the current record.

	unit number	time, in cycles	operational setting 1	operational setting 2	operational setting 3	sensor measurement 1	sensor measurement 2	sensor measurement 3
1	1	1	-0.0007	-0.0004	100	518.67	641.82	1589.7
2	1	2	0.0019	-0.0003	100	518.67	642.15	1591.82
3	1	3	-0.0043	0.0003	100	518.67	642.35	1587.99
4	1	4	0.0007	0	100	518.67	642.35	1582.79
5	1	5	-0.0019	-0.0002	100	518.67	642.37	1582.85
6	1	6	-0.0043	-0.0001	100	518.67	642.1	1584.47
7	1	7	0.001	0.0001	100	518.67	642.48	1592.32
8	1	8	-0.0034	0.0003	100	518.67	642.56	1582.96
9	1	9	0.0008	0.0001	100	518.67	642.12	1590.98
10	1	10	-0.0033	0.0001	100	518.67	641.71	1591.24

Figure 4. NASA Turbofan Jet Dataset

A. POSSIBLE MACHINE-LEARNING METHODS FOR PREDICTIVE ANALYSIS

Common machine-learning models applied towards predictive maintenance include linear-regression and artificial neural networks. Diagnostic data have temporal dependencies that should be considered when generating data structures for modeling.

1. Linear Regression

A simple machine-learning algorithm for predictive analysis is the linear-regression model. Regression estimates a target continuous value from a weighted sum of other numbers. For remaining-life applications, linear-regression commonly predicts a life from other numeric features such as sensor values. The most common metrics of model accuracy are mean-squared error (MSE), root mean-squared error (RMSE), and mean absolute error (MAE). After data preprocessing, a linear-regression model yielded an average RMSE of 36.71 cycles of operation remaining across the four NASA turbofan datasets in the repository (Li et al., 2018). In other words, the model prediction for engines that had not yet failed was off by about 36 cycles.

2. Artificial Neural Networks

Time-series analysis can address predictive-maintenance applications to include remaining-life predictions. In the linear-regression example, inputs for model training and predictions were sensor readings at the end of a single cycle of operation. Because the sensors have temporal dependencies, a better input vector would be sensor readings over multiple operational cycles. This approach would better capture degradation or the health trajectory of an engine. Different types of neural networks are ideal for handling trends.

a. Long Short-Term Memory Networks

Recurrent neural networks (RNN) reason about sequences of states using an internal memory. This is well-suited for handling data with trends and dependencies over time. However, they often require much data to train. Methods such as cross-fold validation train the model over multiple random splits among the data to reduce overfitting. For time series data, dependencies occur between observations that cannot be disrupted so splitting must be applied only to features that distinguish sequences of observations from others. The Long Short-Term Memory (LSTM) network is a popular recurrent neural network for prognostics and health management that uses intelligent splitting. A two-layer LSTM trained on the NASA turbofan dataset yielded an average RMSE of 21.25 cycles of operation from the true time of failure (Zhang, et al., 2020).

Time-series data requires challenging decisions during preprocessing and structuring for input into a model. Preparing the NASA turbojet data for an LSTM network demonstrated this complexity. During preprocessing, constant features and highly correlated features were removed, reducing the features to 13. The data was normalized and divided into training and test sets. Engine units were randomly split into training and test sets while their sequential sensor readings maintained chronological order. For the 100 engines in the first dataset of the repository, 75 were randomly chosen for training and the remaining 25 were chosen for testing. Since it was desirable for the network to handle a constant input size, input vectors smaller than the largest input vector size in a batch were padded with zeros at the end. Figure 5 shows the resulting data before and after preprocessing.

LSTM_input_train

75x1 cell

	1	2
1	13x166 dou...	
2	13x163 dou...	
3	13x213 dou...	
4	13x154 dou...	
5	13x234 dou...	
6	13x179 dou...	
7	13x231 dou...	
8	13x213 dou...	
9	13x213 dou...	
10	13x185 dou...	

LSTM_RUL_train

75x1 cell

	1	2
1	1x166 double	
2	1x163 double	
3	1x213 double	
4	1x154 double	
5	1x234 double	
6	1x179 double	
7	1x231 double	
8	1x213 double	
9	1x213 double	
10	1x185 double	

LSTM_input_train

75x1 cell

	1	2
1	13x362 dou...	
2	13x336 dou...	
3	13x313 dou...	
4	13x293 dou...	
5	13x287 dou...	
6	13x283 dou...	
7	13x283 dou...	
8	13x278 dou...	
9	13x275 dou...	
10	13x269 dou...	

LSTM_RUL_train

75x1 cell

	1	2
1	1x362 double	
2	1x336 double	
3	1x313 double	
4	1x293 double	
5	1x287 double	
6	1x283 double	
7	1x283 double	
8	1x278 double	
9	1x275 double	
10	1x269 double	

Figure 5. NASA Engine Time Series Input Vectors for LSTM Training

The input layer connected to a single LSTM layer with 200 hidden units. This corresponds to recurrently connected LSTM memory that could remember 200 pieces of information (Graves & Schmidhuber, 2005). Each new input is concatenated with the hidden state of 200 units, which then becomes the input to an LSTM memory cell. This layer is followed by a fully connected layer with 50 nodes, a dropout layer, and a final fully connected layer. The dropout layer created a twenty-percent chance of excluding a recurrent connection's input to reduce overfitting the model to training data (Zhang et al., 2020). The final fully connected layer has an output size of one, the predicted number of cycles of operation remaining before failure. Figure 6 is a plot of the model's remaining-life prediction performance on the test data, which resulted in a RMSE of 25.3 cycles of operation from the true time of failure.

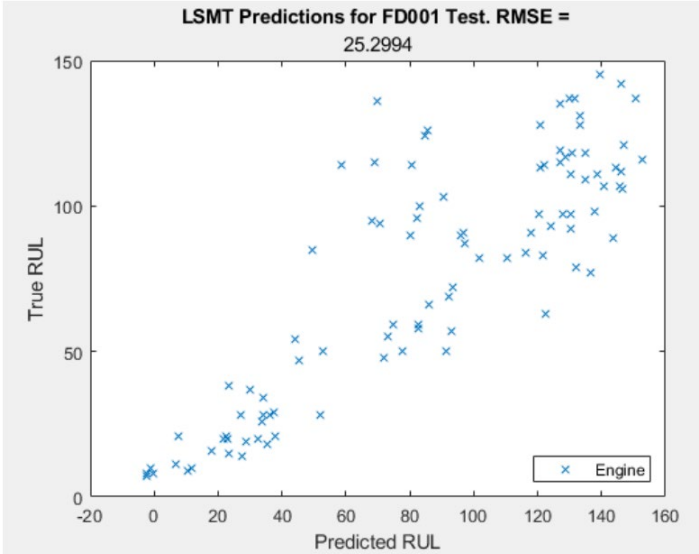

Figure 6. LSTM RUL Predictions for NASA Engines

b. Convolutional Neural Networks

While the LSTM network is well-suited for time-series data, convolutional neural networks (CNN) are good for input data with many attributes (Li, Ding, & Sun, 2018). This approach has recently become popular in lifetime prediction with time-series data. Like the LSTM network, data input for the CNN can be a 2-dimensional time series representation of input features, but it can be split differently than a recurrent neural network. Although less used for lifetime predictions, convolutional models have been used on the NASA turbofan dataset quite a bit recently. A promising way to use them is to train several individual models in parallel with different random data subsets and combine results with averages (Wen, Dong, & Gao, 2019).

Figure 7 shows the convolutional deep-learning architecture proposed by (Li et al., 2018). It used a time window size of 30 and 13 features from preprocessing techniques, and a sequence input layer is followed by four convolutional layers. Each layer has 10 filters with a dimensionality of 10 by 1. With a stride of one, the filters advance over the larger input vector by one dimension at a time. All layer us the hyperbolic tangent activation function. A fifth convolutional layer combines the previous feature maps into one, and connected to a dropout layer. The dropout layer created a fifty-percent chance of excluding a unit's output to reduce overfitting the model to training data (Li et al., 2018).

Subsequent conventional layers reduce the output to a single lifetime prediction Figure 8 plots the improved performance with training on the FD001 engine dataset. With a training RMSE of 21.59 and a test RMSE of 25.30, the deep CNN model demonstrated similar performance to the LSTM network.

ANALYSIS RESULT

	Name	Type	Activations	Learnables
1	input Sequence input with 30×13×1 dimensions	Sequence Input	30×13×1	-
2	fold Sequence folding	Sequence Folding	out 30×13… miniBatchSi… 1	-
3	conv_1 10 10×1 convolutions with stride [1 1] and padding 'same'	Convolution	30×13×10	Weights 10×1×1×10 Bias 1×1×10
4	tanh_1 Hyperbolic tangent	Tanh	30×13×10	-
5	conv_2 10 10×1 convolutions with stride [1 1] and padding 'same'	Convolution	30×13×10	Weights 10×1×10×10 Bias 1×1×10
6	tanh_2 Hyperbolic tangent	Tanh	30×13×10	-
7	conv_3 10 10×1 convolutions with stride [1 1] and padding 'same'	Convolution	30×13×10	Weights 10×1×10×10 Bias 1×1×10
8	tanh_3 Hyperbolic tangent	Tanh	30×13×10	
9	conv_4 10 10×1 convolutions with stride [1 1] and padding 'same'	Convolution	30×13×10	Weights 10×1×10×10 Bias 1×1×10
10	tanh_4 Hyperbolic tangent	Tanh	30×13×10	-
11	conv_5 1 3×1 convolutions with stride [1 1] and padding 'same'	Convolution	30×13×1	Weights 3×1×10 Bias 1×1
12	tanh_5 Hyperbolic tangent	Tanh	30×13×1	-
13	unfold Sequence unfolding	Sequence Unfolding	30×13×1	-
14	flatten Flatten	Flatten	390	-
15	dropout 50% dropout	Dropout	390	-
16	FC_1 100 fully connected layer	Fully Connected	100	Weights 100×390 Bias 100×1
17	tanh_6 Hyperbolic tangent	Tanh	100	-
18	FC_2 1 fully connected layer	Fully Connected	1	Weights 1×100 Bias 1×1
19	output mean-squared-error	Regression Output	1	-

Figure 7. Deep CNN Architecture from Matlab Network Analyzer

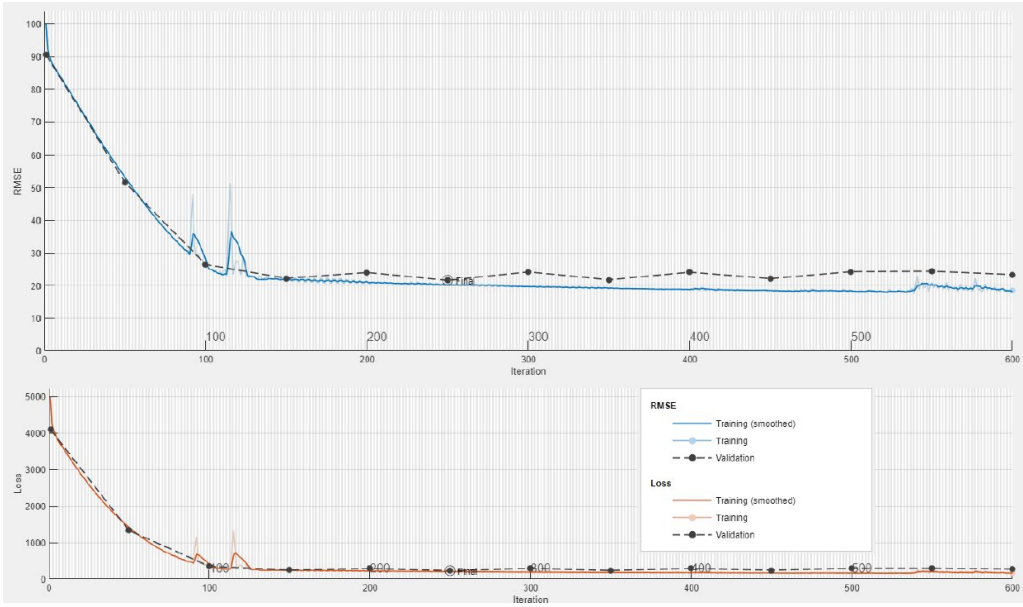

Figure 8. Deep CNN Training for RUL in Matlab

B. SURVIVAL ANALYSIS

For a maintenance department, the timely and accurate prediction of how many flight hours remain until an individual component fails helps maintenance planning. Survival analysis estimates or predicts the probability of a terminal event occurring at or between some interval, and identifies prognostic factors contributing to when the failure occurs (Tilman, 2020). For predictive maintenance, a common survival analysis observation is TTF. The Weibull analysis is commonly used in engineering for survival analysis and can express the mathematical steps towards obtaining conditional probability of failure.

The Weibull analysis models the probability of failure as a function of time. The Weibull probability distribution has two-parameter and the three-parameter forms. The two-parameter form has a shape parameter β and a scale parameter η. The value at which the 63rd percentile of the distribution occurs at is represented by η. For a three-parameter Weibull, γ defines the non-zero "failure-free" period of a component (Rinne, 2008).

$$f(x) = \frac{\beta}{\eta}\left(\frac{x}{\eta}\right)^{\beta-1} e^{\left[-\left(\frac{x}{\eta}\right)^{\beta}\right]}, \; x > 0$$

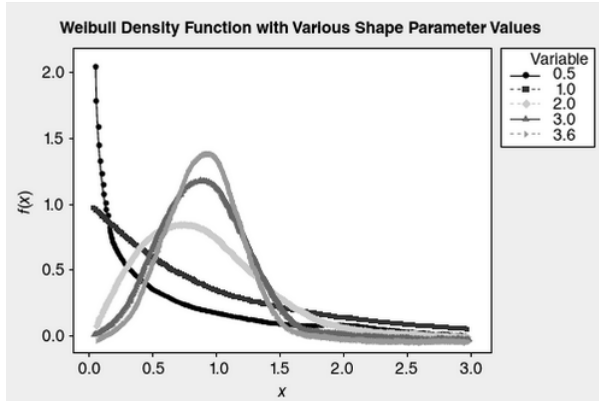

Figure 9. Weibull PDF with Various Shape Parameters, β.
Source: McCool (2012, p. 75).

Taking the integral of the Weibull distribution yields the cumulative distribution shown below. This gives the percentage of components estimated to have failed by a time x, often referred to as unreliability.

$$F(x) = 1 - e^{\left[-\left(\frac{x}{\eta}\right)^{\beta}\right]}$$

In survival analysis, the cumulative distribution is often used to set policy for removal, replacement, or some other maintenance action based on thresholds or percentiles. The COMFRC V22 FST provides lifecycle sustainment services to the fleet to include innovative research. One of their projects used Weibull analysis to develop a better preventative maintenance policy. The objective was a policy for taking preventative measures on a component that has reached a certain number of flight hours. Figure 3 in Chapter I Section A is an example that demonstrated how the Weibull cumulative distribution function (CDF) for each type of failure mode of a component could be used with a threshold of safety to replace the current preventative maintenance policy. Another useful metric is the probability of survival up to a given time given. The hazard rate, $h(x)$ can be calculated using the two previous distributions.

$$h(x) = \frac{f(x)}{1 - F(x)}, \; x \geq 0$$

Often described as the "bathtub curve," the hazard rate of a component is high at installation due to early defects, and at the end of service due to wear. Between these periods, the hazard rate is lower and more constant (Wilkins, 2002). A beta of one indicates that a component's probability of failure is not affected by age; a beta greater than one indicates that the hazard function increases with age (McCool, 2012). Combining multiple Weibull distribution can be a better model than using just one (Dong and Nassif, 2019). The most important survival function in this thesis is the conditional probability of failure, which is the probability of failure at a future time x_b given the duration it has lasted x_a. In Weibull analysis, this is $F\left(x_b | x_a\right)$ using the cumulative probability of failure:

$$F\left(x_b | x_a\right) = 1 - e^{\left[\left(\frac{x_a}{\eta}\right)^{\beta} - \left(\frac{x_b}{\eta}\right)^{\beta}\right]}$$

This function is important in this thesis because it represents a univariate conditional estimate of failure. COMFRC V22 FST want to generate Weibull distributions that account for environmental factors in land or sea operations. Figure 10 shows their proposed method for better predicting a failure of a component based on various characteristics of its current operating life. The proposed approach can individually estimate MTTF in non-austere land-based environments, saltwater-based environments, and austere-based environments.

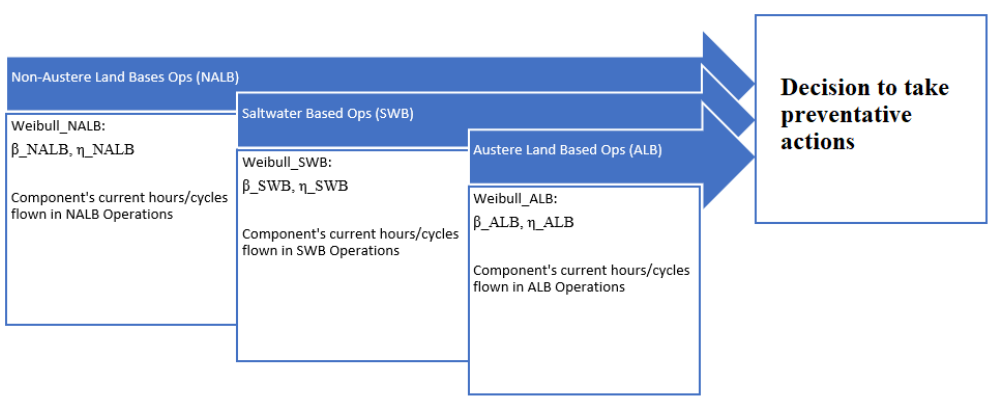

Figure 10. Proposed Approach of Failure Prediction using Weibull Analysis

THIS PAGE INTENTIONALLY LEFT BLANK

III. FAILURES OF MILITARY AIRCRAFT COMPONENTS

This chapter explains why use of components on the Marine Corps' MV-22B aircraft can differ and therefore be problematic for current scheduled maintenance policies. It also discusses naval aviation data repositories as a valuable source of knowledge for machine learning. It distinguishes conditions-based maintenance from current naval aviation practices, and describes reliability analysis and previous work using machine learning to predict conditional probability of failure.

A. AIRCRAFT AND COMPONENT USE RATES

Aircraft-use rates are important metrics that Aircraft Controlling Custodians (ACCs) must monitor and act upon. Often estimated by the average monthly hours flown, the use rate indicates the average operational stress on an aircraft. Aircraft Life Management (ALM) is part of the Marine Corps' Aviation Readiness Program that aims to ensure aircraft reach their intended service life. A properly used fleet of aircraft is defined as falling "within 12 months of, or exceeding projected retirement or transition date, or are within 10% of a published aircraft utilization rate" (Department of the Navy, 2018). A measured rate higher than the published one indicates an aircraft with a shorter service life than what is required. Failures to maintain acceptable use rates not only disrupt short-term planned maintenance schedules and distributions of repairable and consumable supplies but affect long-term strategic plans and acquisitions.

While monitoring the flight hours of aircraft can help determine if they will reach their intended service lives, it does not include use anomalies for individual aircraft. Figure 11 depicts the MV-22B Osprey use rates from March 2021 through February 2022 when the published rate was 16 flight hours per month (Commander, Naval Air Forces, 2022). According to ALM program parameters, the MV-22B use rate was only considered outside of the ten percent limit for four of the twelve months.

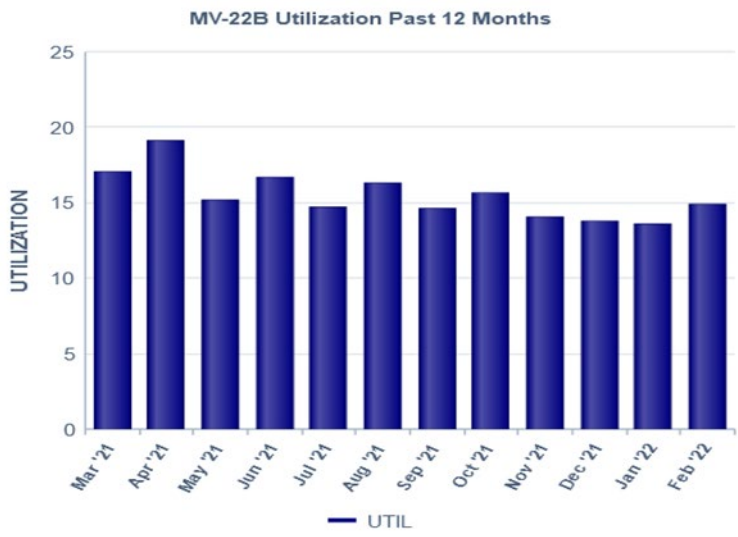

Figure 11. MV-22B Use Rates from March 2021 to February 2022.
Source: NAVAIR Vector Aircraft Readiness Dashboard (2022).

Instead of averaging use across all MV-22Bs, Figure 12 and Figure 13 show the distribution of flight hours flown per month per aircraft for the past year and the past five years, respectively.

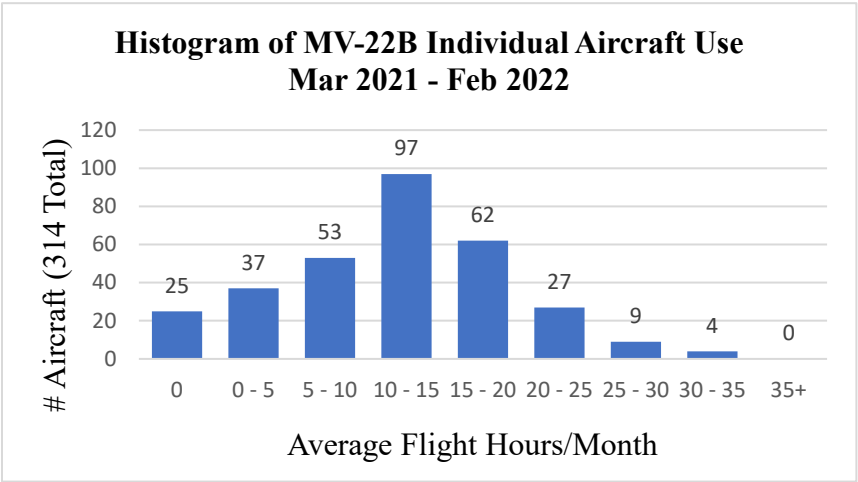

Figure 12. Histogram of MV-22B Use Rates from 2021 to 2022.
Adapted from NAVAIR DECKPLATE Query of Flight Records (2022).

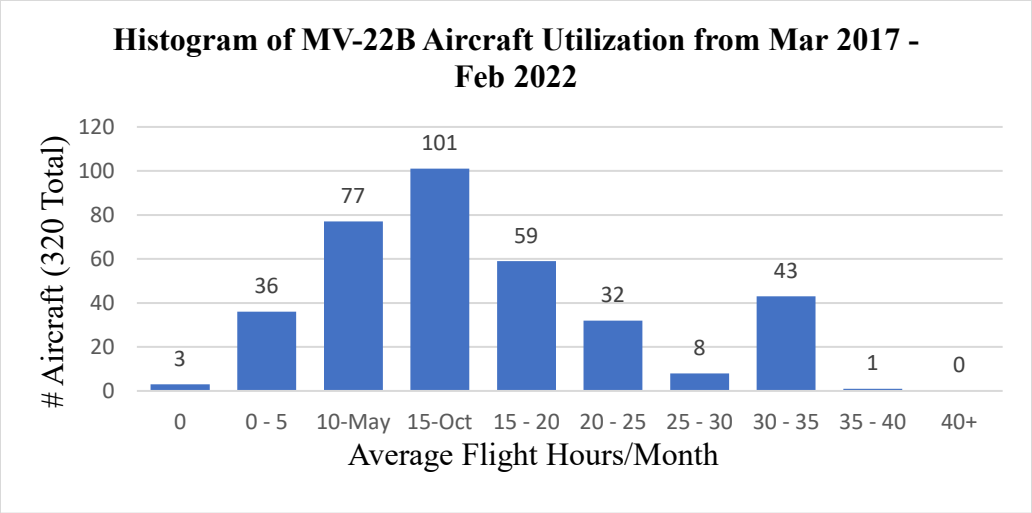

Figure 13. Histogram of MV-22B Use Rates from 2017 to 2022.
Adapted from NAVAIR DECKPLATE Query of Flight Records (2022).

These statistics are the average use for each aircraft in only the months they were active. 44 aircraft were used more than double the recommended rate over the five years. These aircraft are labeled the "flyers" or the "workhorses" in the squadrons. Unsurprisingly, the time it takes one aircraft to reach an important flight-hour milestone can differ by years from another. The "flyers" are more likely to be used in austere environments, and this accelerates their wear and degradation.

In 2007 the Marine Corps deployed ten MV-22B Osprey in Iraq. For the next two years, they flew those same ten aircraft in a very austere environment at double the use rate as planned (O'Rourke, 2009). The practice of "relief in place / transfer of authority" (RIP/TOA) for deployed Osprey squadrons is still followed today. With it, airframes and components can have significantly different operating lives. Fortunately, these differences can be obtained from flight records, such as the number of flight hours flown in an austere environment, or the installation data on a serialized component installed on multiple airframes during its life. The number of days in a Marine Expeditionary Unit (MEU) indicates how long a component has been exposed to corrosive sea spray, and the type of ship the aircraft was on indicates how much closer the flight deck was to the ocean surface.

B. NAVAL AVIATION DATA

Aviation organizations subject to Federal Aviation Administration (FAA) regulations must maintain an air-carrier maintenance program per Title 14 of the Code of Federal Regulations (CFR). While the FAA establishes minimum maintenance requirements for an aircraft, carriers can make stricter policies. United States Naval aviation follows the Naval Aviation Maintenance Program (NAMP) which applies to all Marine Corps aviation. Certain records must be made and kept for a period following FAA requirements for the management of aviation maintenance.

1. Existing Repositories

With over 4,000 aircraft in U.S. Navy aviation, many maintenance records must be stored physically and electronically at individual squadrons and in data repositories online. The three recordkeeping systems most related to this thesis are the Naval Aviation Logistics Command Management Information System (NALCOMIS), DECKPLATE, and the Aviation Maintenance Supply Readiness Report (AMSRR).

The NALCOMIS database is the closest to the aircraft themselves. This database is used by maintenance and supply personnel to record every maintenance action or supply request done for an aircraft, component, or support equipment. Each operational squadron and intermediate-level maintenance squadron keeps electronic records of every tracked component on that air station using NALCOMIS, whether it is installed on an aircraft. Its design and features satisfy the compliance requirements of the NAMP for maintenance and material management (Department of the Navy, 2021). Although NALCOMIS is local, maintenance administration specialists do routine uploads of its data to DECKPLATE.

In the past, DECKPLATE was the data warehouse for NALCOMIS records. Today, DECKPLATE has absorbed over a dozen more maintenance and supply recordkeeping repositories (Teradata, 2016). With over 40 billion records uploaded monthly, DECKPLATE is useful for many purposes. Most of these records are unclassified and for official use only (FOUO), which permits protected quick access. Although the NAE has made much progress towards consolidating data, users are "drowning in data and starving

for information" (Lancaster, Talbert, & Kirk, 2014). More work must be done to collect data that aviation professionals can act upon.

For this research, queries in DECKPLATE produce the data for reliability and survivability analysis. It has historical data to describe how a component failed, when it failed, and the details about the operating life and condition of that component before failure. All maintenance actions are timestamped which allows the user to determine time to failure. Additional information can be extracted using flight records.

The AMSRR is a maintenance reporting system focused on the material status of Naval aircraft. It is used mostly by aviation maintenance and supply officers or senior enlisted, and offers a snapshot of material readiness. It also adds operating-life data that can linked to flight records in DECKPLATE such as location.

2. Data Integrity

Machine-learning algorithms do better as the amount of training and testing data increases. However, the usability of models depends on the quality of the data. Inaccuracies in data can degrade machine-learning applications when models do not accurately reflect the real-world situation they attempt to represent. Maintenance recordkeeping has not historically been considered important; quality requirements of data are often outweighed by functionality requirements (Wilson et al., 2020). Maintenance records often have errors due to a lack of training and experience or the limitations of the maintenance-record software. The lack of emphasis on the quality of data being recorded over the past few decades make the DECKPLATE repository sometimes unreliable.

A contributing factor was that the DECKPLATE data repository was not designed for complex large-scale data extraction. Previous research done by NAVAIR identified the tedious process of manually mining useful data from DECKPLATE as a hindrance to current and future work (Burger, Jaworowski, & Meseroll, 2011). Converting the millions of records to a better format would be too difficult now with budget constraints. An analysis of alternatives to the existing maintenance record system in naval aviation estimated cleansing of data during migration to a more capable system would take from one the three years (Wilson et al., 2020).

This thesis work is limited to data from existing repositories to identify how much data integrity is an issue for future work. We assume with the Commandant of the Marine Corps that extracting actionable knowledge from emerging technologies cannot wait for the costly process of creating new repositories. This thesis uses historical flight records and maintenance records to extract an individual aircraft component's use pattern. The number of flight hours flown during austere land-based or sea-based operations are not readily available for components, nor total flight hours. However, scripts can be written to generate this data by combining flight records and removal/installation (R/I) maintenance records. The challenge is determining in what aircraft a component was installed throughout its service life despite cannibalization and unscheduled maintenance.

The integrity of flight records is relatively high compared to those of R/I maintenance records for serialized components. However, missing or erroneous records occur. Furthermore, some components do not carry the same part number and serial-number combination throughout their service lives. For example, some technical directives that implement a major modification to a component may give it a new part number. If overlooked, one component could be mistaken as two different components. Due to the time constraints of this thesis, assumptions about the data in DECKPLATE are made to facilitate completion and are detailed in Chapter IV Section A.

C. CONDITION BASED MAINTENANCE PLUS

Maintenance actions are either scheduled or unscheduled. The DOD's CBM+ strategy aims to shift maintenance to a more proactive scheduled approach rather than a reactive unscheduled one (Department of Defense, 2020a). Although scheduled maintenance can prevent some failures, much of it may be unnecessary and an inefficient use of resources.

The CBM+ concept has a broader view of data sources and predictive methods than the original CBM maintenance program in which, maintenance was triggered from diagnostics or sensor data. CBM+ works more closely with reliability-centered maintenance to analyze system performance and anticipate maintenance requirements before failure or degradation (Department of Defense, 2020a). Industry leaders such as

General Electric Aviation use CBM programs that aim to "maintain the correct equipment at the right time" (Aviation Pros, 2009).

D. RELIABILITY ANALYSIS

In engineering, reliability is defined as "the probability that a component, device, system or process will perform its intended function without failure for a given time" (Waghmode & Patil, 2016). Unreliable components hurt performance and cost.

1. Censored Data

Predictive maintenance differs from preventative or reactive maintenance by using monitoring data to anticipate when maintenance will be required. The lifetime or time to failure are estimates of when maintenance will be required. The units of measurement may be time, cycles, or uses. However, challenges with these estimates are that multiple modes of failure are possible, and failure is often not confirmable for a long time. This leads to the concept of censored data. Consider ten engines observed over a year for a specific failure, and suppose at the end of the year, only two engines failed and the other eight are still operating as expected. Those eight engines are considered "right censored" and do not provide failure data. Similarly, any engines that failed before the study but are still included in the dataset are considered "left censored." Censoring reduces the usable cases in a dataset.

2. Applying the Weibull Distribution

The Weibull distribution covered in Chapter II Section B has proven useful in modeling reliability due to its flexibility and effectiveness with smaller datasets (Quanterion Solutions, 2015). It assumes that the data is censored and that no repairs have been made to the equipment being analyzed (Quanterion Solutions, 2015). Previous work by the COMFRC V22 FST developed comprehensive datasets of MV-22B components that have failed. Similar components were categorized by failure mode, which divided usable datapoints into smaller subsets for statistical analysis. Some subdivisions had too little data for meaningful analysis and were left out.

A disadvantage of the Weibull distribution is that it is univariate. Experts in aviation maintenance, engineering, and reliability acknowledge various environmental conditions impact a component's health. However, the amount of failure data for each of these conditions for a component is insufficient for machine learning. When a component fails and is repaired, it is considered imperfect preventative maintenance. Altering the hazard function for a component after repair has proven to work for CBM to significantly reduce preventative maintenance hours (Zhou, Xi, & Lee, 2007).

3. CPH

The CPH model fits the relationship between survival time and survivability for censored as well as uncensored data (Chen et al. 2021). The CPH model is popular for medical research for predicting mortality rates of patients based on their condition at an operation or treatment. The data for this type of research is historical medical records, which makes it similar to this thesis work. While sensor data is preferred in predictive maintenance, the CPH model has enabled researchers to use historical maintenance data. It is popular due to its ability to handle censored and sparse data (Chen et al. 2021).

Chapter II Section B suggests using multiple Weibull models for component failure. However, early regression models were limited in that the hazard function could only increase or decrease proportionally with time. The CPH model allows the hazard rate to fluctuate over time, which and better supports splitting a dataset of failed components by failure mode. It determines how covariates affect the hazard function (Korvesis, 2017). It does not assume that the input variables are independent, and the ratios between those covariate hazard rates remain constant with age (Cox, 1972). To validate these assumptions, the Schoenfeld residuals can be evaluated (Tai & Machin, 2014). Regression coefficients should change with time resulting in an increasingly greater than or less than zero mean or "p-value." A residual mean threshold of 0.05 is often used. P-values less than this threshold indicate a covariate violates the CPH assumption.

4. Artificial Neural Networks

To overcome the limitation of a constant ratio of risk factors with time, solving nonlinear problems is required. Artificial neural networks can do this. Their use for survival

analysis in medicine has become increasingly popular, such as prediction of the probability of breast cancer recurrence after patients underwent surgery (Chi et al., 2007). This data was censored since it missed final outcomes of patient status. However, the predictions were to the nearest year (Chi et al., 2007), and predictive maintenance requires more exact predictions. Previous work has compared CPH models to neural networks for automobile maintenance prediction (Chen et al., 2021). However, LSTM networks and CNNs did better (Chen et al., 2021).

THIS PAGE INTENTIONALLY LEFT BLANK

IV. METHODOLOGY

A. RESEARCH AND DESIGN STRATEGY

This research attempted to enhance the reliability analysis of MV-22B components done by the V22 FST. Many statistical machine-learning models and architectures can predict reliability. This research focused on comparing the conditional probability of failure from the univariate Weibull distribution to the multivariate Cox proportional-hazards model and neural networks. Valuable input features beyond the time to maintenance action (TTMA) are collected and added to the input data creating a multidimensional input vector. Based on previous work and best practices, neural-network architectures were used to model and predict the conditional probability of failure as a function of time for select MV-22B components.

The first step in our experiment was to identify key features that have the most impact on component degradation or failure data in the repositories available. Initial factors considered were the number of hours or cycles spent in austere environments, flight mode such as airplane, helicopter, or conversion aircraft, and removals for non-failure reasons such as inspection or cannibalization. Once additional features are identified, the methodology for gathering, relating, and sanitizing the data was developed. Every attempt was made to ensure this process is repeatable.

B. SCOPE AND LIMITATIONS

This thesis used historical MV-22B Osprey data from the NAVAIR DECKPLATE data repositories to train machine learning models to predict the conditional probability of failure for the PCA. Although hundreds of repairable flight-critical components on an MV-22B have electronic records, this thesis attempts to focus on a top-ten component as determined by the program office to be the most logistically and financially challenging. These top ten components are typically the engines, proprotor gearboxes, and other dynamic components. However, limiting this thesis to evaluation of one already evaluated by the V22 FST greatly reduced data collection time and enhanced their previous work.

The NAVAIR DECKPLATE data repositories are live databases that update component records daily, which could skew data if not collected properly. Therefore, training of models was done with a fixed set of labeled training data with supervised classification. When permissible, the research attempted to divided censored and uncensored data into training and test sets randomly and used cross-validation to reduce bias. However, the failure rates for different components can vary greatly. In previous work, the rule of thumb was that a ratio between the number of failure events per variable be at least 10:1 for analysis to be reliable (Spooner et al., 2020). This can significantly impact the feasibility of dividing data into training and test sets. The goal was to use data before 2020 as training data and data from 2020 until the current date as test data. However, for small data subsets, data is split on the date at which 80 percent of datapoints have occurred. Another consideration made for components with fewer failure records is the ratio between data points and features. The V22 FST has already used a minimum of 10 failure events as a rule of thumb.

C. DATA GENERATION

The data used in his thesis is continuous flight data for individual serialized components throughout their service. This required merging data in several DECKPLATE repositories. These included the electronic Naval Flight Records (NAVFLIR) database, the Automated Maintenance Environment (AME) database, and the Serial Number Tracking (DP-0025) report. Figure 14 shows the data generation process we used. The queries from DECKPLATE were saved as Microsoft Excel comma separated values (CSV) files. Three Python scripts extracted this data and generated flight data for serialized components. The programs ran on Microsoft Windows 10 and required the Python datetime, numpy, copy, and csv libraries.

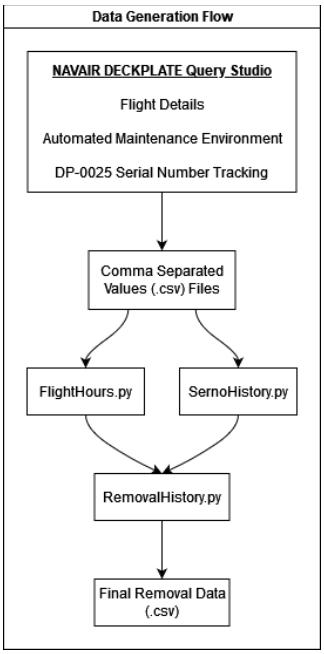

Figure 14. MV-22B PCA Flight Data Generation Flow

Because individual serialized components are often removed and installed on different aircraft, their flight data is rarely identical. Within squadron Maintenance Administration work centers, all major serialized components have a paper log-set that includes removal and installation history. Unfortunately, neither the electronic Auto Log-Set or Configuration Management databases are reliable enough to display a component's installation history, particularly for those not currently in service. While many serialized components record data such as flight hours or cycles flown, these values are cumulative at query time and are often unreliable. Furthermore, this thesis needed additional flight data such as flight hours flown at sea or in austere environments which are not captured in log sets. For this, flight records from aircraft NAVFLIRs must be compared with the dates a component was installed on that aircraft.

We needed to identify which DECKPLATE queries, filters, and sorting could yield pertinent features for model training and testing. To generate this information, maintenance-action details can be queried in the DECKPLATE AME report view. Important criteria are the removed or installed part number and serial number blocks; a non-blank entry in any of these fields implies a component was removed or installed. The

maintenance action form (MAF) origination and completion dates for such entries help determine which serialized components were installed on an aircraft on a given date. Unfortunately, these records can also be unreliable due to recordkeeping errors or missing entries.

1. Component Queries

The work unit code (WUC) identifies equipment, components, or subassemblies in maintenance documentation (Department of the Navy, 2021). The MV-22B flight control actuation equipment WUC is 2750 and has over 160 subassemblies with their own WUCs. The PCA, which adjusts the nacelle angle, is represented by the 275020 WUC for the right-hand nacelle and 275021 WUC for the left-hand nacelle, each with 31 subassembly WUCs. When the entire PCA is removed or installed on an aircraft, the WUC for the entire PCA assembly should be recorded. However, due to inconsistencies in maintenance recordkeeping, erroneous WUCs or those for subassemblies may be recorded instead.

Figure 15 is a snapshot of a query from the AME in DECKPLATE that returns all records of maintenance actions with entries in removal or installation blocks for the 275020 and 275021 WUCs. The type equipment code (TEC) "AYNE" represents the MV-22B. With no date range specified, this query returned 914 removals or installations of conversion actuator assemblies. However, an inconsistency can be seen with the removed part numbers. Part number 42555-43 is the primary hydraulic powered drive unit (HPDU) assembly, one of three such PCA subassemblies. It is ambiguous whether the entire PCA assembly or only the primary HPDU was removed due to the WUC recorded. Adding to the complexity, currently two part numbers identify PCAs in the MV-22B fleet, 42555-400 and 42555-401. They can be installed on either the left or right nacelle, yet different manufacturing details can result in components having multiple part numbers, illustrated in Figure 15 with part numbers 901-301-902-111 and 901-301-902-109. Besides changes in manufacturers, components are upgraded over time by either the squadron or depot facilities. Part-number consolidations or technical directives mandating alterations or inspections of equipment, can also change the part number.

TEC: AYNE AND WUC: 275020, 275021 AND (NOT Rmvd PartNo: OR NOT Inst PartNo:)

TEC	Bu/SerNo	Owner Org Short Name	Rmvd PartNo	Rmvd Equip SerNo	Rmvd NIIN	Rmvd Action Date	Inst PartNo	Inst Equip SerNo	WUC	WUC Desc
AYNE	165946	VMM-162	901-301-902-109	0035	014726110	Sep 12, 2009	901-301-902-109	0035	275021	LEFT CONVERSION ACTUATOR ASSEMBLY
AYNE	166498	VMM-365	901-301-902-109	0091	014726110	Aug 20, 2009	901-301-902-109	0091	275021	LEFT CONVERSION ACTUATOR ASSEMBLY
AYNE	166743	VMM-365	901-301-902-109	0283	014726110	Sep 9, 2009	901-301-902-109	0283	275021	LEFT CONVERSION ACTUATOR ASSEMBLY
AYNE	168652	VMMT-204					901-301-902-109	0055	275021	LEFT CONVERSION ACTUATOR ASSEMBLY
AYNE	166497	VMM-365	901-301-902-109	0176	014726110	Jan 21, 2010	901-301-902-109	0203	275020	RIGHT CONVERSION ACTUATOR ASSEMBLY
AYNE	167911	VMM-161	901-301-902-109	0083	014726110	May 15, 2013	901-301-902-111	0290	275021	LEFT CONVERSION ACTUATOR ASSEMBLY
AYNE	166744	VMM-165	42555-401	0197	015988197	Aug 20, 2013			275020	RIGHT CONVERSION ACTUATOR ASSEMBLY
AYNE	166722	VMM-263	901-301-902-109	0157	014726110	Oct 1, 2009	901-301-902-109	0157	275020	RIGHT CONVERSION ACTUATOR ASSEMBLY
AYNE	165940	VMMT-204	901-301-902-109	0106	014726110	Sep 15, 2008	901-301-902-109	0156	275020	RIGHT CONVERSION ACTUATOR ASSEMBLY
AYNE	165943	VMMT-204	901-301-902-109	0130	014726110	Aug 6, 2008	901-301-902-109	0157	275020	RIGHT CONVERSION ACTUATOR ASSEMBLY
AYNE	166689	VMM-162	901-301-902-109	0085	014726110	Sep 11, 2009	901-301-902-109	0085	275021	LEFT CONVERSION ACTUATOR ASSEMBLY
AYNE	168612	VMM-165	42555-401	0019	015988197	Jan 6, 2020			275020	RIGHT CONVERSION ACTUATOR ASSEMBLY
AYNE	167905	VMM-161	901-301-902-109	0286	014726110	Feb 21, 2010	901-301-902-109	0286	275021	LEFT CONVERSION ACTUATOR ASSEMBLY
AYNE	166722	VMM-264	901-301-902-109	0157	014726110	Jun 14, 2011			275020	RIGHT CONVERSION ACTUATOR ASSEMBLY
AYNE	165941	VMM-263	901-301-902-109	0090	014726110	Apr 25, 2017	901-301-902-109	0884	275020	RIGHT CONVERSION ACTUATOR ASSEMBLY
AYNE	166737	VMM-764	42555-400	0356	015783589	Oct 3, 2017			275020	RIGHT CONVERSION ACTUATOR ASSEMBLY
AYNE	166687	VMM-363	42555-400	0268	015783589	Jun 11, 2014			275020	RIGHT CONVERSION ACTUATOR ASSEMBLY
AYNE	165840	VMM-264	42555-43	0635	014744638	Feb 7, 2013			275021	LEFT CONVERSION ACTUATOR ASSEMBLY
AYNE	166495	VMM-162	901-301-902-109	0265	014726110	Aug 25, 2011			275020	RIGHT CONVERSION ACTUATOR ASSEMBLY
AYNE	168336	VMM-161	42555-401	0534	015988197	Oct 26, 2020			275020	RIGHT CONVERSION ACTUATOR ASSEMBLY

Figure 15. AME Query for MV-22B 275020 and 275021 WUCs.
Source: Report from DECKPLATE AME Query Studio (2022).

Figure 16 illustrates a part-number conversion with serial number 161. Two MAF forms started on August 18, 2009, say the part number 901-301-902-109 with serial number 161 was both removed and installed on aircraft 166387 at the MV-22B training squadron VMMT-204 with a cycle time of 1100 flight hours. Same day removals and installations do not always indicate the component was physically removed, but it could have been administratively transferred, depending on the type of maintenance. The following R/I MAF for the same aircraft on November 1, 2010, shows part number 42555-400 with serial number 161 received maintenance actions with a cycle time of 1379 flight hours. A flight-hour query using the FlightHours.py script we wrote for aircraft 166387 between those dates returns 259 flight hours total flown by the aircraft. A discrepancy of 20 flight hours is not unusual due to separate reporting methods for flights and components. However, it is likely that this is the same PCA with an upgraded part number.

Action Date	Rmvl Inst PartNo	Rmvl Inst Equip SerNo	Action Org Short Name	Buno	R/I	JCN	NCN	Comp Date	Time
04/16/2008	901-301-902-109	161	VMMT-204	166387	I	FC4105256	00GE25A	04/17/2008	783
04/15/2008	901-301-902-109	161	VMMT-204	166481	R	FC4105256	00GE27S	09/15/2008	783
02/20/2009	42555-61	161	MALS-26	166487	R	FC4051495	FC18AC5	02/20/2009	1003
02/20/2009	42555-61	161	VMMT-204	166487	R	FC4051495	00GENPL	02/20/2009	1003
02/20/2009	42555-61	161	VMMT-204	166487	I	FC4049315	00GENO0	02/20/2009	1003
02/20/2009	42555-61	161	VMMT-204	166387	R	FC4049315	00GENPF	03/03/2009	1003
06/09/2009	901-301-902-109	161	VMMT-204	166387	R	FC4160032	00GEVUG	06/17/2009	1078
06/09/2009	901-301-902-109	161	VMMT-204	166387	I	FC4160032	00GEVUG	06/17/2009	1078
08/18/2009	901-301-902-109	161	VMMT-204	166387	R	FC4230450	00GF0AY	09/16/2009	1100
08/18/2009	901-301-902-109	161	VMMT-204	166387	I	FC4230450	00GF0AY	09/16/2009	1100
11/01/2010	42555-400	161	VMMT-204	166387	R	FC4303247	00GFRW1	11/01/2010	1379
11/01/2010	42555-400	161	MALS-26	166387	R	FC4303247	FC19D5Z	11/01/2010	1379

Figure 16. Serial Number 161 PCA Removals and Installations. Adapted from Report run in DECKPLATE AME Query with SernoHistory.py (2022).

The NAMP also lists the Maintenance Data System (MDS) as a source of statistical data for analysis such as equipment reliability (Department of the Navy, 2021). Its Serial Number Tracking (DP-0025) tool covers component repairs that involved a removal or installation. A report was generated for all part numbers and serial numbers with the TEC of "AYNE" for the MV-22B. The resulting CSV file was filtered to exclude all R/I actions whose WUC did not begin with 2750. The data was sorted by serial number and action date and then to ensure a removal entry preceded an installation entry if they shared the same serial number and action date. The SernoHistory.py script we wrote generates hash tables mapping from serial numbers to lists of sorted R/I entries. Only maintenance actions with the WUCs 275020, 275021, 27502015, and 27502115 were included in our experiments, the latter two representing the right or left-hand PCA without the HPDU. Figure 17 shows the same query for serial number 161, which found more data than the query shown by Figure 16.

Action Date	Rmvl Inst PartNo	Rmvl Inst Equip SerNo	WUC	Action Org Short Name	SerNo	R/I	JCN	MCN	Comp Date	Time Cycle 1
04/15/2008	901-301-902-109	161	275020	VMMT-204	166481	R	FC4105256	00GE27S	09/15/2008	783
04/16/2008	901-301-902-109	161	275020	VMMT-204	166387	I	FC4105256	00GE25A	04/17/2008	783
06/09/2009	901-301-902-109	161	275020	VMMT-204	166387	R	FC4160032	00GEVUG	06/17/2009	1078
06/09/2009	901-301-902-109	161	275020	VMMT-204	166387	I	FC4160032	00GEVUG	06/17/2009	1078
08/18/2009	901-301-902-109	161	275020	VMMT-204	166387	R	FC4230450	00GF0AY	09/16/2009	1100
08/18/2009	901-301-902-109	161	275020	VMMT-204	166387	I	FC4230450	00GF0AY	09/16/2009	1100
11/01/2010	42555-400	161	27502015	VMMT-204	166387	R	FC4303247	00GFRW1	11/01/2010	1379
11/01/2010	42555-400	161	275020	MALS-26	166387	R	FC4303247	FC19D5Z	11/01/2010	1379
11/30/2012	42555-400	161	27502015	VMM-263	166720	I	FCG304A35	112GTDJ	11/30/2012	1418
10/07/2013	42555-400	161	27502015	VMM-162	166720	R	FCB280051	121GDR0	10/07/2013	1431
10/07/2013	42555-400	161	27502015	VMM-162	166720	I	FCB280051	121GDR0	10/07/2013	1431
10/13/2016	42555-401	161	27502015	VMM-164	166720	R	GH8287379	13Q4Q1Q	10/24/2016	1686
10/13/2016	42555-401	161	27502015	VMM-164	166720	I	GH8287379	13Q4Q1Q	10/24/2016	1686
11/07/2019	42555-401	161	27502015	VMM-362	166720	R	GJ4311039	4EB1F2H	12/15/2019	2149
11/07/2019	42555-401	161	27502015	VMM-362	166720	I	GJ4311039	4EB1F2H	12/15/2019	2149

Figure 17. Serial Number 161 PCA Removals and Installations. Adapted from Report run in DECKPLATE DP-0025 Report with SernoHistory.py (2022).

2. Component Imputations and Assumptions

The R/I actions from the sorted DP-0025 report provide the critical data needed to determine when components were installed on aircraft. However, many entries lack an installation or a removal entry. The SernoHistory.py script we wrote imputes missing entries based on assumptions about the maintenance. The DP-0025 report does not record components installed on the production line because this was not maintenance. If the first entry is a removal MAF, it is assumed that the component was installed on the production line. If the first entry is an installation MAF, it is assumed that the component was not installed on the production line, but instead a new component from the manufacturer. Once all first-installation MAFs were imputed, the resultant data checked for more than one component occupying an aircraft's left or right-hand PCA WUC and only the earliest installation was kept.

The remaining entries were evaluated for administrative and irrelevant component removals. Component removals can be done due to faults or defects categorized using the documented malfunction, transaction, and action-taken codes. Malfunction codes reflect the need for maintenance while transaction codes reflect the type of data reported (Department of the Navy, 2021). The malfunction codes categorize failure modes. Furthermore, specific codes are used when symptoms or defects prompt the removal of the component. Figure 18 shows the frequency of malfunction codes in the PCA removal dataset.

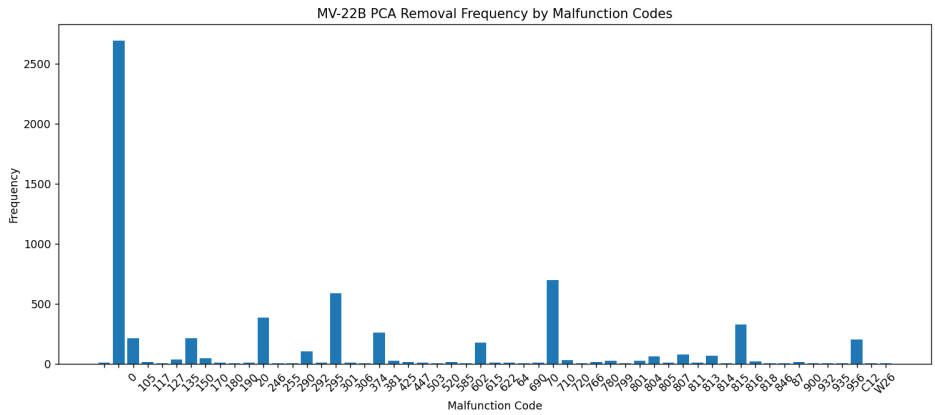

Figure 18. Histogram of PCA Removals by Malfunction Code

The most frequent malfunction code is a blank entry, however, the accompanying transaction code is 47 for technical-directive compliance. As mentioned in Chapter IV Section A, many documented removals are administrative in nature. The only technical directives that have applied to the PCA were inspections and only justified physical removal if they failed the inspection criteria. Of the 2,701 documented removals with a transaction code of 47, only 131 have an accompanying action-taken code "P" for removal. The RemovalHistory.py script we wrote redacts removals to include technical directive compliance not resulting in a removal, intermediate-level maintenance on components already removed, removals where the defect could not be duplicated, or other administrative removals. Figure 19 shows the frequencies of malfunctions in the final dataset.

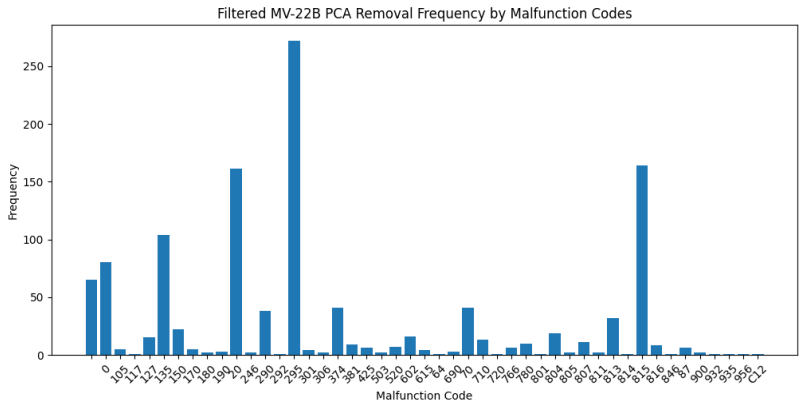

Figure 19. Histogram of Filtered PCA Removals by Malfunction Code

3. Flight Data

The NAVFLIR database maintains historical flight data for maintenance, material, and logistics evaluation (Department of the Navy, 2016). Each record includes the Total Mission Requirement (TMR) hours, TMR code, number of landings, and type of landings. A query for all MV-22B flight records from the NAVFLIR database is done and imported by the FlightHistory.py script to extract input for machine learning.

One attribute is the cumulative TMR hours flown on a component. It is associated with a TMR code which is three characters that represent the general purpose and specific purpose of the flight (Department of the Navy, 2016). For this thesis, TMR hours were categorized by the flight purpose only Table 1 shows the seven flight-purpose categories.

Table 1. NAVFLIR TMR General Purpose Codes

TMR Code	Description
1XX	Training Flights conducted for the purpose of training (both individual and as a crew) to maintain or improve the readiness of the activity to perform its assigned mission.
2XX	Support Services. Flights conducted in support of an assigned mission including tests, logistics, search and rescue, troop transports, etc., either independently or as part of a squadron function
3XX	Operations. Navy flights conducted in support of operational tasking not specifically designated as contingency operations.
4XX	Fleet Marine Forces (FMF) Operations. Marine flights conducted as part of an exercise while deployed with a battle group or task force.
5XX	Contingency Flights. Flights conducted in support of contingency operations as delineated by the type commander.
6XX	Combat Flights. Combat flights shall be used only for aircraft and by units specifically designated by competent authority as being in combat status.
7XX	Exercise Flights. Flights conducted as part of an authorized fleet exercise as designated by the battle group or type commander

Source: Department of the Navy (2016, p. D-8).

Another attribute is the cumulative landings experienced by a component. The associated landing codes represent the environment that the components were flown in. Each landing code is one character representing the type of landing and whether it was conducted at day or at night (Department of the Navy, 2016). For this thesis, landings codes

were categorized as non-austere, austere, or ship landings. Airfield landings are considered non-austere, while all other non-ship landings are considered austere. Table 2 shows the three landing groups by day and night codes and their descriptions.

Table 2. NAVFLIR Landing Groupings

Group	Day Code	Night Code	Description
Non-Austere	0	K	Vertical and/or Short Take-Off and Landing (V/STOL) Vertical Roll
	5	E	Field Carrier Landing Practice (FCLP)
	8	H	V/STOL Slow
	9	J	V/STOL Vertical
	Y	Z	Naval Flight Officer (NFO)
Austere	6	F	Field Full Stop/V/STOL Conventional
	7	G	Field Arrest
	L	M	Unprepared Landing
		P	Night Vision Device (NVD) Land-Field/Field Touch and Go
		Q	NVD Field Deck Landing Practice (FDLP)
	W	T	Field Touch and Go
Ship	1	A	Ship Arrest/Recovery Assist, Secure and Traverse (RAST)
	2	B	Ship Touch and Go
	3	C	Ship Bolter/RAST Free Deck
	4	D	Ship Helicopter/Clear Deck
		N	NVD Ship

Source: Department of the Navy (2016, p. F-3).

The FlightHistory.py script we wrote performs queries for an aircraft for each date between a specified date range. Figure 20 shows flight data for aircraft 168217 between April 1, 2018 and April 13, 2018. Total flight hours, landings, TMR hours and categorical landings are shown.

buno	start_date	flthrs	lndgs	TMR_1_hrs	TMR_2_hrs	TMR_3_hrs	TMR_4_hrs	TMR_5_hrs	TMR_6_hrs	TMR_7_hrs	NON_AUST_lndgs	AUST_lndgs	SHIP_lndgs
168217	04/01/2018	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
168217	04/02/2018	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
168217	04/03/2018	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
168217	04/04/2018	5.0	10.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	8.0
168217	04/05/2018	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
168217	04/06/2018	6.3	13.0	2.9	3.4	0.0	0.0	0.0	0.0	0.0	0.0	2.0	11.0
168217	04/07/2018	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
168217	04/08/2018	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
168217	04/09/2018	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
168217	04/10/2018	6.4	28.0	4.9	1.5	0.0	0.0	0.0	0.0	0.0	0.0	1.0	27.0
168217	04/11/2018	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
168217	04/12/2018	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
168217	04/13/2018	2.0	1.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Figure 20. Flight History of BuNo 168217 from 4/1/2018 to 4/13/2018. Adapted from DECKPLATE NAVFLIR records with FlightHours.py (2022)/

The last step in generating the PCA flight data merged flight hours and landings with PCA serial numbers. The RemovalHistory.py script records the filtered removal MAFs with the cumulative flight hours and landings at removal. Figure 21 shows the final data for PCA serial number 77. The data is now ready for preprocessing and model training.

Action Date	Rmvl Inst PartNo	SerNo	Org	BuNo	Trans	Action	Mal Code	flthrs	lndgs	TMR 1	TMR 2	TMR 3	TMR 4	TMR 5	TMR 6	TMR 7	Non-Aust	Aust	Ship
06/17/2008	901-301-902-109	77	VMMT-204	165940	18	T	815	367.2	1289	324.2	43	0	0	0	0	0	73	1212	4
08/04/2008	901-301-902-109	77	VMMT-204	165943	23	R	70	407.7	1445	364.2	43.5	0	0	0	0	0	73	1368	4
03/02/2009	901-301-902-109	77	VMM-261	166390	11	N	0	407.7	1445	364.2	43.5	0	0	0	0	0	73	1368	4
03/02/2009	901-301-902-109	77	VMM-261	165944	23	R	20	407.7	1445	364.2	43.5	0	0	0	0	0	73	1368	4
09/03/2009	901-301-902-109	77	VMM-264	166390	18	T	815	435.9	1519	389.9	46	0	0	0	0	0	96	1419	4
11/21/2013	901-301-902-109	77	VMM-365	165944	23	R	295	450.6	1574	397.4	53.2	0	0	0	0	0	96	1474	4
12/15/2014	42555-400	77	VMM-365	165944	23	R	295	566.5	1868	449.4	117.1	0	0	0	0	0	100	1693	75

Figure 21. PCA Serial Number 77 Removal Data from RemovalHistory.py

D. TRAINING AND MODEL SELECTION

The models used for estimating the conditional probability of failure included the statistical Weibull analysis and the CPH model. Failure data was categorized by malfunction code, which divided usable data into smaller subsets for analysis. The malfunction code recorded on removal MAFs is not necessarily a specific failure mode, but the type of maintenance required (Department of the Navy, 2021). The V22 FST did Weibull analysis on seven failure modes for their dataset of 526 PCA failures. Their failure modes were determined by analyzing failure descriptions entered in the MAFs by maintainers. Although there were more than seven failure modes in their dataset, some subdivisions had too little data for meaningful analysis. This thesis took a similar approach and only analyzed malfunction codes whose subsets had twenty or more datapoints. Table 3 shows the malfunction codes of the resultant 1,144 failures.

Table 3. Malfunction Codes Analyzed

Mal Code	Description	# Failures	# Censored
Blank	Failed Technical Directive Inspection	63	1,081
135	Binding, Stuck, Jammed	100	1,044
20	Worn, Stripped, Chaffed, or Frayed – not wiring	158	986
290	Fails – diagnostic/automatic tests	38	1,106
295	Fails – check/test	264	880
374	Internal Failure – Foreign Object Damage	39	1,105
70	Broken, Burst, Ruptured, Punctured, Torn, or Cut	40	1,104
150	Chattering	22	1,122

Source: Department of the Navy (2021, p. E-11) and Models.py Python Script (2022).

For machine-learning, the data was split into training and test subsets. Because the data is censored, splits are done by the removal dates. Eighty percent of failures for a malfunction code were used as training data and the remainder were used for test data. Table 4 shows the date at which eighty percent failures for a malfunction code were recorded. Model training was performed on training failures and censored failures. Censored failures had not yet failed with the malfunction code being evaluated. All data points in the test data must be considered censored for model prediction. Therefore, the flight data for each serial number in the test data is not at failure, but rather the time date the data split occurred.

Table 4. Training and Test Data Split by Malfunction Code

Mal Code	Date at 80%	# Training Failures	# Censored	# Test Failures
Blank	3/30/2021	51	830	8
135	6/4/2013	81	727	16
20	8/23/2021	127	784	27
290	10/4/2017	31	928	6
295	11/23/2020	212	759	42
374	4/19/2013	32	767	5
70	7/1/2019	33	910	7
150	5/16/2020	18	969	4

Python libraries provide functions and methods for reliability analysis including Weibull analysis. The V22 FST at FRC East used a two-parameter Weibull model for the analysis they shared for this thesis. They fit the distribution with a rank-regression method. The Python Reliability library provides Weibull model fitting functions and plots for the Weibull PDF, CDF, hazard function, and cumulative hazard function. The Model.py script we wrote imported the PCA failure data and used the reliability library functions for the Weibull analysis. Distributions are modeled by malfunction code, while all other failures are considered right censored.

While determining conditional probability of failure for future RCM policy is the goal, model evaluation was done on the distribution function. The Weibull distribution is defined by the shape and character parameters that best fit the data. To estimate those parameters, fit parameters were adjusted. The methods of fitting were the mean-likelihood estimation (MLE), rank-regression on x (RRX), or rank-regression on y RRY (Reid, 2022). For this data, x is the failure time and y is the unreliability estimate. The default confidence interval for estimating confidence limits was 95%. Models were also fitted with different confidence intervals. The best performing distribution was determined by the log-likelihood function; the higher the value, the better the model fit. A measure of the badness of fit of the distribution is the Akaike Information Criteria (AIC). This value indicates the bias of the log-likelihood; a smaller value indicates less over-fitting to the data (Konishi, & Kitagawa, 2008).

The Lifelines Python library provides functions and methods for CPH analysis. Data was split similarly to Chapter IV Section D, but included the additional multivariate flight data. Once again, analysis of the effects of parameter adjustment were done on the resultant distribution. Besides adjusting the confidence interval, a penalizer is available to penalize coefficients that are highly correlated (Davidson-Pilon, 2019). The best performing distribution was determined by the log-likelihood function.

E. FINAL TESTING

Once the Weibull and CPH models were trained, predictions were made on the remaining test data for each malfunction code. The performance of each model was

evaluated using RMSE. Lastly the conditional probability of failure was plotted using the equation in Chapter II, Section B for survival flight hours.

V. RESULTS AND ANALYSIS

This chapter discusses the findings of the work done. The first analysis compares the time-to-failure (TTF) data generated with the model to the data used by the V22 FST and the performance of the resultant Weibull analyses. Next, the Weibull analysis was done by malfunction code on all the failure data generated in this thesis. Independent models were fit adjusting methods of fit to include rank regression (RRX and RRY), mean likelihood (MLE), and confidence intervals. Lastly, Cox Proportional-Hazards (CPH)models were independently trained while adjusting hyperparameters to include confidence intervals and penalizers. Results show that the CPH models perform better than the Weibull analysis.

A. WEIBULL ANALYSIS COMPARISON

In this phase, the Weibull analysis was done only on failure data that matched those found by the V22 FST. The Maintenance Action Form (MAF) control number (MCN) is a unique code that identifies the form instance. The data generated in Chapter IV of this thesis found 1,144 MCNs for PCA actuator failures resulting in removal. The V22 FST found 546 unique MAFs; however, their latest datapoint was in February 2022 and included failure data from the CV-22. The data generated in this thesis captured MV-22B data only, up to August 2022. Of the 466 MV-22B MAFs captured by the V22 FST, 439 were also captured in this thesis. The average difference in the mean time to failure (MTTF) was 203 flight hours. Table 5 shows the comparison of Weibull analyses done by the V22 FST and those done in this thesis by failure mode. The number of failures by mode, resultant shape and character parameters, coefficient of determination, r^2, and proportional reporting ratio (PRR) are depicted.

Table 5. V22 Fleet Support Team Weibull Model Results

Failure Mode	**Data**	**# Failed**	**MTTF**	**Beta**	**Eta**	**r^2**	**PRR**
Mode 01: Internal Wear Before Scheduled Greasing	V22 FST	84	2089.39	1.99	2357.49	0.93	2.02
	Thesis	69	2290.86	2.08	2586.32	0.95	0
Mode 02: Soft Stop	V22 FST	59	3165.5	1.62	3534.10	0.88	0
	Thesis	51	2696.25	1.93	3040.08	0.98	0
Mode 14: Internal Wear Post Scheduled Greasing	V22 FST	79	3501.40	1.32	3801.88	0.93	2.61
	Thesis	66	2301.41	2.07	2598.07	0.99	0
Mode 15: Cross-Over Jam	V22 FST	43	3982.88	1.49	4408.02	0.98	66.73
	Thesis	43	4479.19	1.47	4950.73	0.97	0
Mode 16: Inefficient Ballscrew	V22 FST	81	2325.30	1.78	2613.17	0.94	3.42
	Thesis	70	2364.38	2.11	2669.58	0.94	0
Mode 16: Ratcheting	V22 FST	80	3128.33	1.42	3438.86	0.88	0
	Thesis	53	2601.84	2.14	2937.87	0.98	0
Mode 21: Seal Damage	V22 FST	20	2766.71	2.75	3109.11	0.92	13.96
	Thesis	17	4143.92	2.18	4679.19	0.94	0

Adapted from data provided by the V22 FST and the Models.py script (2022).

The Cross-Over Jam failure mode analysis for this thesis included all 43 MAFs captured by the V22 FST. Figure 22 and Figure 23 are the PDF plots of the Weibull analyses for this failure mode from the V22 FST and this thesis, respectively. Both used rank regression on Y fit and a 95% confidence interval. The PDF plots for all V22 FST failure modes and our corresponding analyses are show in Appendix A.

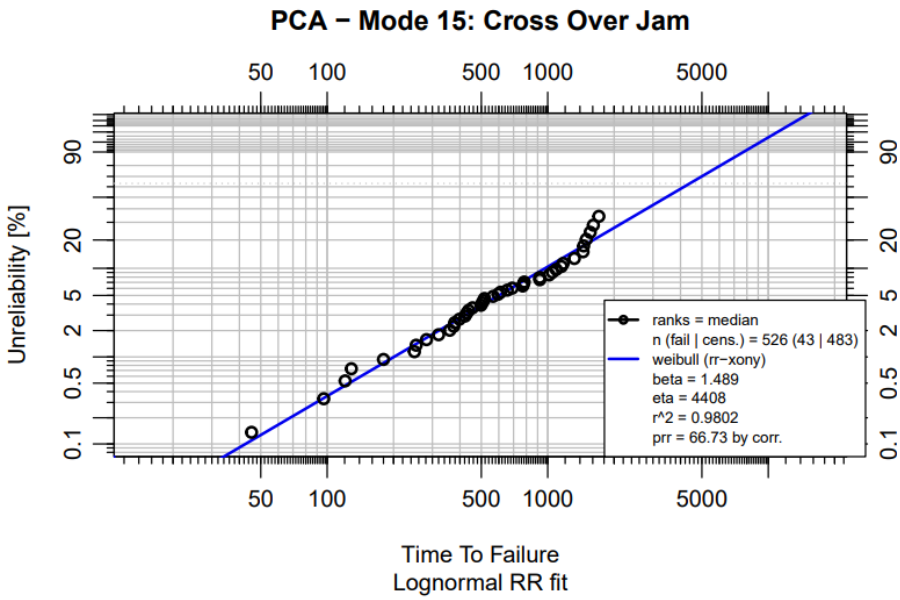

Figure 22. V22 FST Weibull PDF of PCA Mode 15: Cross-Over Jam. Source: FRC East V22 FST Maintenance Optimization (2022).

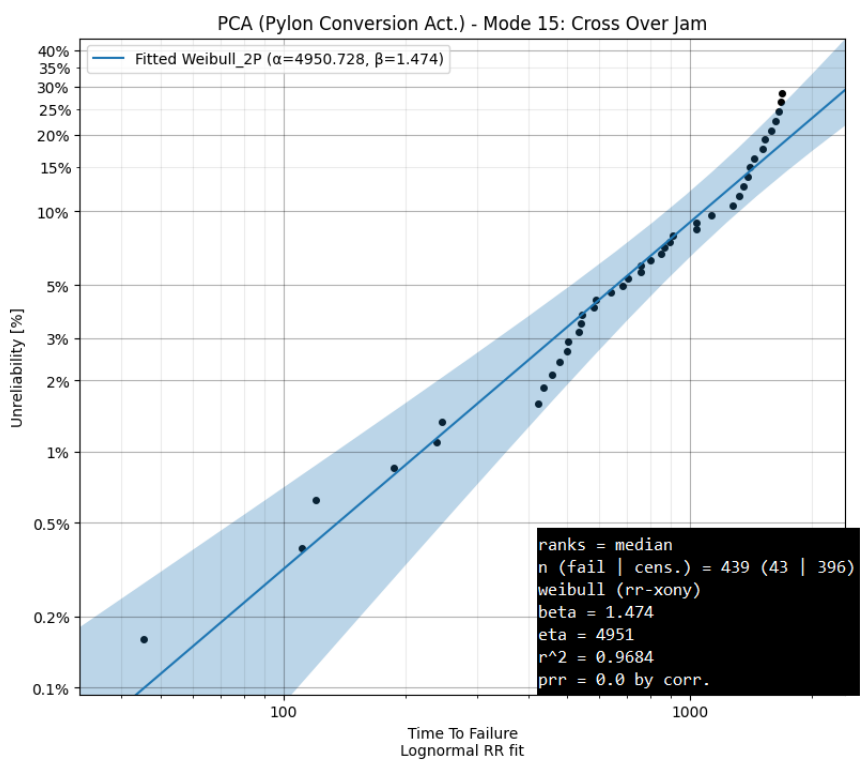

Figure 23. Thesis Weibull PDF of PCA Mode 15: Cross-Over Jam. Source: Models.py Python Script (2022).

B. WEIBULL MODELS WITH DECKPLATE DATA

The second phase of this thesis applied Weibull analysis to failure data generated in Chapter IV with twenty or more datapoints. The model fits included RRX, RRY, and MLE. Confidence intervals ranged from 0.5 to 1.0 in steps of 0.05. To compare relative model performance, the log-likelihood and AIC were recorded. For each malfunction code, the model that produced the highest log-likelihood and the model that produced the lowest AIC were chosen. In all cases, the model with least negative lowest log-likelihood also had the smallest AIC. For each malfunction mode evaluated, the best performing model used the MLE method of fit with a confidence of 95%. Table 6 shows the hyperparameters of the best performing models for each malfunction code.

Table 6. Best Performing Weibull PDF by Malfunction Code

Malfunction Code	**# Failures**	**# Censored**	**Fit Method**	**Conf. Int.**	**Log-Likelihood**	**AIC**
20	63	1081	MLE	0.95	-1482.54	2969.09
70	100	1044	MLE	0.95	-444.02	892.05
135	158	986	MLE	0.95	-1002.92	2009.86
150	38	1106	MLE	0.95	-249.03	502.07
290	264	880	MLE	0.95	-415.88	835.78
295	39	1105	MLE	0.95	-2379.5	4763.02
374	40	1104	MLE	0.95	-426.65	857.3
Blank (TD)	22	1122	MLE	0.95	-676.8	1357.61

Adapted from Models.py Python Script (2022).

As the confidence interval hyperparameter increases, the y-intersection point (unreliability) of the beta values on the plotted PDFs increases (Quanterion Solutions, 2015). For applications of the Weibull analysis, a higher confidence interval may overestimate the percentage of components that have failed at a given service life. In this research, the differences between models using different confidence intervals were minimal. The full table of Weibull models can be found at the end of Appendix B. The MLE fit outperformed both rank-regression methods in every case. Of the seven malfunction codes evaluated, the most datapoints in the subsets was 264. Therefore, at least

77% of datapoints were censored for each malfunction code. The MLE method is generally better at handling highly censored data (ReliaSoft Corporation, 2007).

C. CPH MODELS

The final phase of this thesis applied the CPH analysis methodology to training and test data generated in Chapter IV. The first step in this phase was to train CPH models on the training data subset, evaluate relative performance, and validate assumptions for failure modes with twenty or more datapoints. The confidence interval was tested from 0.5 to 1.0 in steps of 0.05 and the penalizer from 0.15 to 0 in steps of 0.05. To compare relative model performance, the log-likelihood and AIC were recorded. For each malfunction code, the model that produced the highest log-likelihood and the model that produced the lowest AIC were chosen. Table 7 shows the hyperparameters of the best performing models for each malfunction code.

Table 7. Best Performing CPH Model by Malfunction Code

Malfunction Code	**# Failures**	**# Censored**	**Conf. Int.**	**Penalty**	**Log-Likelihood**	**AIC**
20	51	830	0.95	0.0	-386.91	793.82
70	81	727	0.95	0.0	-99.55	219.09
135	127	784	0.95	0.0	-241.93	503.86
150	31	928	0.95	0.0	-52	124
290	212	759	0.95	0.0	-87.98	195.96
295	32	767	0.95	0.0	-641.15	1302.31
374	33	910	0.95	0.0	-87.71	195.42
Blank (TD)	18	969	0.95	0.0	-162.03	344.05

Adapted from Models.py Python Script (2022).

In all cases, the model with least negative lowest log-likelihood also had the smallest AIC. For each malfunction mode evaluated, the best performing model used a confidence interval of 95% and zero penalty. The penalizer prevented models using MLE from overfitting (Pampuri, De Luca, & De Nicolao, 2011). This is a good indication that the models are less likely to be overfitted to the training data. The full table of CPH models can be found at the end of Appendix B.

Section III.D.3 of this thesis assumes that the ratio between covariate hazard rates is constant over time for CPH analysis. The Lifelines library provides a method to check these assumptions using the Schoenfeld residuals (Davidson-Pilon, 2019). Using a threshold of 0.05, on three occasions covariate p-values violated the CPH assumption: ship landings for malfunction code 20, austere landings for malfunction code 70, and TMR_7 hours for removals due to technical-directive inspection failures. Figure 24 shows Schoenfeld residuals of ship landings for malfunction code 20. The sums of the covariate residuals are depicted on vertical axis and the rank and Kaplan Meir-transformed expected survival times are depicted on the horizontal axis. Schoenfeld plots for malfunction code 70 and technical-directive inspection failures are in Appendix C.

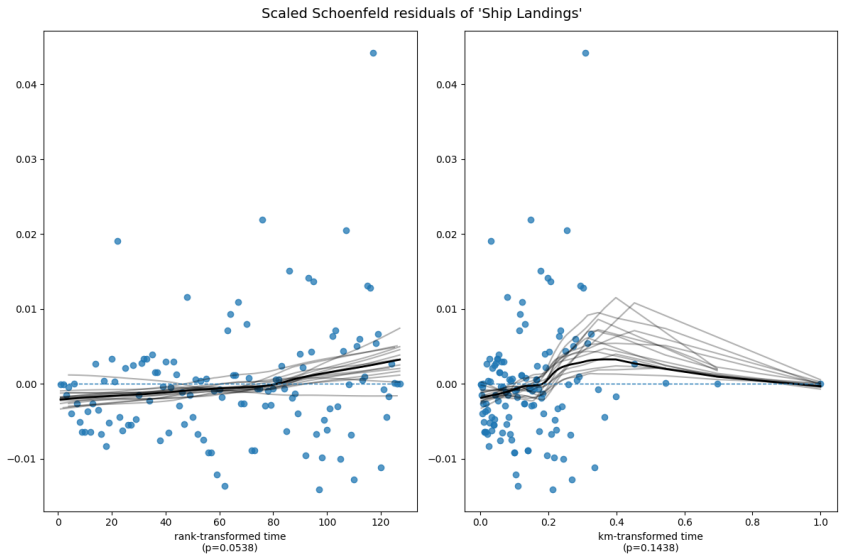

Figure 24. Scaled Schoenfeld Residuals of Ship Landings for Malfunction Code 20 - Worn, Stripped, Chaffed, or Frayed – Not Wiring.
Source: Models.py Python Script (2022).

Although these three models violated the CPH assumption, they are not necessarily unacceptable if no "more correct" model exists to define the distribution (Tai & Machin, 2014). These three models were run again on the training data, excluding the corresponding covariate that failed the CPH assumption. Table 8 shows the relative performance of

models. All three models passed the CPH assumption check, but their difference in performance is negligible.

Table 8. CPH Assumption Model Comparison

Malfunction Code	**Training Data**	**Log-Likelihood**	**AIC**	**Satisfies CPH Assumption?**
20	With Ship Landings	-386.91	793.82	No
	Without Ship Landings	-386.92	791.84	Yes
70	With Austere Landings	-99.55	219.09	No
	Without Austere Landings	-99.62	217.24	Yes
Blank (TD)	With TMR_7	-162.03	344.05	No
	Without TMR_7	-161.97	341.93	Yes

The predicted survival functions from the CPH models were plotted with the test data using the Lifelines library (Davidson-Pilon, 2019). Figure 25 show the probability that each PCA with malfunction code 70 in the test data has failed as a function of flight hours. The legend shows the true time to failure for each PCA, which can be compared to the corresponding survival function. All predicted survival plots are shown in Appendix C.

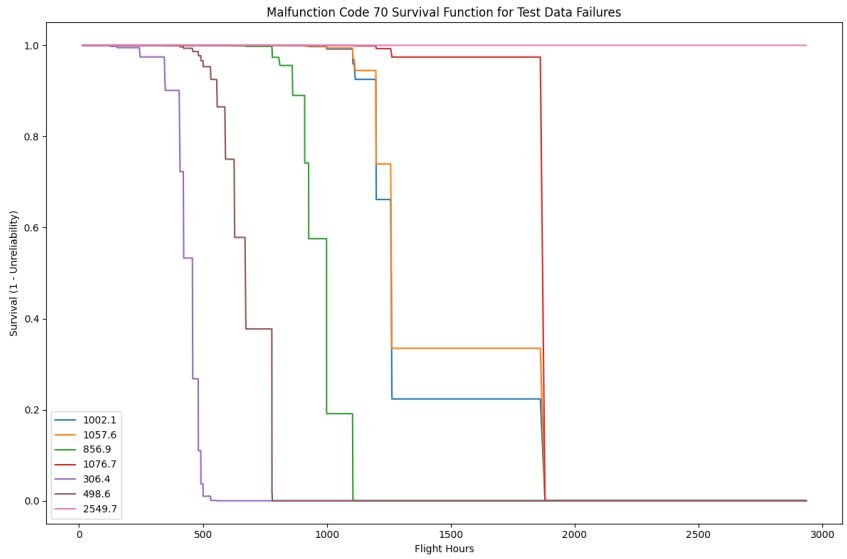

Figure 25. CPH Predicted Survival for Malfunction Code 70 Test Data.
Source: Models.py Python Script (2022).

The final step in this phase was to compare the best performing CPH to the best performing Weibull models for each malfunction code. The CPH models were re-trained on all failure data like the Weibull models with the hyperparameters chosen from Table 7. In all cases, the CPH models had a less negative log-likelihood and lower AIC, which indicated they were better.

Table 9. Weibull and CPH Performance Comparison

Malfunction Code	**Model**	**Log-Likelihood**	**AIC**
20	Weibull	-1482.54	2969.09
	CHP	-511.28	1042.57
70	Weibull	-444.02	892.05
	CHP	-127.1	274.19
135	Weibull	-1002.92	2009.86
	CHP	-329.2	678.4
150	Weibull	-249.03	502.07
	CHP	-69.51	159.03
290	Weibull	-415.88	835.78
	CHP	-116.19	252.38
295	Weibull	-2379.5	4763.02
	CHP	-843.94	1707.89
374	Weibull	-426.65	857.3
	CHP	-121.91	263.83
Blank (TD)	Weibull	-676.8	1357.61
	CHP	-213.58	447.16

VI. CONCLUSIONS AND FUTURE WORK

A. CONCLUSIONS

In this research we have shown a better way that naval-aviation data can be used with machine learning to determine the conditional probability of failure of aircraft components. By merging flight data with component-failure data, multivariate machine-learning models can fit failure distributions with a better log-likelihood than the Weibull model. On the average, the log-likelihood of Cox Proportional-Hazards (CPH) models was 31% better than the Weibull models. The CPH model can also handle the highly censored maintenance records for complex aircraft components without violating assumptions beyond an acceptable level. On average, 92% of failures were right-censored for CPH model training. For the covariates that did violate CPH assumptions, their impact on model performance we negligible.

We evaluated data repositories in DECKPLATE to identify solutions for merging aircraft flight data with components installed during those flights. Through our analysis, we found it useful to categorize flight data as non-austere, austere, and shipboard-operation based on Total Mission Requirement (TMR) and landing codes. Serial-number tracking can be evaluated to determine approximately on what dates components were installed on aircraft. Failure modes can be categorized by the malfunction code, transaction code, and action-taken code, and censored for modelling. A comparison of Weibull and CPH models was made for the same PCA failures identified by the V22 FST using flight data generated in this thesis. Although the CPH models clearly performed better, this is not surprising in comparing univariate and multivariate models.

The Weibull models were run with a wide range of hyperparameters including confidence intervals and methods of fitting. The confidence interval had little significance on performance, but mean-likelihood estimation always did best due to the high percentage of censored data. The CPH models were also run with a wide range of confidence intervals and penalizer values, but a 95% confidence interval and zero penalty always did best. The resultant survival function of trained CPH models often overestimated the time to failure

of test data. This shows that splitting censored failure data by a set date may not be the best method. Data points whose failures were closer to the date on which the split occurred tended to be overestimated more than others.

B. FUTURE WORK

This research met the objectives of developing a methodology to extract meaningful component history from maintenance records and apply machine-learning models to predict the conditional probability of failure. However, challenges still arise that require more research. The number of installation and removal imputations required to give a complete picture of a component's service history is a concern. We believe that the survival function overestimating the time to failure of test data can be attributed to assumptions made in data imputation and the method for splitting censored data. The DECKPLATE electronic-log database is incomplete, which could be rectified by squadron-maintenance work centers having physical log sets of components and their maintenance history.

Due to resource limitations, the neural-network models were not published in this thesis. Time-series data could be extracted from the same repositories used in this thesis as inputs for long short-term memory (LSTM) or other recurrent neural-network architectures. Additionally, flight data are not the only continuous variables available in these repositories for model training. We recommend including maintenance factors such as the number of times a component has received certain types of maintenance to see how they impact the survival function of those components.

Lastly, the V22 FST did a comprehensive failure-mode analysis of the MV-22B PCA actuator records. With 31 tracked subassemblies and far more individual components, a PCA failure cannot be accurately described by the general malfunction codes used in naval aviation. The V22 FST manually reads the system and failure descriptions entered as remarks by maintainers to determine the precise failure mode that justified removal. Text analysis to classify failure modes from the MAF form remarks using artificial intelligence is an opportunity to better categorize and censor failure data.

APPENDIX A. WEIBULL MODEL COMPARISONS

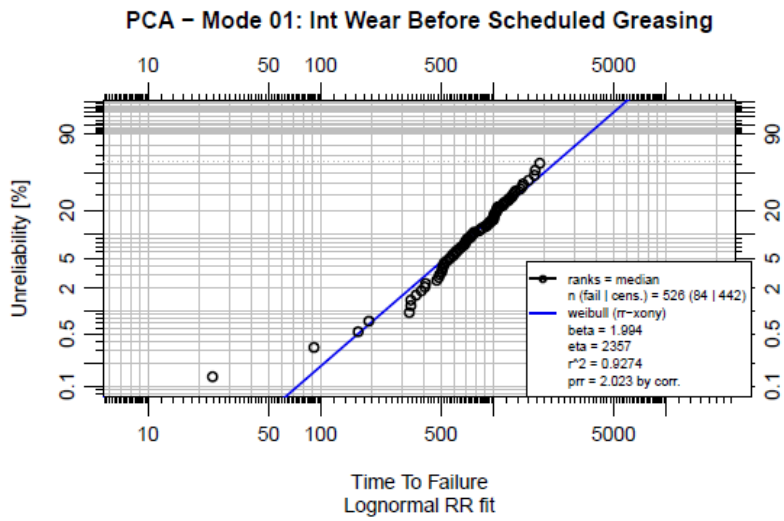

Figure 26. Weibull PDF of PCA Mode 1: Internal Wear Before Scheduled Greasing.
Source: FRC East V22 FST Maintenance Optimization (2022).

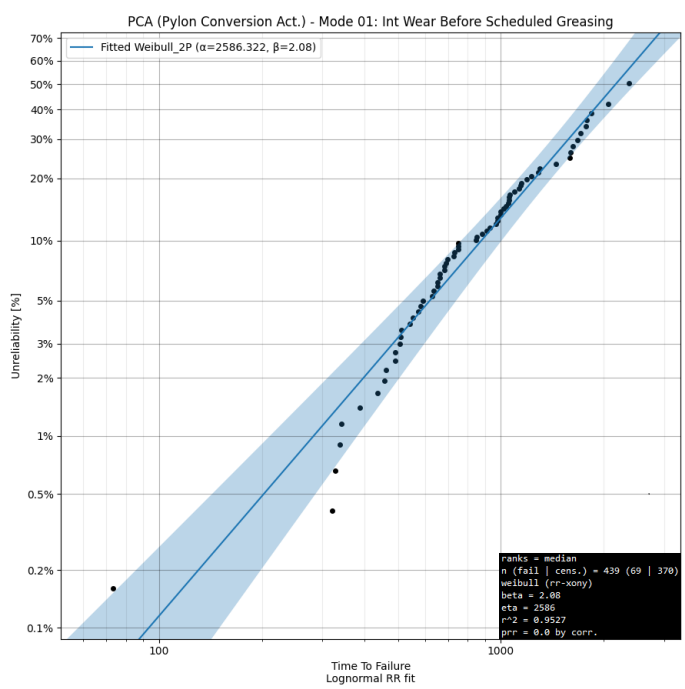

Figure 27. Weibull PDF of PCA Mode 1: Internal Wear Before Scheduled Greasing.
Source: Models.py Python Script (2022).

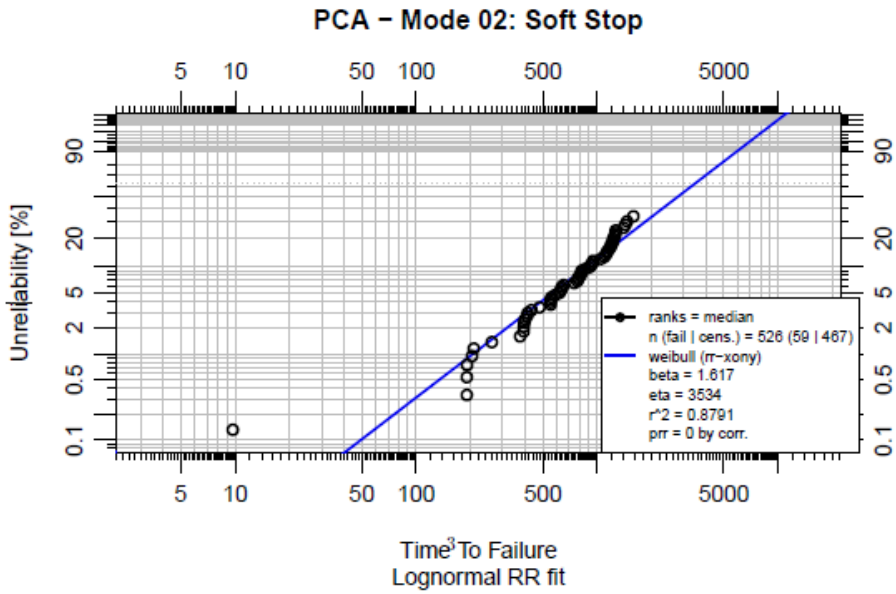

Figure 28. V22 FST Weibull PDF of PCA Mode 2: Soft Stop. Source: FRC East V22 FST Maintenance Optimization (2022).

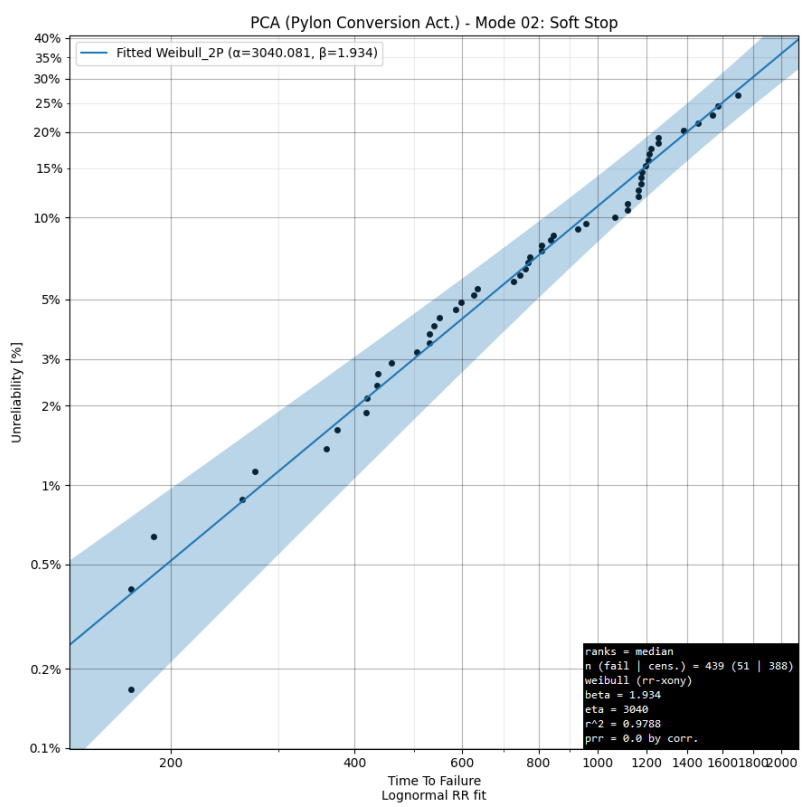

Figure 29. Thesis Weibull PDF of PCA Mode 2: Soft Stop. Source: Models.py Python Script (2022).

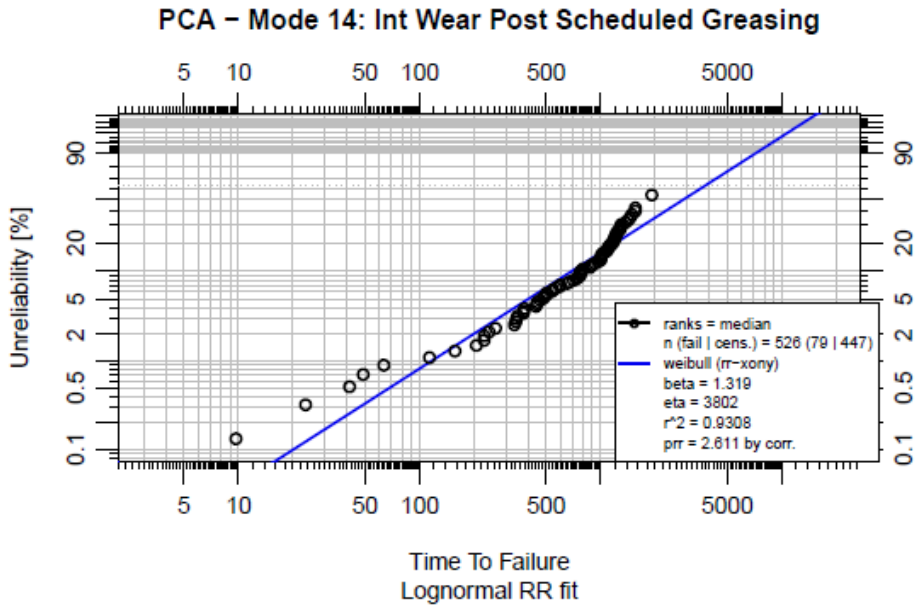

Figure 30. V22 FST Weibull PDF of PCA Mode 14: Internal Wear Post Scheduled Greasing.
Source: FRC East V22 FST Maintenance Optimization (2022).

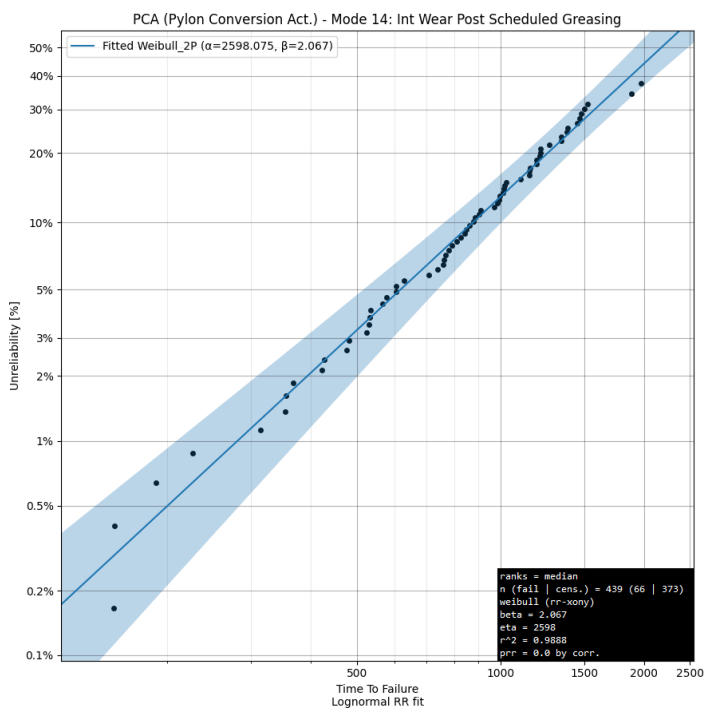

Figure 31. Thesis Weibull PDF of PCA Mode 14: Internal Wear Post Scheduled Greasing.
Source: Models.py Python Script (2022).

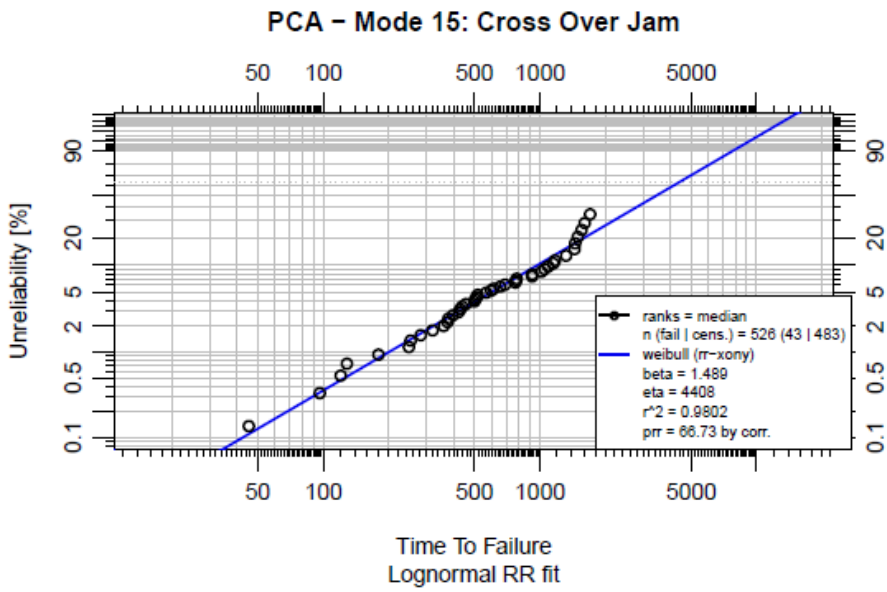

Figure 32. V22 FST Weibull PDF of PCA Mode 15: Cross-Over Jam. Source: FRC East V22 FST Maintenance Optimization (2022).

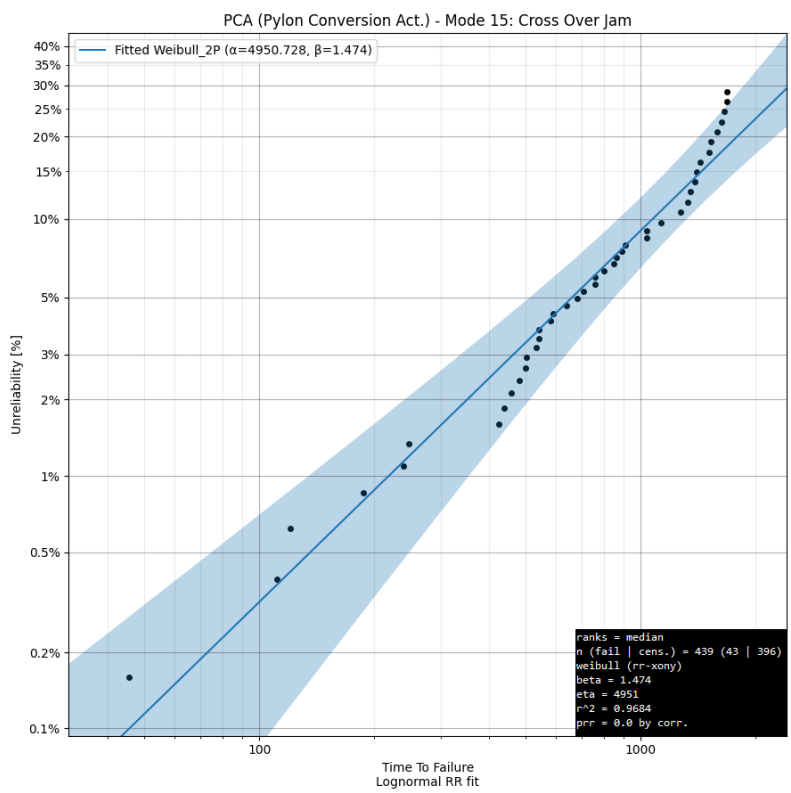

Figure 33. Thesis Weibull PDF of PCA Mode 15: Cross-Over Jam. Source: Models.py Python Script (2022).

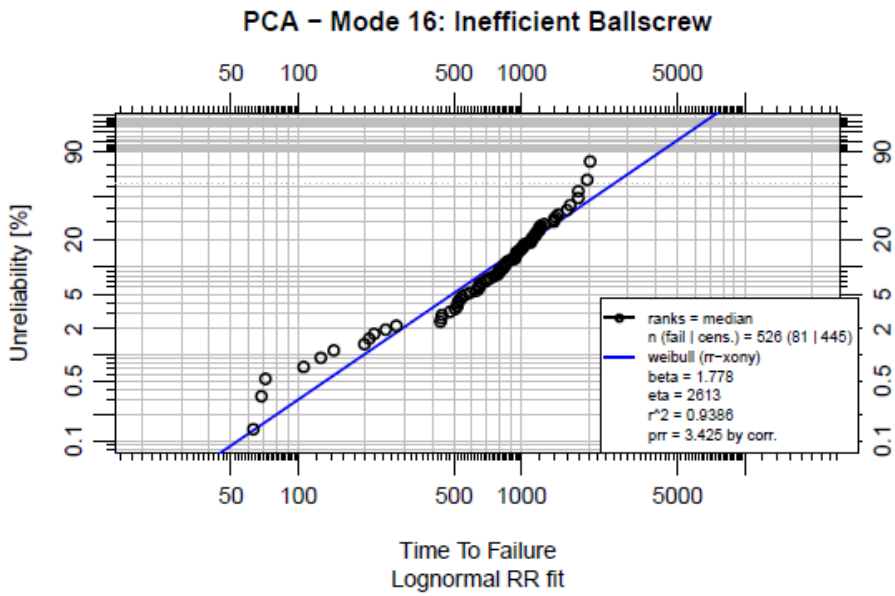

Figure 34. V22 FST Weibull PDF of PCA Mode 16: Insufficient Ballscrew. Source: FRC East V22 FST Maintenance Optimization (2022).

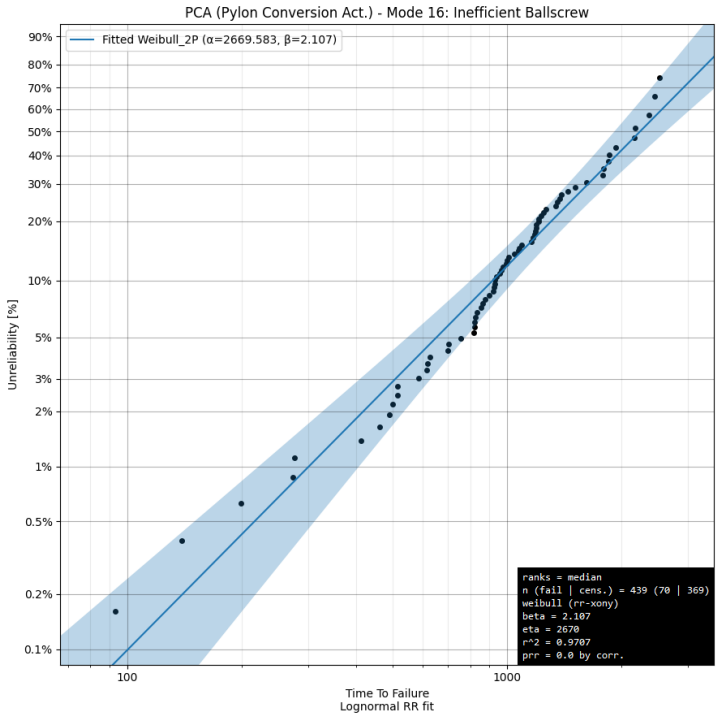

Figure 35. Thesis Weibull PDF of PCA Mode 16: Insufficient Ballscrew. Source: Models.py Python Script (2022).

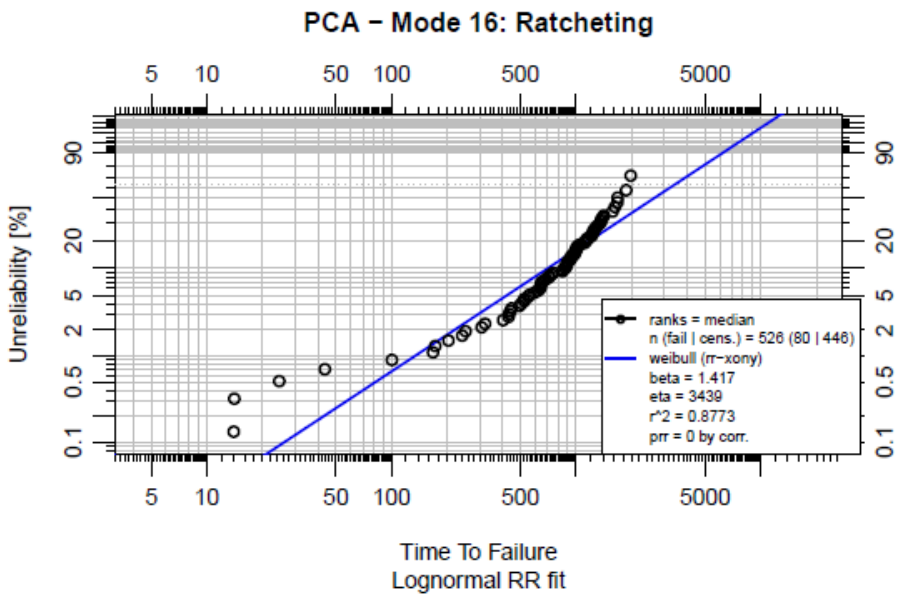

Figure 36. V22 FST Weibull PDF of PCA Mode 16: Ratcheting.
Source: FRC East V22 FST Maintenance Optimization (2022).

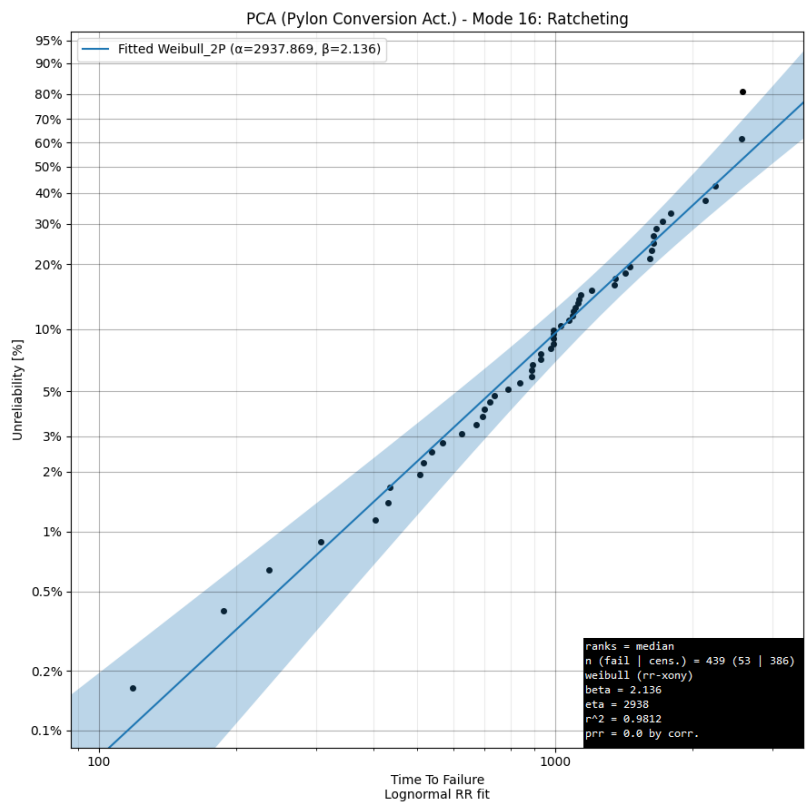

Figure 37. Thesis Weibull PDF of PCA Mode 16: Ratcheting.
Source: Models.py Python Script (2022).

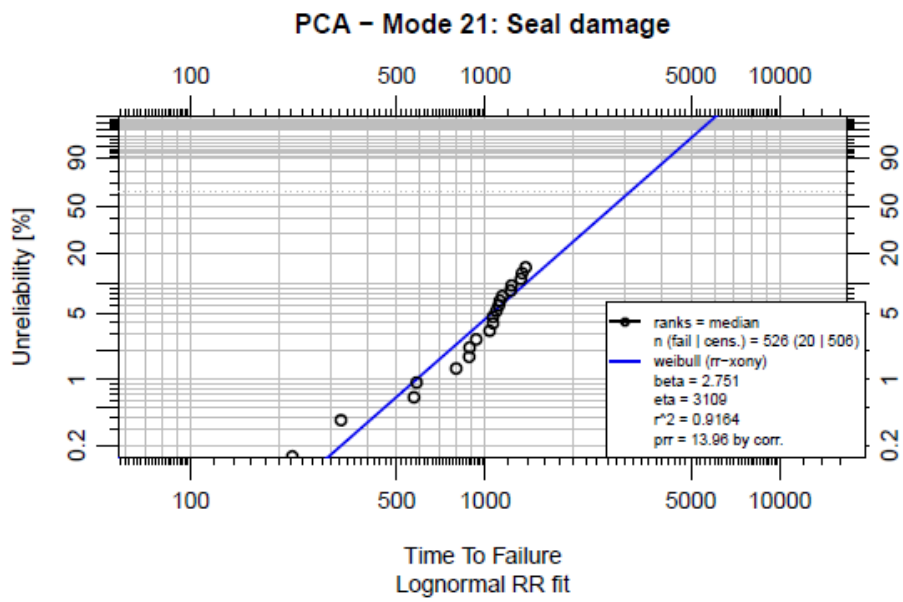

Figure 38. V22 FST Weibull PDF of PCA Mode 21: Seal Damage. Source: FRC East V22 FST Maintenance Optimization (2022).

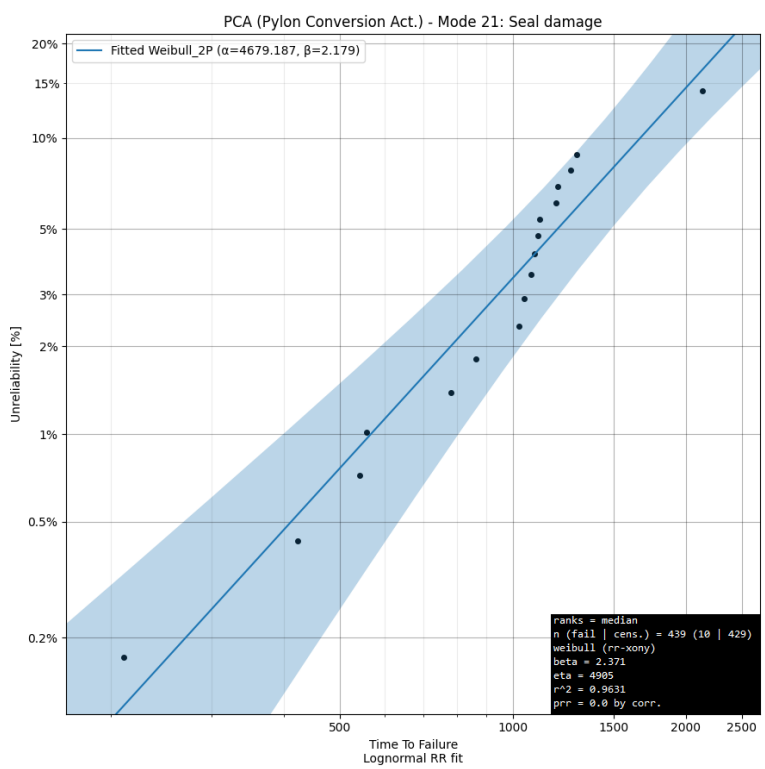

Figure 39. Thesis Weibull PDF of PCA Mode 21: Seal Damage. Source: Models.py Python Script (2022).

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. WEIBULL AND CPH PERFORMANCE

Table 10. Weibull Model Performance by Failure Mode and Hyperparameter

Malfunction Code	Fit Method	Conf. In.t	MTTF	Log-Likelihood	AIC
20	MLE	0.95	2293.71	-1482.54	2969.09
20	MLE	0.9	2293.71	-1482.54	2969.09
20	MLE	0.85	2293.71	-1482.54	2969.09
20	MLE	0.8	2293.71	-1482.54	2969.09
20	MLE	0.75	2293.71	-1482.54	2969.09
20	MLE	0.7	2293.71	-1482.54	2969.09
20	MLE	0.65	2293.71	-1482.54	2969.09
20	MLE	0.6	2293.71	-1482.54	2969.09
20	MLE	0.55	2293.71	-1482.54	2969.09
20	MLE	0.5	2293.71	-1482.54	2969.09
20	RRX	0.95	2213.06	-1482.84	2969.7
20	RRX	0.9	2213.06	-1482.84	2969.7
20	RRX	0.85	2213.06	-1482.84	2969.7
20	RRX	0.8	2213.06	-1482.84	2969.7
20	RRX	0.75	2213.06	-1482.84	2969.7
20	RRX	0.7	2213.06	-1482.84	2969.7
20	RRX	0.65	2213.06	-1482.84	2969.7
20	RRX	0.6	2213.06	-1482.84	2969.7
20	RRX	0.55	2213.06	-1482.84	2969.7
20	RRX	0.5	2213.06	-1482.84	2969.7
20	RRY	0.95	2412.89	-1483.06	2970.13
20	RRY	0.9	2412.89	-1483.06	2970.13
20	RRY	0.85	2412.89	-1483.06	2970.13
20	RRY	0.8	2412.89	-1483.06	2970.13
20	RRY	0.75	2412.89	-1483.06	2970.13
20	RRY	0.7	2412.89	-1483.06	2970.13
20	RRY	0.65	2412.89	-1483.06	2970.13
20	RRY	0.6	2412.89	-1483.06	2970.13
20	RRY	0.55	2412.89	-1483.06	2970.13
20	RRY	0.5	2412.89	-1483.06	2970.13
70	MLE	0.95	8566.98	-444.02	892.05
70	MLE	0.9	8566.98	-444.02	892.05
70	MLE	0.85	8566.98	-444.02	892.05
70	MLE	0.8	8566.98	-444.02	892.05
70	MLE	0.75	8566.98	-444.02	892.05

Malfunction Code	Fit Method	Conf. In.t	MTTF	Log-Likelihood	AIC
70	MLE	0.7	8566.98	-444.02	892.05
70	MLE	0.65	8566.98	-444.02	892.05
70	MLE	0.6	8566.98	-444.02	892.05
70	MLE	0.55	8566.98	-444.02	892.05
70	MLE	0.5	8566.98	-444.02	892.05
70	RRX	0.95	9363.9	-444.15	892.31
70	RRX	0.9	9363.9	-444.15	892.31
70	RRX	0.85	9363.9	-444.15	892.31
70	RRX	0.8	9363.9	-444.15	892.31
70	RRX	0.75	9363.9	-444.15	892.31
70	RRX	0.7	9363.9	-444.15	892.31
70	RRX	0.65	9363.9	-444.15	892.31
70	RRX	0.6	9363.9	-444.15	892.31
70	RRX	0.55	9363.9	-444.15	892.31
70	RRX	0.5	9363.9	-444.15	892.31
70	RRY	0.95	10376.32	-444.24	892.5
70	RRY	0.9	10376.32	-444.24	892.5
70	RRY	0.85	10376.32	-444.24	892.5
70	RRY	0.8	10376.32	-444.24	892.5
70	RRY	0.75	10376.32	-444.24	892.5
70	RRY	0.7	10376.32	-444.24	892.5
70	RRY	0.65	10376.32	-444.24	892.5
70	RRY	0.6	10376.32	-444.24	892.5
70	RRY	0.55	10376.32	-444.24	892.5
70	RRY	0.5	10376.32	-444.24	892.5
135	MLE	0.95	3366.62	-1002.92	2009.86
135	MLE	0.9	3366.62	-1002.92	2009.86
135	MLE	0.85	3366.62	-1002.92	2009.86
135	MLE	0.8	3366.62	-1002.92	2009.86
135	MLE	0.75	3366.62	-1002.92	2009.86
135	MLE	0.7	3366.62	-1002.92	2009.86
135	MLE	0.65	3366.62	-1002.92	2009.86
135	MLE	0.6	3366.62	-1002.92	2009.86
135	MLE	0.55	3366.62	-1002.92	2009.86
135	MLE	0.5	3366.62	-1002.92	2009.86
135	RRY	0.95	3103.17	-1003.31	2010.63
135	RRY	0.9	3103.17	-1003.31	2010.63
135	RRY	0.85	3103.17	-1003.31	2010.63
135	RRY	0.8	3103.17	-1003.31	2010.63

Malfunction Code	Fit Method	Conf. In.t	MTTF	Log-Likelihood	AIC
135	RRY	0.75	3103.17	-1003.31	2010.63
135	RRY	0.7	3103.17	-1003.31	2010.63
135	RRY	0.65	3103.17	-1003.31	2010.63
135	RRY	0.6	3103.17	-1003.31	2010.63
135	RRY	0.55	3103.17	-1003.31	2010.63
135	RRY	0.5	3103.17	-1003.31	2010.63
135	RRX	0.95	3014.41	-1003.67	2011.34
135	RRX	0.9	3014.41	-1003.67	2011.34
135	RRX	0.85	3014.41	-1003.67	2011.34
135	RRX	0.8	3014.41	-1003.67	2011.34
135	RRX	0.75	3014.41	-1003.67	2011.34
135	RRX	0.7	3014.41	-1003.67	2011.34
135	RRX	0.65	3014.41	-1003.67	2011.34
135	RRX	0.6	3014.41	-1003.67	2011.34
135	RRX	0.55	3014.41	-1003.67	2011.34
135	RRX	0.5	3014.41	-1003.67	2011.34
150	MLE	0.95	5072.81	-249.03	502.07
150	MLE	0.9	5072.81	-249.03	502.07
150	MLE	0.85	5072.81	-249.03	502.07
150	MLE	0.8	5072.81	-249.03	502.07
150	MLE	0.75	5072.81	-249.03	502.07
150	MLE	0.7	5072.81	-249.03	502.07
150	MLE	0.65	5072.81	-249.03	502.07
150	MLE	0.6	5072.81	-249.03	502.07
150	MLE	0.55	5072.81	-249.03	502.07
150	MLE	0.5	5072.81	-249.03	502.07
150	RRX	0.95	5070.09	-249.13	502.26
150	RRX	0.9	5070.09	-249.13	502.26
150	RRX	0.85	5070.09	-249.13	502.26
150	RRX	0.8	5070.09	-249.13	502.26
150	RRX	0.75	5070.09	-249.13	502.26
150	RRX	0.7	5070.09	-249.13	502.26
150	RRX	0.65	5070.09	-249.13	502.26
150	RRX	0.6	5070.09	-249.13	502.26
150	RRX	0.55	5070.09	-249.13	502.26
150	RRX	0.5	5070.09	-249.13	502.26
150	RRY	0.95	5583.61	-249.17	502.35
150	RRY	0.9	5583.61	-249.17	502.35
150	RRY	0.85	5583.61	-249.17	502.35

Malfunction Code	Fit Method	Conf. In.t	MTTF	Log-Likelihood	AIC
150	RRY	0.8	5583.61	-249.17	502.35
150	RRY	0.75	5583.61	-249.17	502.35
150	RRY	0.7	5583.61	-249.17	502.35
150	RRY	0.65	5583.61	-249.17	502.35
150	RRY	0.6	5583.61	-249.17	502.35
150	RRY	0.55	5583.61	-249.17	502.35
150	RRY	0.5	5583.61	-249.17	502.35
290	MLE	0.95	5066.34	-415.88	835.78
290	MLE	0.9	5066.34	-415.88	835.78
290	MLE	0.85	5066.34	-415.88	835.78
290	MLE	0.8	5066.34	-415.88	835.78
290	MLE	0.75	5066.34	-415.88	835.78
290	MLE	0.7	5066.34	-415.88	835.78
290	MLE	0.65	5066.34	-415.88	835.78
290	MLE	0.6	5066.34	-415.88	835.78
290	MLE	0.55	5066.34	-415.88	835.78
290	MLE	0.5	5066.34	-415.88	835.78
290	RRX	0.95	6524.2	-416.64	837.28
290	RRX	0.9	6524.2	-416.64	837.28
290	RRX	0.85	6524.2	-416.64	837.28
290	RRX	0.8	6524.2	-416.64	837.28
290	RRX	0.75	6524.2	-416.64	837.28
290	RRX	0.7	6524.2	-416.64	837.28
290	RRX	0.65	6524.2	-416.64	837.28
290	RRX	0.6	6524.2	-416.64	837.28
290	RRX	0.55	6524.2	-416.64	837.28
290	RRX	0.5	6524.2	-416.64	837.28
290	RRY	0.95	7786.52	-417.5	839.01
290	RRY	0.9	7786.52	-417.5	839.01
290	RRY	0.85	7786.52	-417.5	839.01
290	RRY	0.8	7786.52	-417.5	839.01
290	RRY	0.75	7786.52	-417.5	839.01
290	RRY	0.7	7786.52	-417.5	839.01
290	RRY	0.65	7786.52	-417.5	839.01
290	RRY	0.6	7786.52	-417.5	839.01
290	RRY	0.55	7786.52	-417.5	839.01
290	RRY	0.5	7786.52	-417.5	839.01
295	MLE	0.95	1967.81	-2379.5	4763.02
295	MLE	0.9	1967.81	-2379.5	4763.02

Malfunction Code	Fit Method	Conf. In.t	MTTF	Log-Likelihood	AIC
295	MLE	0.85	1967.81	-2379.5	4763.02
295	MLE	0.8	1967.81	-2379.5	4763.02
295	MLE	0.75	1967.81	-2379.5	4763.02
295	MLE	0.7	1967.81	-2379.5	4763.02
295	MLE	0.65	1967.81	-2379.5	4763.02
295	MLE	0.6	1967.81	-2379.5	4763.02
295	MLE	0.55	1967.81	-2379.5	4763.02
295	MLE	0.5	1967.81	-2379.5	4763.02
295	RRX	0.95	1987.42	-2379.57	4763.15
295	RRX	0.9	1987.42	-2379.57	4763.15
295	RRX	0.85	1987.42	-2379.57	4763.15
295	RRX	0.8	1987.42	-2379.57	4763.15
295	RRX	0.75	1987.42	-2379.57	4763.15
295	RRX	0.7	1987.42	-2379.57	4763.15
295	RRX	0.65	1987.42	-2379.57	4763.15
295	RRX	0.6	1987.42	-2379.57	4763.15
295	RRX	0.55	1987.42	-2379.57	4763.15
295	RRX	0.5	1987.42	-2379.57	4763.15
295	RRY	0.95	1994.67	-2379.6	4763.21
295	RRY	0.9	1994.67	-2379.6	4763.21
295	RRY	0.85	1994.67	-2379.6	4763.21
295	RRY	0.8	1994.67	-2379.6	4763.21
295	RRY	0.75	1994.67	-2379.6	4763.21
295	RRY	0.7	1994.67	-2379.6	4763.21
295	RRY	0.65	1994.67	-2379.6	4763.21
295	RRY	0.6	1994.67	-2379.6	4763.21
295	RRY	0.55	1994.67	-2379.6	4763.21
295	RRY	0.5	1994.67	-2379.6	4763.21
374	MLE	0.95	5189.38	-426.65	857.3
374	MLE	0.9	5189.38	-426.65	857.3
374	MLE	0.85	5189.38	-426.65	857.3
374	MLE	0.8	5189.38	-426.65	857.3
374	MLE	0.75	5189.38	-426.65	857.3
374	MLE	0.7	5189.38	-426.65	857.3
374	MLE	0.65	5189.38	-426.65	857.3
374	MLE	0.6	5189.38	-426.65	857.3
374	MLE	0.55	5189.38	-426.65	857.3
374	MLE	0.5	5189.38	-426.65	857.3
374	RRX	0.95	5602.36	-426.77	857.55

Malfunction Code	Fit Method	Conf. In.t	MTTF	Log-Likelihood	AIC
374	RRX	0.9	5602.36	-426.77	857.55
374	RRX	0.85	5602.36	-426.77	857.55
374	RRX	0.8	5602.36	-426.77	857.55
374	RRX	0.75	5602.36	-426.77	857.55
374	RRX	0.7	5602.36	-426.77	857.55
374	RRX	0.65	5602.36	-426.77	857.55
374	RRX	0.6	5602.36	-426.77	857.55
374	RRX	0.55	5602.36	-426.77	857.55
374	RRX	0.5	5602.36	-426.77	857.55
374	RRY	0.95	5939.67	-426.88	857.77
374	RRY	0.9	5939.67	-426.88	857.77
374	RRY	0.85	5939.67	-426.88	857.77
374	RRY	0.8	5939.67	-426.88	857.77
374	RRY	0.75	5939.67	-426.88	857.77
374	RRY	0.7	5939.67	-426.88	857.77
374	RRY	0.65	5939.67	-426.88	857.77
374	RRY	0.6	5939.67	-426.88	857.77
374	RRY	0.55	5939.67	-426.88	857.77
374	RRY	0.5	5939.67	-426.88	857.77
Blank (TD)	MLE	0.95	19543.52	-676.8	1357.61
Blank (TD)	MLE	0.9	19543.52	-676.8	1357.61
Blank (TD)	MLE	0.85	19543.52	-676.8	1357.61
Blank (TD)	MLE	0.8	19543.52	-676.8	1357.61
Blank (TD)	MLE	0.75	19543.52	-676.8	1357.61
Blank (TD)	MLE	0.7	19543.52	-676.8	1357.61
Blank (TD)	MLE	0.65	19543.52	-676.8	1357.61
Blank (TD)	MLE	0.6	19543.52	-676.8	1357.61
Blank (TD)	MLE	0.55	19543.52	-676.8	1357.61
Blank (TD)	MLE	0.5	19543.52	-676.8	1357.61
Blank (TD)	RRY	0.95	19494.86	-676.81	1357.62
Blank (TD)	RRY	0.9	19494.86	-676.81	1357.62
Blank (TD)	RRY	0.85	19494.86	-676.81	1357.62
Blank (TD)	RRY	0.8	19494.86	-676.81	1357.62
Blank (TD)	RRY	0.75	19494.86	-676.81	1357.62
Blank (TD)	RRY	0.7	19494.86	-676.81	1357.62
Blank (TD)	RRY	0.65	19494.86	-676.81	1357.62
Blank (TD)	RRY	0.6	19494.86	-676.81	1357.62
Blank (TD)	RRY	0.55	19494.86	-676.81	1357.62
Blank (TD)	RRY	0.5	19494.86	-676.81	1357.62

Malfunction Code	Fit Method	Conf. In.t	MTTF	Log-Likelihood	AIC
Blank (TD)	RRX	0.95	16797.63	-676.91	1357.83
Blank (TD)	RRX	0.9	16797.63	-676.91	1357.83
Blank (TD)	RRX	0.85	16797.63	-676.91	1357.83
Blank (TD)	RRX	0.8	16797.63	-676.91	1357.83
Blank (TD)	RRX	0.75	16797.63	-676.91	1357.83
Blank (TD)	RRX	0.7	16797.63	-676.91	1357.83
Blank (TD)	RRX	0.65	16797.63	-676.91	1357.83
Blank (TD)	RRX	0.6	16797.63	-676.91	1357.83
Blank (TD)	RRX	0.55	16797.63	-676.91	1357.83
Blank (TD)	RRX	0.5	16797.63	-676.91	1357.83

Adapted from Models.py Python Script (2022).

Table 11. CPH Model Performance by Failure Mode and Hyperparameter

Malfunction Code	Conf. Int.	Penalty	Log-Likelihood	AIC
20	0.95	0	-386.91	793.82
20	0.9	0	-386.91	793.82
20	0.85	0	-386.91	793.82
20	0.8	0	-386.91	793.82
20	0.75	0	-386.91	793.82
20	0.7	0	-386.91	793.82
20	0.65	0	-386.91	793.82
20	0.6	0	-386.91	793.82
20	0.55	0	-386.91	793.82
20	0.5	0	-386.91	793.82
20	0.95	0.05	-627.73	1275.46
20	0.9	0.05	-627.73	1275.46
20	0.85	0.05	-627.73	1275.46
20	0.8	0.05	-627.73	1275.46
20	0.75	0.05	-627.73	1275.46
20	0.7	0.05	-627.73	1275.46
20	0.65	0.05	-627.73	1275.46
20	0.6	0.05	-627.73	1275.46
20	0.55	0.05	-627.73	1275.46
20	0.5	0.05	-627.73	1275.46
20	0.95	0.1	-653.68	1327.36
20	0.9	0.1	-653.68	1327.36
20	0.85	0.1	-653.68	1327.36
20	0.8	0.1	-653.68	1327.36

Malfunction Code	Conf. Int.	Penalty	Log-Likelihood	AIC
20	0.75	0.1	-653.68	1327.36
20	0.7	0.1	-653.68	1327.36
20	0.65	0.1	-653.68	1327.36
20	0.6	0.1	-653.68	1327.36
20	0.55	0.1	-653.68	1327.36
20	0.5	0.1	-653.68	1327.36
20	0.95	0.15	-667.03	1354.06
20	0.9	0.15	-667.03	1354.06
20	0.85	0.15	-667.03	1354.06
20	0.8	0.15	-667.03	1354.06
20	0.75	0.15	-667.03	1354.06
20	0.7	0.15	-667.03	1354.06
20	0.65	0.15	-667.03	1354.06
20	0.6	0.15	-667.03	1354.06
20	0.55	0.15	-667.03	1354.06
20	0.5	0.15	-667.03	1354.06
70	0.95	0	-99.55	219.09
70	0.9	0	-99.55	219.09
70	0.85	0	-99.55	219.09
70	0.8	0	-99.55	219.09
70	0.75	0	-99.55	219.09
70	0.7	0	-99.55	219.09
70	0.65	0	-99.55	219.09
70	0.6	0	-99.55	219.09
70	0.55	0	-99.55	219.09
70	0.5	0	-99.55	219.09
70	0.95	0.05	-180.35	380.69
70	0.9	0.05	-180.35	380.69
70	0.85	0.05	-180.35	380.69
70	0.8	0.05	-180.35	380.69
70	0.75	0.05	-180.35	380.69
70	0.7	0.05	-180.35	380.69
70	0.65	0.05	-180.35	380.69
70	0.6	0.05	-180.35	380.69
70	0.55	0.05	-180.35	380.69
70	0.5	0.05	-180.35	380.69
70	0.95	0.1	-187.07	394.14
70	0.9	0.1	-187.07	394.14
70	0.85	0.1	-187.07	394.14

Malfunction Code	Conf. Int.	Penalty	Log-Likelihood	AIC
70	0.8	0.1	-187.07	394.14
70	0.75	0.1	-187.07	394.14
70	0.7	0.1	-187.07	394.14
70	0.65	0.1	-187.07	394.14
70	0.6	0.1	-187.07	394.14
70	0.55	0.1	-187.07	394.14
70	0.5	0.1	-187.07	394.14
70	0.95	0.15	-190.12	400.25
70	0.9	0.15	-190.12	400.25
70	0.85	0.15	-190.12	400.25
70	0.8	0.15	-190.12	400.25
70	0.75	0.15	-190.12	400.25
70	0.7	0.15	-190.12	400.25
70	0.65	0.15	-190.12	400.25
70	0.6	0.15	-190.12	400.25
70	0.55	0.15	-190.12	400.25
70	0.5	0.15	-190.12	400.25
135	0.95	0	-241.93	503.86
135	0.9	0	-241.93	503.86
135	0.85	0	-241.93	503.86
135	0.8	0	-241.93	503.86
135	0.75	0	-241.93	503.86
135	0.7	0	-241.93	503.86
135	0.65	0	-241.93	503.86
135	0.6	0	-241.93	503.86
135	0.55	0	-241.93	503.86
135	0.5	0	-241.93	503.86
135	0.95	0.05	-410.26	840.52
135	0.9	0.05	-410.26	840.52
135	0.85	0.05	-410.26	840.52
135	0.8	0.05	-410.26	840.52
135	0.75	0.05	-410.26	840.52
135	0.7	0.05	-410.26	840.52
135	0.65	0.05	-410.26	840.52
135	0.6	0.05	-410.26	840.52
135	0.55	0.05	-410.26	840.52
135	0.5	0.05	-410.26	840.52
135	0.95	0.1	-427.63	875.26
135	0.9	0.1	-427.63	875.26

Malfunction Code	Conf. Int.	Penalty	Log-Likelihood	AIC
135	0.85	0.1	-427.63	875.26
135	0.8	0.1	-427.63	875.26
135	0.75	0.1	-427.63	875.26
135	0.7	0.1	-427.63	875.26
135	0.65	0.1	-427.63	875.26
135	0.6	0.1	-427.63	875.26
135	0.55	0.1	-427.63	875.26
135	0.5	0.1	-427.63	875.26
135	0.95	0.15	-436.5	893.01
135	0.9	0.15	-436.5	893.01
135	0.85	0.15	-436.5	893.01
135	0.8	0.15	-436.5	893.01
135	0.75	0.15	-436.5	893.01
135	0.7	0.15	-436.5	893.01
135	0.65	0.15	-436.5	893.01
135	0.6	0.15	-436.5	893.01
135	0.55	0.15	-436.5	893.01
135	0.5	0.15	-436.5	893.01
150	0.95	0	-52	124
150	0.9	0	-52	124
150	0.85	0	-52	124
150	0.8	0	-52	124
150	0.75	0	-52	124
150	0.7	0	-52	124
150	0.65	0	-52	124
150	0.6	0	-52	124
150	0.55	0	-52	124
150	0.5	0	-52	124
150	0.95	0.05	-97.3	214.6
150	0.9	0.05	-97.3	214.6
150	0.85	0.05	-97.3	214.6
150	0.8	0.05	-97.3	214.6
150	0.75	0.05	-97.3	214.6
150	0.7	0.05	-97.3	214.6
150	0.65	0.05	-97.3	214.6
150	0.6	0.05	-97.3	214.6
150	0.55	0.05	-97.3	214.6
150	0.5	0.05	-97.3	214.6
150	0.95	0.1	-99.4	218.79

Malfunction Code	Conf. Int.	Penalty	Log-Likelihood	AIC
150	0.9	0.1	-99.4	218.79
150	0.85	0.1	-99.4	218.79
150	0.8	0.1	-99.4	218.79
150	0.75	0.1	-99.4	218.79
150	0.7	0.1	-99.4	218.79
150	0.65	0.1	-99.4	218.79
150	0.6	0.1	-99.4	218.79
150	0.55	0.1	-99.4	218.79
150	0.5	0.1	-99.4	218.79
150	0.95	0.15	-100.25	220.5
150	0.9	0.15	-100.25	220.5
150	0.85	0.15	-100.25	220.5
150	0.8	0.15	-100.25	220.5
150	0.75	0.15	-100.25	220.5
150	0.7	0.15	-100.25	220.5
150	0.65	0.15	-100.25	220.5
150	0.6	0.15	-100.25	220.5
150	0.55	0.15	-100.25	220.5
150	0.5	0.15	-100.25	220.5
290	0.95	0	-87.98	195.96
290	0.9	0	-87.98	195.96
290	0.85	0	-87.98	195.96
290	0.8	0	-87.98	195.96
290	0.75	0	-87.98	195.96
290	0.7	0	-87.98	195.96
290	0.65	0	-87.98	195.96
290	0.6	0	-87.98	195.96
290	0.55	0	-87.98	195.96
290	0.5	0	-87.98	195.96
290	0.95	0.05	-159.66	339.32
290	0.9	0.05	-159.66	339.32
290	0.85	0.05	-159.66	339.32
290	0.8	0.05	-159.66	339.32
290	0.75	0.05	-159.66	339.32
290	0.7	0.05	-159.66	339.32
290	0.65	0.05	-159.66	339.32
290	0.6	0.05	-159.66	339.32
290	0.55	0.05	-159.66	339.32
290	0.5	0.05	-159.66	339.32

Malfunction Code	Conf. Int.	Penalty	Log-Likelihood	AIC
290	0.95	0.1	-165.51	351.02
290	0.9	0.1	-165.51	351.02
290	0.85	0.1	-165.51	351.02
290	0.8	0.1	-165.51	351.02
290	0.75	0.1	-165.51	351.02
290	0.7	0.1	-165.51	351.02
290	0.65	0.1	-165.51	351.02
290	0.6	0.1	-165.51	351.02
290	0.55	0.1	-165.51	351.02
290	0.5	0.1	-165.51	351.02
290	0.95	0.15	-168.16	356.32
290	0.9	0.15	-168.16	356.32
290	0.85	0.15	-168.16	356.32
290	0.8	0.15	-168.16	356.32
290	0.75	0.15	-168.16	356.32
290	0.7	0.15	-168.16	356.32
290	0.65	0.15	-168.16	356.32
290	0.6	0.15	-168.16	356.32
290	0.55	0.15	-168.16	356.32
290	0.5	0.15	-168.16	356.32
295	0.95	0	-641.15	1302.31
295	0.9	0	-641.15	1302.31
295	0.85	0	-641.15	1302.31
295	0.8	0	-641.15	1302.31
295	0.75	0	-641.15	1302.31
295	0.7	0	-641.15	1302.31
295	0.65	0	-641.15	1302.31
295	0.6	0	-641.15	1302.31
295	0.55	0	-641.15	1302.31
295	0.5	0	-641.15	1302.31
295	0.95	0.05	-1018.27	2056.53
295	0.9	0.05	-1018.27	2056.53
295	0.85	0.05	-1018.27	2056.53
295	0.8	0.05	-1018.27	2056.53
295	0.75	0.05	-1018.27	2056.53
295	0.7	0.05	-1018.27	2056.53
295	0.65	0.05	-1018.27	2056.53
295	0.6	0.05	-1018.27	2056.53
295	0.55	0.05	-1018.27	2056.53

Malfunction Code	Conf. Int.	Penalty	Log-Likelihood	AIC
295	0.5	0.05	-1018.27	2056.53
295	0.95	0.1	-1071.59	2163.17
295	0.9	0.1	-1071.59	2163.17
295	0.85	0.1	-1071.59	2163.17
295	0.8	0.1	-1071.59	2163.17
295	0.75	0.1	-1071.59	2163.17
295	0.7	0.1	-1071.59	2163.17
295	0.65	0.1	-1071.59	2163.17
295	0.6	0.1	-1071.59	2163.17
295	0.55	0.1	-1071.59	2163.17
295	0.5	0.1	-1071.59	2163.17
295	0.95	0.15	-1100.52	2221.05
295	0.9	0.15	-1100.52	2221.05
295	0.85	0.15	-1100.52	2221.05
295	0.8	0.15	-1100.52	2221.05
295	0.75	0.15	-1100.52	2221.05
295	0.7	0.15	-1100.52	2221.05
295	0.65	0.15	-1100.52	2221.05
295	0.6	0.15	-1100.52	2221.05
295	0.55	0.15	-1100.52	2221.05
295	0.5	0.15	-1100.52	2221.05
374	0.95	0	-87.71	195.42
374	0.9	0	-87.71	195.42
374	0.85	0	-87.71	195.42
374	0.8	0	-87.71	195.42
374	0.75	0	-87.71	195.42
374	0.7	0	-87.71	195.42
374	0.65	0	-87.71	195.42
374	0.6	0	-87.71	195.42
374	0.55	0	-87.71	195.42
374	0.5	0	-87.71	195.42
374	0.95	0.05	-162.77	345.53
374	0.9	0.05	-162.77	345.53
374	0.85	0.05	-162.77	345.53
374	0.8	0.05	-162.77	345.53
374	0.75	0.05	-162.77	345.53
374	0.7	0.05	-162.77	345.53
374	0.65	0.05	-162.77	345.53
374	0.6	0.05	-162.77	345.53

Malfunction Code	Conf. Int.	Penalty	Log-Likelihood	AIC
374	0.55	0.05	-162.77	345.53
374	0.5	0.05	-162.77	345.53
374	0.95	0.1	-168.29	356.57
374	0.9	0.1	-168.29	356.57
374	0.85	0.1	-168.29	356.57
374	0.8	0.1	-168.29	356.57
374	0.75	0.1	-168.29	356.57
374	0.7	0.1	-168.29	356.57
374	0.65	0.1	-168.29	356.57
374	0.6	0.1	-168.29	356.57
374	0.55	0.1	-168.29	356.57
374	0.5	0.1	-168.29	356.57
374	0.95	0.15	-170.8	361.59
374	0.9	0.15	-170.8	361.59
374	0.85	0.15	-170.8	361.59
374	0.8	0.15	-170.8	361.59
374	0.75	0.15	-170.8	361.59
374	0.7	0.15	-170.8	361.59
374	0.65	0.15	-170.8	361.59
374	0.6	0.15	-170.8	361.59
374	0.55	0.15	-170.8	361.59
374	0.5	0.15	-170.8	361.59
Blank (TD)	0.95	0	-162.03	344.05
Blank (TD)	0.9	0	-162.03	344.05
Blank (TD)	0.85	0	-162.03	344.05
Blank (TD)	0.8	0	-162.03	344.05
Blank (TD)	0.75	0	-162.03	344.05
Blank (TD)	0.7	0	-162.03	344.05
Blank (TD)	0.65	0	-162.03	344.05
Blank (TD)	0.6	0	-162.03	344.05
Blank (TD)	0.55	0	-162.03	344.05
Blank (TD)	0.5	0	-162.03	344.05
Blank (TD)	0.95	0.05	-282.04	584.08
Blank (TD)	0.9	0.05	-282.04	584.08
Blank (TD)	0.85	0.05	-282.04	584.08
Blank (TD)	0.8	0.05	-282.04	584.08
Blank (TD)	0.75	0.05	-282.04	584.08
Blank (TD)	0.7	0.05	-282.04	584.08
Blank (TD)	0.65	0.05	-282.04	584.08

Malfunction Code	Conf. Int.	Penalty	Log-Likelihood	AIC
Blank (TD)	0.6	0.05	-282.04	584.08
Blank (TD)	0.55	0.05	-282.04	584.08
Blank (TD)	0.5	0.05	-282.04	584.08
Blank (TD)	0.95	0.1	-294.62	609.25
Blank (TD)	0.9	0.1	-294.62	609.25
Blank (TD)	0.85	0.1	-294.62	609.25
Blank (TD)	0.8	0.1	-294.62	609.25
Blank (TD)	0.75	0.1	-294.62	609.25
Blank (TD)	0.7	0.1	-294.62	609.25
Blank (TD)	0.65	0.1	-294.62	609.25
Blank (TD)	0.6	0.1	-294.62	609.25
Blank (TD)	0.55	0.1	-294.62	609.25
Blank (TD)	0.5	0.1	-294.62	609.25
Blank (TD)	0.95	0.15	-300.72	621.44
Blank (TD)	0.9	0.15	-300.72	621.44
Blank (TD)	0.85	0.15	-300.72	621.44
Blank (TD)	0.8	0.15	-300.72	621.44
Blank (TD)	0.75	0.15	-300.72	621.44
Blank (TD)	0.7	0.15	-300.72	621.44
Blank (TD)	0.65	0.15	-300.72	621.44
Blank (TD)	0.6	0.15	-300.72	621.44
Blank (TD)	0.55	0.15	-300.72	621.44
Blank (TD)	0.5	0.15	-300.72	621.44

Adapted from Models.py Python Script (2022).

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. COX PROPORTIONAL HAZARD PLOTS

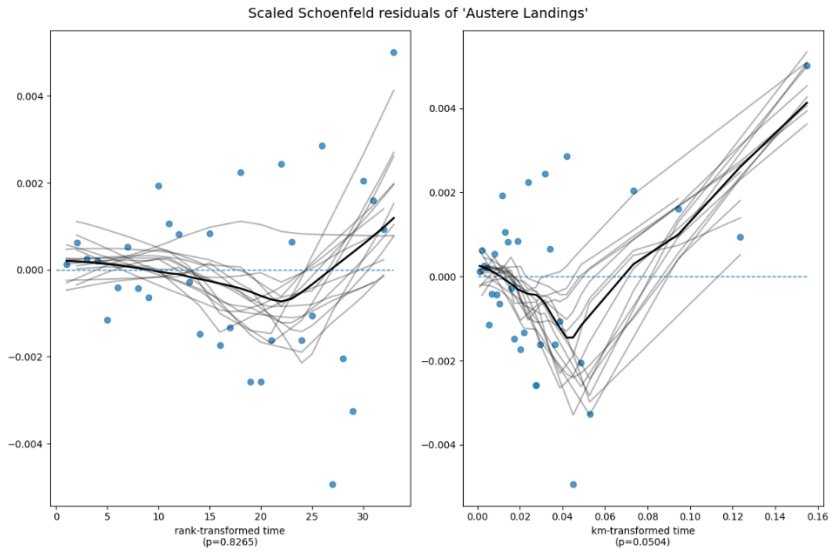

Figure 40. Scaled Schoenfeld Residuals of Austere Landings for Malfunction Code 70 – Broken, Burst, Ruptured, Punctured, Torn, or Cut.
Source: Models.py Python Script (2022).

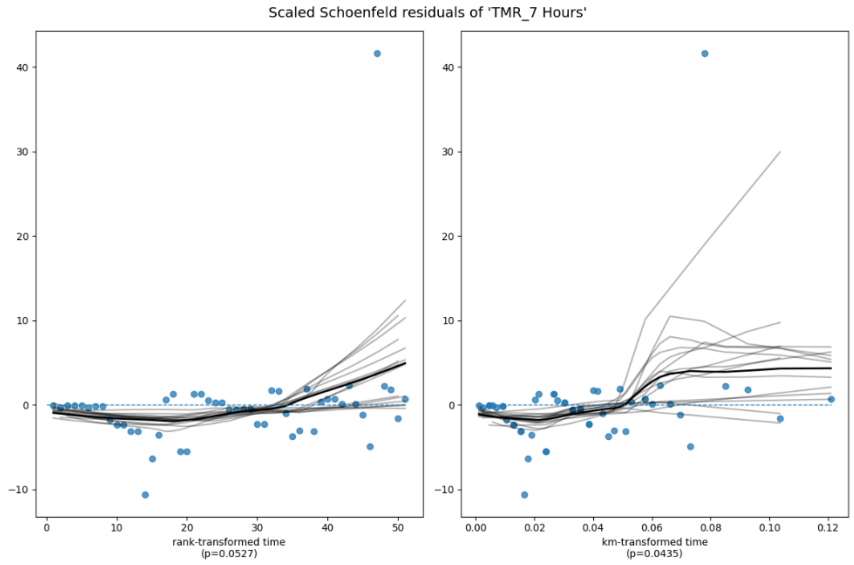

Figure 41. Scaled Schoenfeld Residuals of TMR_7 Hours for Blank – Technical Directive Inspection Failure.
Source: Models.py Python Script (2022).

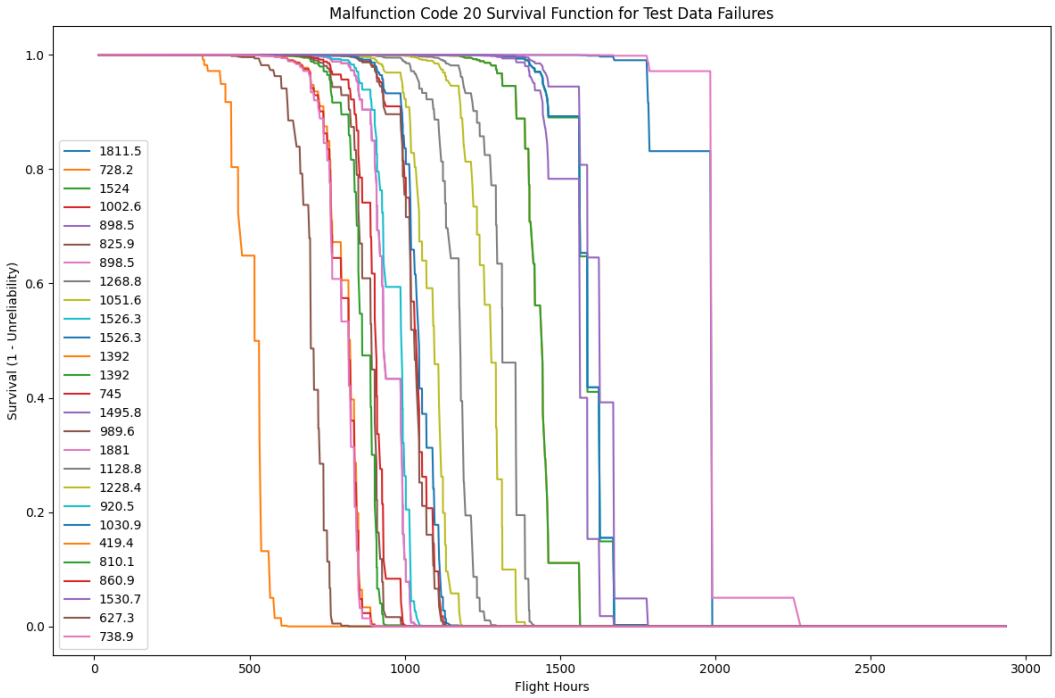

Figure 42. CPH Predicted Survival for Malfunction Code 20 Test Data. Source: Models.py Python Script (2022).

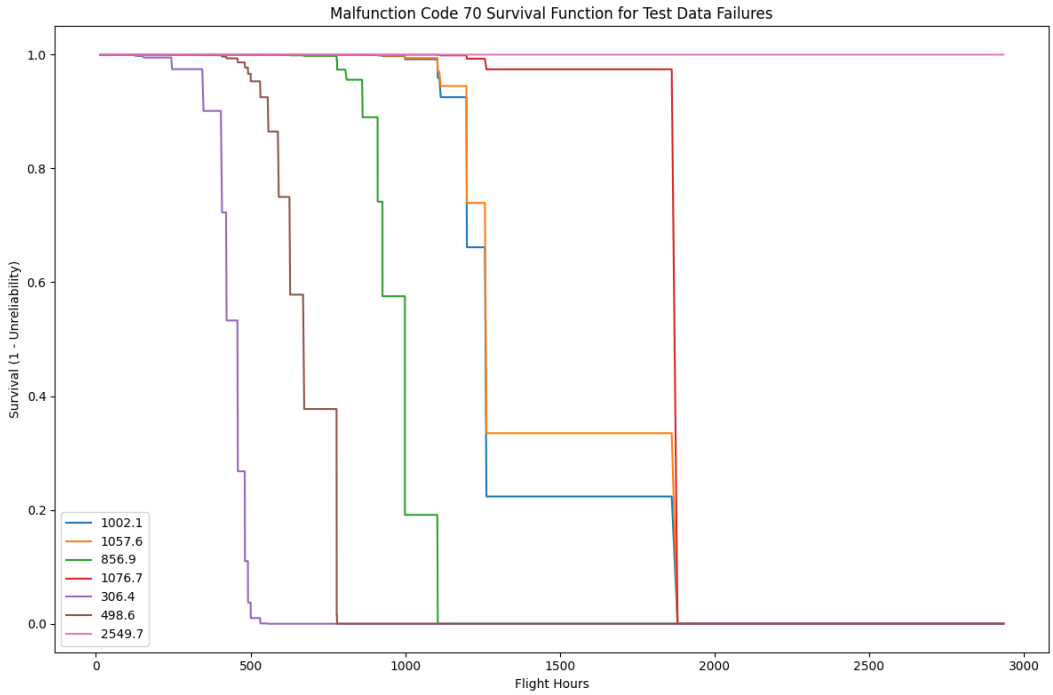

Figure 43. CPH Predicted Survival for Malfunction Code 70 Test Data. Source: Models.py Python Script (2022).

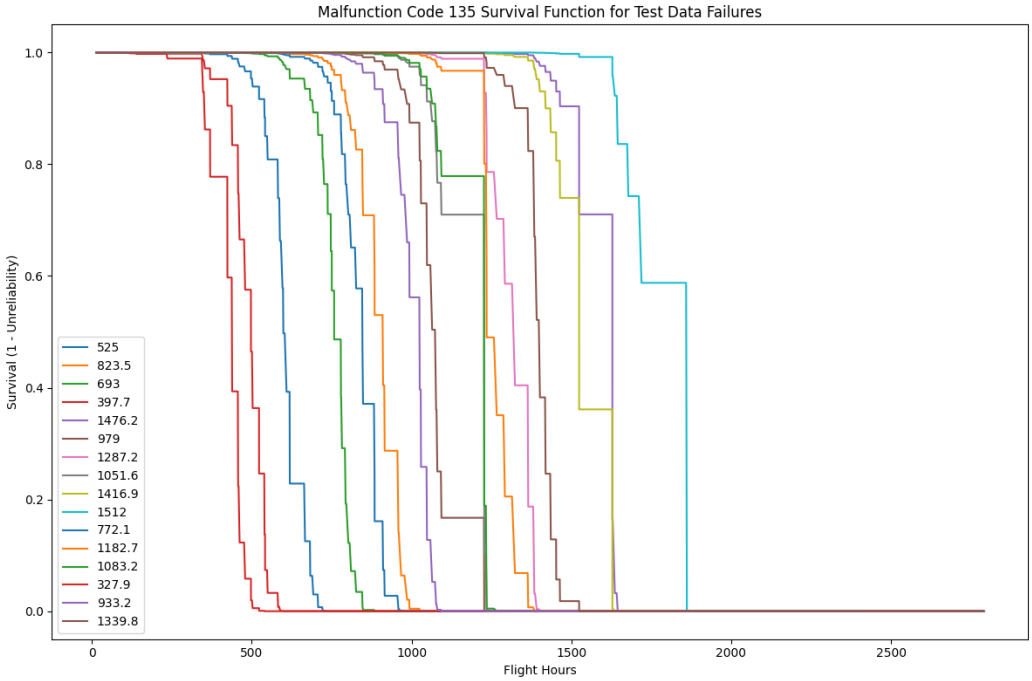

Figure 44. CPH Predicted Survival for Malfunction Code 135 Test Data. Source: Models.py Python Script (2022).

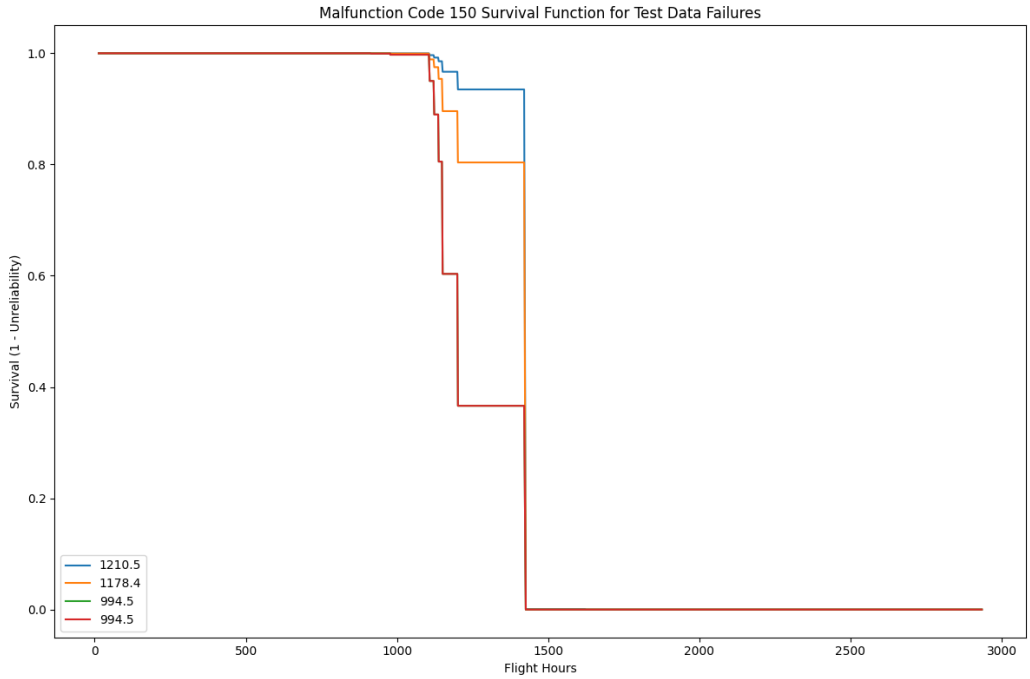

Figure 45. CPH Predicted Survival for Malfunction Code 150 Test Data. Source: Models.py Python Script (2022).

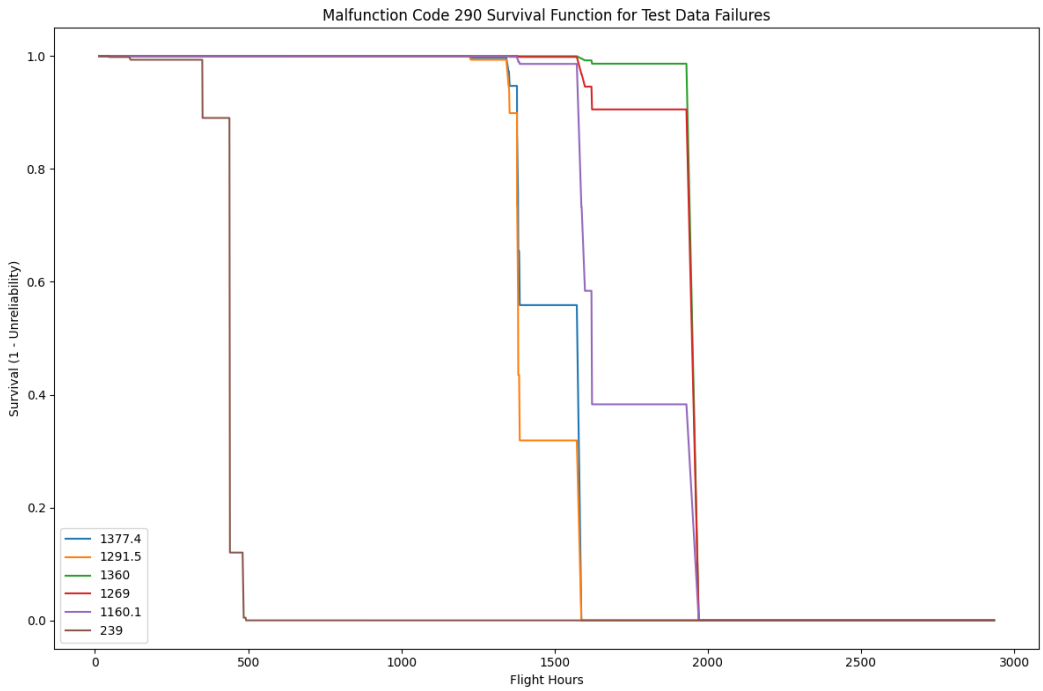

Figure 46. CPH Predicted Survival for Malfunction Code 290 Test Data. Source: Models.py Python Script (2022).

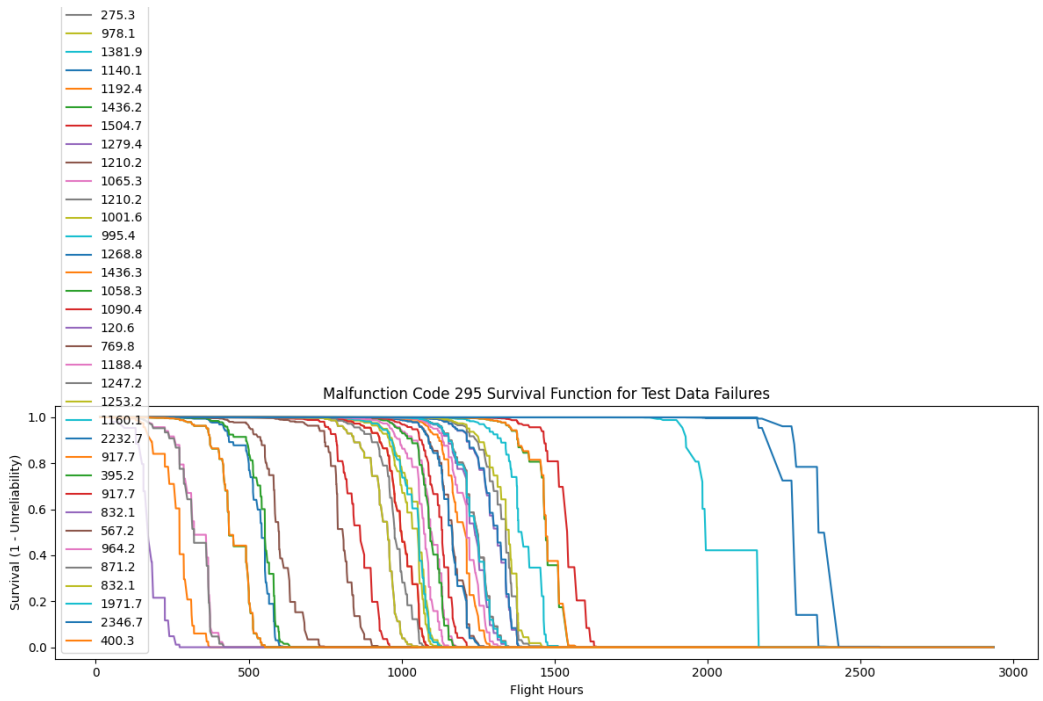

Figure 47. CPH Predicted Survival for Malfunction Code 295 Test Data. Source: Models.py Python Script (2022).

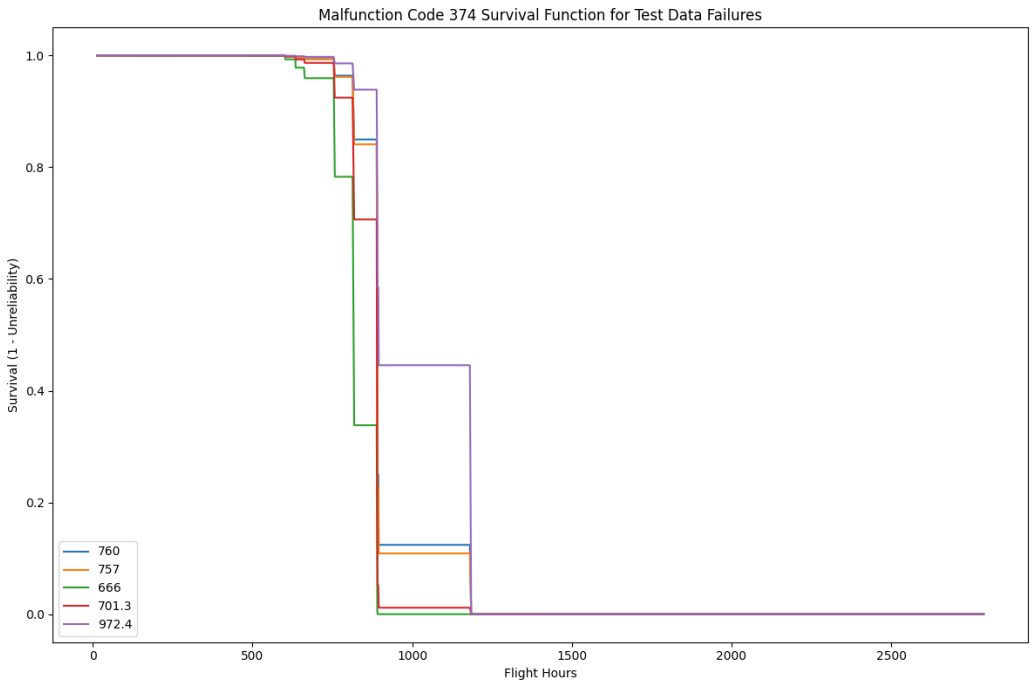

Figure 48. CPH Predicted Survival for Malfunction Code 374 Test Data.
Source: Models.py Python Script (2022).

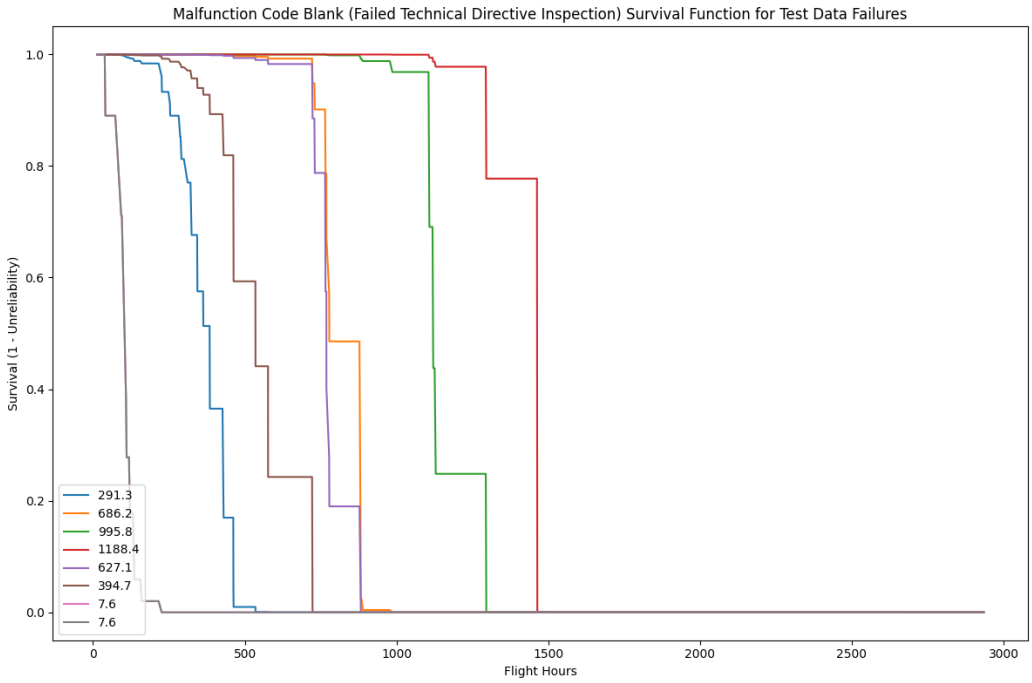

Figure 49. CPH Predicted Survival for Failed Technical Directive Inspection Test Data.
Source: Models.py Python Script (2022).

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. MATLAB SOURCE CODE FOR REMAINING USEFUL LIFE PREDICTIONS USING LSTM NETWORK

```
%% NASA Turbofan LSTM Net
% Captain William Frazier
% NPS Thesis research
% Predicting RUL of NASA Turbofan Dataset
% https://www.kaggle.com/c/predictive-maintenance

clear
clc

labels = ["id","cycle","op1","op2","op3","sensor1","sensor2",...
  "sensor3","sensor4","sensor5","sensor6","sensor7","sensor8",...
  "sensor9","sensor10","sensor11","sensor12","sensor13","sensor14",...
  "sensor15","sensor16","sensor17","sensor18","sensor19","sensor20",...
  "sensor21","RUL","TTF_Window"];

% Load first set of training data
% train_FD001_new.csv is the updated version of the original train_FD001.csv
% it includes two new columns at the end which were created in orange for
% RUL and a TTF window
%train_FD001 =
train_FD001 =
readtable('C:\Users\willi\OneDrive\Desktop\Thesis\Kaggle\NASA\train_FD001_new.
csv');
train_FD001.Properties.VariableNames = labels;
X_all_train = table2array(train_FD001(:,1:26));
Y_train = table2array(train_FD001(:,27));

% Load first set of test data
test_FD001 =
readtable('C:\Users\willi\OneDrive\Desktop\Thesis\Kaggle\NASA\test_FD001_new.c
sv');
test_FD001.Properties.VariableNames = labels;
X_all_test = table2array(test_FD001(:,1:26));
Y_test = table2array(test_FD001(:,27));

% Grab the number of engines in the training/test sets
num_units_train = X_all_train(length(X_all_train),1);
num_units_test = X_all_test(length(X_all_test),1);

results = table();
% Declare how many input features to keep
for numFeatures = 3:16

  % Remove constant features
  [X_train, X_labels] = removeConstants(X_all_train, labels);
  [X_test, X_labels] = removeConstants(X_all_test, labels);

  % Identify any missing data
```

```
  missing = imputeMissing(X_all_train, Y_train, X_all_test, Y_test);

  % Calculate VIF of training inputs
  [VIF, VIFTable] = getVIF(X_train(:,(3:18)), X_labels(:,(3:18)));

  % Retain only engine ID, cycle, and the numFeatures least correlated
  [X_train, X_labels, X_test] = removeCorrInputs(X_train, X_labels, ...
    X_test, VIF, numFeatures);

  % Normalize data
  [X_train, RUL_train, X_test, RUL_test] = normalize(X_train, Y_train, ...
    X_test, Y_test);

  [LSTM_input_train, LSTM_RUL_train] = createCells(num_units_train, X_train,
RUL_train);
  [LSTM_input_test, LSTM_RUL_test] = createCells(num_units_test, X_test,
RUL_test);

  % Healthy State
  health_index = 150;
  LSTM_RUL_train = adjustHealth(LSTM_RUL_train, health_index);

  % Split training into train/val
  train_perc = .75;
  val_perc = .25;

  % Padding
  [LSTM_input_train, LSTM_RUL_train] = padding(LSTM_input_train,
LSTM_RUL_train);


  % De
  numHiddenUnits = 200;
  FCnumNodes = 50;
  dropoutrate = 0.2; % 0.5
  maxEpochs = 1000;
  miniBatchSize = 49;
  InitialLearnRate = 0.01;
  GradientThreshold = 1;

  % No splitting
  [net,info] = LSTM(LSTM_input_train, LSTM_RUL_train, numHiddenUnits, ...
    FCnumNodes, dropoutrate, maxEpochs, miniBatchSize, ...
    InitialLearnRate, GradientThreshold);

  % Predict
  % Apply the over-padding prevention by setting minibatchsize to 1
  Y_pred_test = predict(net, LSTM_input_test,'MiniBatchSize',1);
  results.(numFeatures - 2) = [info.TrainingRMSE(end); getError(LSTM_RUL_test,
Y_pred_test)];

end
results.Properties.Description = ["Loss by number of input features used"];
```

```
results.Properties.VariableNames =
["3","4","5","6","7","8","9","10","11","12","13","14","15","16"];
results.Properties.RowNames = ["Final Training RMSE"; "Test RMSE"];
results


%% Remove constant features
function [X_new, X_labels] = removeConstants(X, labels)
  % Sensors 1, 5, 6, 10, 16, 18, 19, Operation Setting 3 are all constant
  X_new = X(:,[1:4,7:9,12:14,16:20,22,25:26]);
  X_labels = labels([1:4,7:9,12:14,16:20,22,25:26]);
end
%% Find missing data
function missing = imputeMissing(X_all_train, Y_train, X_all_test, Y_test)

  missing = [];
  check_for_missing = isnan(X_all_train);
  [rowmissing, colmissing] = find(check_for_missing);
  %missing(:,1) = [rowmissing, colmissing];

  check_for_missing = isnan(Y_train);
  [rowmissing, colmissing] = find(check_for_missing);
  %missing(:,2) = [rowmissing, colmissing];

  check_for_missing = isnan(X_all_test);
  [rowmissing, colmissing] = find(check_for_missing);
  %missing(:,3) = [rowmissing, colmissing];

  check_for_missing = isnan(Y_test);
  [rowmissing, colmissing] = find(check_for_missing);
  %missing(:,4) = [rowmissing, colmissing];
end
%% Calculate VIF
% Looks at multicolinearity between input features
function [VIF, VIFTable] = getVIF(X_train, X_labels)
  R = corrcoef(X_train);
  rowNames = X_labels;
  VIF = diag(inv(R));
  colNames = ["VIF"];
  VIFTable = array2table(VIF,'RowNames',rowNames,'VariableNames',colNames);
end
%% Calculate PCA
%% Remove highly correlated input features
function [X_train_new, X_labels_new, X_test_new] = ...
  removeCorrInputs(X_train, X_labels, X_test, VIF, numFeatures)
% Initialize output datasets with engine and cycle columns
X_train_new = X_train(:,1:2);
X_labels_new = X_labels(1:2);
X_test_new = X_test(:,1:2);

for i = 3:2+numFeatures
  [val, pos] = min(VIF);
  VIF(pos) = [];
```

```
  X_train_new(:,i) = X_train(:,pos+2);
  X_labels_new(i) = X_labels(pos+2);
  X_labels(pos+2) = [];
  X_test_new(:,i) = X_test(:,pos+2);
end


end

%% Normalize/Standardize
% Transpose for MatLab mapminmax(); by default normalizes between -1 and 1
function [X_train, Y_train, X_test, Y_test] = normalize(X_train, Y_train, ...
  X_test, Y_test)
% Transpose for MatLab mapminmax(); by default normalizes between -1 and 1
X_train = X_train';
X_test = X_test';
% Normalize the training data and apply that mapping to the test data
% Exclude engine id and cycle
[X_train_scaled, PS] = mapminmax(X_train(3:size(X_train,1),:));
X_test_scaled = mapminmax('apply',X_test(3:size(X_test,1),:),PS);
% Rejoin with engine and cycle
X_train(3:size(X_train,1),:) = X_train_scaled;
X_test(3:size(X_train,1),:) = X_test_scaled;
% Don't need to transpose back, keep for Cell creation
% Transpose the target though, to match
Y_train = Y_train';
Y_test = Y_test';
end
%% Create LSTM Network Training Input and Output Cells
% MatLab LSTM expects a cell
% Dimensions of the cell are n_samples x 1
% Each element in the cell is a unique engine id's sequential data
function [input_cell, target_cell] = createCells(num_units, X, Y)
% Initialize the input/target cells for MatLab LSTM expectations
% Number of units is equal to the last cell's engine id
input_cell = cell(num_units,1);
target_cell = cell(num_units,1);
% Counters so we can grab all the rows of data for an engine unit and
% insert that as one element into the cell
unit_id = 1;
row_index = 1;
cell_index = 1;
first_cycle = 1;

% Index from 1 to the length of all rows in x data
while row_index <= length(X)

  % Insert the rows of data for each unique engine ID
  % If the current row's engine ID is not = unit
  % Fill from first index to i-1
  % update first = i, unit and cell index increment by 1
  if X(1,row_index) ~= unit_id
    input_cell{cell_index} = X(3:size(X,1),[first_cycle:row_index-1]);
    target_cell{cell_index} = Y(:,[first_cycle:row_index-1]);
```

```
    cell_index = cell_index + 1;
    first_cycle = row_index;
    unit_id = unit_id + 1;
  end
  % If this is the last row, there are no more engines. fill from first
  if row_index == length(X)
    input_cell{cell_index} = X(3:size(X,1),[first_cycle:row_index]);
    target_cell{cell_index} = Y(:,[first_cycle:row_index]);
  end
  row_index = row_index + 1;
end
end
%% Splitting
function [LSTM_input_train, LSTM_RUL_train, LSTM_input_val, LSTM_RUL_val] =
...
  dataSplit(X_train, Y_train, test_perc, val_perc, num_units)

  LSTM_input_train = cell(num_units*test_perc,1);
  LSTM_RUL_train = cell(num_units*test_perc,1);
  LSTM_input_val = cell(num_units*val_perc,1);
  LSTM_RUL_val = cell(num_units*val_perc,1);

  a = randperm(numel(X_train));
  for i = 1:num_units*test_perc
    LSTM_input_train(i) = X_train(a(i));
    LSTM_RUL_train(i) = Y_train(a(i));
  end
  j = i;
  for i = 1:num_units*val_perc
   LSTM_input_val(i) = X_train(a(i+j));
   LSTM_RUL_val(i) = Y_train(a(i+j));
  end
end
%% Padding
% Because each engine has variable length samples/cycles, padding can be
% really aggressive for an engine that only has 10 cycles recorded if it
% is mini-batched with one with 150 cycles. To avoid this, we are going
% to sort our sequential data in descending order by number of cycles.
% We will also be sure to set our network options to never shuffle.
function [LSTM_input, LSTM_target] = padding(X, Y)
  % map/sort for training cell
  for i=1:numel(X)
    unit_id = X{i};
    num_cycles(i) = size(unit_id,2);
  end

  [num_cycles,index] = sort(num_cycles,'descend');
  LSTM_input = X(index);
  LSTM_target = Y(index);
end
%% How far back do we look?
% We've been discussing how far back to look in historical data before
% the data is no longer useful. Set a healthy RUL threshold and anything
% above that RUL value is set equal to that threshold
```

```
% Treat any RUL over 150 as 150
function Y_train = adjustHealth(Y_train, health_index)

  for i = 1:numel(Y_train)
    Y_train{i}(Y_train{i} > health_index) = health_index;
  end
end
%% RMSE Function
function RMSE = getError(NN_target_test, Y_pred_test)

  for i = 1:numel(NN_target_test)
    YTestLast(i) = NN_target_test{i}(end);
    YPredLast(i) = Y_pred_test{i}(end);
  end
  RMSE = sqrt(mean((YPredLast - YTestLast).^2));
end
%% Function for LSTM
function [net, info] = LSTM(LSTM_input_train, LSTM_RUL_train, ...
  numHiddenUnits, FCnumNodes, dropoutrate, maxEpochs, miniBatchSize, ...
  InitialLearnRate, GradientThreshold)

numFeatures = size(LSTM_input_train{1},1);
numResponses = size(LSTM_RUL_train{1},1);
% Hyperparameters / Model Architecture
layers = [ ...
  sequenceInputLayer(numFeatures)

  lstmLayer(numHiddenUnits,'Name', 'LSTM_1', 'OutputMode','sequence')
  fullyConnectedLayer(FCnumNodes, 'Name', 'FC_1')
  dropoutLayer(dropoutrate, 'Name', 'Dropout_1')

  %lstmLayer(numHiddenUnits,'Name', 'LSTM_2', 'OutputMode','sequence')
  %fullyConnectedLayer(FCnumNodes, 'Name', 'FC_2')
  %dropoutLayer(dropoutrate, 'Name', 'Dropout_2')

  fullyConnectedLayer(numResponses)
  regressionLayer];
% Training Options
options = trainingOptions('adam', ...
  'MaxEpochs',maxEpochs, ...
  'MiniBatchSize',miniBatchSize, ...
  'InitialLearnRate',InitialLearnRate, ...
  'GradientThreshold',GradientThreshold, ...
  'Shuffle','never', ...
  'Plots','none', ...
  'Verbose',0);

[net,info] = trainNetwork(LSTM_input_train, LSTM_RUL_train, layers, options);
end
```

APPENDIX E. MATLAB SOURCE CODE FOR REMAINING USEFUL LIFE PREDICTIONS USING CNN

```
%% NASA Turbofan LSMT Net
%  Captain William Frazier
%  NPS Thesis practice/research
%  Predicting RUL of NASA Turbofan Dataset
%  https://www.kaggle.com/c/predictive-maintenance

clear
clc

labels = ["id","cycle","op1","op2","op3","sensor1","sensor2",...
    "sensor3","sensor4","sensor5","sensor6","sensor7","sensor8",...
    "sensor9","sensor10","sensor11","sensor12","sensor13","sensor14",...
    "sensor15","sensor16","sensor17","sensor18","sensor19","sensor20",...
    "sensor21","RUL","TTF_Window"];

% Load first set of training data
% train_FD001_new.csv is the updated version of the original train_FD001.csv
% it includes two new columns at the end which were created in orage for
% RUL and a TTF window
train_FD001 =
readtable('C:\Users\willi\OneDrive\Desktop\Thesis\Orange\NASA\MatLab\train_FD0
01_new.csv');
train_FD001.Properties.VariableNames = labels;
X_all_train = table2array(train_FD001(:,1:26));
Y_train = table2array(train_FD001(:,27));

% Load first set of test data
test_FD001 =
readtable('C:\Users\willi\OneDrive\Desktop\Thesis\Orange\NASA\MatLab\test_FD00
1_new.csv');
test_FD001.Properties.VariableNames = labels;
X_all_test = table2array(test_FD001(:,1:26));
Y_test = table2array(test_FD001(:,27));

% Grab the number of engines in the training/test sets
num_units_train = X_all_train(length(X_all_train),1);
num_units_test = X_all_test(length(X_all_test),1);

%% Remove constant features
% Sensors 1, 5, 6, 10, 16, 18, 19, Operation Setting 3 are all constant
X_train = X_all_train(:,[1:4,7:9,12:14,16:20,22,25:26]);
x_labels = labels([1:4,7:9,12:14,16:20,22,25:26]);

%% Calculate Coeff Correlation and VIF of training data
% Lets first just look at multicolinearity between input features

R = corrcoef(X_train);
rowNames = x_labels;
```

```
colNames = x_labels;
RTable = array2table(R,'RowNames',rowNames,'VariableNames',colNames);

VIF = diag(inv(R));
rowNames = x_labels;
colNames = ["VIF"];
VIFTable = array2table(VIF,'RowNames',rowNames,'VariableNames',colNames);

%% Remove extremely highly correlated and recalculate VIF
% Remove sensor 9, 11, and 14 due to high VIF

X_train = X_all_train(:,[3:4,7:9,12,13,17,18,20,22,25,26]);
x_labels = labels([3:4,7:9,12,13,17,18,20,22,25,26]);

R = corrcoef(X_train);
rowNames = x_labels;
colNames = x_labels;
RTable = array2table(R,'RowNames',rowNames,'VariableNames',colNames);

VIF = diag(inv(R));
rowNames = x_labels;
colNames = ["VIF"];
VIFTable = array2table(VIF,'RowNames',rowNames,'VariableNames',colNames);

% Apply the reduction to the test data as well
X_test = X_all_test(:,[3:4,7:9,12,13,17,18,20,22,25,26]);

%% Normalize/Standardize
% Transpose for MatLab mapminmax(); by default normalizes between -1 and 1
X_train = X_train';
X_test = X_test';

% Normalize the training data and apply that mapping to the test data
[X_train_scaled, PS] = mapminmax(X_train);
X_test_scaled = mapminmax('apply',X_test,PS);

X_train = X_train';
X_test = X_test';

X_train_scaled = X_train_scaled';
X_test_scaled = X_test_scaled';
%% OPTIONAL
% rejoin with cycle/engine ID for train/validation splitting
X_train_pre_split = cat(2,X_all_train(:,1:2),X_train_scaled);
X_test_pre_split = cat(2,X_all_test(:,1:2),X_test_scaled);
%% How far back do we look?
%  We've been discussing how far back to look in historical data before
%  the data is no longer useful. Set a healthy RUL threshold and anything
%  above that RUL value is set equal to that threshold

% Treat any RUL over 150 as 150
healthy = 150;
for i = 1:numel(Y_train)
    if Y_train(i) > healthy
```

```
        Y_train(i) = healthy;
    end
end

%% This takes the training data set and creates a new NN input data set
% Takes 30 sonsecutive cycles with a step size of 1
window_size = 30;

NN_input_train = [];
NN_RUL_train = [];

unit_id = 1;
row_index = 1;
cell_index = 1;
first_cycle = 1;

while row_index <= length(X_all_train) - window_size
    if row_index == 1
        NN_input_train(:,:,:,row_index) =
X_train_scaled(row_index:row_index+window_size-1,:);
        %NN_RUL_train(:,:,:,row_index) =
Y_train(row_index:row_index+window_size-1,:);
        NN_RUL_train(:,row_index) = Y_train(row_index:row_index+window_size-
1,:);
        row_index = row_index+1;
    end

    % Ensure there are at least window_size cycles recorded before failure
    if Y_train(row_index,1) >= window_size
        % While unit number is still the same, insert samples into final
        while X_all_train(row_index+window_size-1,1) == unit_id
            % Insert rows i to i+window_size, all columns
            NN_input_train(:,:,:,cell_index) =
X_train_scaled(row_index:row_index+window_size-1,:);
            %NN_RUL_train(:,:,:,cell_index) =
Y_train(row_index:row_index+window_size-1,:);
            NN_RUL_train(:,cell_index) =
Y_train(row_index:row_index+window_size-1,:);
            cell_index = cell_index + 1;
            row_index = row_index+1;
            if row_index > length(X_train_scaled) - window_size
                return
            end
        end
        % Once the unit number is changed, need to 'hop' down, change unit
        row_index = row_index + window_size-1;
        unit_id = X_all_train(row_index,1);
    % If there aren't enough, move index to the new unit
    else
        while row_index < length(X_all_train)
            if X_all_train(row_index,1) == unit_id
                row_index = row_index+1;

            else
```

```
                unit_id = X_all_train(row_index,1);
                break
            end
        end
    end
end


%% This takes the test data set and creates a new NN input data set
NN_input_test = [];
NN_RUL_test = [];

unit_id = 1;
row_index = 1;
cell_index = 1;
first_cycle = 1;

while row_index <= length(X_all_test) - window_size
    if row_index == 1
        NN_input_test(:,:,:,row_index) =
X_test_scaled(row_index:row_index+window_size-1,:);
        NN_RUL_test(:,row_index) = Y_test(row_index:row_index+window_size-
1,:);
        row_index = row_index+1;
    end

    % Ensure there are at least window_size cycles recorded before failure
    if Y_test(row_index,1) >= window_size
        % While unit number is still the same, insert samples into final
        while X_all_test(row_index+window_size-1,1) == unit_id
            % Insert rows i to i+window_size, all columns
            NN_input_test(:,:,:,cell_index) =
X_test_scaled(row_index:row_index+window_size-1,:);
            NN_RUL_test(:,cell_index) =
Y_test(row_index:row_index+window_size-1,:);
            cell_index = cell_index + 1;
            row_index = row_index+1;
            if row_index > length(X_test_scaled) - window_size
                return
            end
        end
        % Once the unit number is changed, need to 'hop' down, change unit
        row_index = row_index + window_size-1;
        unit_id = X_all_test(row_index,1);
    % If there aren't enough, move index to the new unit
    else
        while row_index < length(X_all_test)
            if X_all_test(row_index,1) == unit_id
                row_index = row_index+1;

            else
                unit_id = X_all_test(row_index,1);
                break
            end
```

```
        end
    end
end
%% OPTIONAL
% attempt to split 75 - 25
% ensure the inputs and target are in the same cell indexes
% create an array from 1 to 100 (number of engines) psuedo randomly
% without repeating or omitting any numbers
test_perc = .75;
val_perc = .25;
a = randperm(num_units_train);
CNN_input_train = [];
CNN_RUL_train = [];
index = 1;
% for the first 75% of engine ids
for i = 1:num_units_test*test_perc
    % iterate through all
    for j = 1:length(X_train_pre_split)
        if X_train_pre_split(j,1) == a(i)
            CNN_input_train(index,:) = X_train_pre_split(j,:);
            CNN_RUL_train(index,:) = Y_train(j,:);
            index = index + 1;
        end
    end
end
k = i;

CNN_input_val = [];
CNN_RUL_val = [];
index = 1;
for i = 1:num_units_test*val_perc
    % iterate through all
    for j = 1:length(X_train_pre_split)
        if X_train_pre_split(j,1) == a(i+k)
            CNN_input_val(index,:) = X_train_pre_split(j,:);
            CNN_RUL_val(index,:) = Y_train(j,:);
            index = index + 1;
        end
    end
end
%% This takes the training data set and creates a new NN input data set (75 %)
% Takes 30 sonsecutive cycles with a step size of 1
window_size = 30;

NN_input_train = [];
NN_RUL_train = [];

unit_id = 1;
row_index = 1;
cell_index = 1;
first_cycle = 1;

while row_index <= length(CNN_input_train) - window_size
    if row_index == 1
```

```
        NN_input_train(:,:,:,row_index) =
CNN_input_train(row_index:row_index+window_size-1,3:15);
        NN_RUL_train(:,row_index) =
CNN_RUL_train(row_index:row_index+window_size-1,:);
        row_index = row_index+1;
    end

    if CNN_RUL_train(row_index,1) >= window_size
        while CNN_input_train(row_index+window_size-1,1) == unit_id
            NN_input_train(:,:,:,cell_index) =
CNN_input_train(row_index:row_index+window_size-1,3:15);
            NN_RUL_train(:,cell_index) =
CNN_RUL_train(row_index:row_index+window_size-1,:);
            cell_index = cell_index + 1;
            row_index = row_index+1;
            if row_index > length(CNN_input_train) - window_size
                return
            end
        end
        % Once the unit number is changed, need to 'hop' down, change unit
        row_index = row_index + window_size-1;
        unit_id = CNN_input_train(row_index,1);
    % If there aren't enough, move index to the new unit
    else
        while row_index < length(CNN_input_train)
            if CNN_input_train(row_index,1) == unit_id
                row_index = row_index+1;

            else
                unit_id = CNN_input_train(row_index,1);
                break
            end
        end
    end
end

%% This takes the test data set and creates a new NN validation data set (25
%)
% Takes 30 consecutive cycles with a step size of 1
window_size = 30;

NN_input_val = [];
NN_RUL_val = [];

unit_id = 1;
row_index = 1;
cell_index = 1;
first_cycle = 1;

while row_index <= length(CNN_input_val) - window_size
    if row_index == 1
        NN_input_val(:,:,:,row_index) =
CNN_input_val(row_index:row_index+window_size-1,3:15);
```

```
        NN_RUL_val(:,row_index) = CNN_RUL_val(row_index:row_index+window_size-
1,:);
        row_index = row_index+1;
    end

    if CNN_RUL_val(row_index,1) >= window_size
        while CNN_input_val(row_index+window_size-1,1) == unit_id
            NN_input_val(:,:,:,cell_index) =
CNN_input_val(row_index:row_index+window_size-1,3:15);
            NN_RUL_val(:,cell_index) =
CNN_RUL_val(row_index:row_index+window_size-1,:);
            cell_index = cell_index + 1;
            row_index = row_index+1;
            if row_index > length(CNN_input_val) - window_size
                return
            end
        end
        % Once the unit number is changed, need to 'hop' down, change unit
        row_index = row_index + window_size-1;
        unit_id = CNN_input_val(row_index,1);
    % If there aren't enough, move index to the new unit
    else
        while row_index < length(CNN_input_val)
            if CNN_input_val(row_index,1) == unit_id
                row_index = row_index+1;

            else
                unit_id = CNN_input_val(row_index,1);
                break
            end
        end
    end
end

%% Neural Net Model Building and Training Function

% Architecture Hyperparameters/
% Training Hyperparameters
dropoutrate = 0.50;
maxEpochs = 600;
miniBatchSize = 512;
numFeatures = [30 13 1];
numHiddenNodes = 100;
numResponses = 30;
InitialLearnRate = 0.01;
GradientThreshold = 1;

% CNN inputs are 2D: time_window x n_features
% Four convolutional layers with 10 filters and filter size 10x1
% numResponses should be a 30x1 RUL prediction

lgraph = layerGraph;
lgraph = addLayers(lgraph, [sequenceInputLayer(numFeatures,"Name","input");
    sequenceFoldingLayer("Name", "fold")]);
```

```
lgraph = addLayers(lgraph, [convolution2dLayer([10
1],10,"Padding","Same","Name","conv_1");
    tanhLayer("Name","tanh_1")]);

lgraph = addLayers(lgraph, [convolution2dLayer([10
1],10,"Padding","Same","Name","conv_2");
    tanhLayer("Name","tanh_2")]);

lgraph = addLayers(lgraph, [convolution2dLayer([10
1],10,"Padding","Same","Name","conv_3");
    tanhLayer("Name","tanh_3")]);

lgraph = addLayers(lgraph, [convolution2dLayer([10
1],10,"Padding","Same","Name","conv_4");
    tanhLayer("Name","tanh_4")]);

lgraph = addLayers(lgraph, [convolution2dLayer([3
1],1,"Padding","Same","Name","conv_5");
    tanhLayer("Name","tanh_5")]);

lgraph = addLayers(lgraph, [sequenceUnfoldingLayer("Name", "unfold")
    flattenLayer("Name", "flatten")
    dropoutLayer(dropoutrate, "Name", "dropout")]);

lgraph = addLayers(lgraph, [fullyConnectedLayer(numHiddenNodes, "Name",
"FC_1");
    tanhLayer("Name","tanh_6")]);

lgraph = addLayers(lgraph, [fullyConnectedLayer(1, "Name", "FC_2");
    regressionLayer("Name", "output")]);

lgraph = connectLayers(lgraph, "fold/out", "conv_1");
lgraph = connectLayers(lgraph, "tanh_1", "conv_2");
lgraph = connectLayers(lgraph, "tanh_2", "conv_3");
lgraph = connectLayers(lgraph, "tanh_3", "conv_4");
lgraph = connectLayers(lgraph, "tanh_4", "conv_5");
lgraph = connectLayers(lgraph, "tanh_5", "unfold/in");
lgraph = connectLayers(lgraph, "dropout", "FC_1");
lgraph = connectLayers(lgraph, "fold/miniBatchSize", "unfold/miniBatchSize");
lgraph = connectLayers(lgraph, "tanh_6", "FC_2");

% Training Options
options = trainingOptions('adam', ...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'InitialLearnRate',InitialLearnRate, ...
    'GradientThreshold',GradientThreshold, ...
    'Shuffle','never', ...
    'Plots','training-progress',...
    'ValidationData', {NN_input_val, NN_RUL_val_new}, ...
    "OutputNetwork", "best-validation-loss",... % added to keep best model
    'Verbose',0);
```

```
%%
for i=1:size(NN_RUL_train,2)
    NN_RUL_train_new(i) = NN_RUL_train(30,i);
end
%%
for i=1:size(NN_RUL_test,2)
    NN_RUL_test_new(i) = NN_RUL_test(30,i);
end
%%
for i=1:size(NN_RUL_val,2)
    NN_RUL_val_new(i) = NN_RUL_val(30,i);
end
%%

net = trainNetwork(NN_input_train, NN_RUL_train_new,lgraph,options);

%%

t_hat_test = predict(net, NN_input_test);

%RMSE_train = {sqrt(mean((Y_train - t_hat_train).^2))}
RMSE_test = {sqrt(mean((NN_RUL_test_new - t_hat_test).^2))};

%%
for i = 1:width(NN_RUL_test)
    YTestLast(i) = NN_RUL_test(30,i);
end
%end
figure()
plot(t_hat_test, YTestLast, 'X')
hold on
title("LSMT Predictions for FD001 Test. RMSE = ", RMSE_test)
xlabel("Predicted RUL")
ylabel("True RUL")

hold off
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. PYTHON SOURCE CODE FOR GENERATING FLIGHT HOUR DATA

```
# FlightHours.py
# Captain William Frazier
# NPS Thesis research
import csv
from datetime import date, timedelta
import matplotlib.pyplot as plt
class FlightHours(object):
''' Dictionary of BUNOs:calendar{date:fltshrs} '''
def __init__(self):
self.data = []
self.bunos = {}
self.beginning = date(1997, 4, 3)
self.end = date.today()
self.day_delta = (self.end - self.beginning).days
self.input_filename = 'Buno Flight Hours Between Dates_22_Aug_22.csv'
self.output_filename = 'Buno Flight Data Between Dates_22_Aug_22.csv'
self.temp_output = []
# Indicies
self.TEC = 0
self.BUNO = 1
self.HRS = 2
self.TMR_CODE = 3
self.TMR_DESC = 4
self.TTL_LNDG = 5
self.LNDG_TYPE = 6
self.LNDG_DESC = 7
self.DATE = 8
self.NON_AUST_LNDG_CODES = ['0','5','8','9','E','H','J','K','Y','Z']
self.AUST_LNDG_CODES = ['6','7','F','G','L','M','P','Q','T','W']
self.SHIP_LNG_CODES = ['1','2','3','4','A','B','C','D','N']
self.BUNO_HRS = 0
self.BUNO_LNDG = 1
self.TMR_1 = 2
self.TMR_2 = 3
self.TMR_3 = 4
self.TMR_4 = 5
self.TMR_5 = 6
self.TMR_6 = 7
self.TMR_7 = 8
self.NON_AUST_LNDGS = 9
C:\Users\willi\OneDrive\Desktop\Latest_9\FlightHours.py
Page 2, last modified 08/31/22 18:03:19
self.AUST_LNDGS = 10
self.SHIP_LNDGS = 11
def read_data(self):
''' Reads in NAVFLIR data from the DECKPLATE saved csv file. '''
```

```
with open(self.input_filename, 'r') as file:
reader = csv.reader(file)
for row in reader:
self.data.append(row)
def write_data(self):
''' Writes the flight data generated by this class to a csv. '''
with open(self.output_filename, 'w', newline='') as file:
writer = csv.writer(file)
writer.writerow(['flthrs', 'lndgs', 'TMR_1_hrs', 'TMR_2_hrs', 'TMR_3_hrs',
'TMR_4_hrs', 'TMR_5_hrs', 'TMR_6_hrs', 'TMR_7_hrs', 'NON_AUST_lndgs',
'AUST_lndgs', 'SHIP_lndgs'])
for buno in self.bunos.keys():
writer.writerows(self.bunos[buno])
def format_date(self):
''' Converts flight hour date string from .csv into datetime data type.
Assumes .csv date is in MM/DD/YYYY and needs to be [YYYY, MM, DD]'''
for row in range(1,len(self.data)):
mdy = self.data[row][self.DATE].split("/")
self.data[row][self.DATE] = date(int(mdy[2]), int(mdy[0]), int(mdy[1]))
def build_calendar(self):
''' Fills the dictionary with the key:value pair of date:[0]
The date range is from self.beginning to date.today()
'''
new_calendar = {}
# Each day has a list of the flight data:
# [Total flthrs, Total Landings, TMR_1 flthrs, TMR_2 flthrs, TMR_3 flthrs,
# TMR_4 flthrs, TMR_5 flthrs, TMR_6 flthrs, TMR_7 flthrs,
# NON_AUST_LNDGS, AUST_LNDGS, SHIP_LNDGS]
for day in range(self.day_delta):
new_calendar[self.beginning + timedelta(day)] = [0,0,0,0,0,0,0,0,0,0,0,0]
return new_calendar
def build_bunos(self):
''' Generates master dictionary of BUNOs:calendar{} which also
calls build_calendar to generate the value which is a dictionary as well. '''
C:\Users\willi\OneDrive\Desktop\Latest_9\FlightHours.py
Page 3, last modified 08/31/22 18:03:19
for row in range(1,len(self.data)):
if self.data[row][self.BUNO] not in self.bunos:
self.bunos[self.data[row][self.BUNO]] = self.build_calendar()
def fill_flthrs(self):
''' Fills the BUNO's calendar with flighthours using flight summary .csv data
The date column is already in datetime format which matches dictionary key'''
# Step 1: Fill flight record hours into the dates flown
for row in range(1,len(self.data)):
TMR_1_hrs = 0
TMR_2_hrs = 0
TMR_3_hrs = 0
TMR_4_hrs = 0
TMR_5_hrs = 0
TMR_6_hrs = 0
TMR_7_hrs = 0
```

```
# Column values from csv
data = self.data[row]
current_buno = self.data[row][self.BUNO] # Key for self.bunos dictionary
current_date = self.data[row][self.DATE] # Key for buno's calendar dictionary
flt_hrs_flown = self.data[row][self.HRS] # Value to increment for buno's
calendar dictionary
tmr_code = self.data[row][self.TMR_CODE]
if tmr_code != '':
tmr_code = str(tmr_code)[0]
if tmr_code == "1":
TMR_1_hrs = flt_hrs_flown
elif tmr_code == "2":
TMR_2_hrs = flt_hrs_flown
elif tmr_code == "3":
TMR_3_hrs = flt_hrs_flown
elif tmr_code == "4":
TMR_4_hrs = flt_hrs_flown
elif tmr_code == "5":
TMR_5_hrs = flt_hrs_flown
elif tmr_code == "6":
TMR_6_hrs = flt_hrs_flown
elif tmr_code == "7":
TMR_7_hrs = flt_hrs_flown
if flt_hrs_flown != '':
current_hrs = self.bunos[current_buno][current_date][self.BUNO_HRS]
C:\Users\willi\OneDrive\Desktop\Latest_9\FlightHours.py
Page 4, last modified 08/31/22 18:03:19
current_TMR_1_hrs = self.bunos[current_buno][current_date][self.TMR_1]
current_TMR_2_hrs = self.bunos[current_buno][current_date][self.TMR_2]
current_TMR_3_hrs = self.bunos[current_buno][current_date][self.TMR_3]
current_TMR_4_hrs = self.bunos[current_buno][current_date][self.TMR_4]
current_TMR_5_hrs = self.bunos[current_buno][current_date][self.TMR_5]
current_TMR_6_hrs = self.bunos[current_buno][current_date][self.TMR_6]
current_TMR_7_hrs = self.bunos[current_buno][current_date][self.TMR_7]
# Increment date's flthrs for that BUNO
# Allows for multiple flights documented for one day
self.bunos[current_buno][current_date][self.BUNO_HRS] =
round(float(flt_hrs_flown) + float(current_hrs), 2)
self.bunos[current_buno][current_date][self.TMR_1] = round(float(TMR_1_hrs)
+ float(current_TMR_1_hrs), 2)
self.bunos[current_buno][current_date][self.TMR_2] = round(float(TMR_2_hrs)
+ float(current_TMR_2_hrs), 2)
self.bunos[current_buno][current_date][self.TMR_3] = round(float(TMR_3_hrs)
+ float(current_TMR_3_hrs), 2)
self.bunos[current_buno][current_date][self.TMR_4] = round(float(TMR_4_hrs)
+ float(current_TMR_4_hrs), 2)
self.bunos[current_buno][current_date][self.TMR_5] = round(float(TMR_5_hrs)
+ float(current_TMR_5_hrs), 2)
self.bunos[current_buno][current_date][self.TMR_6] = round(float(TMR_6_hrs)
+ float(current_TMR_6_hrs), 2)
self.bunos[current_buno][current_date][self.TMR_7] = round(float(TMR_7_hrs)
```

```
+ float(current_TMR_7_hrs), 2)
# Step 2: Iterate through each date in the calendar and sum current with previous
for current_buno in self.bunos.keys():
for day in range(1, self.day_delta):
current_day = self.beginning + timedelta(day)
prev_day = current_day - timedelta(1)
current_day_hrs = self.bunos[current_buno][current_day][self.BUNO_HRS]
current_day_TMR_1_hrs = self.bunos[current_buno][current_day][self.TMR_1]
current_day_TMR_2_hrs = self.bunos[current_buno][current_day][self.TMR_2]
current_day_TMR_3_hrs = self.bunos[current_buno][current_day][self.TMR_3]
current_day_TMR_4_hrs = self.bunos[current_buno][current_day][self.TMR_4]
current_day_TMR_5_hrs = self.bunos[current_buno][current_day][self.TMR_5]
current_day_TMR_6_hrs = self.bunos[current_buno][current_day][self.TMR_6]
current_day_TMR_7_hrs = self.bunos[current_buno][current_day][self.TMR_7]
prev_day_hrs = self.bunos[current_buno][prev_day][self.BUNO_HRS]
prev_day_TMR_1_hrs = self.bunos[current_buno][prev_day][self.TMR_1]
prev_day_TMR_2_hrs = self.bunos[current_buno][prev_day][self.TMR_2]
prev_day_TMR_3_hrs = self.bunos[current_buno][prev_day][self.TMR_3]
C:\Users\willi\OneDrive\Desktop\Latest_9\FlightHours.py
Page 5, last modified 08/31/22 18:03:19
prev_day_TMR_4_hrs = self.bunos[current_buno][prev_day][self.TMR_4]
prev_day_TMR_5_hrs = self.bunos[current_buno][prev_day][self.TMR_5]
prev_day_TMR_6_hrs = self.bunos[current_buno][prev_day][self.TMR_6]
prev_day_TMR_7_hrs = self.bunos[current_buno][prev_day][self.TMR_7]
self.bunos[current_buno][current_day][self.BUNO_HRS] = round(prev_day_hrs +
current_day_hrs,2)
self.bunos[current_buno][current_day][self.TMR_1] = round(prev_day_TMR_1_hrs
+ current_day_TMR_1_hrs,2)
self.bunos[current_buno][current_day][self.TMR_2] = round(prev_day_TMR_2_hrs
+ current_day_TMR_2_hrs,2)
self.bunos[current_buno][current_day][self.TMR_3] = round(prev_day_TMR_3_hrs
+ current_day_TMR_3_hrs,2)
self.bunos[current_buno][current_day][self.TMR_4] = round(prev_day_TMR_4_hrs
+ current_day_TMR_4_hrs,2)
self.bunos[current_buno][current_day][self.TMR_5] = round(prev_day_TMR_5_hrs
+ current_day_TMR_5_hrs,2)
self.bunos[current_buno][current_day][self.TMR_6] = round(prev_day_TMR_6_hrs
+ current_day_TMR_6_hrs,2)
self.bunos[current_buno][current_day][self.TMR_7] = round(prev_day_TMR_7_hrs
+ current_day_TMR_7_hrs,2)
def fill_landings(self):
''' Fills the BUNO's calendar with number of landings and type. '''
# Step 1: Fill number of landings into the dates flown
for row in range(1,len(self.data)):
non_aust_lndgs = 0
aust_lndgs = 0
ship_lndgs = 0
# Column values from csv
current_buno = self.data[row][self.BUNO] # Key for self.bunos dictionary
current_date = self.data[row][self.DATE] # Key for buno's calendar dictionary
num_landings = self.data[row][self.TTL_LNDG] # Value to increment for buno's
```

```
calendar dictionary
lndg_code = self.data[row][self.LNDG_TYPE]
if lndg_code != '':
lndg_code = str(lndg_code)
if lndg_code in self.NON_AUST_LNDG_CODES:
non_aust_lndgs = num_landings
elif lndg_code in self.AUST_LNDG_CODES:
aust_lndgs = num_landings
elif lndg_code in self.SHIP_LNG_CODES:
ship_lndgs = num_landings
C:\Users\willi\OneDrive\Desktop\Latest_9\FlightHours.py
Page 6, last modified 08/31/22 18:03:19
if num_landings != '':
current_landings = self.bunos[current_buno][current_date][self.BUNO_LNDG]
current_non_aust_lndgs =
self.bunos[current_buno][current_date][self.NON_AUST_LNDGS]
current_aust_lndgs = self.bunos[current_buno][current_date][self.AUST_LNDGS]
current_ship_lndgs = self.bunos[current_buno][current_date][self.SHIP_LNDGS]
# Increment date's flthrs for that BUNO
# Allows for multiple flights documented for one day
self.bunos[current_buno][current_date][self.BUNO_LNDG] =
round(float(num_landings) + float(current_landings), 2)
self.bunos[current_buno][current_date][self.NON_AUST_LNDGS] =
round(float(non_aust_lndgs) + float(current_non_aust_lndgs), 2)
self.bunos[current_buno][current_date][self.AUST_LNDGS] =
round(float(aust_lndgs) + float(current_aust_lndgs), 2)
self.bunos[current_buno][current_date][self.SHIP_LNDGS] =
round(float(ship_lndgs) + float(current_ship_lndgs), 2)
# Step 2: Iterate through each date in the calendar and sum current with previous
for current_buno in self.bunos.keys():
for day in range(1, self.day_delta):
current_day = self.beginning + timedelta(day)
prev_day = current_day - timedelta(1)
current_day_lndgs = self.bunos[current_buno][current_day][self.BUNO_LNDG]
current_day_non_aust_lndgs =
self.bunos[current_buno][current_day][self.NON_AUST_LNDGS]
current_day_aust_lndgs = self.bunos[current_buno][current_day][self.AUST_LNDGS]
current_day_ship_lndgs = self.bunos[current_buno][current_day][self.SHIP_LNDGS]
prev_day_lndgs = self.bunos[current_buno][prev_day][self.BUNO_LNDG]
prev_day_non_aust_lndgs =
self.bunos[current_buno][prev_day][self.NON_AUST_LNDGS]
prev_day_aust_lndgs = self.bunos[current_buno][prev_day][self.AUST_LNDGS]
prev_day_ship_lndgs =
self.bunos[current_buno][prev_day][self.SHIP_LNDGS]
self.bunos[current_buno][current_day][self.BUNO_LNDG] = round(prev_day_lndgs
+ current_day_lndgs,2)
self.bunos[current_buno][current_day][self.NON_AUST_LNDGS] =
round(prev_day_non_aust_lndgs + current_day_non_aust_lndgs,2)
self.bunos[current_buno][current_day][self.AUST_LNDGS] =
round(prev_day_aust_lndgs + current_day_aust_lndgs,2)
self.bunos[current_buno][current_day][self.SHIP_LNDGS] =
```

```
round(prev_day_ship_lndgs + current_day_ship_lndgs,2)
C:\Users\willi\OneDrive\Desktop\Latest_9\FlightHours.py
Page 7, last modified 08/31/22 18:03:19
def buno_query(self, buno, start_date, end_date):
''' Returns the tracked numbers flown by a buno between two dates '''
records = [0,0,0,0,0,0,0,0,0,0,0,0]
if buno in self.bunos.keys():
flthrs = round(self.bunos[buno][end_date][self.BUNO_HRS] -
self.bunos[buno][start_date][self.BUNO_HRS] , 2)
lndgs = round(self.bunos[buno][end_date][self.BUNO_LNDG] -
self.bunos[buno][start_date][self.BUNO_LNDG] , 2)
TMR_1_hrs = round(self.bunos[buno][end_date][self.TMR_1] -
self.bunos[buno][start_date][self.TMR_1] , 2)
TMR_2_hrs = round(self.bunos[buno][end_date][self.TMR_2] -
self.bunos[buno][start_date][self.TMR_2] , 2)
TMR_3_hrs = round(self.bunos[buno][end_date][self.TMR_3] -
self.bunos[buno][start_date][self.TMR_3] , 2)
TMR_4_hrs = round(self.bunos[buno][end_date][self.TMR_4] -
self.bunos[buno][start_date][self.TMR_4] , 2)
TMR_5_hrs = round(self.bunos[buno][end_date][self.TMR_5] -
self.bunos[buno][start_date][self.TMR_5] , 2)
TMR_6_hrs = round(self.bunos[buno][end_date][self.TMR_6] -
self.bunos[buno][start_date][self.TMR_6] , 2)
TMR_7_hrs = round(self.bunos[buno][end_date][self.TMR_7] -
self.bunos[buno][start_date][self.TMR_7] , 2)
NON_AUST_lndgs = round(self.bunos[buno][end_date][self.NON_AUST_LNDGS] -
self.bunos[buno][start_date][self.NON_AUST_LNDGS] , 2)
AUST_lndgs = round(self.bunos[buno][end_date][self.AUST_LNDGS] -
self.bunos[buno][start_date][self.AUST_LNDGS] , 2)
SHIP_lndgs = round(self.bunos[buno][end_date][self.SHIP_LNDGS] -
self.bunos[buno][start_date][self.SHIP_LNDGS] , 2)
records = [flthrs, lndgs, TMR_1_hrs, TMR_2_hrs, TMR_3_hrs, TMR_4_hrs, TMR_5_hrs,
TMR_6_hrs, TMR_7_hrs, NON_AUST_lndgs, AUST_lndgs, SHIP_lndgs]
start_date = start_date.strftime('%m/%d/%Y')
temp_records = records
temp_records = list(str(x) for x in temp_records)
temp_records.insert(0,start_date)
temp_records.insert(0,buno)
self.temp_output.append(temp_records)
return records
def buno_query_all_data(self, buno, start_date, end_date):
''' Returns the tracked numbers flown by a buno between two dates '''
C:\Users\willi\OneDrive\Desktop\Latest_9\FlightHours.py
Page 8, last modified 08/31/22 18:03:19
if buno in self.bunos.keys():
flthrs = round(self.bunos[buno][end_date][self.BUNO_HRS] -
self.bunos[buno][start_date][self.BUNO_HRS] , 2)
lndgs = round(self.bunos[buno][end_date][self.BUNO_LNDG] -
self.bunos[buno][start_date][self.BUNO_LNDG] , 2)
TMR_1_hrs = round(self.bunos[buno][end_date][self.TMR_1] -
self.bunos[buno][start_date][self.TMR_1] , 2)
```

```
TMR_2_hrs = round(self.bunos[buno][end_date][self.TMR_2] -
self.bunos[buno][start_date][self.TMR_2] , 2)
TMR_3_hrs = round(self.bunos[buno][end_date][self.TMR_3] -
self.bunos[buno][start_date][self.TMR_3] , 2)
TMR_4_hrs = round(self.bunos[buno][end_date][self.TMR_4] -
self.bunos[buno][start_date][self.TMR_4] , 2)
TMR_5_hrs = round(self.bunos[buno][end_date][self.TMR_5] -
self.bunos[buno][start_date][self.TMR_5] , 2)
TMR_6_hrs = round(self.bunos[buno][end_date][self.TMR_6] -
self.bunos[buno][start_date][self.TMR_6] , 2)
TMR_7_hrs = round(self.bunos[buno][end_date][self.TMR_7] -
self.bunos[buno][start_date][self.TMR_7] , 2)
NON_AUST_lndgs = round(self.bunos[buno][end_date][self.NON_AUST_LNDGS] -
self.bunos[buno][start_date][self.NON_AUST_LNDGS] , 2)
AUST_lndgs = round(self.bunos[buno][end_date][self.AUST_LNDGS] -
self.bunos[buno][start_date][self.AUST_LNDGS] , 2)
SHIP_lndgs = round(self.bunos[buno][end_date][self.SHIP_LNDGS] -
self.bunos[buno][start_date][self.SHIP_LNDGS] , 2)
else:
return [0,0,0,0,0,0,0,0,0,0,0,0]
return [flthrs, lndgs, TMR_1_hrs, TMR_2_hrs, TMR_3_hrs, TMR_4_hrs, TMR_5_hrs,
TMR_6_hrs, TMR_7_hrs, NON_AUST_lndgs, AUST_lndgs, SHIP_lndgs]
def display_records(self):
''' Prints the gathered flight records with proper formatting. '''
self.temp_output.insert(0,['buno', 'start_date', 'flthrs', 'lndgs', 'TMR_1_hrs',
'TMR_2_hrs', 'TMR_3_hrs', 'TMR_4_hrs', 'TMR_5_hrs', 'TMR_6_hrs', 'TMR_7_hrs',
'NON_AUST_lndgs', 'AUST_lndgs', 'SHIP_lndgs'])
max_lens = [len(str(max(i, key=lambda x: len(str(x))))) for i in
zip(*self.temp_output)]
print('\n'.join(' '.join(item[i].ljust(max_lens[i]) for i in range(len(max_lens)))
for item in self.temp_output))
print()
self.temp_output = []
C:\Users\willi\OneDrive\Desktop\Latest_9\FlightHours.py
Page 9, last modified 08/31/22 18:03:19
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G. PYTHON SOURCE CODE FOR GENERATING SERIAL NUMBER HISTORIES

```
# SernoHistory.py
# Captain William Frazier
# NPS Thesis research
# Save each P/N reports from Deckplate using DP-0025 Serial Number Tracking as a .csv
import csv
from datetime import date, timedelta
import matplotlib.pyplot as plt
from copy import deepcopy
import FlightHours
# Steps taken by this program:
#
# - Read in .csv files for each P/N associated with the component being evaluated
# - Those files are sorted oldest to newest completion date
# - The dates are formatted
# - A dictionary is created for the unique SerNo's as the keys
# - Each row of the data is read and new SerNo's are added to the dict
# - SerNo's already added have the row of data appended to the value list
class SernoHistory(object):
''' Dictionary of Sernos whose key:value pair will be
serno:[list of historical flight data].
This class has methods that will map out by calendar date
which BUNO the serno was installed on throughout its service.
It will then utilize methods from the FlightHours class to
populate data from flight records for that buno during a date
range subsequently providing the flight data of a serno during
a date range. '''
def __init__(self):
self.data = []
self.final_v22_PCA_data = []
self.sernos = {}
self.beginning = date(1997, 4, 3)
self.end = date.today()
self.day_delta = (self.end - self.beginning).days
self.raw_data_filename = 'DP-0025_WUC_27502X.csv'
self.save_data_filename = 'SORTED_WUC_27502X.csv'
self.save_decon_data_filename = 'DECONFLICTED_WUC_27502X.csv'
self.header = ''
self.header_extended = ''
self.serno_flight_data = {}
self.sorted_data = []
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 2, last modified 08/31/22 18:12:17
# Indicies
self.ACTION_DATE = 0
self.PART_NO = 1
self.CAGE = 2
```

```
self.SERNO = 3
self.WUC = 4
self.WEC_DESC = 5
self.ORG = 6
self.ORG_DESC = 7
self.TEC = 8
self.TMS = 9
self.BUNO = 10
self.REM_INS_FLAG = 11
self.JCN = 12
self.MCN = 13
self.TYPE_MAINT = 14
self.TRANS_CODE = 15
self.ACT_TAKEN = 16
self.MAL_CODE = 17
self.COMP_DATE = 18
self.TIME_1_CODE = 19
self.TIME_1_CYCLE = 20
self.TIME_2_CODE = 21
self.TIME_2_CYCLE = 22
self.TIME_3_CODE = 23
self.TIME_3_CYCLE = 24
# Indicies
self.HRS = 25
self.LNDGs = 26
self.TMR_1 = 27
self.TMR_2 = 28
self.TMR_3 = 29
self.TMR_4 = 30
self.TMR_5 = 31
self.TMR_6 = 32
self.TMR_7 = 33
self.NON_AUST_LNDGS = 34
self.AUST_LNDGS = 35
self.SHIP_LNDGS = 36
# WUC Criteria
self.WUC_criteria = ['275021', '275020', '27502015', '27502115']
self.FlightHours = FlightHours.FlightHours()
self.FlightHours.read_data()
self.FlightHours.format_date()
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 3, last modified 08/31/22 18:12:17
self.FlightHours.build_bunos()
self.FlightHours.fill_flthrs()
self.FlightHours.fill_landings()
#self.FlightHours.write_data()
def read_data(self):
''' Reads in the DP-0025 report csv file generated from Deckplate. '''
with open(self.raw_data_filename, 'r') as file:
reader = csv.reader(file)
# Create local list that includes first row header
```

```
new_data = []
for row in reader:
new_data.append(row)
# save the header row and pop it off
self.header = new_data[0]
self.header_extended = self.header
self.header_extended.extend(['flthrs', 'lndgs'])
new_data.pop(0)
for row in new_data:
self.data.append(row)
def format_date(self):
''' Converts maintenance action date string from .csv into datetime data type.
Assumes .csv date is in MM/DD/YYYY and needs to be [YYYY, MM, DD]'''
for row in range(0,len(self.data)):
# Action Taken Date
mdy = self.data[row][self.ACTION_DATE].split("/")
if len(mdy) == 3:
self.data[row][self.ACTION_DATE] = date(int(mdy[2]), int(mdy[0]), int(mdy[1]))
else:
mdy = self.data[row][self.ACTION_DATE].split("-")
if len(mdy) == 3:
self.data[row][self.ACTION_DATE] = date(int(mdy[0]), int(mdy[1]),
int(mdy[2]))
# Action Completion Date
mdy = self.data[row][self.COMP_DATE].split("/")
if len(mdy) == 3:
self.data[row][self.COMP_DATE] = date(int(mdy[2]), int(mdy[0]), int(mdy[1]))
else:
mdy = self.data[row][self.COMP_DATE].split("-")
if len(mdy) == 3:
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 4, last modified 08/31/22 18:12:17
self.data[row][self.COMP_DATE] = date(int(mdy[0]), int(mdy[1]),
int(mdy[2]))
def write_data(self):
''' Saves the resultant data to a csv file. '''
with open(self.save_data_filename, 'w', newline="") as file:
writer = csv.writer(file)
writer.writerow(self.header_extended)
for key in self.sernos.keys():
entries = self.sernos[key]
writer.writerows(entries)
def sort_by_date(self, history):
''' Sorts an input list of historical R/I records by their action date value. '''
history.sort(key=lambda x: x[self.ACTION_DATE])
return history
def display_records(self, records):
''' Prints the gathered R/I action records with proper formatting. '''
max_lens = [len(str(max(i, key=lambda x: len(str(x))))) for i in zip(*records)]
print('\n'.join(' '.join(item[i].ljust(max_lens[i]) for i in range(len(max_lens)))
for item in records))
```

```
print()
def build_calendar(self, serno):
''' Fills the dictionary with the key:value pair of date:buno
The date range is from self.beginning to date.today()
Default BUNO installed on is blank '''
# All R/I are sorted. There is one I and one R for an installation period
# Iterate every 2 entries
new_calendar = self.FlightHours.build_calendar()
serno_history = self.sernos[serno]
index = 1
while index < len(serno_history):
buno = serno_history[index][self.BUNO]
install_date = serno_history[index-1][self.ACTION_DATE]
removal_date = serno_history[index][self.ACTION_DATE]
day_delta = (removal_date - install_date).days
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 5, last modified 08/31/22 18:12:17
for day in range(1, day_delta):
#current_day = install_date + timedelta(day)
current_day = install_date + timedelta(day)
prev_day = current_day - timedelta(1)
new_calendar[current_day] = self.FlightHours.buno_query(buno, prev_day,
current_day)
index += 2
return new_calendar
def build_sernos(self):
''' Generates master dictionary of sernos:[R/I actions].
If the SerNo is not yet a key in the dict, it is added and its
value is the corresponding row of data i.e. the oldest action
completed from the data. Subsequent rows with that same SerNo will
have that data appended to the value list. '''
for row in range(0,len(self.data)):
# Ensures only entries whose WUC begins with the WUC_criteria
if str(self.data[row][self.WUC]) in self.WUC_criteria:
#key = str(self.data[row][self.SERNO]) + '/' +
str(self.data[row][self.PART_NO])
key = str(self.data[row][self.SERNO])
if key not in self.sernos:
self.sernos[key] = [self.data[row]]
else:
self.sernos[key].extend([self.data[row]])
def check_missing(self):
''' Compares all of the JCNs for FST Weibull analysis with JCNs found for
R/I data in deckplate to ensure each component used in the analysis has an R/I MAF. '''
# load all MCNs from v22 fst file
with open('v22_fst_results_pca.csv', 'r') as file:
reader = csv.reader(file)
failures = []
for row in reader:
failures.append(row)
MCNs_fst = [str(row[2]) for row in failures]
```

```
MCNs_deckplate = [str(row[self.MCN]) for row in self.data]
for mcn in MCNs_fst:
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 6, last modified 08/31/22 18:12:17
if mcn not in MCNs_deckplate:
print("MCN missing:", mcn)
def sort_serno_history(self, serno_history):
''' Applies the logic to the serno:[history] dict key:value pairs that sorts
the R/I actions in the correct chronological sequence. This rearranging sets
the conditions to begin recording the ranges of dates for which a SerNo was
installed on a BUNO. '''
new_history = []
entry = 0
while entry < len(serno_history):
# Get records
current_record = deepcopy(serno_history[entry])
previous_record = None
next_record = None
# If this isnt the first entry, get the previous record
if entry != 0:
previous_record = deepcopy(serno_history[entry-1])
# If this isnt the last entry, get the next record
if entry < len(serno_history) - 1:
next_record = deepcopy(serno_history[entry+1])
# First Entry
if entry == 0:
# First Entry is Install
if current_record[self.REM_INS_FLAG] == "I":
new_history.append(current_record)
# Is there another same day entry
if next_record and current_record[self.ACTION_DATE] ==
next_record[self.ACTION_DATE]:
# Same day is another Install, removal missing, impute
if next_record[self.REM_INS_FLAG] == "I":
imputed_record = deepcopy(current_record)
imputed_record[self.REM_INS_FLAG] = "R"
# First Entry is Install, second is not same date, Second is Install
elif next_record and next_record[self.REM_INS_FLAG] == "I":
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 7, last modified 08/31/22 18:12:17
# A removal is missing, impute it with the current record's Action date
imputed_record = deepcopy(current_record)
imputed_record[self.REM_INS_FLAG] = "R"
new_history.append(imputed_record)
# First Entry is Install, second is not same date, Second is Removal
# First Entry is Removal, install is missing
elif entry == 0 and current_record[self.REM_INS_FLAG] == "R":
# Assumed component has been installed from aircraft's delivery
imputed_record = deepcopy(current_record)
imputed_record[self.ACTION_DATE] = self.beginning
imputed_record[self.REM_INS_FLAG] = "I"
```

```
new_history.append(imputed_record)
new_history.append(current_record)
# Last entry
elif entry == len(serno_history) - 1:
# Last Entry is Install and previous is install, missing removal, impute
if current_record[self.REM_INS_FLAG] == "I" and
previous_record[self.REM_INS_FLAG] == "I":
imputed_record = deepcopy(current_record)
imputed_record[self.REM_INS_FLAG] = "R"
new_history.append(imputed_record)
new_history.append(current_record)
# Middle entry
else:
# Entry is Install
if current_record[self.REM_INS_FLAG] == "I":
# Previous is also Install, missing removal, impute
if previous_record[self.REM_INS_FLAG] == "I":
imputed_record = deepcopy(current_record)
imputed_record[self.REM_INS_FLAG] = "R"
new_history.append(imputed_record)
# Entry is Removal
```

```
else:
# Previous is also Removal, missing removal, impute
if previous_record[self.REM_INS_FLAG] == "R":
imputed_record = deepcopy(current_record)
imputed_record[self.REM_INS_FLAG] = "I"
new_history.append(imputed_record)
new_history.append(current_record)
entry += 1
return new_history
def add_flight_data_to_history(self, serno):
# History is now sorted there is an Install and Removal pair every aircraft
total_hrs = 0
total_lndgs = 0
total_TMR_1_hrs = 0
total_TMR_2_hrs = 0
total_TMR_3_hrs = 0
total_TMR_4_hrs = 0
total_TMR_5_hrs = 0
total_TMR_6_hrs = 0
total_TMR_7_hrs = 0
total_NON_AUST_lndgs = 0
total_AUST_lndgs = 0
total_SHIP_lndgs = 0
data = [str(total_hrs), str(total_lndgs), str(total_TMR_1_hrs), str(total_TMR_2_hrs),
str(total_TMR_3_hrs), str(total_TMR_4_hrs), str(total_TMR_5_hrs),
str(total_TMR_6_hrs),
str(total_TMR_7_hrs), str(total_NON_AUST_lndgs), str(total_AUST_lndgs),
str(total_SHIP_lndgs)]
```

```
entry = 0
while entry < len(self.sernos[serno]):
if entry == 0:
self.sernos[serno][entry].extend(data)
elif self.sernos[serno][entry][self.REM_INS_FLAG] == "I":
self.sernos[serno][entry].extend(data)
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 9, last modified 08/31/22 18:12:17
else:
install_date = self.sernos[serno][entry-1][self.ACTION_DATE]
removal_date = self.sernos[serno][entry][self.ACTION_DATE]
buno = self.sernos[serno][entry][self.BUNO]
[flthrs, lndgs, TMR_1_hrs, TMR_2_hrs, TMR_3_hrs,
TMR_4_hrs, TMR_5_hrs, TMR_6_hrs, TMR_7_hrs,
NON_AUST_lndgs, AUST_lndgs, SHIP_lndgs] =
self.FlightHours.buno_query_all_data(buno, install_date, removal_date)
total_hrs += flthrs
total_lndgs += lndgs
total_TMR_1_hrs += TMR_1_hrs
total_TMR_2_hrs += TMR_2_hrs
total_TMR_3_hrs += TMR_3_hrs
total_TMR_4_hrs += TMR_4_hrs
total_TMR_5_hrs += TMR_5_hrs
total_TMR_6_hrs += TMR_6_hrs
total_TMR_7_hrs += TMR_7_hrs
total_NON_AUST_lndgs += NON_AUST_lndgs
total_AUST_lndgs += AUST_lndgs
total_SHIP_lndgs += SHIP_lndgs
data = [str(total_hrs), str(total_lndgs), str(total_TMR_1_hrs),
str(total_TMR_2_hrs),
str(total_TMR_3_hrs), str(total_TMR_4_hrs), str(total_TMR_5_hrs),
str(total_TMR_6_hrs),
str(total_TMR_7_hrs), str(total_NON_AUST_lndgs),
str(total_AUST_lndgs), str(total_SHIP_lndgs)]
self.sernos[serno][entry].extend(data)
entry += 1
def flight_hour_query_steps(self):
''' Takes the user through steps of inputting flight hour query parameters. '''
try:
# Enter BUNO
buno = str(input("Enter the BUNO\n"))
# Start date or enter 0 to use the earliest flight date of the V22 program
start = str(input("Enter the start date of the query such as MM/DD/YYYY or 0 for
the beginning of the V22 program\n"))
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 10, last modified 08/31/22 18:12:17
if start != "0":
mdy = start.split("/")
start = date(int(mdy[2]), int(mdy[0]), int(mdy[1]))
else:
start = self.beginning
```

```
end = str(input("Enter the end date of the query such as MM/DD/YYYY\n"))
mdy = end.split("/")
end = date(int(mdy[2]), int(mdy[0]), int(mdy[1]))
except Exception:
print("not a valid component or response format\n")
# input parameters for the FlightHours class
return buno, start, end
def serno_query_steps(self):
''' Takes the user through input steps to choose parameters for a SerNo R/I history
query. '''
try:
serno = str(input("Enter the SerNo you wish to evaluate\n"))
self.query_serno(serno)
except Exception:
print(str(serno) + "is not a valid component or response format\n")
def auto_serno_query_steps(self):
''' Takes the user through input steps to choose parameters for a SerNo R/I history
query. '''
try:
for serno in self.sernos.keys():
self.auto_serno_query(serno)
self.write_data()
except Exception:
print(str(serno) + "had an issue in auto_serno_query_steps\n")
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 11, last modified 08/31/22 18:12:17
def auto_serno_query(self, serno):
''' Takes the input serno and returns all R/I records for that
SerNo to include all part numbers. Used to identify errors. '''
try:
# Iterate through serno values in the dict
temp_results = []
temp_value = deepcopy(self.sernos[serno])
for row in temp_value:
temp_results.append(row)
# Properly sort/impute R/I entries and replace the old history with the new
new_history = self.sort_serno_history(self.get_history(serno))
self.sernos[serno] = new_history
# append cumulative flight data to R/I entry
self.add_flight_data_to_history(serno)
# Call for the creation of flight data calendar and insert into serno_calendar
serno_calendar = self.build_calendar(serno)
self.serno_flight_data[serno] = serno_calendar
except Exception as e:
print("Error with Serno ", str(serno), e)
def get_history(self, serno):
''' Generates the list of R/I actions based on a serno/part combo and adds the
header. '''
history_list = deepcopy(self.sernos[serno])
return history_list
def SerNo_query_short(self, history_list):
```

```
''' Returns the R/I history of a SerNo '''
history_list.insert(0,self.header)
# Convert dates to strings for formating
for row in range(1,len(history_list)):
history_list[row][self.ACTION_DATE] =
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 12, last modified 08/31/22 18:12:17
history_list[row][self.ACTION_DATE].strftime('%m/%d/%Y')
history_list[row][self.COMP_DATE] =
history_list[row][self.COMP_DATE].strftime('%m/%d/%Y')
# only the essential columns of data
#indicies = [0,1,3,7,10,11,12,13,22,24]
indicies = [self.ACTION_DATE,
self.PART_NO,
self.SERNO,
self.WUC,
self.ORG_DESC,
self.BUNO,
self.REM_INS_FLAG,
self.JCN,
self.MCN,
self.COMP_DATE,
self.TIME_1_CYCLE]
short_list = []
for row in history_list:
short_list.append([row[index] for index in indicies])
# Print using formatting
self.display_records(short_list)
def SerNo_query_long(self, history_list, header):
''' Returns the R/I history of a SerNo '''
history_list.insert(0,header)
# Convert dates to strings for formating
for row in range(1,len(history_list)):
history_list[row][self.ACTION_DATE] =
history_list[row][self.ACTION_DATE].strftime('%m/%d/%Y')
history_list[row][self.COMP_DATE] =
history_list[row][self.COMP_DATE].strftime('%m/%d/%Y')
# Print using formatting
self.display_records(history_list)
def query_serno(self, user_serno):
''' Takes the users input serno and returns all R/I records for that
SerNo to include all part numbers. Used to identify errors. '''
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 13, last modified 08/31/22 18:12:17
try:
# Iterate through serno values in the dict
temp_results = []
for key in self.sernos.keys():
# make a copy of the dict key
temp_key = deepcopy(key)
# compare the current dict key with the user input serno
```

```
if str(user_serno) == str(temp_key):
# If they match, add a copy of the dict key's value to the return list
temp_value = deepcopy(self.sernos[temp_key])
for row in temp_value:
temp_results.append(row)
short_results = deepcopy(temp_results)
long_results = deepcopy(temp_results)
# Automaticall provide short version of results
self.SerNo_query_short(short_results)
long_response = str(input("Enter y to see long version or n to continue\n"
"Enter s to sort and build flight data\n"))
if long_response == 'y':
self.SerNo_query_long(long_results, self.header)
elif long_response == 's':
# Properly sort/impute R/I entries and replace the old history with the new
new_history = self.sort_serno_history(self.get_history(user_serno))
self.sernos[user_serno] = new_history
# append cumulative flight data to R/I entry
self.add_flight_data_to_history(user_serno)
# Call for the creation of flight data calendar and insert into serno_calendar
serno_calendar = self.build_calendar(user_serno)
self.serno_flight_data[user_serno] = serno_calendar
response = str(input("Enter full to see all serno flight entries\n"
"Enter short to see cululative flight data for R/I
entries only\n"))
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 14, last modified 08/31/22 18:12:17
if response == 'full':
calendar = self.serno_flight_data[user_serno]
for day in calendar.keys():
print(calendar[day])
elif response == 'short':
self.SerNo_query_long(self.get_history(user_serno), self.header_extended)
except Exception:
print("something went wrong")
def query_buno(self, buno):
''' Iterates through all serno/part keys in the dict and their
value lists of R/I actions to search for the BUNO. '''
try:
# Iterate through all serno/part_num key values in the dict
temp_results = []
for key in self.sernos.keys():
# make a copy of the dict key
temp_key = deepcopy(key)
temp_value = deepcopy(self.sernos[temp_key])
for row in temp_value:
if row[self.BUNO] == buno:
temp_results.append(row)
# sort the results by completion date oldest to newest
sorted_results = self.sort_by_date(temp_results)
results = deepcopy(sorted_results)
```

```
return results
except Exception:
print("something went wrong quurying buno ", str(buno))
def deconflict_duplicates(self):
''' Once all missing R/I data has been imputed, deconflicts multiple
SerNos that were assumed to have been installed during production that
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 15, last modified 08/31/22 18:12:17
occupy and aircrafts left or right hand WUC. '''
try:
self.data = []
self.new_data = []
self.sernos = {}
with open(self.save_data_filename, 'r') as file:
reader = csv.reader(file)
# Create local list that includes first row header
new_data = []
for row in reader:
new_data.append(row)
# save the header row and pop it off
self.header_extended = new_data[0]
new_data.pop(0)
for row in new_data:
self.data.append(row)
self.format_date()
self.build_sernos()
# Returns all R/I entries for a buno
for buno in self.FlightHours.bunos.keys():
buno_history = self.query_buno(buno)
buno_history.insert(0,self.header_extended)
# DECONFLICTION
# Get all entries that are installations from the beginning until a removal
is introduced
buno_history.pop(0)
right_PCA = None
left_PCA = None
right_PCA_index = None
left_PCA_index = None
i = 0
# Get the first removals for right and left PCAs
while right_PCA == None and i < len(buno_history):
if buno_history[i][self.WUC][:6] == '275020' and
buno_history[i][self.REM_INS_FLAG] == "R":
right_PCA = buno_history[i][self.SERNO]
right_PCA_index = i
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 16, last modified 08/31/22 18:12:17
break
i += 1
i = 0
while left_PCA == None and i < len(buno_history):
```

```
if buno_history[i][self.WUC][:6] == '275021' and
buno_history[i][self.REM_INS_FLAG] == "R":
left_PCA = buno_history[i][self.SERNO]
left_PCA_index = i
break
i += 1
i = 0
# Check if there is an install for a different serno but the same WUC before
the first removed indecies
while right_PCA != None and i < right_PCA_index:
if buno_history[i][self.REM_INS_FLAG] == "I":
if buno_history[i][self.WUC][:6] == '275020' and
buno_history[i][self.SERNO] != right_PCA:
buno_history.pop(i)
i -= 1
i += 1
i = 0
# Check if there is an install for a different serno but the same WUC before
the first removed indecies
while left_PCA != None and i < left_PCA_index:
if buno_history[i][self.REM_INS_FLAG] == "I":
if buno_history[i][self.WUC][:6] == '275021' and
buno_history[i][self.SERNO] != left_PCA:
buno_history.pop(i)
i -= 1
i += 1
print("Buno ", buno, "deconflicted")
for entry in buno_history:
new_data.append(entry)
with open(self.save_decon_data_filename, 'w', newline="") as file:
header = self.header_extended
header.extend(['total_TMR_1_hrs', 'total_TMR_2_hrs', 'total_TMR_3_hrs',
'total_TMR_4_hrs', 'total_TMR_5_hrs', 'total_TMR_6_hrs',
'total_TMR_7_hrs',
'total_NON_AUST_lndgs', 'total_AUST_lndgs',
'total_SHIP_lndgs'])
C:\Users\willi\OneDrive\Desktop\Latest_9\SernoHistory.py
Page 17, last modified 08/31/22 18:12:17
writer = csv.writer(file)
writer.writerow(header)
writer.writerows(new_data)
except Exception:
print("what happened")
x = SernoHistory()
x.read_data()
x.format_date()
x.build_sernos()
x.auto_serno_query_steps()
x.deconflict_duplicates()
x.auto_serno_query_steps()
```

APPENDIX H. PYTHON SOURCE CODE FOR GENERATING SERIAL NUMBER REMOVAL DATA

```
C:\Users\willi\OneDrive\Desktop\Latest_9\RemovalHistory.py
Page 1, last modified 08/31/22 18:14:54
# RemovalHistory.py
# Captain William Frazier
# NPS Thesis research
import csv
from datetime import date, timedelta
import matplotlib.pyplot as plt
import numpy as np
from copy import deepcopy
class SernoRemovals(object):
def __init__(self):
self.data = []
self.sernos = {}
self.PCA_removals_filename = 'PCA_Data.csv'
self.new_filename = 'FINAL_REMOVALS_WUC_27502X.csv'
self.header = ''
# Indicies
self.ACTION_DATE = 0
self.PART_NO = 1
self.CAGE = 2
self.SERNO = 3
self.WUC = 4
self.WEC_DESC = 5
self.ORG = 6
self.ORG_DESC = 7
self.TEC = 8
self.TMS = 9
self.BUNO = 10
self.REM_INS_FLAG = 11
self.JCN = 12
self.MCN = 13
self.TYPE_MAINT = 14
self.TRANS_CODE = 15
self.ACT_TAKEN = 16
self.MAL_CODE = 17
self.COMP_DATE = 18
self.TIME_1_CODE = 19
self.TIME_1_CYCLE = 20
self.TIME_2_CODE = 21
self.TIME_2_CYCLE = 22
self.TIME_3_CODE = 23
self.TIME_3_CYCLE = 24
self.HRS = 25
C:\Users\willi\OneDrive\Desktop\Latest_9\RemovalHistory.py
Page 2, last modified 08/31/22 18:14:54
```

```
self.LNDGs = 26
self.TMR_1 = 27
self.TMR_2 = 28
self.TMR_3 = 29
self.TMR_4 = 30
self.TMR_5 = 31
self.TMR_6 = 32
self.TMR_7 = 33
self.NON_AUST_LNDGS = 34
self.AUST_LNDGS = 35
self.SHIP_LNDGS = 36
def read_data(self):
''' Reads in the DP-0025 report csv file generated from Deckplate. '''
with open(self.PCA_removals_filename, 'r') as file:
reader = csv.reader(file)
for row in reader:
self.data.append(row)
# save the header row and pop it off
self.header = self.data[0]
self.data.pop(0)
def format_date(self):
''' Converts maintenance action date string from .csv into datetime data type.
Assumes .csv date is in MM/DD/YYYY and needs to be [YYYY, MM, DD]'''
for row in range(0,len(self.data)):
# Action Taken Date
mdy = self.data[row][self.ACTION_DATE].split("/")
self.data[row][self.ACTION_DATE] = date(int(mdy[2]), int(mdy[0]), int(mdy[1]))
# Action Completion Date
mdy = self.data[row][self.COMP_DATE].split("/")
self.data[row][self.COMP_DATE] = date(int(mdy[2]), int(mdy[0]), int(mdy[1]))
def sort_by_date(self, history):
''' Sorts an input list of historical R/I records by their action date value. '''
history.sort(key=lambda x: x[self.ACTION_DATE])
return history
def display_records(self, records):
''' Prints the gathered R/I action records with proper formatting. '''
C:\Users\willi\OneDrive\Desktop\Latest_9\RemovalHistory.py
Page 3, last modified 08/31/22 18:14:54
max_lens = [len(str(max(i, key=lambda x: len(str(x))))) for i in zip(*records)]
print('\n'.join(' '.join(item[i].ljust(max_lens[i]) for i in range(len(max_lens)))
for item in records))
print()
def build_sernos(self):
''' Generates master dictionary of sernos:[R/I actions].
If the SerNo is not yet a key in the dict, it is added and its
value is the corresponding row of data i.e. the oldest action
completed from the data. Subsequent rows with that same SerNo will
have that data appended to the value list. '''
for row in range(0,len(self.data)):
key = str(self.data[row][self.SERNO])
if key not in self.sernos:
```

```
self.sernos[key] = [self.data[row]]
else:
self.sernos[key].extend([self.data[row]])
def SerNo_query_short(self, history_list):
''' Returns the R/I history of a SerNo '''
history_list.insert(0,self.header)
# Convert dates to strings for formating
for row in range(1,len(history_list)):
history_list[row][self.ACTION_DATE] =
history_list[row][self.ACTION_DATE].strftime('%m/%d/%Y')
history_list[row][self.COMP_DATE] =
history_list[row][self.COMP_DATE].strftime('%m/%d/%Y')
# only the essential columns of data
#indicies = [0,1,3,7,10,11,12,13,22,24]
indicies = [self.ACTION_DATE,
self.PART_NO,
self.SERNO,
self.ORG_DESC,
self.BUNO,
self.TRANS_CODE,
self.ACT_TAKEN,
self.MAL_CODE,
self.HRS,
self.LNDGs,
self.TMR_1,
self.TMR_2,
self.TMR_3,
C:\Users\willi\OneDrive\Desktop\Latest_9\RemovalHistory.py
Page 4, last modified 08/31/22 18:14:54
self.TMR_4,
self.TMR_5,
self.TMR_6,
self.TMR_7,
self.NON_AUST_LNDGS,
self.AUST_LNDGS,
self.SHIP_LNDGS]
short_list = []
for row in history_list:
short_list.append([row[index] for index in indicies])
# Print using formatting
self.display_records(short_list)
def get_history(self, serno):
''' Generates the list of R/I actions based on a serno/part combo and adds the
header. '''
history_list = deepcopy(self.sernos[serno])
history_list = self.sort_by_date(history_list)
return history_list
def remove_duplicates(self):
for serno in self.sernos.keys():
new_history = []
temp_history = deepcopy(self.sernos[serno])
```

```
for removal in temp_history:
if removal not in new_history:
new_history.append(removal)
self.sernos[serno] = new_history
def remove_admin_removals(self):
''' Removes any removal MAFs that are administrative in nature.
Determined by transaction, action taken, and malfunction codes.
Comments are from the NAMP 4790.2D Appendix E. '''
for serno in self.sernos.keys():
C:\Users\willi\OneDrive\Desktop\Latest_9\RemovalHistory.py
Page 5, last modified 08/31/22 18:14:54
temp_history = deepcopy(self.sernos[serno])
row = 0
while row < len(temp_history):
trans_code = temp_history[row][self.TRANS_CODE]
mal_code = temp_history[row][self.MAL_CODE]
act_code = temp_history[row][self.ACT_TAKEN]
if str(trans_code) == '11':
# Could not duplicate or found within tolerances
if act_code == "A":
temp_history.pop(row)
row -= 1
elif str(trans_code) == '16':
# No Defect Removal
temp_history.pop(row)
row -= 1
elif str(trans_code) == '17':
# Installation, should not have a removal serno
temp_history.pop(row)
row -= 1
elif str(trans_code) == '18':
# Cannibalization or removal for defect
pass
elif str(trans_code) == '20':
# Cannibalization
pass
elif str(trans_code) == '21':
# repairable component is removed
pass
elif str(trans_code) == '23':
# Removal and replacement of a defective, suspected defective, or scheduled
# maintenance of a repairable component from an end item
pass
elif str(trans_code) == '30':
# I Level test and check, component already removed
temp_history.pop(row)
row -= 1
C:\Users\willi\OneDrive\Desktop\Latest_9\RemovalHistory.py
Page 6, last modified 08/31/22 18:14:54
elif str(trans_code) == '31':
# I Level maintenance, component already removed
```

```
temp_history.pop(row)
row -= 1
elif str(trans_code) == '47':
# TDs not resulting in a removal
if act_code == "C":
temp_history.pop(row)
row -= 1
row += 1
row = 0
self.sernos[serno] = temp_history
def write_data(self):
with open(self.new_filename, 'w', newline='') as file:
writer = csv.writer(file)
writer.writerow(self.header)
for serno in self.sernos.keys():
writer.writerows(self.sernos[serno])
x = SernoRemovals()
x.read_data()
x.format_date()
x.build_sernos()
x.remove_duplicates()
x.remove_admin_removals()

x.write_data()
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX I. PYTHON SOURCE CODE FOR WEIBULL AND CPH MODELS

```
# Models.py
# Captain William Frazier
# NPS Thesis research
from copy import deepcopy
import csv
from datetime import date, timedelta
import scipy.stats as s
from scipy.special import gammaln
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import weibull_min, exponweib, weibull_max, norm, chi2, beta, linregress,
trim_mean
import math
from predictr import Analysis
import weibull
import reliability
import pandas as pd
from lifelines import KaplanMeierFitter
from lifelines import CoxPHFitter
from statistics import mean
import FlightHours
class Weibull(object):
def __init__(self):
# Indicies for data columns
self.TTF = 0
self.MODE = 1
self.MCN = 2
self.RAW_MCN = 0
self.RAW_SERNO = 1
self.RAW_PN= 2
# PCA Data from V22 FST Weibull Analysis
self.PCA_Data_filename = 'FINAL_REMOVALS_WUC_27502X.csv'
self.Weibull_data_filename = 'PCA_Weibull.csv'
self.PCA_Test_Data_filename = 'FINAL_REMOVALS_WUC_27502X.csv'
self.failures = []
self.num_failures = 0
self.data_header = ''
self.failures_by_mal = {}
self.modes_to_ignore = ['0', '813', '815']
self.sernos = {}
C:\Users\willi\OneDrive\Desktop\Thesis\Latest_Final\Weibull.py
Page 2, last modified 09/04/22 19:58:55
self.data_at_test_data = {}
# Indicies
self.ACTION_DATE = 0
self.PART_NO = 1
```

```
self.CAGE = 2
self.SERNO = 3
self.WUC = 4
self.WEC_DESC = 5
self.ORG = 6
self.ORG_DESC = 7
self.TEC = 8
self.TMS = 9
self.BUNO = 10
self.REM_INS_FLAG = 11
self.JCN = 12
self.MCN = 13
self.TYPE_MAINT = 14
self.TRANS_CODE = 15
self.ACT_TAKEN = 16
self.MAL_CODE = 17
self.COMP_DATE = 18
self.TIME_1_CODE = 19
self.TIME_1_CYCLE = 20
self.TIME_2_CODE = 21
self.TIME_2_CYCLE = 22
self.TIME_3_CODE = 23
self.TIME_3_CYCLE = 24
self.HRS = 25
self.LNDGs = 26
self.TMR_1 = 27
self.TMR_2 = 28
self.TMR_3 = 29
self.TMR_4 = 30
self.TMR_5 = 31
self.TMR_6 = 32
self.TMR_7 = 33
self.NON_AUST_LNDGS = 34
self.AUST_LNDGS = 35
self.SHIP_LNDGS = 36
self.CENS = 37
# Results tables
self.results_header = ['Failure Mode','# Failures', 'MTTF',
'Eta','Beta','Rsqr','AbPval']
self.results = [self.results_header]
self.final_results = [self.results_header]
C:\Users\willi\OneDrive\Desktop\Thesis\Latest_Final\Weibull.py
Page 3, last modified 09/04/22 19:58:55
def read_data(self):
''' Reads in the final data of TTF and Mode from a csv file. '''
with open(self.PCA_Data_filename, 'r') as file:
reader = csv.reader(file)
for row in reader:
self.failures.append(row)
self.data_header = self.failures[0]
self.failures.pop(0)
```

```
def write_data(self):
''' Connects the SerNo from FRC's Raw Data to the resultant data via MCN. '''
with open(self.Weibull_data_filename, 'w', newline='') as file:
writer = csv.writer(file)
writer.writerow(self.data_header)
writer.writerows(self.failures)
def remove_zeros(self):
''' Removes any TTFs == 0 which could cause NaN or Inf in weibull calc. '''
without_zeros = [row for row in self.failures if (float(row[self.HRS]) != float(0))]
self.failures = without_zeros
def ensure_floats(self):
''' Ensures TTF in data is in proper format. '''
for row in range(len(self.failures)):
self.failures[row][self.HRS] = float(self.failures[row][self.HRS])
def censor_flag(self):
''' Adds the censored boolean column for all TTF data for weibull module. '''
self.data_header.append('Censor Flag')
for row in range(len(self.failures)):
self.failures[row].append(False)
def sort_by_mal(self):
failures = deepcopy(self.failures)
for row in failures:
C:\Users\willi\OneDrive\Desktop\Thesis\Latest_Final\Weibull.py
Page 4, last modified 09/04/22 19:58:55
if row[self.MAL_CODE] not in self.failures_by_mal:
self.failures_by_mal[row[self.MAL_CODE]] = row
else:
self.failures_by_mal[row[self.MAL_CODE]].append(row)
def split_data_weibull(self, mal_code, num_failures, split_perc):
train_failures = []
train_suspensions = []
test_failures = []
num_train_failures = 0
num_test_failures = 0
num_train_suspensions = 0
count = 0
end_date = 0
date_flag = 0
for row in self.failures:
if count <= float(num_failures * split_perc):
if str(row[self.MAL_CODE]) == str(mal_code):
train_failures.append(row[self.HRS])
num_train_failures += 1
count += 1
else:
train_suspensions.append(row[self.HRS])
num_train_suspensions += 1
else:
if date_flag == 0:
end_date = row[self.ACTION_DATE]
date_flag = 1
```

```
if str(row[self.MAL_CODE]) == str(mal_code):
test_failures.append(row[self.HRS])
num_test_failures += 1
return train_failures, num_train_failures, train_suspensions, num_train_suspensions,
test_failures, num_test_failures, end_date
def split_data_CPH(self, mal_code, num_failures, split_perc):
train_failures = []
test_failures = []
num_train_failures = 0
C:\Users\willi\OneDrive\Desktop\Thesis\Latest_Final\Weibull.py
Page 5, last modified 09/04/22 19:58:55
num_test_failures = 0
num_train_suspensions = 0
count = 0
end_date = 0
date_flag = 0
for row in self.failures:
if count <= float(num_failures * split_perc):
if str(row[self.MAL_CODE]) == str(mal_code):
row[self.CENS] = 1
train_failures.append(row)
num_train_failures += 1
count += 1
else:
row[self.CENS] = 0
train_failures.append(row)
num_train_suspensions += 1
else:
if date_flag == 0:
end_date = row[self.ACTION_DATE]
date_flag = 1
if str(row[self.MAL_CODE]) == str(mal_code):
row[self.CENS] = 1
test_failures.append(row)
num_test_failures += 1
else:
row[self.CENS] = 0
test_failures.append(row)
num_train_suspensions += 1
return train_failures, num_train_failures, num_train_suspensions, test_failures,
num_test_failures, end_date
def reliability_RRY(self, train_failures, train_suspensions, m, conf_int, mode):
model = reliability.Fitters.Fit_Weibull_2P(failures=train_failures,
right_censored=train_suspensions, show_probability_plot=True, print_results=True,
CI=conf_int, quantiles=None, CI_type='time', method=m, optimizer=None,
force_beta=None, downsample_scatterplot=True)
MTTF = model.alpha * math.exp( gammaln( 1 + (1 / model.beta)) )
print('MTTF: ' + str(round(MTTF,2)) + '\n'
C:\Users\willi\OneDrive\Desktop\Thesis\Latest_Final\Weibull.py
Page 6, last modified 09/04/22 19:58:55
'Eta: ' + str(round(model.alpha,2)) + '\n'
```

```
'Beta: ' + str(round(model.beta,2)))
return round(float(MTTF),2), round(float(model.loglik),2),
round(float(model.AICc),2) #, model.loglik2, model.AICc, model.BIC, model.AD]
def CoxPropHazard(self, mal_code, train_failures, a, p):
test_failures = []
cph_data = []
for key in self.data_at_test_data.keys():
if str(self.data_at_test_data[key][self.MAL_CODE]) == str(mal_code):
test_failures.append(self.data_at_test_data[key])
df_train = pd.DataFrame({
'TTF': [row[self.HRS] for row in train_failures],
'TMR_1 Hours': [row[self.TMR_1] for row in train_failures],
'TMR_2 Hours': [row[self.TMR_2] for row in train_failures],
'TMR_3 Hours': [row[self.TMR_3] for row in train_failures],
'TMR_4 Hours': [row[self.TMR_4] for row in train_failures],
C:\Users\willi\OneDrive\Desktop\Thesis\Latest_Final\Weibull.py
Page 7, last modified 09/04/22 19:58:55
'TMR_5 Hours': [row[self.TMR_5] for row in train_failures],
'TMR_6 Hours': [row[self.TMR_6] for row in train_failures],
'TMR_7 Hours': [row[self.TMR_7] for row in train_failures],
'Non-Austere Landings': [row[self.NON_AUST_LNDGS] for row in train_failures],
'Austere Landings': [row[self.AUST_LNDGS] for row in train_failures],
'Ship Landings': [row[self.SHIP_LNDGS] for row in train_failures],
'Fail': [row[self.CENS] for row in train_failures]
})
df_test = pd.DataFrame({
'TTF': [row[self.HRS] for row in test_failures],
'TMR_1 Hours': [row[self.TMR_1] for row in test_failures],
'TMR_2 Hours': [row[self.TMR_2] for row in test_failures],
'TMR_3 Hours': [row[self.TMR_3] for row in test_failures],
'TMR_4 Hours': [row[self.TMR_4] for row in test_failures],
'TMR_5 Hours': [row[self.TMR_5] for row in test_failures],
'TMR_6 Hours': [row[self.TMR_6] for row in test_failures],
'TMR_7 Hours': [row[self.TMR_7] for row in test_failures],
'Non-Austere Landings': [row[self.NON_AUST_LNDGS] for row in test_failures],
'Austere Landings': [row[self.AUST_LNDGS] for row in test_failures],
'Ship Landings': [row[self.SHIP_LNDGS] for row in test_failures],
})
print(df_test)
cph = CoxPHFitter(alpha=a, penalizer=p)
cph.fit(df_train, duration_col="TTF", event_col="Fail")
cph.print_summary()
plt.figure()
if mal_code == ' ':
mal_code = 'Blank (Failed Technical Directive Inspection)'
plt.title('Malfunction Code ' + str(mal_code) + ' Coefficient Ranges')
cph.plot(hazard_ratios=True)
plt.show()
cph.check_assumptions(df_train,p_value_threshold=0.05,show_plots=True)
plt.figure()
test_rows = df_test.iloc[:,1:11]
```

```
print(test_rows)
cph.predict_survival_function(test_rows).plot()
plt.title('Malfunction Code ' + str(mal_code) + ' Survival Function for Test Data
Failures')
plt.ylabel('Survival (1 - Unreliability)')
plt.xlabel('Flight Hours')
plt.legend(labels=df_test['TTF'], loc = 'lower left')
plt.show()
cph.predict_median(test_rows)
C:\Users\willi\OneDrive\Desktop\Thesis\Latest_Final\Weibull.py
Page 8, last modified 09/04/22 19:58:55
cph.predict_expectation(test_rows)
return round(float(cph.log_likelihood_),2), round(float(cph.AIC_partial_),2)
def v22_fst_weibull(self):
TTF = 0
MODE = 1
MCN = 2
diff = []
v22_fst_data = []
with open('v22_fst_results_PCA.csv', 'r') as file:
reader = csv.reader(file)
for row in reader:
v22_fst_data.append(row)
my_data = deepcopy(self.failures)
for entry in range(len(v22_fst_data)):
for entry_2 in my_data:
if v22_fst_data[entry][MCN] == entry_2[self.MCN]:
v22_fst_data[entry].extend(entry_2[self.HRS:self.CENS+1])
failure_modes = []
for entry in v22_fst_data:
if entry[MODE] not in failure_modes:
failure_modes.append(entry[MODE])
for mode in failure_modes:
train_failures = []
train_suspensions = []
for row in v22_fst_data:
if len(row) > 3:
if str(row[MODE]) == str(mode):
train_failures.append(row[3])
diff.append(float(row[0]) - float(row[3]))
else:
train_suspensions.append(row[3])
print('Failure Mode: ' + str(mode) + ' reliability - RRY ' +
str(len(train_failures)) + ' failures, ' + str(len(train_suspensions)) + ' right
censored\n')
w_mttf, w_llh, w_aic = self.reliability_RRY(train_failures, train_suspensions,
'RRY', 0.95, mode)
C:\Users\willi\OneDrive\Desktop\Thesis\Latest_Final\Weibull.py
Page 9, last modified 09/04/22 19:58:55
print(len(diff))
print(mean(diff))
```

```
def create_serno_removal_history(self):
with open(self.PCA_Test_Data_filename, 'r') as file:
reader = csv.reader(file)
for row in reader:
if row[self.SERNO] not in self.sernos.keys():
self.sernos[row[self.SERNO]] = [row]
else:
self.sernos[row[self.SERNO]].append(row)
def build_test_flight_data(self, test_data, end_date):
for failure in test_data:
serno = failure[self.SERNO]
if serno not in self.data_at_test_data.keys():
stopper = 0
total_hrs = 0
total_lndgs = 0
total_TMR_1_hrs = 0
total_TMR_2_hrs = 0
total_TMR_3_hrs = 0
total_TMR_4_hrs = 0
total_TMR_5_hrs = 0
total_TMR_6_hrs = 0
total_TMR_7_hrs = 0
total_NON_AUST_lndgs = 0
total_AUST_lndgs = 0
total_SHIP_lndgs = 0
data = [str(total_hrs), str(total_lndgs), str(total_TMR_1_hrs),
str(total_TMR_2_hrs),
str(total_TMR_3_hrs), str(total_TMR_4_hrs), str(total_TMR_5_hrs),
str(total_TMR_6_hrs),
str(total_TMR_7_hrs), str(total_NON_AUST_lndgs),
str(total_AUST_lndgs), str(total_SHIP_lndgs)]
entry = 0
while entry < len(self.sernos[serno]):
if stopper == 0:
C:\Users\willi\OneDrive\Desktop\Thesis\Latest_Final\Weibull.py
Page 10, last modified 09/04/22 19:58:55
if entry == 0:
self.sernos[serno][entry][self.HRS:self.CENS] = data
elif self.sernos[serno][entry][self.REM_INS_FLAG] == "I":
self.sernos[serno][entry][self.HRS:self.CENS] = data
else:
install = self.sernos[serno][entry-1][self.ACTION_DATE]
removal = self.sernos[serno][entry][self.ACTION_DATE]
mdy = install.split("-")
install = date(int(mdy[0]), int(mdy[1]), int(mdy[2]))
mdy = removal.split("-")
removal = date(int(mdy[0]), int(mdy[1]), int(mdy[2]))
mdy = str(end_date).split("-")
end_date = date(int(mdy[0]), int(mdy[1]), int(mdy[2]))
if removal > end_date:
removal = end_date
```

```
stopper = 1
buno = self.sernos[serno][entry][self.BUNO]
[flthrs, lndgs, TMR_1_hrs, TMR_2_hrs, TMR_3_hrs,
TMR_4_hrs, TMR_5_hrs, TMR_6_hrs, TMR_7_hrs,
NON_AUST_lndgs, AUST_lndgs, SHIP_lndgs] =
self.FlightHours.buno_query_all_data(buno, install, removal)
total_hrs += flthrs
total_lndgs += lndgs
total_TMR_1_hrs += TMR_1_hrs
total_TMR_2_hrs += TMR_2_hrs
total_TMR_3_hrs += TMR_3_hrs
total_TMR_4_hrs += TMR_4_hrs
total_TMR_5_hrs += TMR_5_hrs
total_TMR_6_hrs += TMR_6_hrs
total_TMR_7_hrs += TMR_7_hrs
total_NON_AUST_lndgs += NON_AUST_lndgs
total_AUST_lndgs += AUST_lndgs
total_SHIP_lndgs += SHIP_lndgs
data = [str(total_hrs), str(total_lndgs), str(total_TMR_1_hrs),
str(total_TMR_2_hrs),
str(total_TMR_3_hrs), str(total_TMR_4_hrs),
C:\Users\willi\OneDrive\Desktop\Thesis\Latest_Final\Weibull.py
Page 11, last modified 09/04/22 19:58:55
str(total_TMR_5_hrs), str(total_TMR_6_hrs),
str(total_TMR_7_hrs), str(total_NON_AUST_lndgs),
str(total_AUST_lndgs), str(total_SHIP_lndgs)]
self.sernos[serno][entry][self.HRS:self.CENS] = data
print(self.sernos[serno][entry])
if stopper == 1:
self.data_at_test_data[serno] = self.sernos[serno][entry]
entry += 1
entry += 1
def build_test_flight_data_new(self, test_data):
for failure in test_data:
serno = failure[self.SERNO]
serno_history = deepcopy(self.sernos[serno])
self.data_at_test_data[serno] = serno_history[-1]
# Main
x = Weibull()
x.read_data()
x.remove_zeros()
x.ensure_floats()
x.censor_flag()
x.sort_by_mal()
x.write_data()
x.create_serno_removal_history()
#x.v22_fst_weibull()
# Iterate through each component
split_perc = .80
results = [['mal_code', 'method', 'CI', 'MTTF', 'Log Likelihood', 'AIC']]
for mal in [150]:
```

```
num_failures = 0
# Check that the total number of failures is over 20
for row in x.failures:
if str(row[x.MAL_CODE]) == str(mal):
num_failures += 1
C:\Users\willi\OneDrive\Desktop\Thesis\Latest_Final\Weibull.py
Page 12, last modified 09/04/22 19:58:55
if num_failures >= 20 and str(mal) not in x.modes_to_ignore:
# Split the data into train and test data
train_failures, num_train_failures, num_train_suspensions, test_data,
num_test_failures, end_date = x.split_data_CPH(mal, num_failures, split_perc)
# based on the date the split occured, call RemovalHistory to get the flight data
for the test serial numbers at the time of split
x.build_test_flight_data_new(test_data)
alphas = [0.05]
pens = [0]
for alpha in alphas:
for pen in pens:
#cph_llh, cph_aic = x.CoxPropHazard(mal, train_failures, test_failures,
1-(alpha/100), 1-(pen/100))
cph_llh, cph_aic = x.CoxPropHazard(mal, train_failures, alpha, pen)
if mal == ' ':
mal = 'Blank (TD)'
results.append([str(mal), str(alpha), str(pen), str(cph_llh), str(cph_aic)])
print()
print()
with open('CPH.csv', 'w', newline='') as file:
writer = csv.writer(file)
writer.writerows(results)
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Aviation Pros. (2009). *GE, Boeing implement condition-based maintenance standard.* Retrieved August 22, 2022, from https://www.aviationpros.com/home/press-release/10398932/ge-and-boeing-implement-open-system-architecture-for-conditionbased-maintenance

Burger, M., Jaworowski, C., & Meseroll, R. (2011). V-22 aircraft flight data mining. *IEEE AUTOTESTCON, 2011,* 443–447. https://doi.org/10.1109/AUTEST.2011.6058773

Chen, C., Liu, Y., Sun, X., Cairano-Gilfedder, C. D., & Titmus, S. (2021). An integrated deep learning-based approach for automobile maintenance prediction with GIS data. *2021 Reliability Engineering & System Safety, 216*(C)*,* 107919. https://doi.org/10.1016/j.ress.2021.107919

Chi, Street, W. N., & Wolberg, W. H. (2007). Application of artificial neural network-based survival analysis on two breast cancer datasets. *AMIA Annual Symposium Proceedings, 2007,* 130–134.

Commander, Naval Air Forces. (2021). *Periodic Maintenance Information Cards Inspection Requirements Manual Navy Model CMV-22B and MV-22B Aircraft and Air Force Model CV-22B Aircraft.* (Technical Manual A1-V22AB-MRC-00).

Commander, Naval Air Forces. (2022). *MV-22B Aircraft Readiness Dashboard*. NAVAIR Vector.

Cox, D. R. (1972). Regression Models and Life-Tables. *Journal of the Royal Statistical Society. Series B (Methodological), 34*(2), 187–220. http://www.jstor.org/stable/2985181

Davidson-Pilon. (2019). Lifelines: Survival Analysis in Python. *Journal of Open Source Software, 4*(40), 1317. https://doi.org/10.21105/joss.01317

Department of Defense. (2011). *Reliability Centered Maintenance.* (DOD Manual 4151.22-M). Washington, DC: Department of Defense.

Department of Defense. (2020a). *Condition Based Maintenance Plus (CBM+) Order.* (MCO 4151.22). Washington, DC: Headquarters United States Marine Corps.

Department of Defense. (2020b). *Joint Artificial Intelligence Center (JAIC) Holistic Aircraft Component Health Predictor (HAC-HP).* Retrieved July 16, 2021, from https://www.govconwire.com/2020/05/DOD-ai-center-requests-info-on-predictive-maintenance-tech-for-h-60-helicopters/

Department of the Navy. (2016). *NATOPS General Flight and Operating Instructions Manual*. (CNAF M-3710.7). San Diego, CA: Commander, Naval Air Forces.

Department of the Navy. (2018) *Marine Corps Aviation Current Readiness Program.* (MCO 3710.7). Washington, DC: Headquarters United States Marine Corps.

Department of the Navy. (2021). *The Naval Aviation Maintenance Program.* (COMNAVAIRFORINST 4790.2D CH-1). Washington, DC: Headquarters, Department of the Navy.

Dong, & Nassif, A. B. (2019). Combining Modified Weibull Distribution Models for Power System Reliability Forecast. *IEEE Transactions on Power Systems, 34*(2), 1610–1619. https://doi.org/10.1109/TPWRS.2018.2877743

Eckstein, Megan. (2017). *Marine Aviation Going After Small Maintenance Issues that Create Big Readiness Problems.* Retrieved February 14, 2022, from https://news.usni.org/2017/02/08/marines-going-after-small-maintenance-issues-that-result-big-readiness-problems

Crusher, M. (2020). *Weapon system sustainment: Aircraft mission capable rates generally did not meet goals and cost of sustaining selected weapons systems varied widely, GAO-21-101SP.* Government Accountability Office. https://www.gao.gov/assets/720/710794.PDF

Graves, A. & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM networks. *IEEE International Joint Conference on Neural Networks, 2005*(4), 2047-2052. doi: 10.1109/IJCNN.2005.1556215

Konishi, Kitagawa, G., & Kitagawa, G. (2008). I*nformation criteria and statistical modeling.* Springer. https://doi.org/10.1007/978-0-387-71887-3

Korvesis, P. (2017). *Machine Learning for Predictive Maintenance in Aviation.* Université Paris-Saclay. Retrieved October 21, 2021, from https://pastel.archives-ouvertes.fr/tel-02003508

Lancaster, R., Talbert, M., & Kirk, R. (2014). Drowning in Data, Starving for Information*. United States Naval Institute Proceedings, 140*(2), 78–79.

Li, X., Ding, Q., & Sun, J.-Q. (2018). Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering & System Safety, 172*, 1–11. https://doi.org/10.1016/j.ress.2017.11.021

Mathew, V., Toby, T., Singh, V., Rao, B. M., & Kumar, M. G. (2017). Prediction of Remaining Useful Lifetime (RUL) of turbofan engine using machine learning. *2017 IEEE International Conference on Circuits and Systems (ICCS),* 306–311. https://doi.org/10.1109/ICCS1.2017.8326010

McCool, J. I. (2012). *Using the Weibull distribution: Reliability, modeling, and inference.* John Wiley & Sons, Incorporated.

O'Rourke, Ronald O. (2009). V-22 Osprey Tilt-Rotor Aircraft: Background and Issues for Congress. *Congressional Research Service.* Retrieved February 2, 2022, from https://www.everycrsreport.com/files/20091020_RL31384_71437ff23c4e6a5fa02cf011809cf5f4beca0745.PDF

Pampuri, Schirru, A., De Luca, C., & De Nicolao, G. (2011). Proportional hazard model with ℓ1 Penalization applied to Predictive Maintenance in semiconductor manufacturing. *2011 IEEE International Conference on Automation Science and Engineering,* 250–255. https://doi.org/10.1109/CASE.2011.6042436

Quanterion Solutions (2015). *System Reliability Toolkit - V: New Approaches and Practical Applications.* Utica, NY: Quanterion Solutions Inc.

Reid, M. (2022). *Reliability – a Python library for reliability engineering (Version 0.8.2)* [Computer software]. Zenodo. https://doi.org/10.5281/ZENODO.3938000

ReliaSoft Corporation. (2007). *Reliabiliy HotWire Tooltips.* Retrieved June 22, 2022, from https://www.weibull.com/hotwire/issue80/tooltips80.htm

Rinne, H. (2008). *The Weibull Distribution: A Handbook.* Chapman and Hall/CRC. https://doi.org/10.1201/9781420087444

Saxena, A., & Goebel, K. (2008). *Turbofan Engine Degradation Simulation Data Set. NASA Prognostics Data Repository.* [Data set]. NASA Ames Research Center, Moffett Field, CA. https://data.nasa.gov/Aerospace/CMAPSS-Jet-Engine-Simulated-Data/ff5v-kuh6

Spooner, C., Sowmya, A., Sachdev, P., Kochan, N., Trollor, J., & Brodaty, H. (2020). A comparison of machine learning methods for survival analysis of high-dimensional clinical data for dementia prediction. *Scientific Reports, 10*(1), 20410–20410. https://doi.org/10.1038/s41598-020-77220-w

Susto, S., Pampuri, A., McLoone, S., & Beghi, A. (2015). Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics, 11*(3), 812–820. https://doi.org/10.1109/TII.2014.2349359

Tai, B. C., & Machin, D. (2014). *Regression methods for medical research.* John Wiley & Sons, Incorporated.

Teradata. (2016). *Data Dominance for the DOD Analytics for Data-Driven Decision Making.* Retrieved August 13, 2021, from https://assets.teradata.com/resourceCenter/downloads/Brochures/EB9332.PDF

Tilman, N. (2020). *Prediction models for survival data with machine learning: An application to soft tissue sarcoma cohort*. Leiden University Medical Center.

United States and D. H. Berger. (2019). *Commandant's Planning Guidance.* U. S. Marine Corps, Quantico, VA.

Waghmode, & Patil, R. B. (2016). Reliability analysis and life cycle cost optimization: a case study from Indian industry. *The International Journal of Quality & Reliability Management, 33*(3), 414–429. https://doi.org/10.1108/IJQRM-11-2014-0184

Wen, L., Dong, Y., Gao, L. (2019). A new ensemble residual convolutional neural network for remaining useful life estimation. *Mathematical Biosciences and Engineering, 16*(2), 862–880. https://doi.org/10.3934/mbe.2019040

Wilkins, D. (2002). *The Bathtub Curve and Product Failure Behavior.* ReliaSoft Corporation. Retrieved May 20, 2022, from https://www.maths.tcd.ie/~donmoore/project/project/Write%20up/22%20mar%202006/hottopics21.htm

Wilson, B., Riposo, J., Goughnour, T., Burns, R., Vermeer, M., Kochhar, A., Bohman, A., & Eisman, M. (2020). *Naval Aviation Maintenance System: Analysis of Alternatives.* RAND Corporation: Santa Monica, CA.

Zhang, Y., Hutchinson, P., Lieven, N., & Nunez-Yanez, J. (2020). Remaining Useful Life Estimation Using Long Short-Term Memory Neural Networks and Deep Fusion. *IEEE Access, 8,* 19033–19045. https://doi.org/10.1109/ACCESS.2020.2966827

Zhou, Xi, L., & Lee, J. (2007). Reliability-centered predictive maintenance scheduling for a continuously monitored system subject to degradation. *Reliability Engineering & System Safety, 92*(4), 530–534. https://doi.org/10.1016/j.ress.2006.01.006

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California