**Artificial Intelligence in Transportation**

Naga Jyothi Madamanchi

***Abstract—*** The purpose of this project is to recreate the airline industry’s Aircraft Condition Monitoring System (ACMS) through the creation of neural networks.

***Index Terms—*** ACMS,Fuzzy Logic, Sensor Types, Stochastic Uncertainty, Fuzzy Rules, Control System, Neural Network, Crisp, Deep Learning, Episode, SGD, MAPE, Prediction, Training, Partial Derivative, LSTM, Logistic Regression

**I. Introduction**

Since the early 1970s, the airline industry has relied on artificial Intelligence to aid in the safety and efficiency of the aircraft. From Air Traffic Control Automation to pilot assistance, automation algorithms have greatly improved the efficiency of the workers of the industry [2].

Improvements in computing capabilities in recent years have allowed for the development of more sophisticated autopilot algorithms, and the creation of user experience oriented programs [3]. With mechanical failure as an ever present danger, software engineers collaborated with maintenance workers to develop the Aircraft Condition Monitoring System (ACMS).

**II. ACMS Control System**

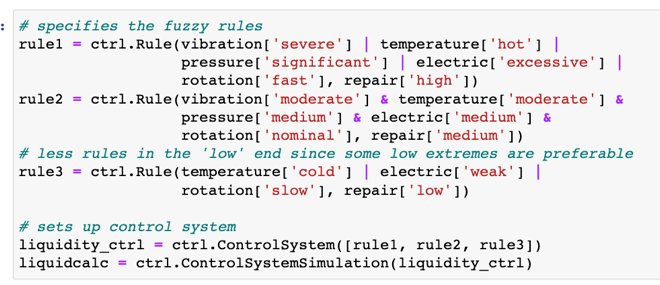
The ACMS aircraft health monitoring tool consists of a multitude of sensor type inputs and data processing algorithms to determine the time of maintenance and predict the probability of failure before flight. The program compares the real-time numerical input of the sensors to the healthy baseline values [4]. The ACMS system utilizes the numerical stochastic uncertainty logic to determine the probability of failure [1].

The previous iteration of this project contained a Jupyter-notebook python file, titled “AircraftHealth.ipynb”. This file contained 5 out of the dozens of the real-life aircraft maintenance variables [9].

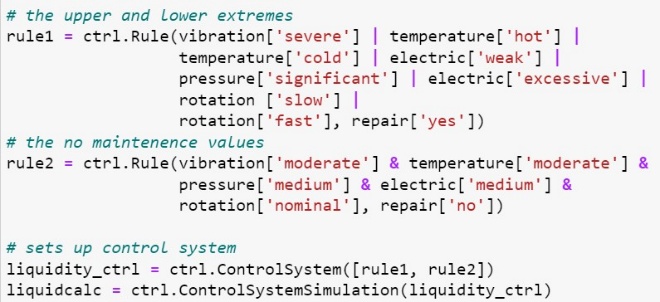
* Vibration
* Temperature
* Pressure
* Electric current
* Rotational speed

A set of fuzzy rules were applied to each of the 5 variables. These rules defined what constitutes a low, medium, and high extreme of values. The three fuzzy rules become the input of the control system which calculates the output value.

Old Three Rule Code:

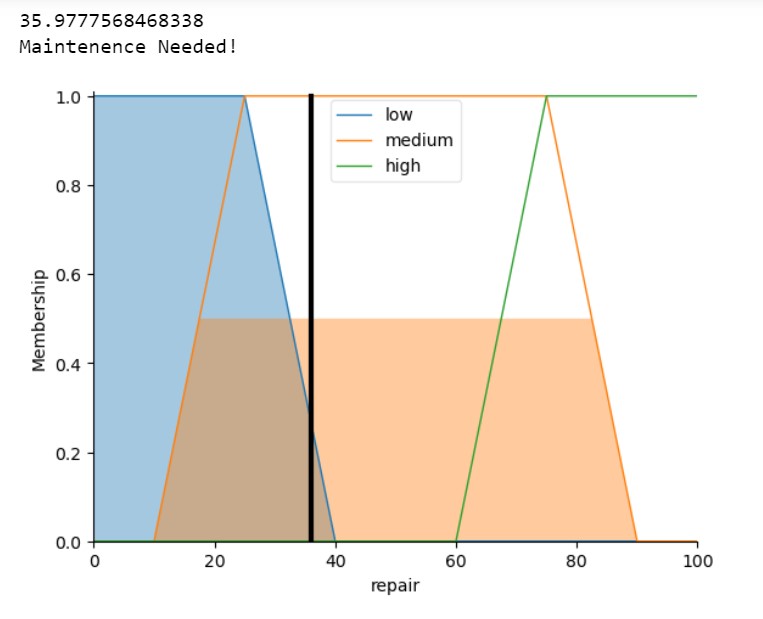


New Two Rule Code:

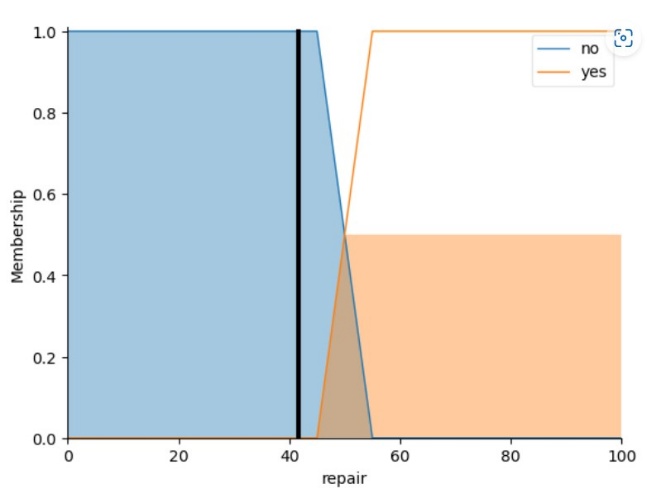


The original iteration had a set of three rules that corresponded to: low, medium, and high values. However, the new iteration was revised to utilize a two rule system that corresponds a simple: “yes” or “no” maintenance needed value. The room for uncertainty is also significantly smaller in the new two rule control system when compared to the old three rule system [5].

Three Rule Control System:



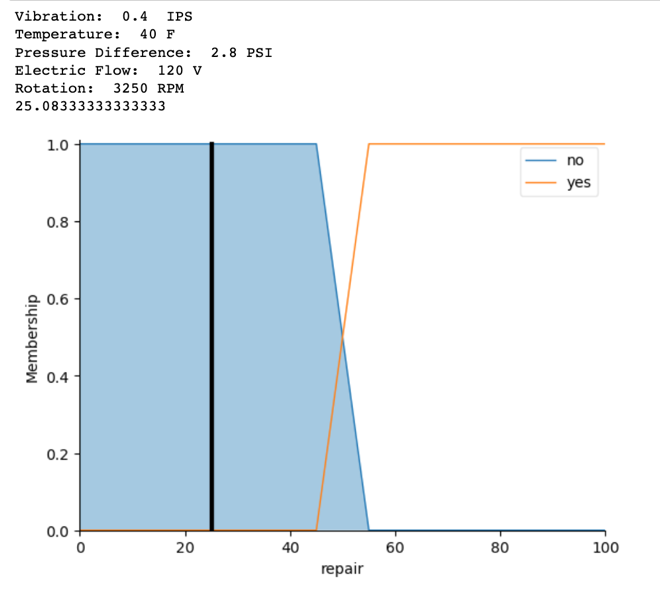
Two Rule Control System:



The Stochastic Gradient Descent (SGD) loss formula was used to generate the graphs. One of the primary reasons the two rule control system was placed was to rectify a bug that I discovered in the old system. For example, if one variable “vibration” had a high value extreme, and another variable “rotation” had a low extreme value, the two extremes canceled out and produced an average value that fell into the middle category.

Such an error in logic allowed some mechanical faults to go unnoticed. I had to eliminate the medium set of rules to eliminate the possibility of extremes canceling one another out to produce a control graph similar to the ideal baseline graph.

Ideal Baseline Graph:

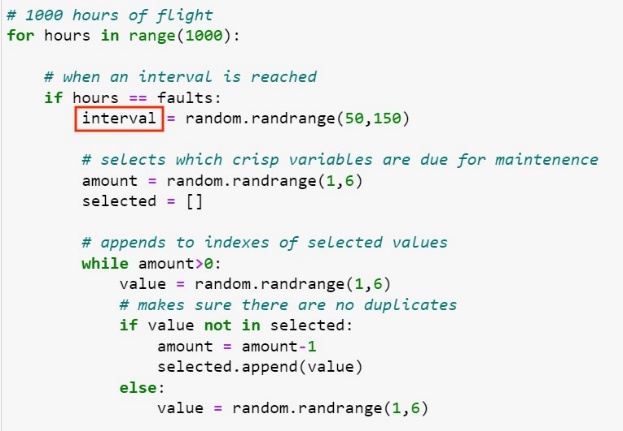


**III. Neural Network Setup**

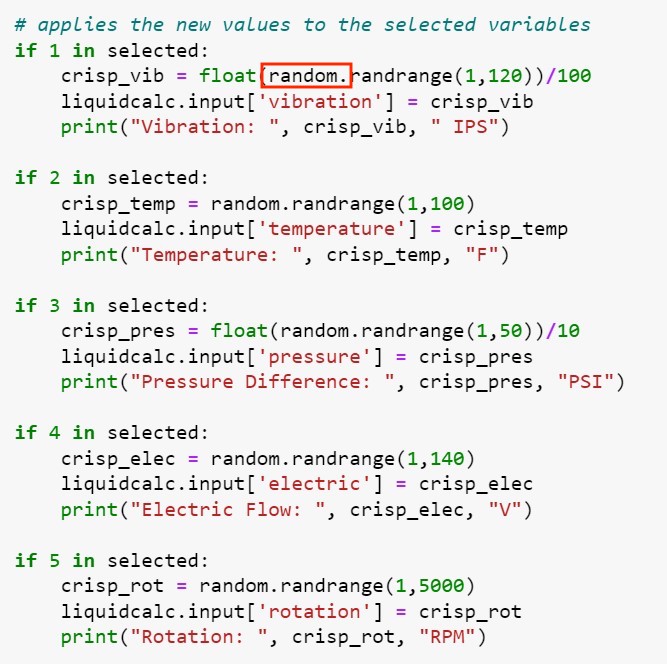
The neural network of the previous iteration was practically non-existent. While the old system did utilize fuzzy logic to calculate a crisp score, what it lacked was a deep learning algorithm [7]. To build the network, first I modified the generation of the crisp values. Instead of fixed static values, the 5 variables: vibration, temperature, pressure, electric current, and rotation speed, were now assigned random values.

The random values of the maintenance variables are assigned in a for loop that represents 1000 hours of flight. Aircraft usually require maintenance every 100 hours or so. This is the reason I also assigned a random “interval” value to represent the time the aircraft required repairs [6].

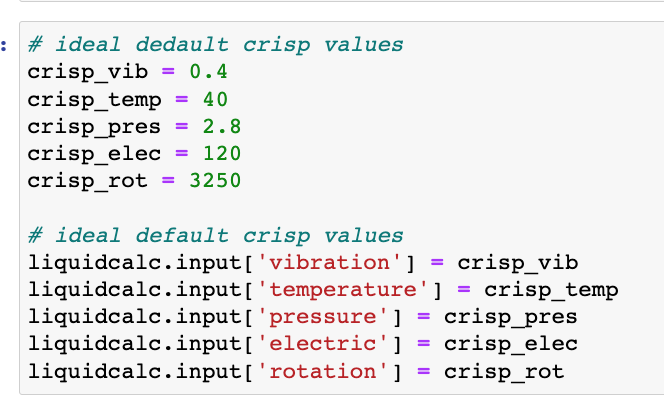
Maintenance Interval:



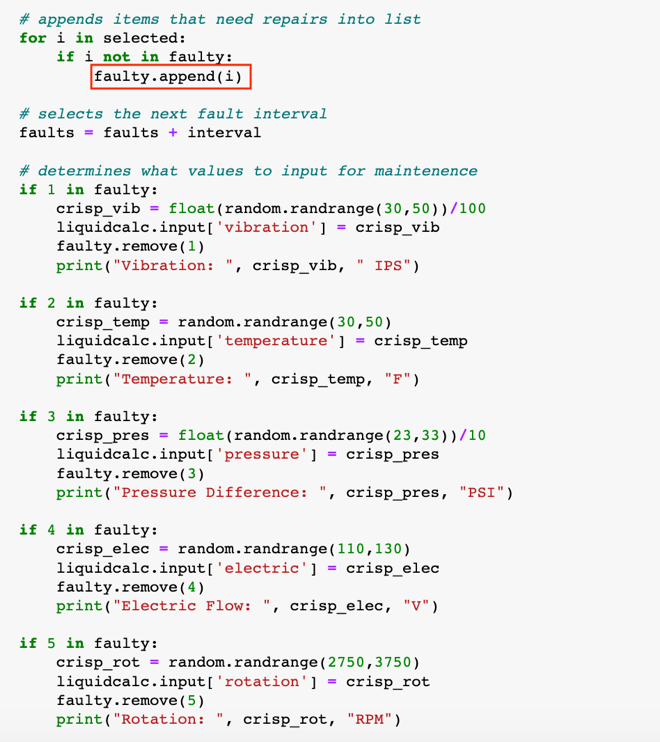
Random Value Generator:



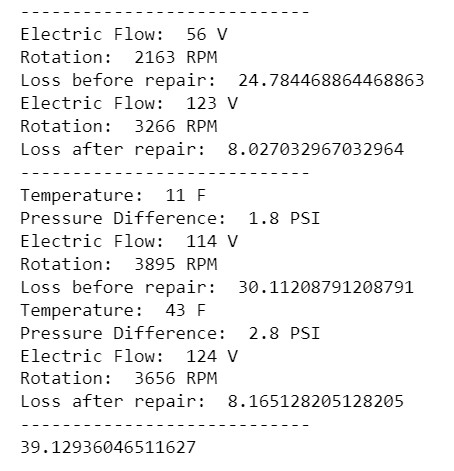
Each episode in the 1000-hour loop represents the time a randomly selected set of components of the aircraft is given a fault value that falls outside of the ideal baseline values.



Every component that experiences failure is added to a list titled “faulty” that stores the variables that require repairs. The values on that list are passes through a new set of “if” statements that assign a new random values that fall within the acceptable maintenance parameters.

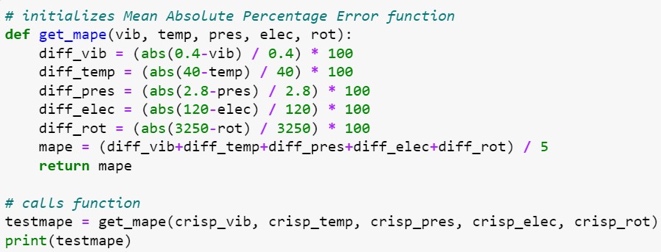


Each run through of an episode prints out what components were repaired, the values of those components, the loss before the repairs, and the loss after the repairs.



The print statements in the image in the previous page use a new, more accurate loss formula as opposed to the old SGD formula. The new formula, the Mean Absolute Percentage Error (MAPE), measures the deviation from the baseline. The numbers inside the absolute value “abs()” functions represent the ideal aircraft variable values.

MAPE Loss Function:



To account for the variations in the sizes of the variables, I divided the absolute value of the differences by the size of the ideal values. I then multiplied the resulting value by 100 to get a percentage value. Once all the percent values are added up and divided by the number of variables, the result should be a number between 0 to 100 that represents the loss [10].

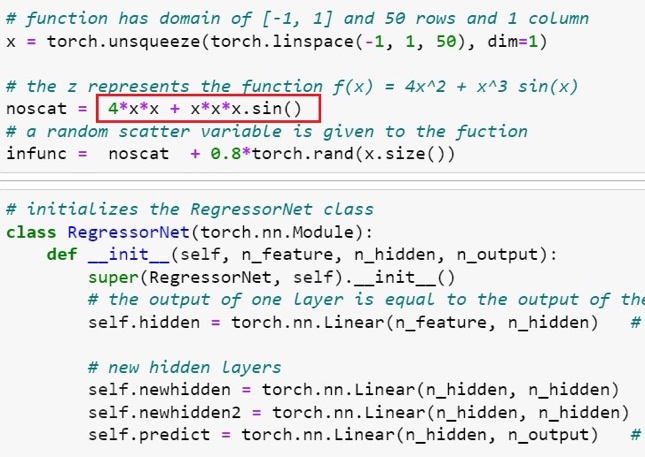
The neural network was still far from complete however. The loss only calculated the deviation from the baseline immediately after the faults were declared in the loop. The network still required a prediction and training system [8].

**IV. Prediction and Training**

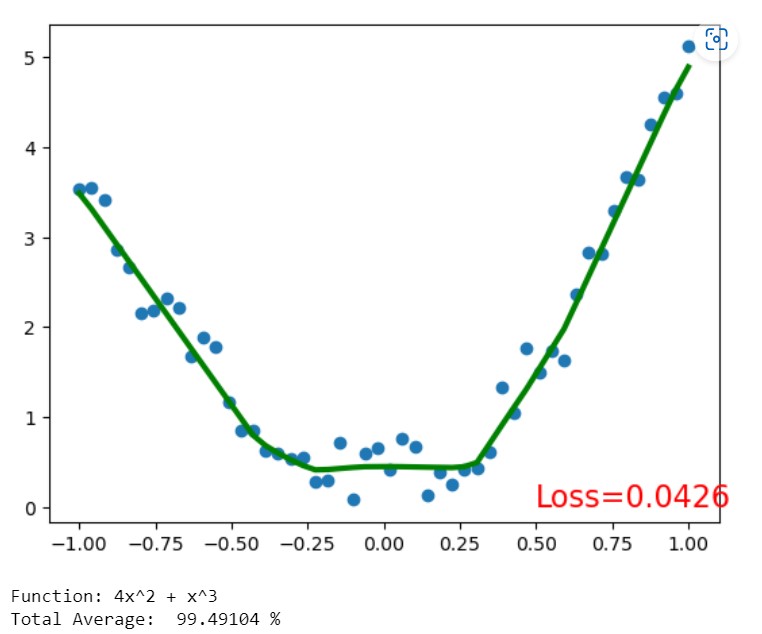
During my first attempt to create a prediction and training system, I referenced Assignment 5. I copied and then modified the code to suit the requirements of the “AircraftHealth.ipynb” file. The “infunc” formula was altered to represent a flat “x\*0” equation.

Assignment 5

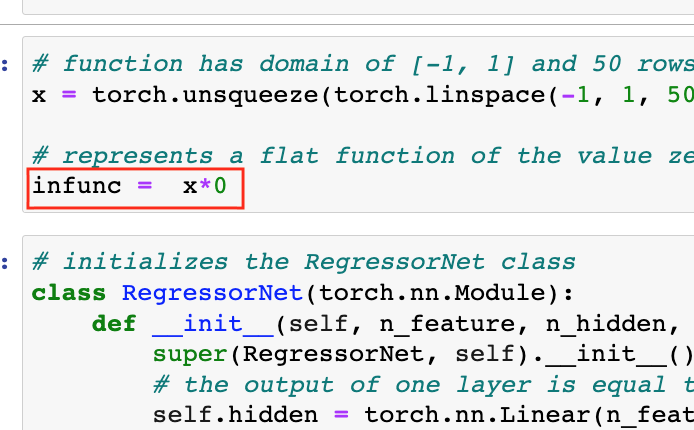
4x^2 + x^3 + sin() Function [5]:



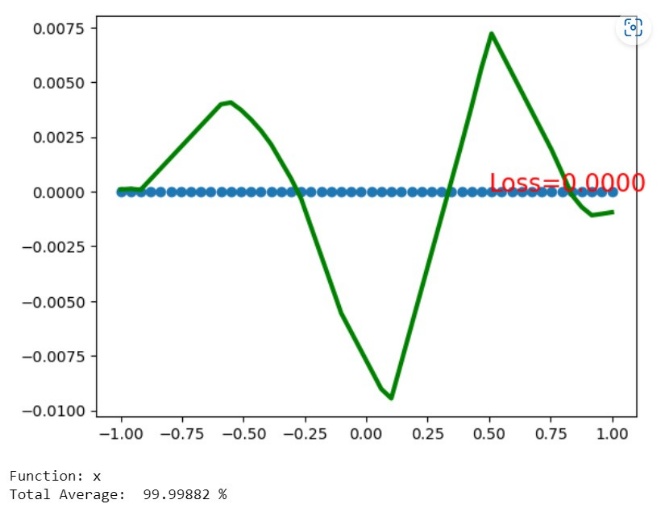
Assignment 5 Graph [5]:



Aircraft Health x\*0 Function:



Aircraft Health Graph:

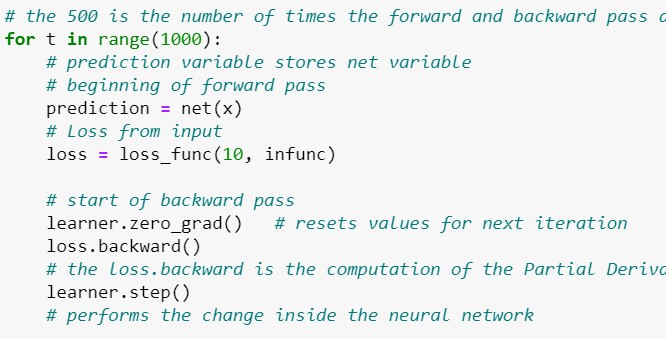


I selected the “X\*0” equation to represent the loss instead of “X\*0 + 25” because any attempt to utilize a flat non-zero formula led to an error. The equation above was meant to represent the medium partial derivative value of 25.08333 that corresponded to the baseline values. I intended to offset the zero equation with the 25.08333 value to calculate the partial derivate of the other values.



The for loop in Assignment 5 passes the equation through the “loss\_func()” function to obtain the loss. The loop uses the “net(x)” equation to obtain the prediction the “loss.backward()” line obtains the partial derivative. The “learner.step()” line performs the change for the next step in the learning loop.

Assignment 5 For loop:



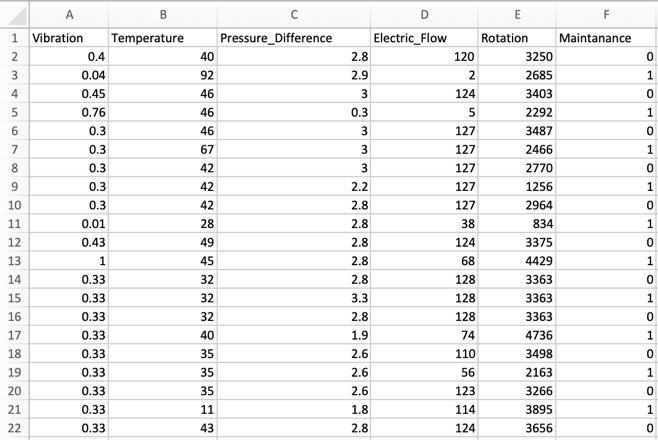
The loop above can perform the prediction for one function [9]. However, the ACMS program requires a prediction that uses 5 variables. The partial derivative only takes into account the final value of the equation, not the individual variables that lead to the equation. To obtain a more accurate prediction, I had to create a new program.

**V. LSTM Prediction**

The term Long Short-Term Memory (LSTM), describes a prediction algorithm that is ideal for predictions based on a dataset where time is a factor. Although the Maintenance holds values of 0 and 1, the original program that was used to create the data did run on a 1000-hour timeframe.

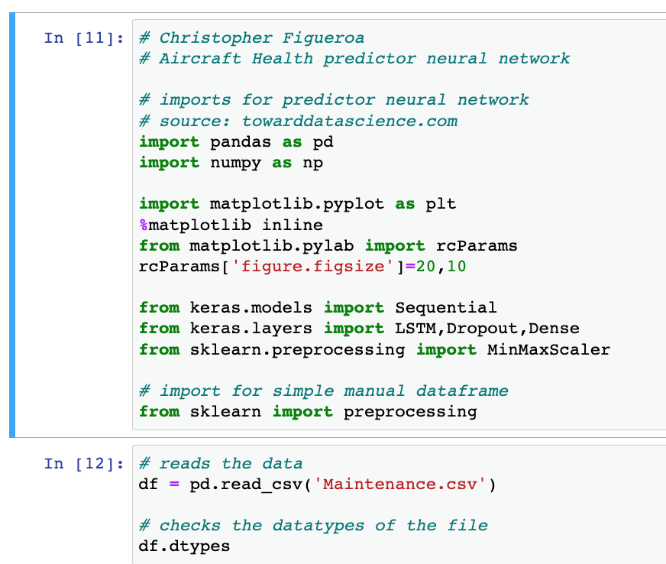
Before I created the new program, I first had to create a .csv file that contained an example of a maintenance schedule. The “Maintenance.csv” file contained a maintenance schedule that corresponded with an example output of the “AircraftHealth.ipynb” program.

AircraftHealth Output Table:



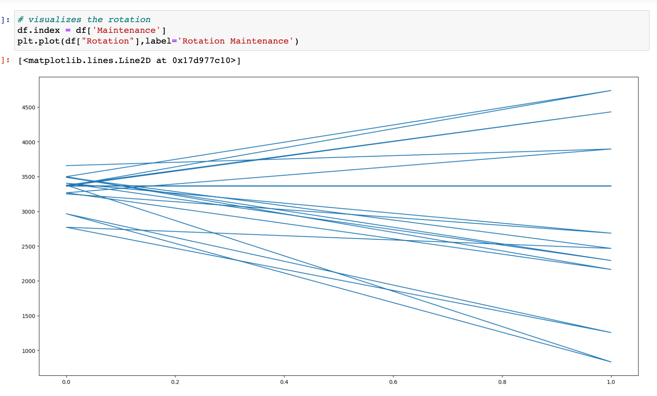
The 0’s in the “Maintenance” column represent instances when repairs were not necessary. The 1’s represent the need for at least one repair. With that information available, I proceeded to create the “AircraftPredictor.ipynb” file.

My next step was to install the “pandas” and the “keras” libraries on to my Anaconda Python environment. Those two libraries were necessary for the prediction algorithms. The “matplotlib” library was needed to display the graphs.



I used the data from the .csv file to create a data frame. I then utilized the information stored in the data frame to generate 5 graphs, each representing one of the 5 maintenance variables. The graph below represents the variable “Rotation”.

Rotation Prediction Graph [11]:



One aspect of the graph that is immediately apparent is the contrast in the distribution of the values that correspond with 0 and the values that correspond with 1. The 0 values, the ones that do not require maintenance, all sit within a narrow range of values that do not deviate too far from those of the ideal baseline of 3250 rpm.

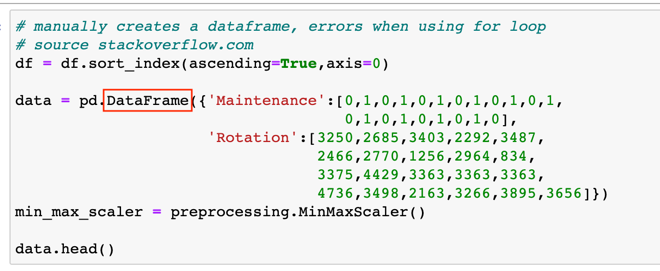
The graph provides insightful view of the overall structure of the ACMS’s binary “yes” or “no” system. The fact that the values for Maintenance only equal 0 or 1 suggests that perhaps the algorithm used in “towardsdatascience.com” perhaps does not suit the maintenance data. For contrast, the graph below represents the “Close price history” with a structure that is easier for the program to analyze [11].

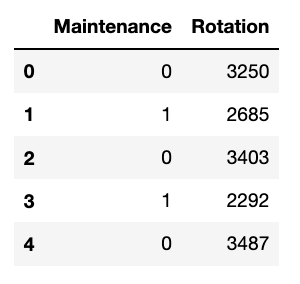
Close Price History: Source - Towardsdatascience.com [11]



Proof of the program’s difficulty can be exemplified by the number of error messages that I received when trying to process the linear binary Maintenance data. The first error I encountered was a “label exception” which I bypassed by typing in the data into a “pd.DataFrame” manually.

Manually generated Table: source – Stackoverflow.com [13]



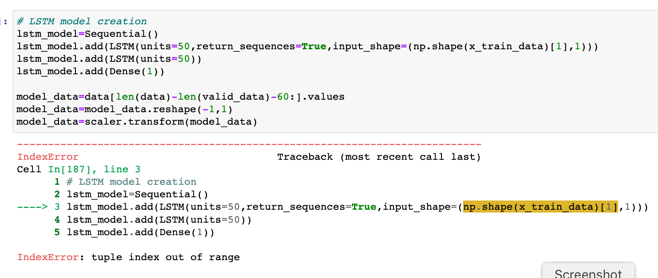


After I created the “Rotation” data frame, I proceeded to expand the manual data frame to accommodate the other variables.



But then I encountered another error when I tried to create the LSTM model. The error I encountered was a “tuple out of range” error. I did some additional research on the possible causes of the error, however none of the results that I found have yielded any usable syntax.

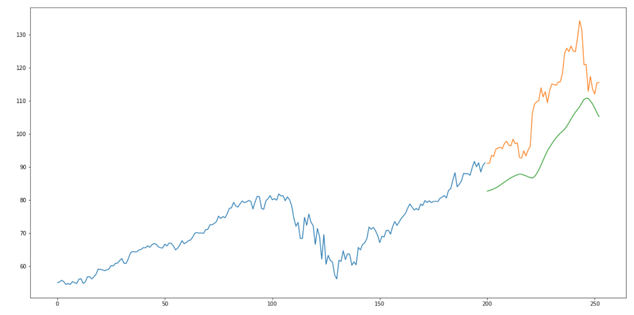
Tuple Out of Range:



While I could have found a solution given enough time, because of some unforeseen circumstances, I had to leave the program as it is.

The example output graph on the “towardsdatascience.com” webpage succeeds in where the code in the “AircraftPredictor.ipynb” file fails. The graph in the upper right represents what a successful prediction graph should like.

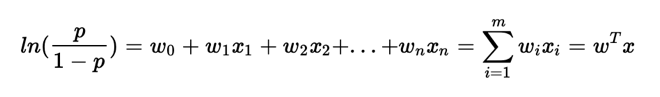
Successful Prediction: Source - Towardsdatascience.com [13]

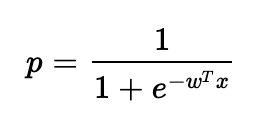


**VI. Logistic Regression**

With the failure of the “AircraftPredictor.ipynb” file to perform a prediction, I searched for another algorithm to make “yes” or “no” decisions out of multiple variables. The additional research yielded another prediction algorithm known as Logistic Regression.

Logistical Regression Formula: source – datajungler.com [12]



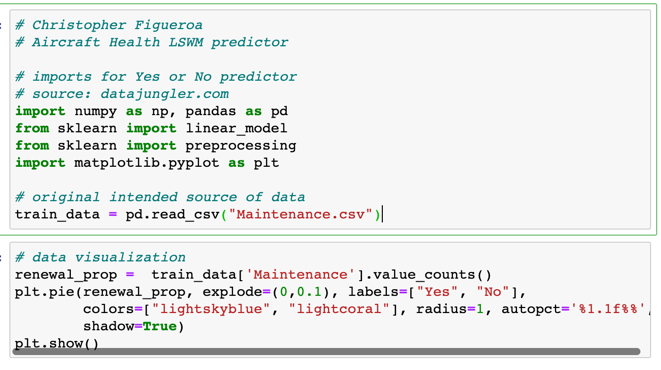


The smaller of the two equations represents a simplified version of the original Logistic Regression formula. This simpler variant intended to produce a “p” value between 0 and 1. The threshold of 0.5 represents the threshold that determines a “yes” or “no” decision.

This particular formula is less computationally expensive when compared to SGD or MAPE. However, many of the Logistic Regression based algorithms are prone to errors caused by under-fitting data instances. The volatile nature of the program in “datajungler.com” will become apparent as I attempt to recreate it in my own file.

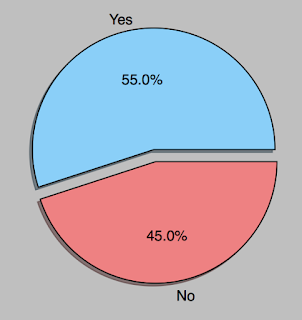
To test the usability of the new file “AircraftBinary.ipynb” contains code sourced from “datajungler.com”. The code below represents my attempt at recreating the website’s program. The one change I made was to have my “Maintenance.csv” file as the input for the “read\_csv()” function.

Data Visualization [12]:



The purpose of the code above is to put the “yes” and “no” variables into a visual format that represent the intended purpose of the source’s program. The program requires the libraries “pandas” and sklearn” to run properly The screenshot below represents the website’s code.

Output Chart [12]:



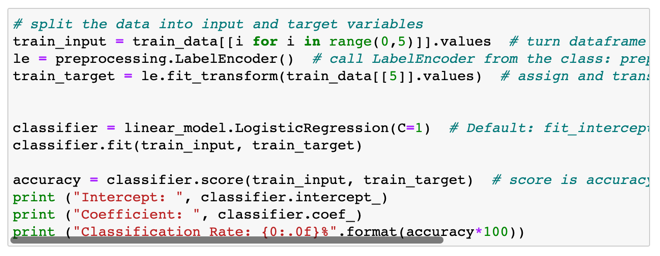
Although the code in the new file is practically identical to the one in the source, due to perhaps either some updates in Python or some differences in syntax between regular Python and Anaconda, the rest of my code does not work as intended. I attempted to alter the index range to “0-5” for the “train\_data()” function. But no alterations made a difference.

Key Error [12]:



The next intended step of the logistic regression program was to use the indexes 0-5 (2-6 in source code) of the .csv file’s table as the input for training. Each one of the indexes represents one of the factors that would be used for the training: vibration, temperature, pressure difference, electric flow, rotation.

Logistic Regression Training [12]:



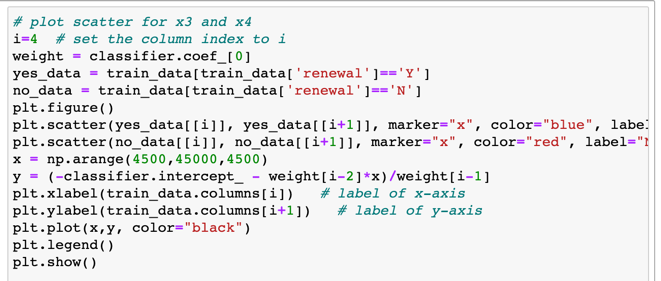
A classifier variable is to store the values given by the command “linear\_model.LogisticRepression()”. The function “fit()” is then called by the classifier to place in the “train\_input” and “train\_target” values.

Since there are 5 variables, the training model would be described as [12]:

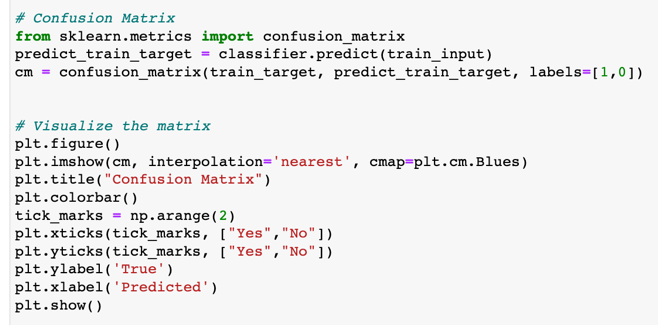
hw(x) = w0 + w0 + w1x1 + w2x2 + w3x3 + w4x4 + w5x5

Due to the failure for the code in the “AircraftBinary.ipybn” file to compile the rest of the example code below refers to the source code in its original state. No modifications have been made to accommodate the “Maintenance.csv” file.

Plot Generator Code [12]:



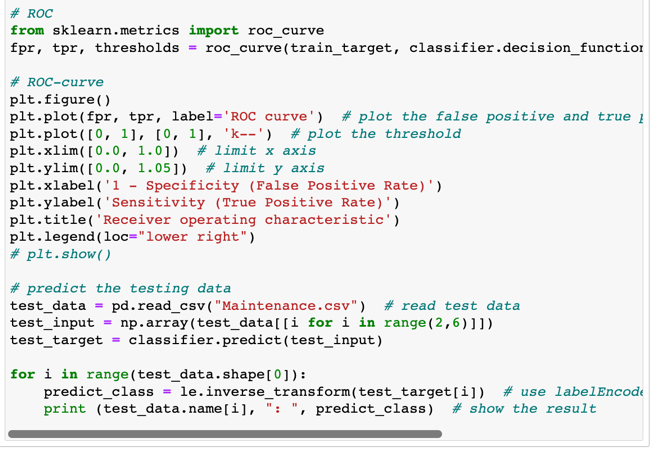
Confusion Matrix [12]:





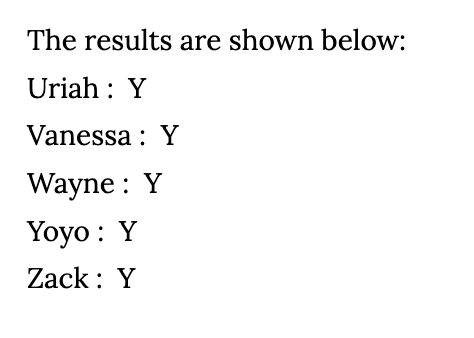
The plot generator code utilizes the classifier to find the classification rate. The classification rate is used in the construction of the confusion matrix which visualizes the Scikit-learn and MatplotLib respectively. The columns represent the predicted values while the rows represent the actual values.

The Prediction Code [12]:



The confusion matrix output is then used to create the other performance evaluation model Receiver Operating Characteristic (ROC). The result of the ROC curve is used to find the prediction necessary to determine if the data in the .csv file reaches the threshold of 0.5.

Source Results [12]:

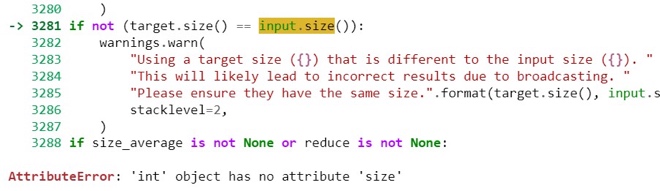


The source code in “datajungler.com” yielded the results above. However, since the code in the “AircraftBinary.ipynb” did not compile, there is no way to know what the results of that program would have been.

**VII. Challenges in Coding**

The objective of this project was to expand upon the preliminary iteration of this project and to create a neural network. Although I have made some significant alterations to the original code, I have encountered a number of road blocks that have halted the development of the neural network.

I stumbled upon the first of my issues when I attempted to model the neural network after Assignment 5. I attempted to use the rounded partial derivative result of 25 as the value for the “predict” function. Since Python was expecting a tuple with the attribute size, I got the following error:



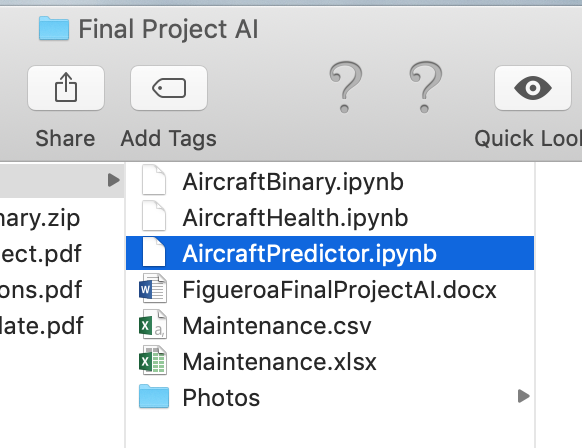
Prediction cannot use int values, therefore it cannot be used to calculate the input values. Loss function() is a built in library function. Therefore I cannot modify the function to take in values that are not tensors. This particular issue was the reason I opted to use the MAPE loss function, and then to create the “AircraftBinary.ipynb” and “AircraftPredictor.ipynb” files.

Besides the aforementioned issues above and the ones I mentioned in the other sections of the report, another major hurdle I faced was the unexpected shutdown of my primary school computer. For the past few weeks, I have been been greeted with “the blue screen of death”.

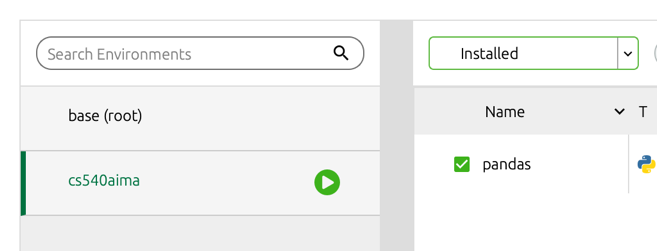
Blue Screen of Death: source - h30434.ww3.hp.com[14]



For the past few weeks, my Windows computer has been shutting itself down, and interrupting my research. When I could take the interruptions no more, I reinstalled my Anaconda environment onto my MacBook pro.



I encountered some issues while installing my Anaconda environment. Terminal did not always show me in what environment I installed the Python libraries. I had to open the Anaconda Navigator showed to find out I had installed the pandas and keras libraries outside of the Conda environment.



On top of that, when I got the news that there was going to be a snow storm, I had to rush to turn in my project before I was no longer able to find a taxi to leave campus. Needless to say I was not able to debug all my code.



**IIX. Conclusion**

Although I have not completed every goal I intended to do in this project, I have gained the necessary knowledge to debug my code and make a proper neural network in the near future. I have a program that can create randomly generated aircraft maintenance schedules. And with that program, I would potentially build more complex artificial intelligence programs when the time is right.

**References**

[1] B. Andriamanalimana, “Introduction to Fuzzy Inference Systems (FIS)”, *Module 18*, p. 2-25, 2022.

[2] D. A. Spencer, “Applying Artificial Intelligence Techniques to Air Traffic Control Automation”, *The Lincoln Laboratory Journal*, p. 1-5, 1989.

[3] C. Rovinescu, “How Artificial Intelligence is Changing the Aviation Industry”, *Social Hospitality*, p. 1-3, 2022.

[4] D. Buck and T. Burgidou, “Aircraft Condition Monitoring System (ACMS)”, *Skybrary*, p. 1-4, 2021.

[5] B. Andriamanalimana, “Assignment 6 Statement”, *Module 21*, p. 1, 2022.

[6] B. Elsevier, “Condition Monitoring System”, Sciencedirect.com, p. 8-14, 2017.

[7] B. Andriamanalimana, “Deep Reinforcement Learning”, *Module 24*, p. 5-10, 2022.

[8] D. Starnes, “Building a Neural Network & Making Predictions With Python AI”, *Realpython*, p. 1-3. 2021.

[9] B. Andriamanalimana, “Assignment 5 Statement”, *Module 17*, p. 1, 2022.

[10] F. Zach, “How to Calculate Mean Absolute Percentage Error (MAPE)”, *Statology*, p. 1-4, 2020.

[11] B. Guven, “Step-by-Step-Guide-Building a Prediction Model in Python”, *Towardsdatascience*, p. 2-8, 2020.

[12] J. Horace, “Logistic Regression-Tell Us “Yes” Or “No”, *Datajungler*, p. 1-7, 2016.

[13] Desertnaut, “Pandas Dataframe Columns Scaling With SKlearn”, *Stackoverflow*, p. 1, 2014.

[14] Q. Tedi, “Blue screen of death (BSOD) 116”, *www3.hp.com*, p. 1, 2019.