**DESIGN OF A COMPLIMENTARY MACHINE LEARNING AERODYNAMIC STALL WARNING**

**George Mason University**

**DAEN 690**

**Summer 2021**

**Team Machine**

Sean Case

Jake Ryan

Adam Ferguson

Ajay Kurian

**Table of Contents**

[***Abstract 4***](#_Toc79047153)

[***1 Introduction 5***](#_Toc79047154)

[***1.1 Background and Rationale 5***](#_Toc79047155)

[***1.2 Research 6***](#_Toc79047156)

[***1.3 Project Objectives 10***](#_Toc79047157)

[***1.4 Problem Space 10***](#_Toc79047158)

[***1.5 Primary User Story: 10***](#_Toc79047159)

[***1.6 Solution Space 10***](#_Toc79047160)

[***1.7 Product Vision - Sample scenarios (why would someone want to use this) 11***](#_Toc79047161)

[*■ Scenario #1 11*](#_Toc79047162)

[***2 Data Acquisition 12***](#_Toc79047163)

[***2.1 Overview: 12***](#_Toc79047164)

[***2.2 Field Descriptions: 12***](#_Toc79047165)

[***2.3 Data Context 15***](#_Toc79047166)

[***2.4 Data Conditioning 15***](#_Toc79047167)

[***2.5 Data Quality Assessment: 15***](#_Toc79047168)

[***2.6 Other Data Sources 15***](#_Toc79047169)

[***3 Analytics and Algorithms 16***](#_Toc79047170)

[***3.1 Random Cut Forest: 16***](#_Toc79047171)

[***3.2 DeepAR (Amazon SageMaker): 17***](#_Toc79047172)

[***4 Visualization 19***](#_Toc79047173)

[***4.1 Dataset: 19***](#_Toc79047174)

[***4.2 RCF Multiple Simulations: 20***](#_Toc79047175)

[***4.3 DeepAR: 22***](#_Toc79047176)

[***5 Findings 25***](#_Toc79047177)

[***5.1 Random Cut Forest: 25***](#_Toc79047178)

[***5.1.1 Final Algorithm: 25***](#_Toc79047179)

[***5.1.2 Algorithm Results: 25***](#_Toc79047180)

[***5.1.3 RCF Model Summary: 28***](#_Toc79047181)

[***5.2 DeepAR: 28***](#_Toc79047182)

[***6 Summary 33***](#_Toc79047183)

[***7 Future Work 33***](#_Toc79047184)

[***7.1 Random Cut Forest: 33***](#_Toc79047185)

[***7.2 DeepAR: 36***](#_Toc79047186)

[***● Appendix A- GitHub Links 37***](#_Toc79047187)

[***● Appendix B- Risk Analysis and Mitigation 38***](#_Toc79047188)

[***● Appendix C- Agile Process 39***](#_Toc79047189)

[***● Appendix D- Definition of Terms 40***](#_Toc79047190)

[***● Appendix E - Works Cited 42***](#_Toc79047191)

# Abstract

Aerodynamic stall events are fairly rare events where influxes of turbulence reduces the flight-enabling lift force for a plane, leading to an airline crash. This project aims to create a complimentary Artificial Intelligence (AI) Stall Warning model based on Machine Learning (ML) from historic flight data. The models could significantly increase safety margins for the stall hazard by identifying the onset of stall reliably before a human operator could. The models utilize simulated flight data as a proxy for historical flight data parameters, housed within an AWS SageMaker Jupyter Notebook. The two models we researched were anomaly detection through a Random Cut Forest and a predictive LSTM model from AWS SageMaker DeepAR.

This project is scoped as a research project with a focus on investigating potential AI solutions and not as a fully formed solution to the problem. A final ML solution would provide a pilot adequate time to perform evasive measures to mitigate the chances of a stall. Although the project dives into two algorithms and their use cases, the project does not have a fully vetted and ready algorithm to implement into a real-world aircraft. The results of this project provide the basis for future algorithm development and implementation for a real-world aircraft system.

# Introduction

## Background and Rationale

Aerodynamic stall is a hazardous condition for fixed wing aircraft. In a stall event , the laminar airflow over the wings that is conducive to generating a lift force from the wings becomes turbulent. The turbulence in the flow over the wing reduces the lift force and then abruptly ends the lift force. Without the Lift force on the wing, the aircraft “stops flying” and becomes a heavy object subject only to the gravitational force that falls out of the sky. Depending on the conditions the time from onset of stall to the stall can take less than 20 seconds. For this reason the recognition of the onset of stall and the stall condition is critical for initiating a stall recovery maneuver.

Due to the severity of the stall hazard, fixed wing aircraft are designed with several barriers to prevent the aircraft from reaching the stall condition. First, the aircraft is designed to operate at 30% above the stall speed (i.e. 1.3 VStall). This speed is known as the minimum safe operating airspeed. The onset of stall typically occurs 7% above the stall speed (i.e. 1.07 VStall). This “speed buffer” provides protection from stall even when the aircraft airspeed dips below the minimum safe operating airspeed. Second, cockpit warnings in the form of a haptic alert on the yoke/side-stick, known as the “stick shaker,” occur at the onset of stall 1.07 VStall. This haptic alert is also accompanied by aural and visual display alerts. Third, in some aircraft, an automated flight control system takes over control of the aircraft and automatically performs a nose-down maneuver associated with the stall recovery procedure. In other aircraft the flight crew are responsible for initiating the nose-down maneuver associated with the stall recovery procedure.

Several aircraft accidents (ABX827, XL Germany, AF 447, …) have been associated with failure of the flight crew to initiate the nose-down maneuver associated with the stall recovery procedure. In aircraft with an automated stall recovery, this function was disabled due to the absence of valid sensor data. The difficulty in flight crews identifying the onset of stall is due to the non-linear nature of the phenomenon. The onset of stall is identified by severe shaking of the aircraft as the airflow transitions from laminar to turbulent flow. This shaking is similar to the turbulence such as clear air turbulence, and to wake turbulence from a leading aircraft. Further, the onset of stall is typically accompanied by an uncommanded roll of the aircraft. This is the result of one wing of the aircraft reaching stall conditions and losing its Lift force before the other wing. The imbalance of Lift forces results in the uncommanded roll.

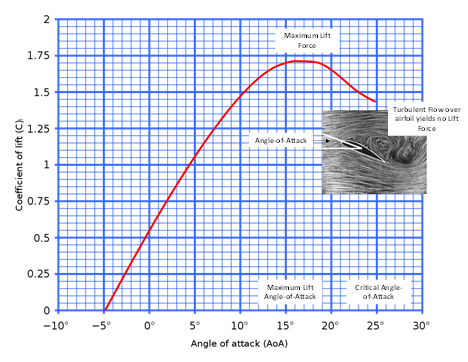
In some circumstances, the onset of stall is also accompanied by an increase in the Angle-of-Attack (also known as Alpha, and A-o-A). The increase in A-o-A is the result of natural aerodynamics as the wings lose Lift and the aircraft rotates around its Center of Gravity (CoG). With a high A-o-A, the aircraft will have the nose-up, but the flight path angle (i.e. the trajectory of the vehicle) will be negative resulting in an uncommanded descent. Although these aerodynamic and trajectory cues appear straightforward to identify as written above, keep in mind the flight crew may be faced with inaccurate airspeed, angles, and or rate of descent data. The flight crew may also be dealing with airspace, traffic, and/or terrain restrictions. The flight crew also may be dealing with aircraft system failures and alerts. There may also be communication between crew members on the flight deck, some of it accurate and other parts inaccurate. There may also be other factors such as the design of the upset procedure, the content and type of training on upset recovery, the accuracy and fidelity of simulators used for training, corporate policies and other personal mitigating factors.

At the end of the day, the detection of the onset of stall is a very complex cognitive activity that cannot be performed reliably by human operators. A complimentary Artificial Intelligence (AI) Stall Warning system based on Machine Learning (ML) from historic flight data could significantly increase safety margins for the stall hazard by identifying the onset of stall reliably before a human operator could.

## Research

All of the research, here in section 1.2 of this paper, was done by this project's sponsor Dr. Lance Sherry. Dr. Sherry and his team are the aviation experts we relied on heavily throughout this project to generate our dataset and gain deeper understanding into this project problem space. Without his knowledge and help this project would not have been possible. We are extremely grateful to Dr. Sherry and his entire team for the time and resources they dedicated to help us achieve the results we were after.

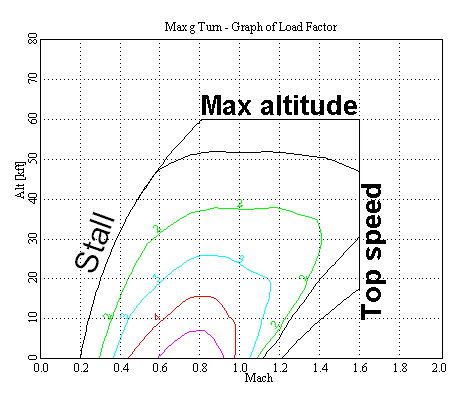
Laminar air flow over a cambered airfoil (i.e. a wing) generates an upward Lift force that enables a heavier air aircraft to “fly” and overcome the force of gravity (Figure 1.1). The magnitude of the Lift force increases as the angle between the oncoming flow and the angle of the airfoil (i.e. Angle of Attack) increases. At an Angle-of-Attack, known as the Maximum Lift Angle-of-Attack, the magnitude of Lift reaches its maximum. As the Angle-of-Attack increases beyond the Maximum Lift Angle-of-Attack, the magnitude of the Lift force decreases. At an Angle-of-Attack, known as the Critical Angle-of-Attack, the airflow over the airfoil transitions from laminar to turbulent flow and the Lift force abruptly goes to zero. At this point the wing (and aircraft) are no longer “flying.” Instead the wing and the aircraft are overcome by the force of gravity and “drop” out of the sky.



*Figure 1.1 - Lift Force vs Angle of Attack (AoA)*

Stall in a fixed wing aircraft is generally defined by a minimum airspeed (Figure 1.2 below). The equation for Lift Force is a function of atmospheric density shown here:

***Lift Force= (0.5) \* (atmospheric density) \* (Wing Surface Area) \* (Airspeed)2 \* (Lift Coefficient)***



*Figure 1.2 - Max Altitude vs Top Speed for Onset of Stall*

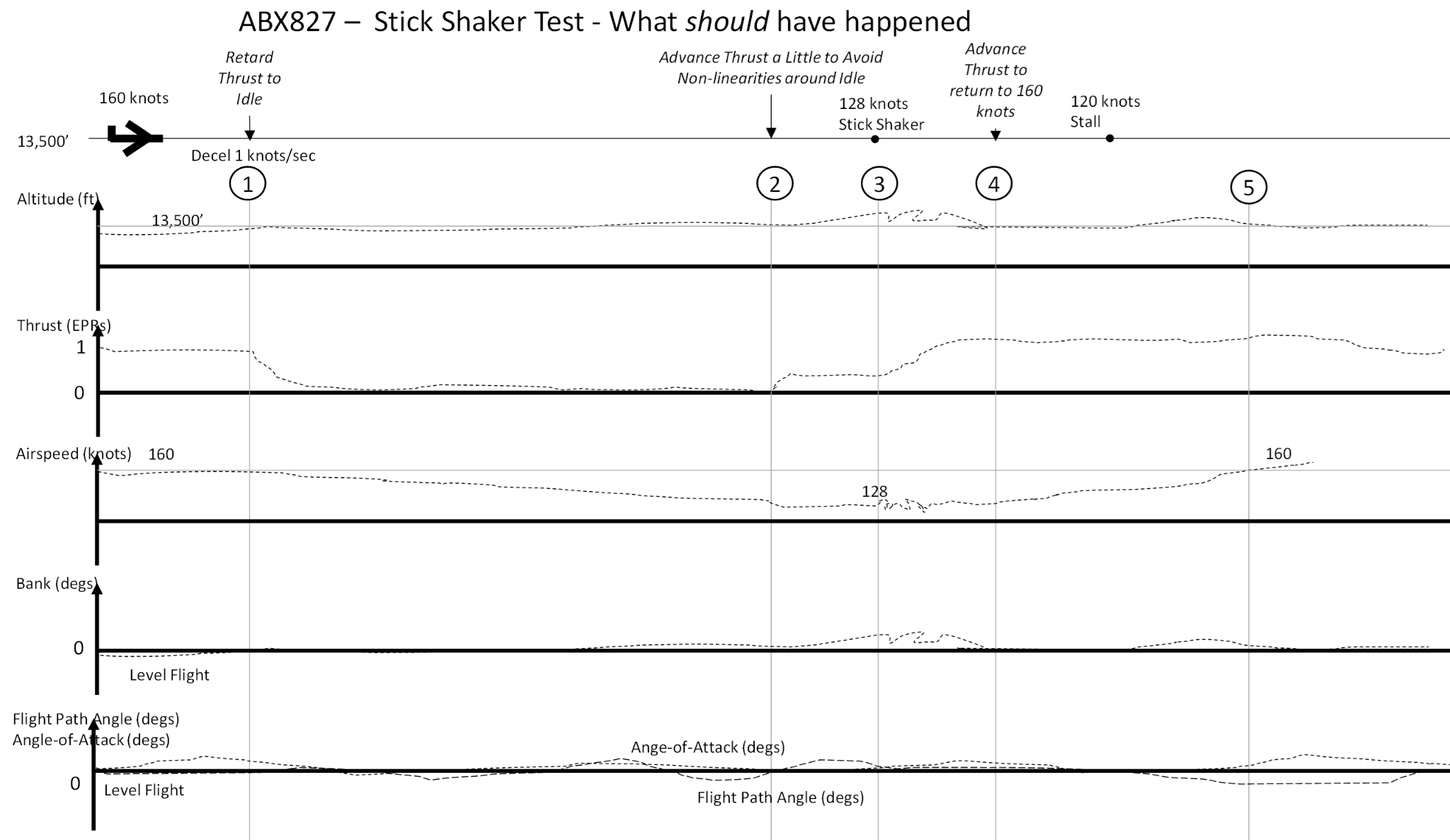
Fixed-wing aircraft can be equipped with devices to prevent or postpone a stall or to make it less (or in some cases more) severe, or to make recovery easier. Definition for these terms will be in Appendix D.

* Aerodynamic Twist
* Stall Strip
* Stall Fence
* Vortex Generators
* Anti-stall strake
* Stick pusher
* Stick Shaker
* Stall Warning
* Angle-of-attack indicator
* angle-of-attack limiter

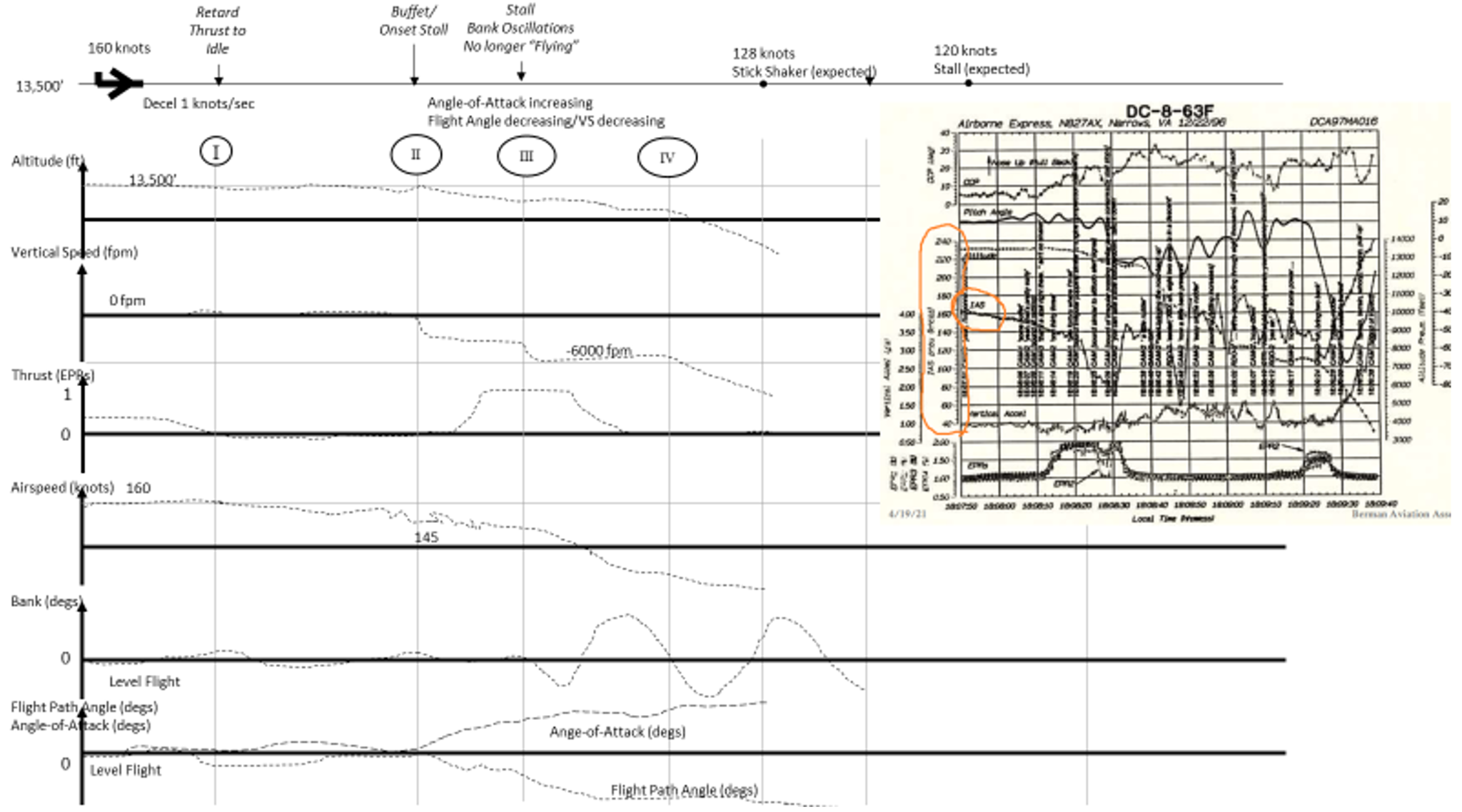
Stall warning systems often involve inputs from a broad range of sensors and systems to include a dedicated angle of attack sensor. Blockage, damage, or inoperation of stall and angle of attack (AOA) probes can lead to unreliability of the stall warning, and cause the stick shaker, overspeed warning, autopilot, and yaw damper to malfunction.

If a forward [canard](https://en.wikipedia.org/wiki/Canard_(aeronautics)) is used for pitch control, rather than an aft tail, the canard is designed to meet the airflow at a slightly greater angle of attack than the wing. Therefore, when the aircraft pitch increases abnormally, the canard will usually stall first, causing the nose to drop and so preventing the wing from reaching its critical AOA. Thus, the risk of main wing stalling is greatly reduced. However, if the main wing stalls, recovery becomes difficult, as the canard is more deeply stalled and angle of attack increases rapidly.

If an aft tail is used, the wing is designed to stall before the tail. In this case, the wing can be flown at higher lift coefficient (closer to stall) to produce more overall lift. Most military combat aircraft have an angle of attack indicator among the pilot's instruments, which lets the pilot know precisely how close to the stall point the aircraft is. Modern airliner instrumentation may also measure angle of attack, although this information may not be directly displayed on the pilot's display, instead driving a stall warning indicator or giving performance information to the flight computer (for fly by wire systems).



*Figure 1.3 - Theoretical Flight ABX827*



*Figure 1.4 - Actual Flight ABX827*

Boeing 737 Quick Reference Handbook (QRH), an upset condition had been defined as unintentionally exceeding any one or more of the following conditions: pitch attitude greater than 25° nose up, pitch attitude greater than 10° nose down, bank angle greater than 45°, less than the above parameters but flying at an airspeed inappropriate for the conditions.

Upset recovery training was included as a mandatory training program, which was required for recurrence within 24 months, and was also included as one of the training modules that had to be conducted during proficiency check.

The Sriwijaya Air Boeing 737 Quick Reference Handbook (QRH), page MAN.1.7, described the upset recovery procedure as follows: Historically, an upset has been defined as unintentionally exceeding any one or more of the following conditions: pitch attitude greater than 25°/nose up, pitch attitude greater than 10°/nose down, bank angle greater than 45°, or less than the above parameters but flying at an airspeed inappropriate for the conditions.

An upset condition is now considered any time an airplane is diverting from the intended airplane state. An airplane upset can involve pitch or roll angle deviations as well as inappropriate airspeeds for the conditions. The following actions represent a logical progression for recovering the airplane. The sequence of actions is for guidance only and represents a series of options to be considered and used depending on the situation. Not all actions may be needed once recovery is underway. If needed, use minimal pitch trim during initial recovery. Consider careful use of rudder to aid roll control only if roll control is ineffective and the airplane is not stalled.

These actions assume that the airplane is not stalled. A stall condition can exist at any attitude and can be recognized by one or more of the following:

* Stick shaker
* Buffet that can be heavy at times
* Lack of pitch authority
* Lack of roll control
* Inability to stop a descent.

If the airplane is stalled, first recover from the stall by applying and maintaining nose down elevator until stall recovery is complete and stick shaker stops

## Project Objectives

The primary objective of our project is to use machine learning techniques to identify instances of stall against normal (i.e. “non-stall”) conditions and generate a machine notification at the onset of stall.

## Problem Space

Aerodynamic stall is a hazardous condition which can contribute to airline accidents, damage of property, and loss of life. Today, pilots rely on physical vibrations from a “stick shaker” to be notified of stall, a condition where influxes of turbulence hinders the ability of a plane to fly. A reliable means to recognize the onset of stall against normal “non-stall” conditions and to notify pilots to take corrective action is required.

## Primary User Story:

Based on the user context and value proposition, we developed the following primary user story to guide our project:

***“As a pilot, I want to be notified of stall conditions before they occur, with adequate time, so that I may take the appropriate corrective action to save the aircraft and the lives of those on the plane.”***

## Solution Space

Our system delivers values to its end users (i.e. pilots) by accurately identifying and reporting stall conditions while being able to distinguish stall events from non-stall events. The system reports these conditions in the form of a machine notification which will be picked up and interpreted by the plane system. Users derive value from these notifications when the plan system uses those notifications to alert the pilot of stall activity. The pilots can take the appropriate corrective actions to address the stall event or mitigate risk associated with the stall event.

## Product Vision - Sample scenarios (why would someone want to use this)

* *For*: Pilots experiencing stall
* *Who*: Mr. Bigshot
* *The*: Captain
* *Is a*: Pilot
* *That*: Is about to experience stall
* *Unlike*: Stick Shaker
* *Our product*: Detects stall before it is happening and gives a warning
* *Caveats:* Needs to notify pilots with at least 10 seconds notice before a stall occurs. Needs to have 99.99999% accuracy

### Scenario #1



*Figure 1.5 - Storyboard of Potential Solution*

# Data Acquisition

## Overview:

Aviation data is oftentimes tightly regulated and not easily accessible. For the purposes of this project, synthetic data will be generated using parameters and guidelines provided by subject matter experts (SMEs) in this domain. All of the data for this project will be simulated using Python code.

## Field Descriptions:

* flight\_id (Type: Integer) - A constant variable for each simulation. The unique identifier for each flight we simulated. The first flight starts at “1” and continues to increase by 1 for every flight we simulate.
* initial\_alt (Type: Integer) - A constant variable for each simulation. The initial altitude of the aircraft, measured in feet. This is randomly assigned to each simulated flight and is held constant for the duration of the flight\_id simulation. It is a number we randomly select between 5,000 and 43,000.
* time\_to\_buffet (Type: Integer) - A constant variable for each simulation. The time until the flight is to encounter a buffet, measured in seconds. This is randomly assigned to each simulated flight and is held constant for the duration of the flight\_id simulation. It is a randomly generated number between 0 and 100.
* time\_from\_buffet\_to\_uncommanded\_descent (Type: Integer) - A constant variable for each simulation. The time from when an air buffet occurs to the uncommanded descent of the flight, measured in seconds. This is randomly assigned to each simulated flight and is held constant for the duration of the flight\_id simulation. It is a randomly generated number between 0 and 10.
* magnitude\_of\_uncommanded\_descent (Type: Integer) - A constant variable for each simulation. The magnitude, or severity, of the flight's uncommanded descent, measured in feet per second (fps). There is a condition on this variable where it cannot be less than time\_from\_buffet\_to\_uncommanded\_descent . Due to this, we chose to select a random number between (time\_from\_buffet\_to\_uncommanded\_descent+1) and (time\_from\_buffet\_to\_uncommanded\_descent+300).
* time\_from\_buffet\_to\_uncommanded\_roll (Type: Integer) - A constant variable for each simulation. The time from when an air buffet occurs to the uncommanded roll of the flight, measured in seconds. There is a condition on this variable where it cannot be less than time\_from\_buffet\_to\_uncommanded\_descent . Due to this, we chose to select a random number between (time\_from\_buffet\_to\_uncommanded\_descent) and (time\_from\_buffet\_to\_uncommanded\_descent+30).
* magnitude\_of\_uncommanded\_roll (Type: Integer) - A constant variable for each simulation. The magnitude, or severity, of the flight's uncommanded roll, measured in degrees (degs). This is randomly assigned to each simulated flight and is held constant for the duration of the flight\_id simulation. It is a randomly generated number between 0 and 90.
* period\_of\_uncommanded\_roll (Type: Float) - A constant variable for each simulation. The rate at which the uncommanded roll changes directions. This is randomly assigned to each simulated flight and is held constant for the duration of the flight\_id simulation. It is a randomly distributed normal variable, with mean=0 and sd=10.
* initial\_airspeed (Type: Integer) - A constant variable for each simulation. The initial airspeed of the aircraft, measured in knots. This is randomly assigned to each simulated flight and is held constant for the duration of the flight\_id simulation. It is a number we randomly select between 120 and 300.
* time\_from\_buffet\_to\_uncommanded\_descent\_high (Type: Integer) - A constant variable for each simulation. The time between when buffet begins and when the plane begins its high uncommanded descent. This is randomly assigned to each simulated flight and is held constant for the duration of the flight\_id simulation. There is a condition on this variable where it cannot be less than time\_from\_buffet\_to\_uncommanded\_descent . Due to this, we chose to select a random number between (time\_from\_buffet\_to\_uncommanded\_descent+1) and (time\_from\_buffet\_to\_uncommanded\_descent+30).
* magnitude\_of\_uncommanded\_descent\_high (Type: Integer) - A constant variable for each simulation. The magnitude of the high uncommanded descent in feet per minute, which we then converted to feet per second for our calculations. This is randomly assigned to each simulated flight and is held constant for the duration of the flight\_id simulation. It is a randomly generated number between 120 and 1200.
* time\_from\_buffet\_to\_positive\_angle\_of\_attack (Type: Integer) - A constant variable for each simulation. The time between the onset of buffet and the beginning of a positive angle of attack, measured in seconds. This is randomly assigned to each simulated flight and is held constant for the duration of the flight\_id simulation. There is a condition on this variable where it cannot be less than time\_from\_buffet\_to\_uncommanded\_descent . Due to this, we chose to select a random number between (time\_from\_buffet\_to\_uncommanded\_descent+1) and (time\_from\_buffet\_to\_uncommanded\_descent+10).
* max\_angle\_of\_attack (Type: Integer) - A constant variable for each simulation. The maximum angle of attack the simulated flight can reach during the simulation measured in degrees. There is a condition where this may never be larger than 30 degrees. It is a randomly generated number between 10 and 30.
* rate\_of\_change\_in\_angle\_of\_attack (Type: Float) - A constant variable for each simulation. The rate at which the angle of attack can change during the flight simulation, measured in degrees/second (degs/sec). There is a condition where this may never be larger than 1 degree/second. It is a randomly generated number between 0 and 1.
* cur\_time (Type: Float) - A dynamic variable that is re-generated every 1/10th of a second during the simulation. The current time of the flight, measured in seconds, which increases every 1/10th of a second throughout the simulation.
* alt\_noise (Type: Float) - A dynamic variable that is re-generated every 1/10th of a second during the simulation. It is the random noise in altitude change the flight encounters during the simulation, measured in feet. It is a randomly distributed normal variable, with mean=0 and sd=5.
* alt\_noise\_buffet (Type: Float) - A dynamic variable that is re-generated every 1/10th of a second during the simulation. It is the random noise in altitude change, caused by wind buffet, the flight encounters during the simulation, measured in feet. It is a randomly distributed normal variable, with mean=0 and sd=10.
* airspeed\_noise (Type: Float) - A dynamic variable that is re-generated every 1/10th of a second during the simulation. It is the random noise in air speed change the flight encounters during the simulation, measured in knots. It is a randomly distributed normal variable, with mean=0 and sd=5.
* airspeed\_noise\_buffet (Type: Float) - A dynamic variable that is re-generated every 1/10th of a second during the simulation. It is the random noise in air speed change, caused by wind buffet, the flight encounters during the simulation, measured in knots. It is a randomly distributed normal variable, with mean=0 and sd=10.
* cur\_altitude (Type: Float) - A dynamic variable that is re-generated every 1/10th of a second during the simulation. It is the current altitude at which the plane is flying, measured in feet. It is calculated by taking the previous altitude and adding or subtracting the magnitude of uncommanded descent multiplied by the time interval of .1 seconds.
* cur\_airspeed (Type: Float) - A dynamic variable that is re-generated every 1/10th of a second during the simulation. It is the current horizontal airspeed at which the plane is flying, measured in knots. It is calculated by taking the previous airspeed and adding or subtracting the airspeed\_noise or airspeed\_noise\_buffet, depending on if the onset of buffet has occurred.
* roll (Type: Float) - A dynamic variable that is re-generated every 1/10th of a second during the simulation. It is the current angle at which the plane is flying considering the front-to-back axis of the plane, which is caused by one wing experiencing stall prior to the other wing and measured in degrees. It is calculated by taking the sine of the magnitude of uncommanded roll.
* vertical\_speed (Type: Float) - A dynamic variable that is re-generated every 1/10th of a second during the simulation. It is the current vertical speed of the plane, measured in knots and converted to feet per second for calculations. It is calculated by subtracting the previous altitude from the current altitude and dividing by the time interval of .1 seconds.
* angle\_of\_attack (Type: Float) - A dynamic variable that is re-generated every 1/10th of a second during the simulation. It is the current angle of the nose of the plane in relation to the trajectory of the plane, measured in degrees. Once the onset of buffet occurs, it is calculated by adding the rate of change in angle of attack to the previous angle of attack, until it reaches the designed max angle of attack.
* flight\_path\_angle (Type: Float) - A dynamic variable that is re-generated every 1/10th of a second during the simulation. It is the current angle of the plane’s trajectory in relation to the ground irrespective of angle of attack, measured in degrees. It is calculated by taking the inverse sine of vertical speed divided by current airspeed.
* pitch\_angle (Type: Float) - A dynamic variable that is re-generated every 1/10th of a second during the simulation. It is the current difference between the flight path angle and angle of attack, measured in degrees. It is calculated by subtracting the angle of attack from the flight path angle.

## Data Context

Before we could work with the data we had to understand exactly what the fields above represented. From the fields above the ones we had to focus on were cur\_altitude, cur\_airspeed, roll, vertical\_speed, angle\_of\_attack, flight\_path\_angle, and pitch\_angle. These were the variables that would change during a flight and could be used to detect when a plane is about to or starts falling out of the sky due to stall.

## Data Conditioning

Data conditioning for this project was different from the normal cleansing and pruning, because all the data used in this project was synthetic. Random generation of the data prior to each simulation caused the focus of data conditioning to shift more to refining our formulas and calculations that created each piece of data. Numerous formulas relied on other variables for generation and the variables used in these formulas were not always in the same unit. Our team decided to convert most speed related fields to feet per second to provide uniformity throughout those calculations. For flight path angle and roll conversions between radians and degrees were also required in order to create the correct output.

Random number generation and normal distributions were used throughout the data generation based on the input we received from the subject matter expert we were partnering with. Through the iterations of these formulas and calculations, the main goal was to create a dataset that as closely as possible resembled real world flight data. This is a necessity in order to fully train and test an algorithm that could potentially be used in planes in the future. Once a real world dataset has been obtained it would be very important to compare side by side with our synthetic dataset to ensure a realistic dataset was produced through our simulated means.

## Data Quality Assessment:

**Dataset** - Simulated Synthetic Flight Data

* Completeness: our dataset has no null values and every row is populated with a value
* Uniqueness: They are not all unique, by nature of using randomly distributed normal variables we will expect to have some duplicate values in some of our records.
* Accuracy: All data values in our dataset are based on calculations and documentation provided to us by Dr. Sherry. They are all consistent, unambiguous, and therefore, correct.
* Conformity: Since this is a synthetic dataset the main area of concern is whether the generated values conform to what is reasonable in a flight scenario.
* Overall Quality: With the formulas provided, the team asserts that the quality of the dataset is high. The data conforms to the guidelines outlined by Dr. Sherry.

## Other Data Sources

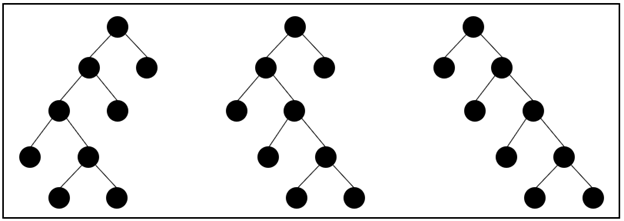
We had the possibility of having a data source provided from real flight data, but ultimately were unable to receive this from our partner by the time the semester ended. Future work built on this project should look to obtain and utilize flight data from actual planes.

# Analytics and Algorithms

## Random Cut Forest:

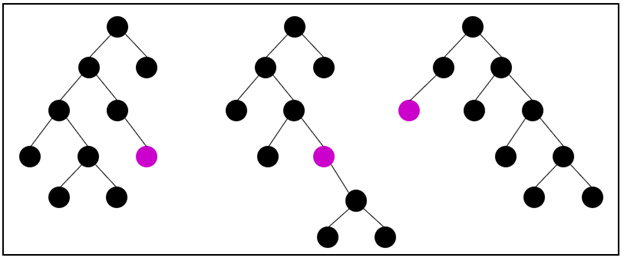
A random cut forest (RCF) algorithm is a modified version of a random forest (RF) algorithm, a ML algorithm that is widely used and highly successful in its ability to detect anomalies. RCF was developed by AWS and is an unsupervised algorithm that uses a cataloged set of random data points, which it partitions (or cuts) into equal numbers of points, and builds a collection of multiple models. The “Forest” name in the RCF algorithm is because the models are decision trees. Due to the fact that RFs are not able to incrementally update with ease, the RCF algorithm was built to provide quick incremental updates with the use of variables in the tree construction. RCF relies on cluster analysis to discover data outliers, cuts in seasonality or periodicity, and to identify anomalies in time series data.

Anomalies are points in a data set that draw attention away from the “normal” points in the data. For example, imagine a parking lot full of 99 black cars and 1 red car, the red car draws attention away from the 99 black cars and is the anomaly in the parking lot. While humans may be able to identify data points that seem to be anomalies, the RCF algorithm can detect anomalies the human eye can not see. It detects them by building multiple decision trees, known as a Forest of Trees, to track how newly added data points affect the forest. A RCF model may have between 1 and 1000 trees, the figure below is an example forest with three trees.



*Figure 3.1 - Example RCF with Three Trees*

Each anomaly in the data is assigned a score by the RCF algorithm by calculating the mean score from each of the decision trees and appropriately scaling the output by the sample size. All anomaly scores from the different models are combined, due to the individual models being weak identifiers on their own, and a final anomaly score is calculated. Amazon Sagemaker pinpoints anomalous data points when the score of the data point is statistically significantly dissimilar from other points in the models. In the example forest below, we can see the addition of the data point to tree 2 causes the complexity and depth of the tree to increase seen by the additional branch below the data point being created. Tree 3 has a moderate change as its width increases and Tree 1 has almost no change. By taking the average change between all trees in this example, we see a statistically insignificant result for the data point leading us to determine it is not an anomaly.



*Figure 3.2 - Example RCF with “Anomalous” Data Point*

This project decided to utilize a RCF model to help us determine when an aircraft’s flight began to turn from normal to something abnormal. The RCF model, built in this project, allowed us to pick up on small changes in multiple variables. When these changes occurred, the algorithm was able to compute mean scores for the entire simulation and alert us to an uncommanded change in one or many of the variables when their score deviated from the mean by 2 or more standard deviations. Standard practice for RCF is usually to look at scores at or beyond 3 standard deviations from the score mean, but we felt that 2 standard deviations would be better for this project. This is because we would rather be over cautious and provide pilots more time and more information about the aircraft even if a stall was not coming. A false positive for this project was not as concerning for us compared to a false negative where the algorithm would not pick up anomalies in time for the pilot to take action, often resulting in catastrophic results for the aircraft.

## DeepAR (Amazon SageMaker):

DeepAR is a forecasting algorithm included within Amazon SageMaker that utilizes recurrent neural networks (RNNs) to forecast time series data. In contrast to traditional neural networks, RNN takes information from inputs in the past to influence inputs and outputs in the future.

A traditional feed-forward neural network architecture consists of three layers: an input, hidden, and output layer. The input layer consists of the data that is fed to the model. The hidden layer is where the inputs are multiplied by a weight value then summed up to get a sum of weighted values. The sum of weighted values are fed into an activation function which is used to forecast an output value. The resulting output is then fed to the output layer (BuiltIn 2019, IBM 2020).

A loss function calculates the gradient between the model output and the actual output and propagates this loss back to the hidden layer. The parameters and hyperparameters in the hidden layer are subsequently tuned to reduce the loss function (i.e. reducing the error between the model output and actual values). This is a concept known as backpropagation and is the mechanism by which a neural network “learns” from data.

A recurrent layer adds a looping mechanism that allows previous information, represented through a hidden state, to influence future layers. In contrast, feed-forward networks only move forward and have no memory of the input that they receive. RNN also uses a form of backpropagation known as backpropagation through time (BPTT). In BPTT, the timesteps of the input values are unrolled and the loss is calculated at each timestamp. The network is subsequently rolled up and the weights are updated to apply backpropagation in time series data (BuiltIn 2019).

DeepAR applies concepts of recurrent neural networks in its application. DeepAR is trained on one or more target time series by randomly sampling training windows from each time series in the training set. The training window consists primarily of a context window, which indicates how far in the past the network can use to influence its forecast, and a prediction window, which reflects how far in the future the model should forecast for the time series data. The context and prediction windows are fixed lengths and are defined as hyperparameters in the model (Amazon 2021).

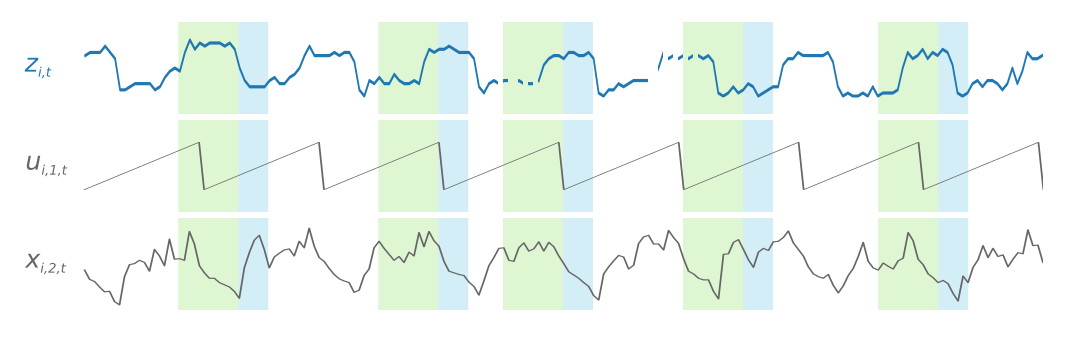


Figure 3.3. Sagemaker DeepAR Diagram

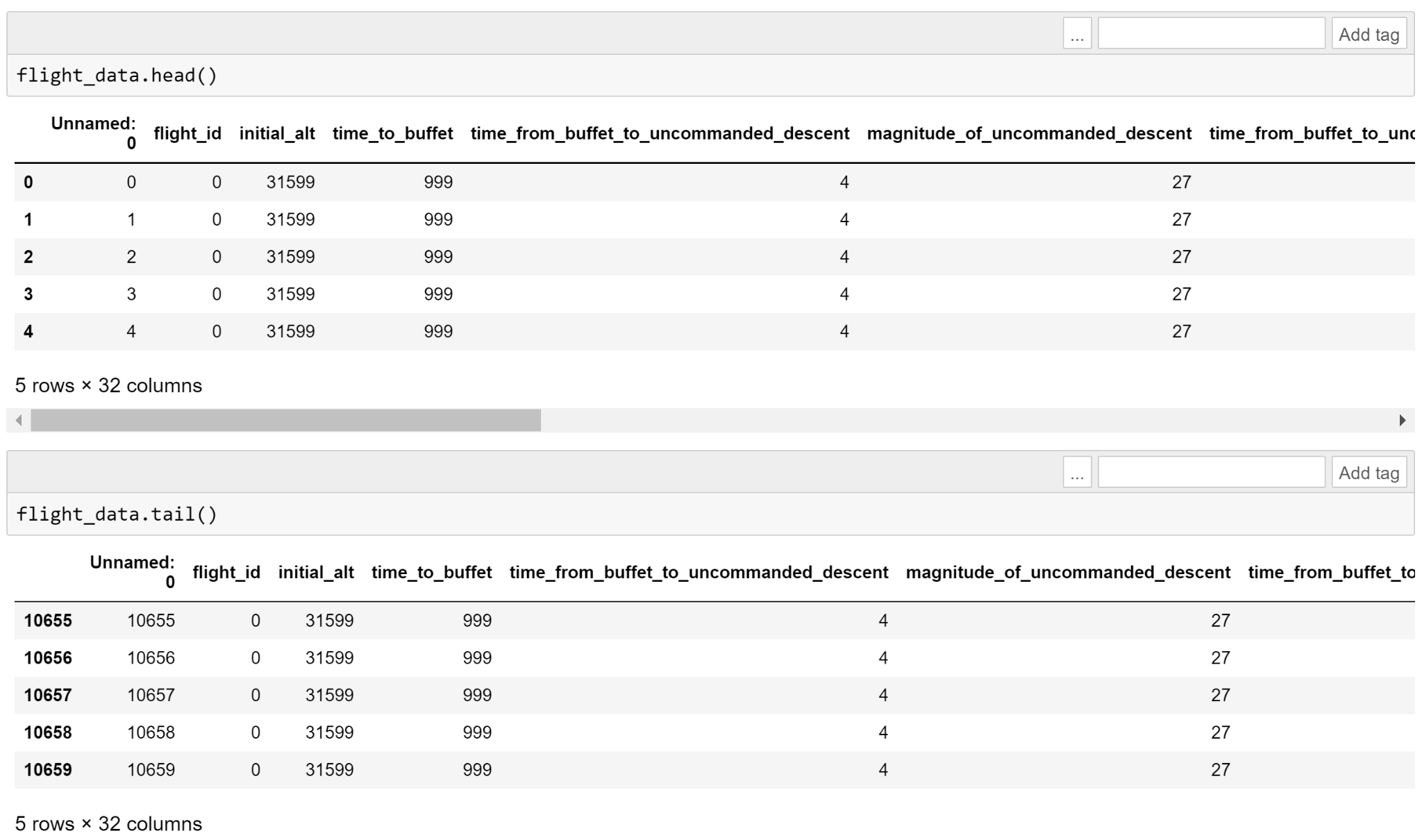
Rather than fitting individual models for multiple different time series, DeepAR creates one global model from related time series to identity patterns and handles varying scales. The architecture utilizes Gaussian binomial likelihood to produce a probabilistic forecast. When training the model, the current likelihood is used to calculate the error. When performing backpropagation, the weights of the hidden network are tuned, adjusting the parameterization of every Gaussian likelihood, to reach an optimal value ( Salinas, Flunkert, Gasthaus 2017). After training, forward propagation occurs and distribution parameters are calculated for the output distribution. The default distribution enabled by DeepAR is Gaussian so the end result will be a distribution rather than a single value (Arrigoni 2018).

The project decided to utilize the DeepAR algorithm to determine how a forecasting algorithm would function in detecting stall events. Since DeepAR was built to identify patterns on numerous similar time-series data and forecast based on a likelihood distribution, it was determined to be an algorithm of interest. However, during testing it was discovered that the DeepAR algorithm is limited in its level of granularity for the frequency parameter. Frequency is used in conjunction with the start time to assign timestamp values for each data point in a time series. Currently, the smallest time unit accepted by DeepAR is minutes. Since the stall dataset has a row for every tenth of a second, an initial attempt was made to convert tenths of a second to minutes (i.e. frequency = 0.00167minutes). It was subsequently discovered that DeepAR only accepts integer values before a unit of time (i.e. 1 min is the smallest time stamp). Therefore, the frequency value was set to 1 min with a larger context and prediction window to account for the new time scale. While this limits the applicability of this algorithm in an operational context (since the project wanted to detect stall events within seconds), it was determined that an investigation into DeepARs forecasting abilities was still applicable. The results and findings are listed in the following sections.

# Visualization

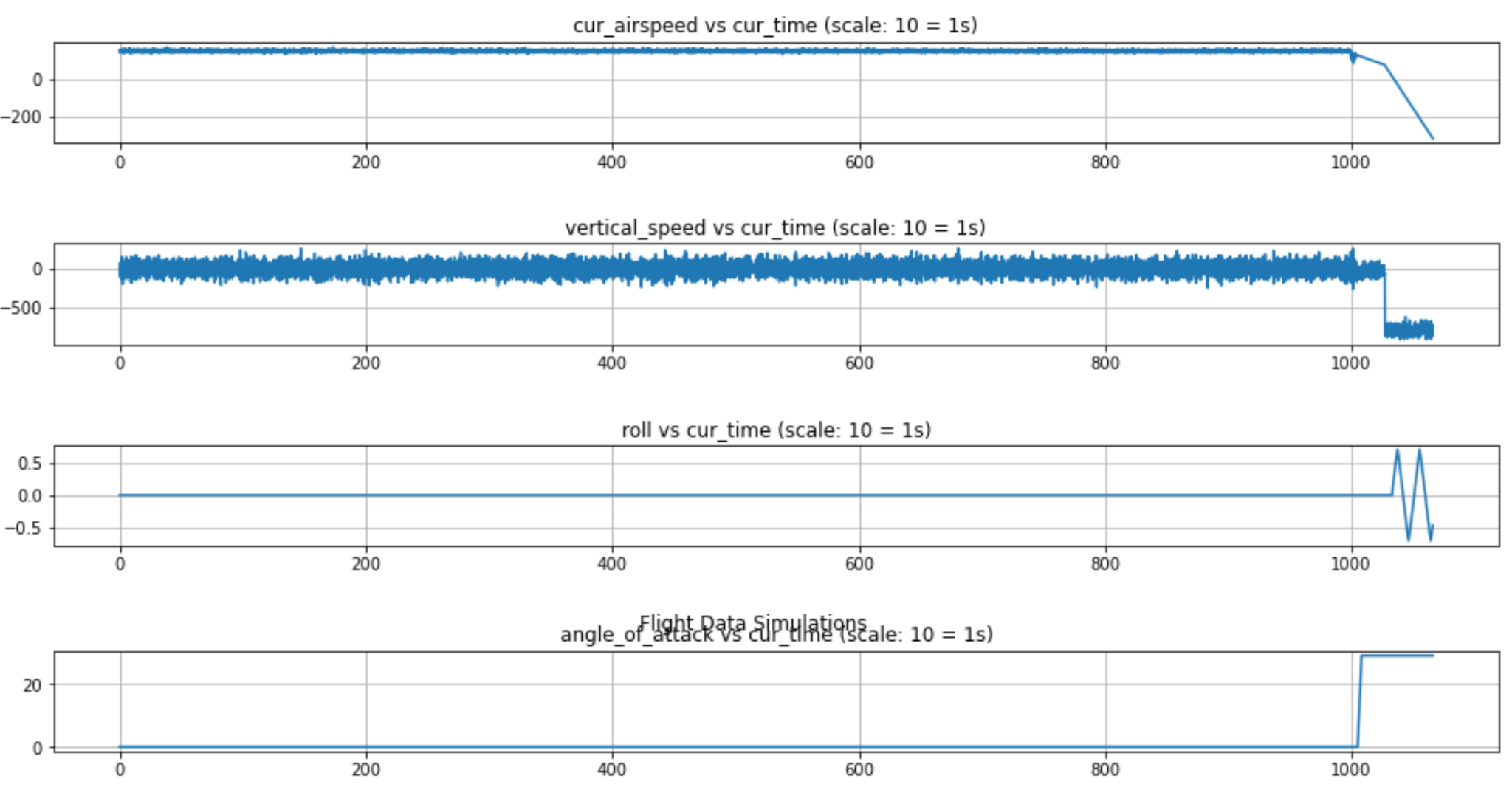
## Dataset:

Figure 4.1 shows the top five and bottom five rows of data for our flight simulation we used for testing our RCF model. We can see it is only for one simulated flight as the flight\_id variable is 0 at row 1 and at row 10659. We also can see other useful information such as the time until the aircraft is going to experience air buffet from the ‘time\_to\_buffet’ variable (999 seconds), and when the aircraft will begin its uncommanded descent seen in the ‘time\_from\_buffet\_to\_uncommanded\_descent’ variable (4 seconds). There is additional inference we can make looking at this data by looking at multiple variables at once. By adding the two variables mentioned above together, we know that the time the aircraft will face uncommanded descent and be at risk for a stall is at 1003 seconds (999 + 4).

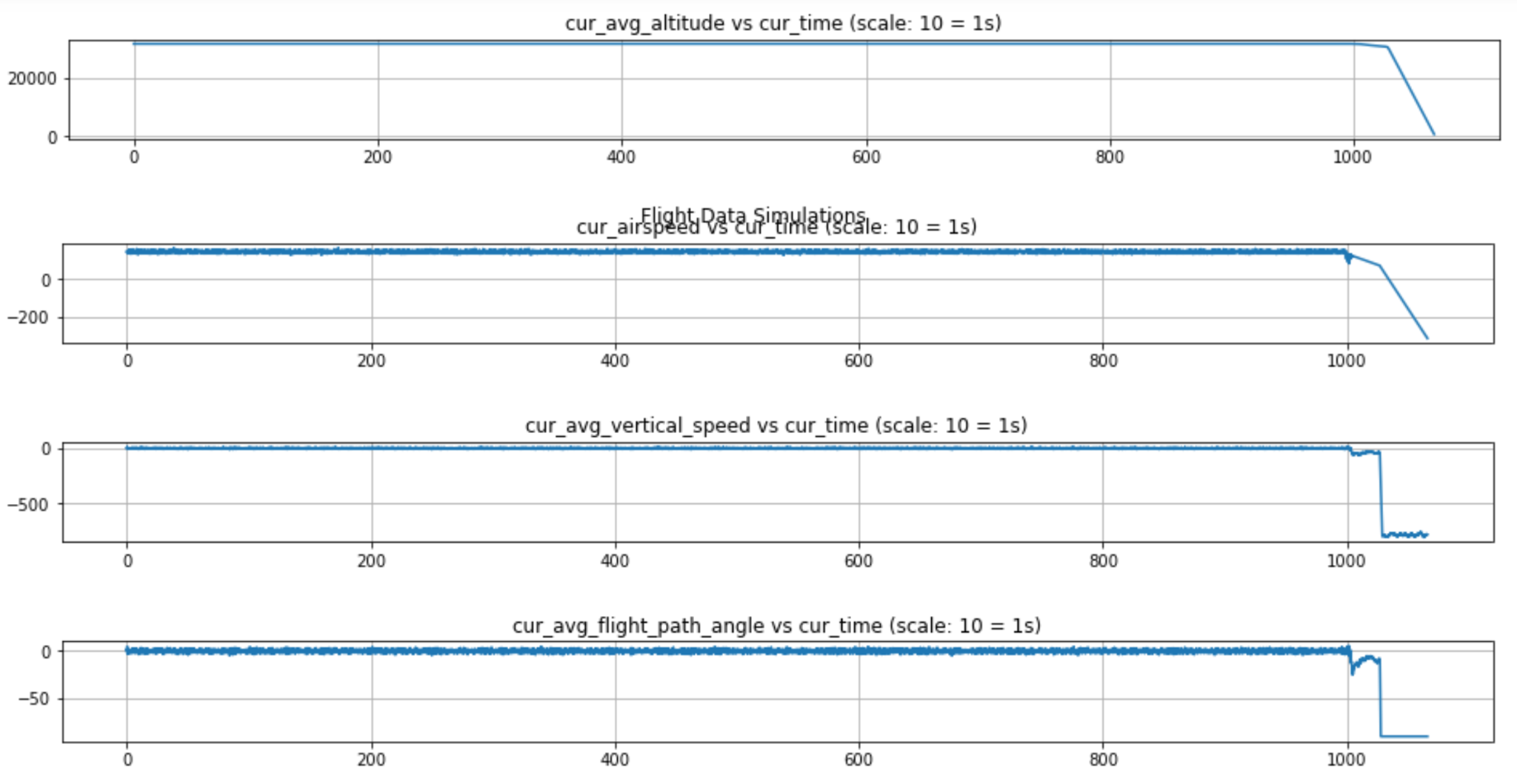


*Figure 4.1 - Simulated Dataset top 5 and bottom 5 rows*

Figures 4.2 and 4.3 represent eight of the variables we are using to determine if an aircraft is at risk for stall or any uncommanded descent/rolls. Figure 4.2 is the raw variable datapoints, plotted every 1/10th of a second. That is the reason for the noticeable “noise” in a few of the variables before the 999 second of the simulation, mainly ‘cur\_airspeed’ and ‘vertical\_speed’. Figure 4.3 is similar to Figure 4.2, but instead takes an average of the data points every 2 seconds to smooth out our plots. We can see the “noise” in the ‘cur\_airspeed’ and ‘vertical\_speed’ variable graphs is greatly reduced by using this method.



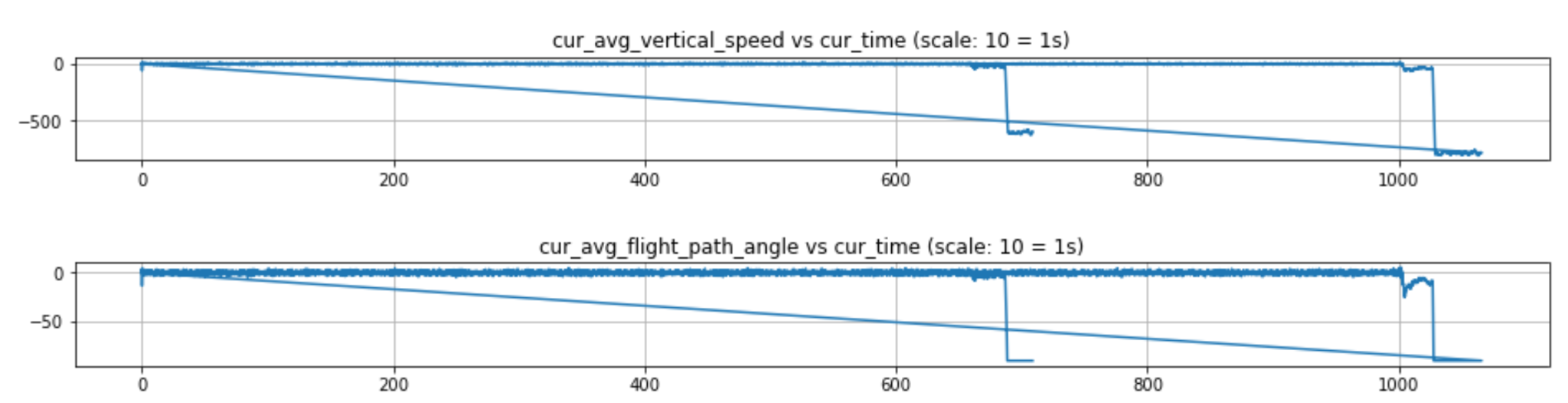
*Figure 4.2 - Simulated Dataset Variables 1*



*Figure 4.3 - Simulated Dataset Variables 2*

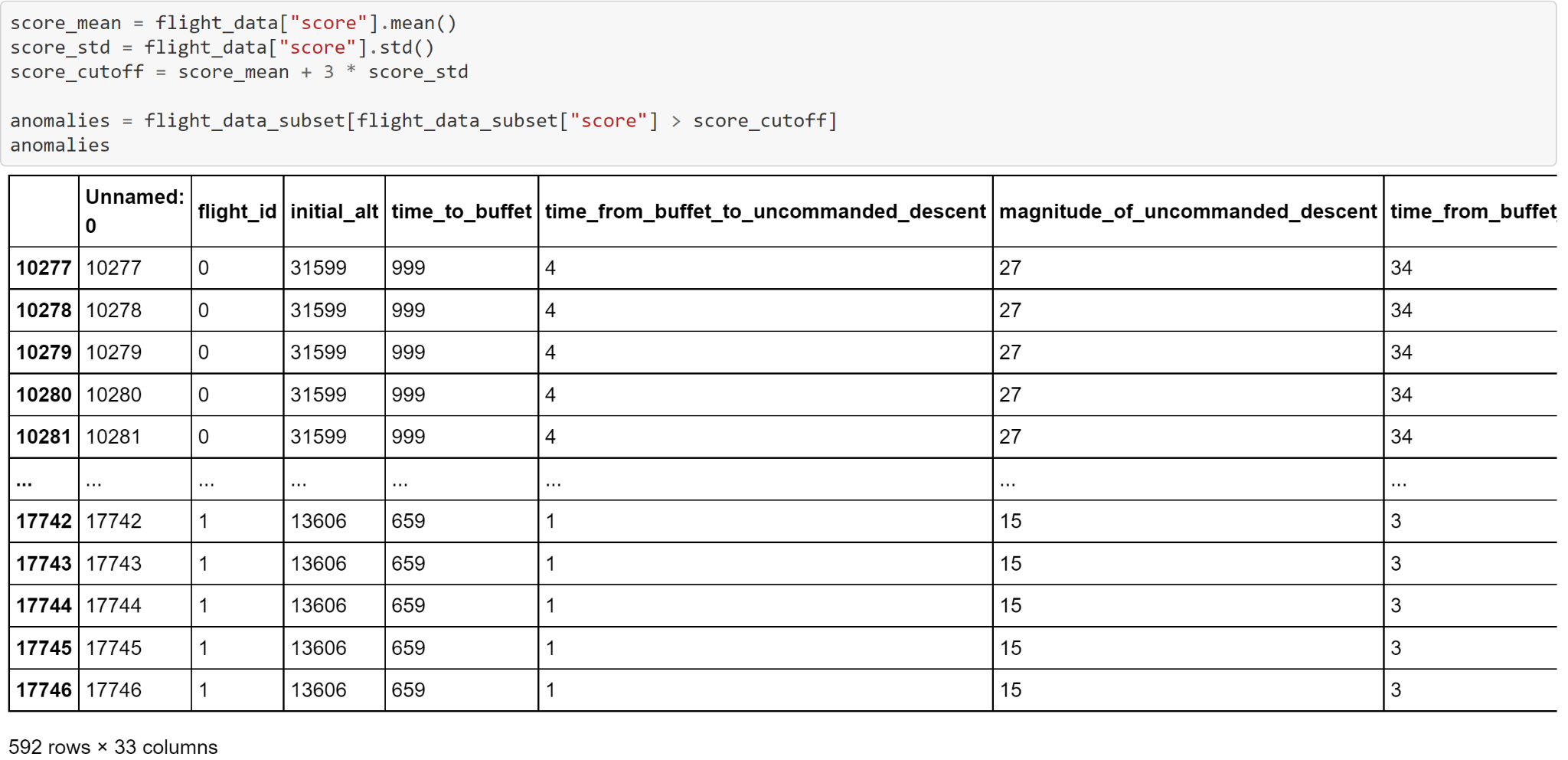
## RCF Multiple Simulations:

As a part of this project, we attempted to run multiple simulated flights through a RCF model to see if adding additional data provided us with any earlier anomaly detection rates. We struggled through this portion of the project as getting the variables and individual flights to plot and read in separately was a difficult task. Figure 4.4 below shows what two variables, from two different simulated flights, look like graphed on the same axis. We can see two distinct trend lines for each flight’s variable, one ends somewhere around 700 seconds and the other ends toward 1100 seconds.



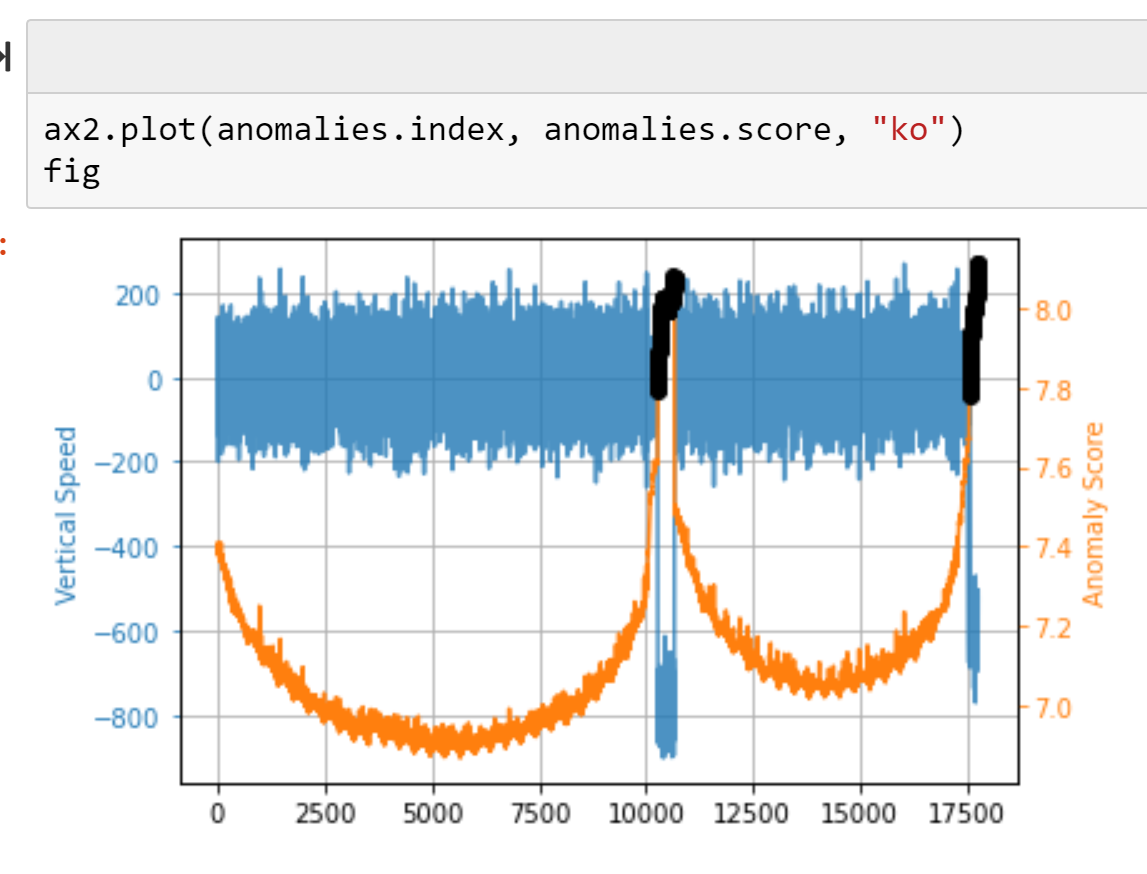
*Figure 4.4 - Variable Plot from Multiple Simulations Loaded at Once*

As we looked into ways to plot the variables in a more intuitive way, we began testing loading the data into a RCF model. We were hoping to train our RCF model on multiple simulations to expectantly improve the algorithms ability to detect normal and abnormal behavior on the dependent variables. Figure 4.5 below shows the head and tail of our dataset that included 2 simulated flights. Again we can see there are two flights in this dataset because the flight\_id starts at flight 0 and ends the file at flight 1 representing the second simulated flight. The first flight, flight\_id = 0, is the flight shown in Figure 4.1 where ‘time\_to\_buffet’ is still 999 seconds and ‘time\_from\_buffet\_to\_uncommanded\_descent’ is still 4 seconds. We can see the second simulated flight, flight\_id = 1, begins to buffet at 659 seconds and starts its uncommanded descent one second after the buffet, instead of four seconds in the first flight.



*Figure 4.5 - Multiple Flight Simulation Dataset*

After a week of testing, we ended up determining that running multiple flight simulations at once through a RCF model would not work. The RCF is best suited for time series data that begins and ends at a specified point. When running two flight simulations through the model at once, we were introducing two different sets of time series dependent data. One flight began at zero seconds and ran until the stall at 1,066 seconds, while the other flight in the dataset began at zero seconds and ran until it stalled around 708 seconds. We can see in Figure 4.6 below how running the two different flights at once produces a confusing and non-intuitive plot. While the model did end up producing and calculating anomaly scores for both flights, the detection time was slower when compared to running one flight simulation through the model at a time. Because of this slower detection time, we decided to end our investigation and research on running multiple flights through our algorithm at once, and focused back on improving our model for each individual flight.

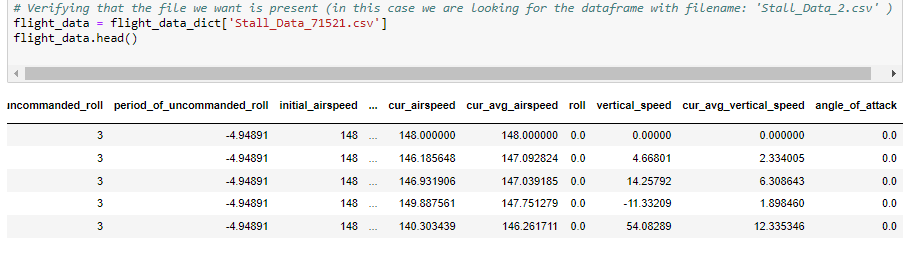
****

*Figure 4.6 - RCF Anomalies from Multiple Flight Simulations*

## DeepAR:

In addition to the Random Cut Forest algorithm, DeepAR was used to prototype and research the effectiveness of a forecasting algorithm for detecting stall events. Similar to the RCF example, a synthetically generated dataset was used to train and test the DeepAR algorithm. Figure 4.6 shows some of the initial rows of the dataset. As indicated earlier, the frequency parameter had to be set to 1 min as its the smallest level of granularity allowed between time steps. Figure 4.7 shows the time configurations used when building the data objects. For example, the start timestamp was decided to be 2019-07-13 00:00:00 with a frequency of 1 min. AWS documentation indicated that it was best practice to start by making the context and prediction windows equal so the context and prediction windows were set to 300 minutes. Initially, the goal was to forecast 30 seconds into the future but due to the timescale adjustment we made, the prediction window was scaled upward to account for the larger time window reflected in our dataset. The current average vertical speed was utilized as the variable to forecast as it is an important determinant of stall events (i.e. sudden downward influxes of vertical speed could indicate the onset of buffet or uncommanded descent).

In order to create the test and training data, the values for current average vertical speed was extracted into a new dataframe with a time index. There were 17747 entries in this data object representing 17747 minutes of data. For the training set, the last 300 minutes of data was withheld to form a training set of 17447 entries. In DeepAR, the entire dataset is used for testing and the algorithm attempts to forecast a certain length of entries (defined by the prediction window hyperparameter) before comparing against the actual values to facilitate backpropagation. The final test dataset was essentially the original current average vertical speed dataset with 10660 entries.

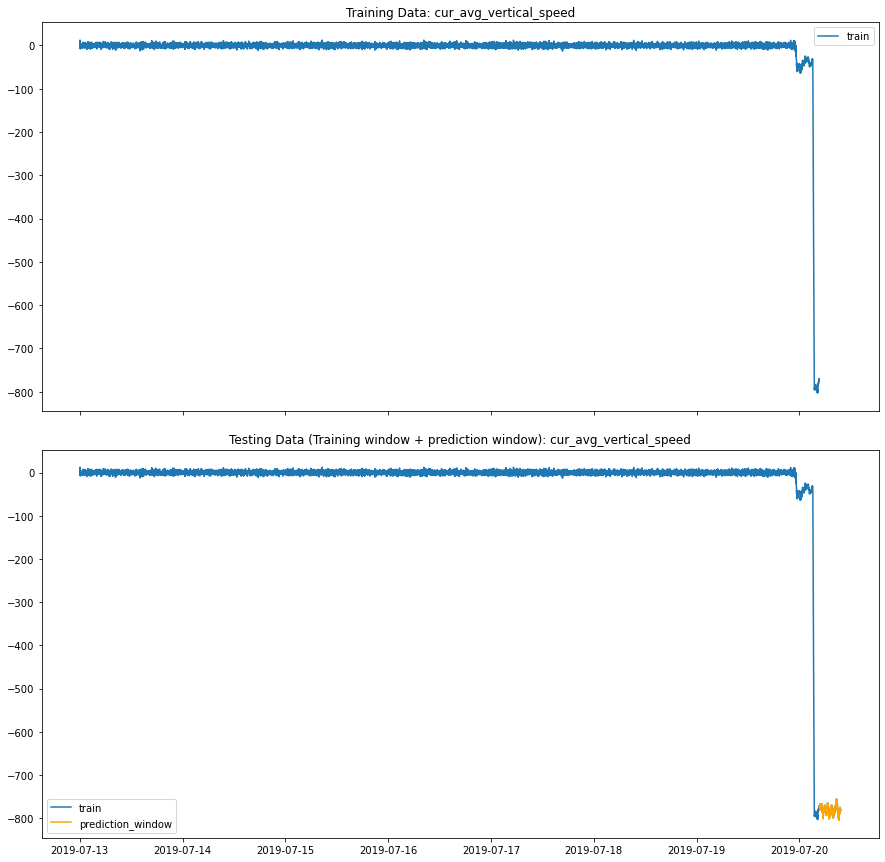


*Figure 4.6 Simulated dataset sample for DeepAR Algorithm*

As indicated earlier, the frequency hyper parameter was set to 1 min. Therefore, when building the test data object, one minute was added to the previous timestamp for every entry in the data frame. Figure 4.8 shows a plot of the training and test sets. The time series data minus the prediction window was used to train DeepAR. The entire dataset including the prediction window was used to test the dataset with the predicted values being compared to the actual values. The blue line signifies the training data and the orange indicates the prediction window data (i.e. the data points that DeepAR is attempting to forecast).



*Figure 4.7. Building Training and Test datasets for DeepAR*



*Figure 4.8 Training and Test dataset plot for DeepAR. The blue line indicates the training data and reflects the entire time window of the dataset minus the 300 minute prediction window. The orange line is what DeepAR is attempting to forecast (i.e. 300 minutes into the future)*

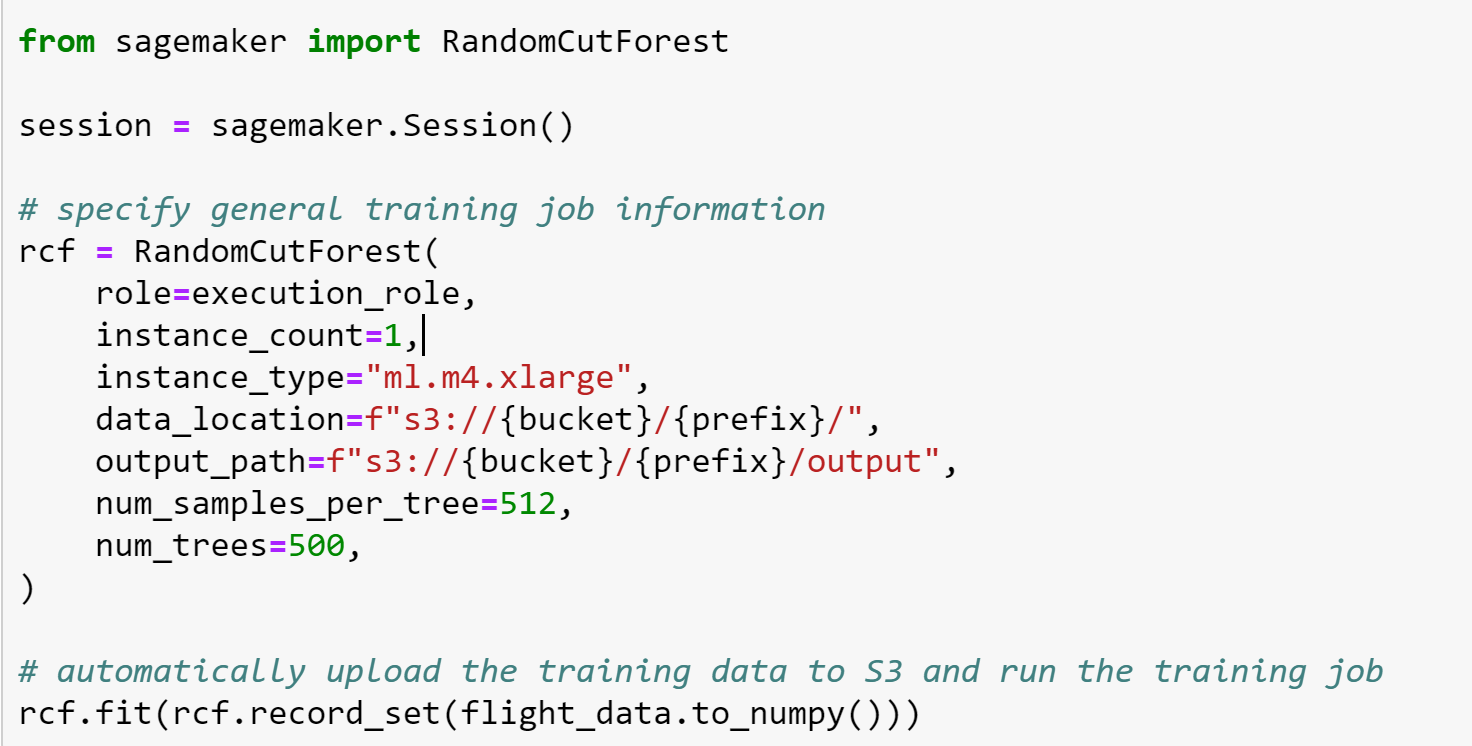
# Findings

## Random Cut Forest:

## Final Algorithm:

The objective of our project was to develop an explainable ML/AI algorithm and model that, based off of input variable data from an aircraft, would be able to detect anomalies in the target variables, such that a warning would be sent to the pilot with enough time for them to perform the maneuvers required to avoid the stall. We determined that a warning time of 10 seconds or greater, before a stall would occur, was needed to have a “successful” model for this project. Our team was able to develop a Random Cut Forest model, with our simulated data, that provides the minimum 10 seconds of warning time a pilot would need to avoid a stall of the aircraft. In our final versions of the model we were able to provide up to 20 seconds of warning time, double what we set out to accomplish.

Figure 5.1.1 below is the final tuned model we ended up with for this project. We did 3 weeks of testing different simulations and different mixes of hyperparameters before arriving at the model seen in Figure 5.1.1. A ‘ml.m4.xlarge’ instance was used in an AWS SageMaker Jupyter notebook due to its high processing power and low cost to run. With the simulated data we generated, and after weeks of testing, we found that a forest of 500 trees, each with 512 samples per tree, was the optimal model. Again Figure 5.1.1 below shows the number of trees in the model, num\_trees = 500, and the number of samples within each tree, num\_samples\_per\_tree = 512.

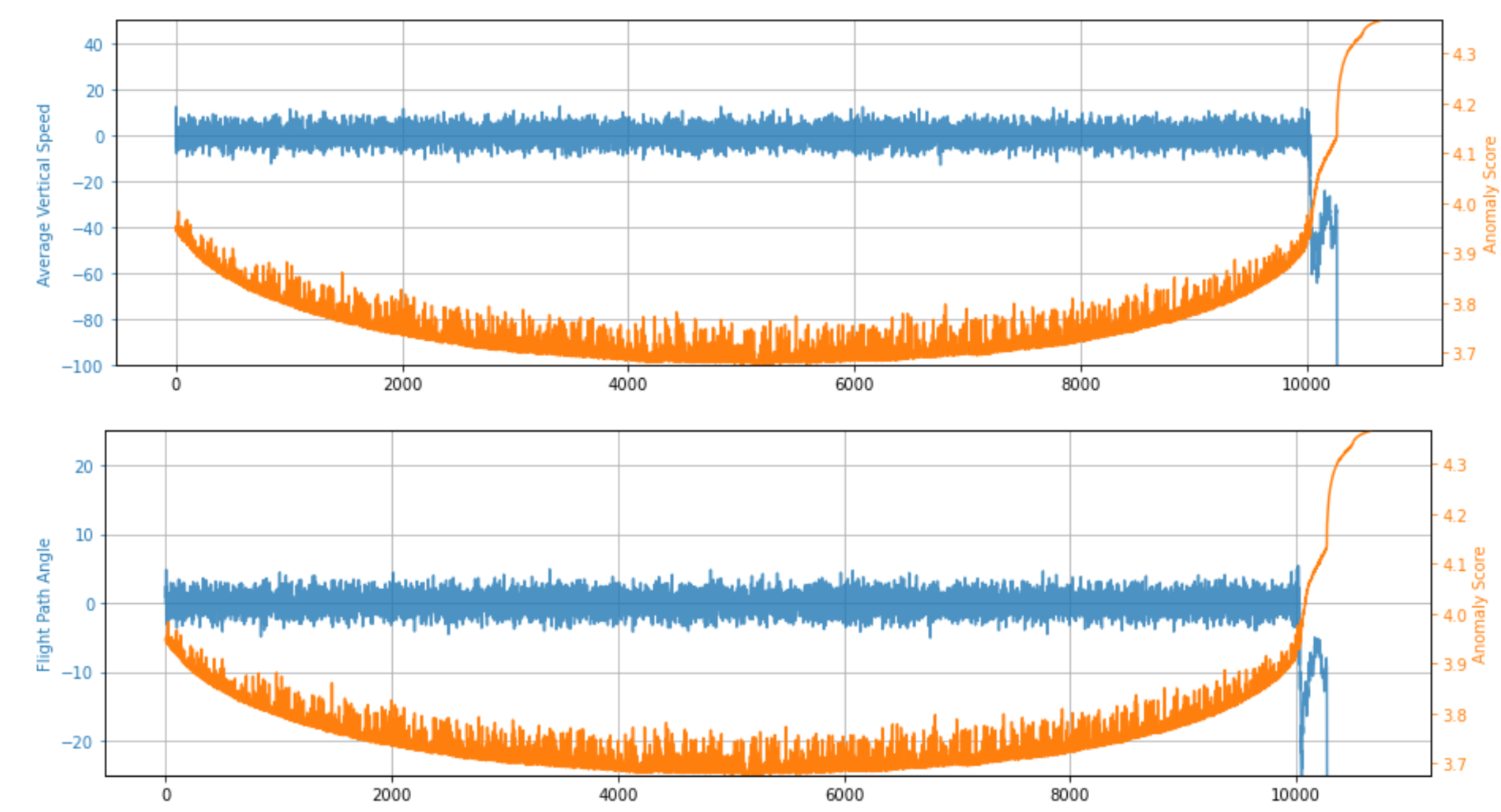


*Figure 5.1.1 - Random Cut Forest Algorithm (Python Code)*

## Algorithm Results:

The RCF model was run with the above mentioned hyperparameters, to produce a score for each row of data in our simulated flight dataset. By nature of the RCF model, all of the variables had their scores calculated together to produce one score for each row of data in the dataset. What this meant was the algorithm was looking at all of the input variables changing as a whole, instead of looking only at specific variables. This could be a real world use case as some airlines or companies may not have the resources or finances to evaluate every input variable from a flight separately, as the cost for doing so would increase exponentially as more variables were added. The scores for the dataset are seen below in Figure 5.1.2.1 in orange, while the ‘Average Vertical Speed’ and ‘Flight Path Angle’ variables are shown in blue. We notice again, that the orange 'Anomaly Score’ line is the same for both variables shown below.

Figure 5.1.2.1 below, allows us to visually see how the model is working with our data. From its inception, the anomaly scores begin around 3.95 and slowly decrease as the RCF begins to understand the noise in the variables is normal behavior and nothing anomalous is going on with the flight. As the flight progresses to around the 600 second mark, the RCF model starts to detect some anomalous behavior, as seen by the increase in orange scores after the 6000 mark on the x-axis, until it jumps to scores over 4.3 towards the end of the simulation as the flight would have been in a stall by that point in time.



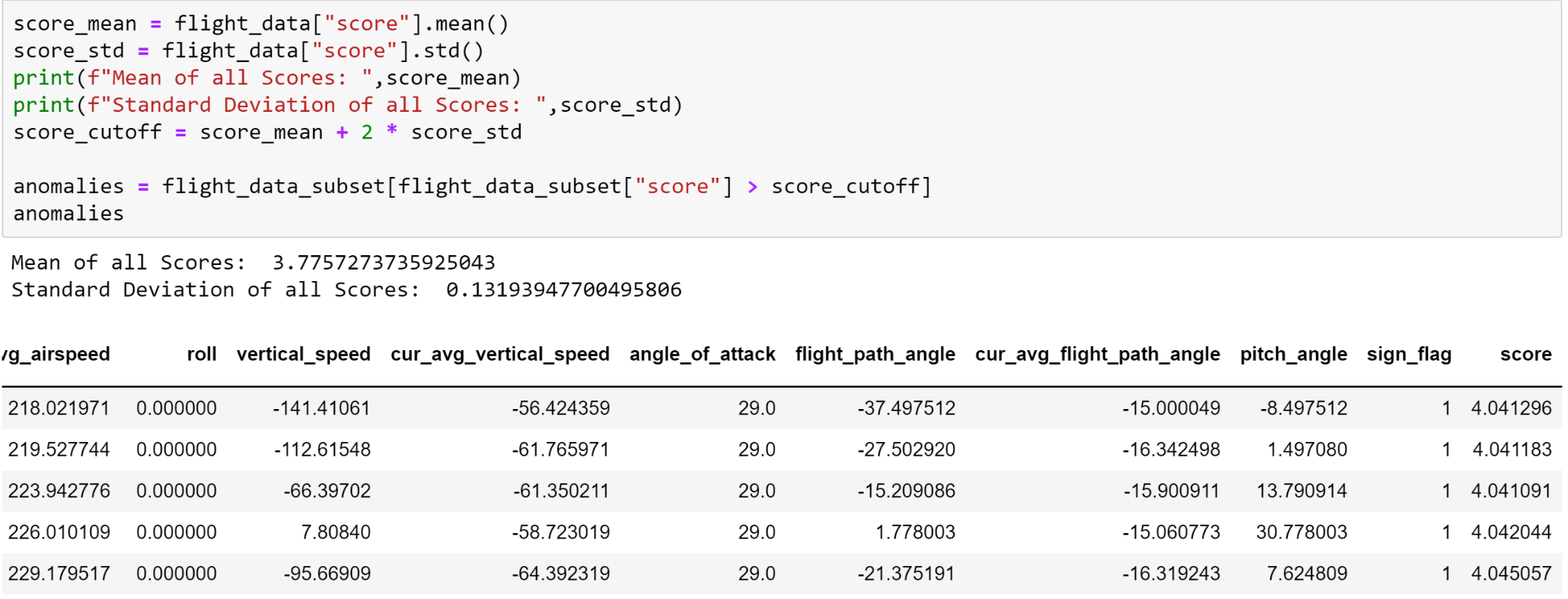
*Figure 5.1.2.1 - Initial RCF Scores for Vertical Speed (VS) and Flight Path Angle (FPA)*

Our model visualizations were encouraging and showed us we had a working algorithm. Next it was time to do some calculations to allow our model to tell us when an anomaly occurred that we should be paying attention to. To get our model to output concerning anomalies, we first had to calculate the mean and standard deviations of all the scores within our dataset. Figure 5.1.2.2 below shows the mean for all the anomaly scores to be ~3.776, and the standard deviation for all the anomaly scores to be ~0.1319. We then determined that an anomaly score, beyond 2 standard deviations (95%+ percentile) from the mean of all the anomaly scores, would constitute a necessary stall warning to the pilot that conditions of the flight were changing and a stall may be forthcoming. So the formula for an anomaly we wanted to catch in our model is seen below:

***Anomaly > 3.776(mean) + 2\*0.1319(standard deviation)***

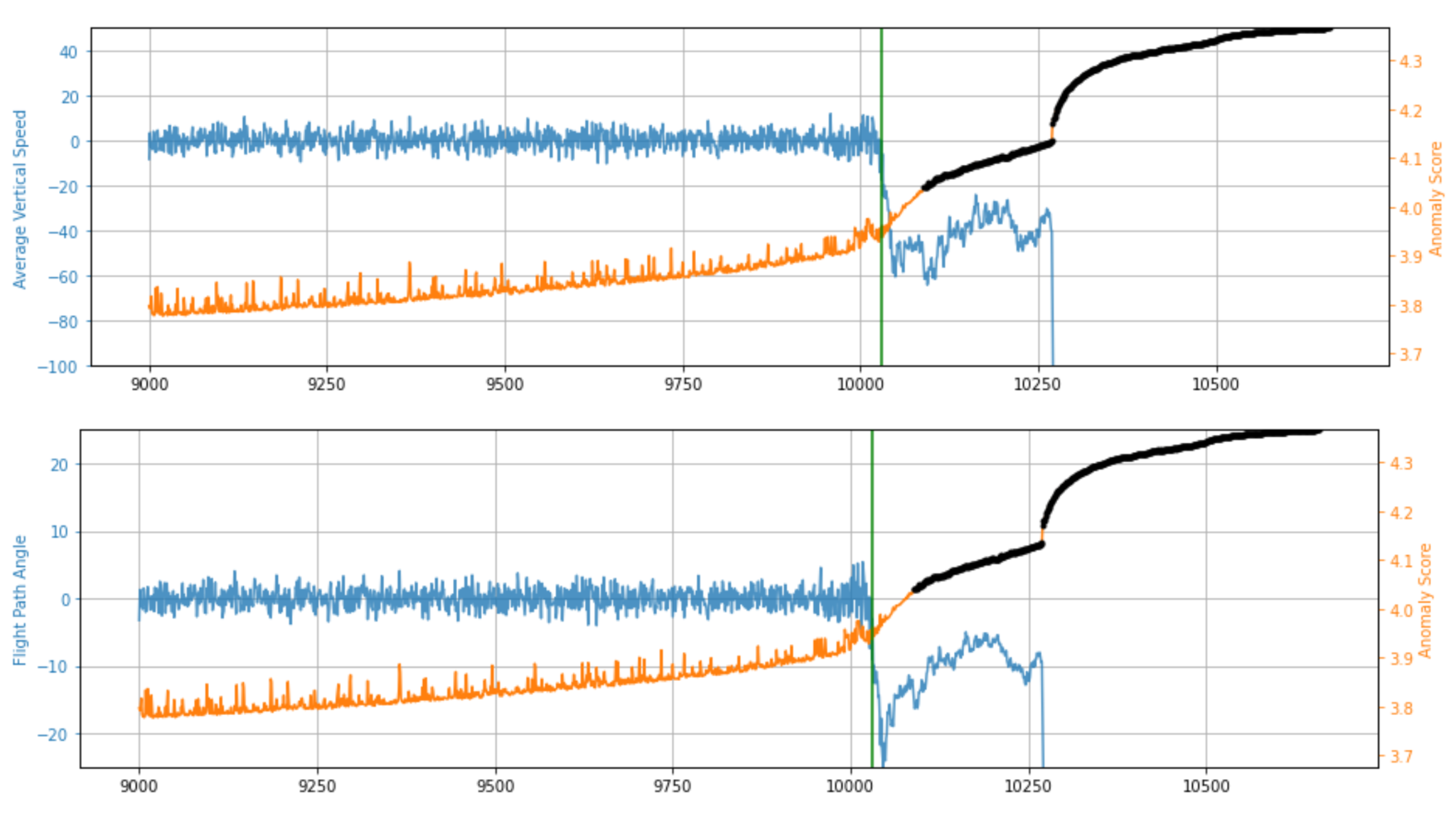
***Anomaly > 4.0399***

We can additionally see, from Figure 5.1.2.2 below, the formula above.The resulting first five rows, that had anomaly scores larger than the threshold of 4.0399, are printed as a check to ensure our model is identifying the correct anomaly threshold.



*Figure 5.1.2.2 - Mean and Standard Deviation for RCF*

Finally, we took the anomaly scores over our threshold of 4.0399, and plotted them against our variables along with all the other anomaly scores. Figure 5.1.2.3 below is our final visualization for our finished RCF model. Note that Figure 5.1.2.3 is shown in 1/10ths of a second on the x-axis, i.e. every increase of 10 on the x-axis represents one second of time for the flight. The vertical green bar in the figure represents the onset of stall conditions, while the steep drop off around 10,260 of the blue lines represent the stall and crash of the aircraft. Each black dot on the orange 'Anomaly Score’ line is a datapoint that has an anomaly score greater than our threshold. We can visually see by this point that our algorithm is detecting the anomalies over our threshold score, 17 to 18 seconds before the stall occurs for this flight. Almost double the necessary warning time we set of 10 seconds.



*Figure 5.1.2.3 - Anomaly Scores for VS and FPA in Black*

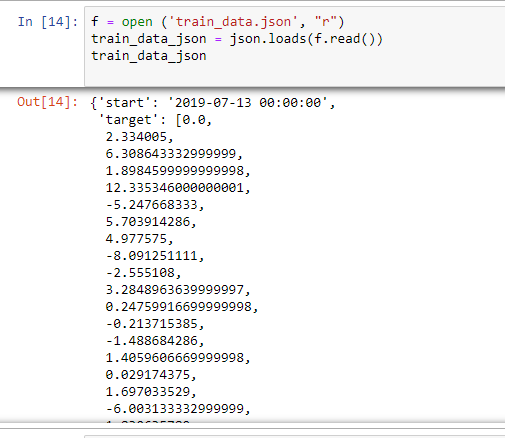
## RCF Model Summary:

The final model we have created using the RCF Algorithm was highly successful. While additional work could have been done to further improve the algorithm and our overall detection time, the final algorithm we produced performed above and beyond the 10 second threshold we set out to accomplish. With this model, or an improved version, we believe it could be used in a real aircraft to save countless lives in the future and avoid aerodynamic stall situations as we know them now.

## DeepAR:

Our objective with DeepAR was more of a research and learning opportunity. Due to the limitations in frequency granularity, the algorithm is not ideal for operational environments where stall events need to be detected in a matter of seconds. From a research perspective, we inferred that trends in the data could be observed - just represented in a large time window. As a result, the forecast window was adjusted from 30 seconds to 300 minutes to account for the larger dataset time window. Additionally, challenges obtaining access to AWS Sagemaker limited our ability to pursue more advanced applications with DeepAR due to our limited time frame. Regardless of these challenges, the model did prove to be a worthwhile investigation as it yielded insight into how forecasting models operate and are configured. Future applications in this domain could utilize this or other forecasting models to detect conditions of stall events occurring in smaller time frames.

After creating a training dataset, the data was saved into an S3 bucket and opened as a JSON object. DeepAR requires the data to be in JSONLines format which consists of a start timestamp (indicated by the start key) and the data entries (indicated by the target key). Figure 5.2.1 shows how the training data is formatted.



*Figure 5.2.1 Creation of Input Data object for DeepAR*

After creating the training object, a DeepAR image was downloaded and the model was trained with the following hyperparameters in Table 5.2.1

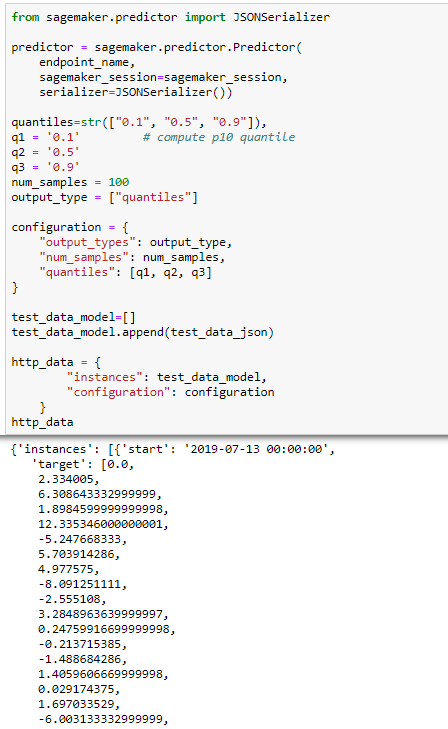
**Table 5.2.1** Hyperparameters for DeepAR Algorithm

|  |  |  |
| --- | --- | --- |
| Hyperparameter | Value | Description |
| context\_length | 300 | Number of time points to look at before making prediction |
| prediction length | 300 | Number of time-steps the model is trained to predict |
| time\_freq | 1 min | Granularity of time series in dataset |
| likelihood | Gaussian | Noise model used for uncertainty estimates |
| num\_cells | 40 | Number of cells to use in each hidden layer of the RNN |
| num\_layers | 3 | The number of hidden layers in the RNN |
| epochs | 20 | Maximum number of passes over the training data. Typical values are between 10 and 1000 |

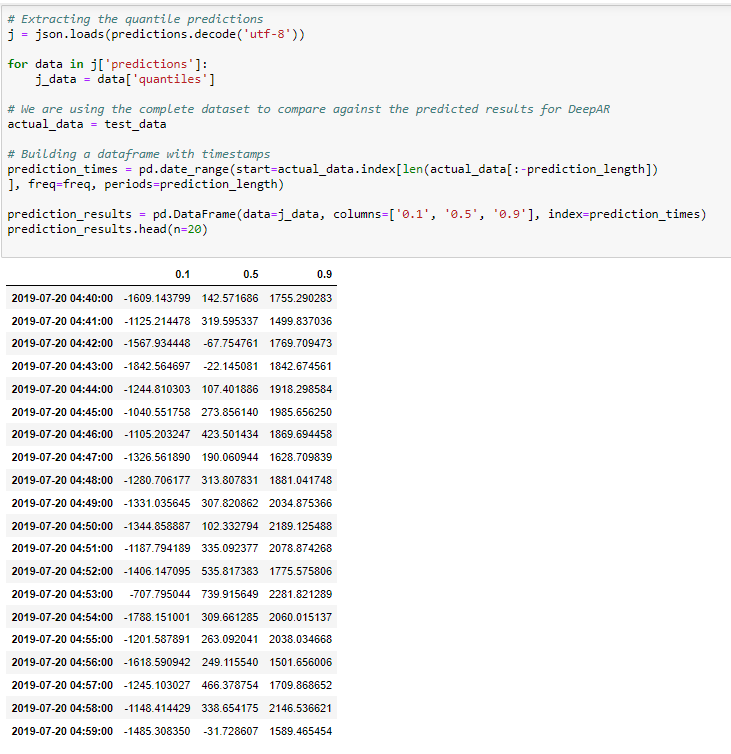
*Definitions source: AWS Sagemaker Documentation*

After training the dataset and deploying an endpoint, a training object (http\_data) was created to pass the training data to the endpoint and return the forecasted results The training object contained both the data and configuration parameters (Figure 5.2.2). . As mentioned earlier, DeepAR returns a probability distribution for each value. Quantiles were specified as the output type with the quantile ranges being specified as [0.1, 0.5, 0.9]. That range communicates to the model to return a distribution with an 80% confidence interval. After receiving the model predictions, a data frame was created using the last timestamp of the test data to report the forecasted values for each quantile at every timestamp in the prediction window (Figure 5.2.3). The resulting forecasted values were plotted against the actual data in a time series plot (Figure 5.2.4). Due to the relatively tight plot, a separate plot showing the predicted values along with the quantile range was developed (Figure 5.2.5)

Overall, Figure 5.2.4 shows oscillation around the 0 value. Since the average vertical speed hovers around the 0 mark the majority of the time, the algorithm is predicting that the vertical speed will continue to spike before gradually returning back to its normal behavior. Since there was a significant decrease in vertical speed around the 20 July 2019 mark, the algorithm could be interpreting that as the start of a notable pattern. Since there are no previous patterns to work off, the algorithm is forecasting a wide prediction range. Since our model is based on a single simulation, the model is limited in its learning context. Since one of the strengths of DeepAR is its ability to form one global mode out of multiple similar time series; it would be beneficial for future renditions of this model to incorporate multiple simulations of the same time series variable. Figure 5.2.5 shows that the prediction mean of the model is hovering between 1000 and -1000 ft/s. Additional tuning of the hyperparameters and adjustment of the quantile range could show a tighter and more refined prediction median.



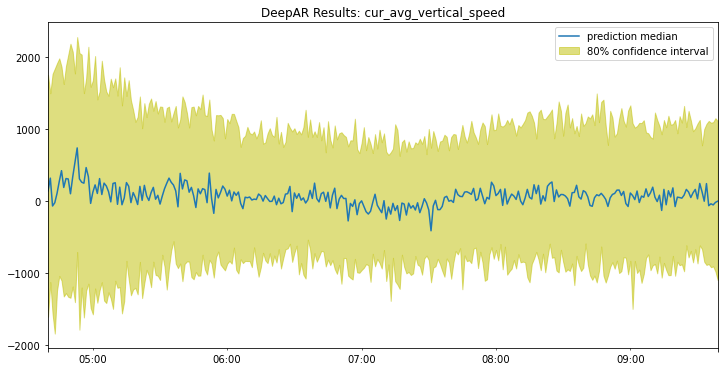
*Figure 5.2.2 Training object creation for DeepAR forecasting*



*Figure 5.2.3 DeepAR forecasted value dataframe per quantile (10th, 50th, 90th percentiles)*



*Figure 5.2.4 DeepAR forecasted value and 80% confidence interval for cur\_avg\_vertical\_speed*



*Figure 5.2.5 DeepAR forecasted value and 80% confidence interval for cur\_avg\_vertical\_speed without the training data*

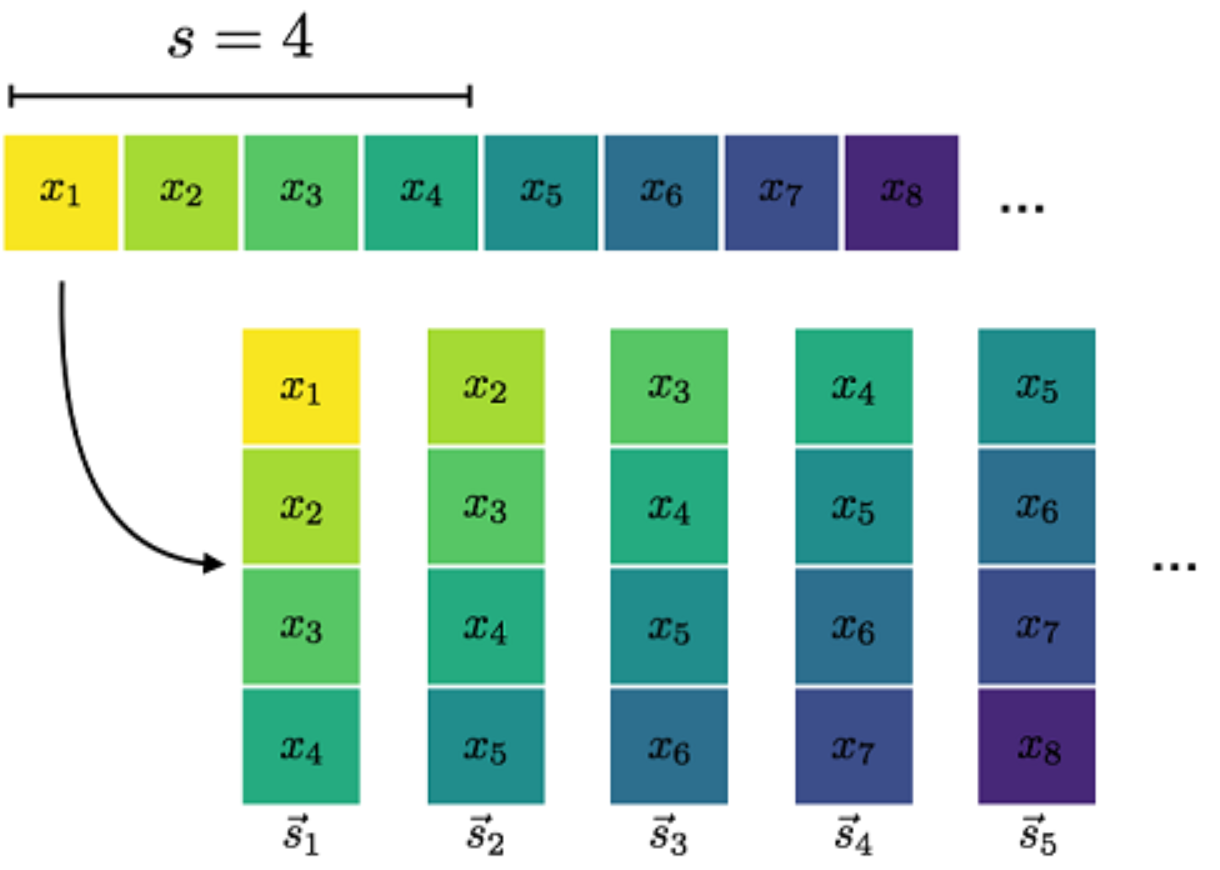
# Summary

The objective of our project was to develop algorithms/methods that anticipate when stall is about to happen based on real-time flight data. Team Machine worked with Sagemaker algorithms such as Random Cut Forest and DeepAR. We determined that Random Cut Forest would ideally be the way to go. DeepAR is mostly forecasting and with that specific algorithm it could only go as granular as minutes. Therefore DeepAR was limited in the amount it could do for us. Random Cut forest is a model used for anomaly detection. When detecting stalls we believe that looking for anomalies is the way to go. As our finding showed we could detect stall with anomaly detection well before the pilot needs to perform corrective maneuvers. In our example the pilot had 17-18 seconds before the plane would stall to perform the necessary actions.

# Future Work

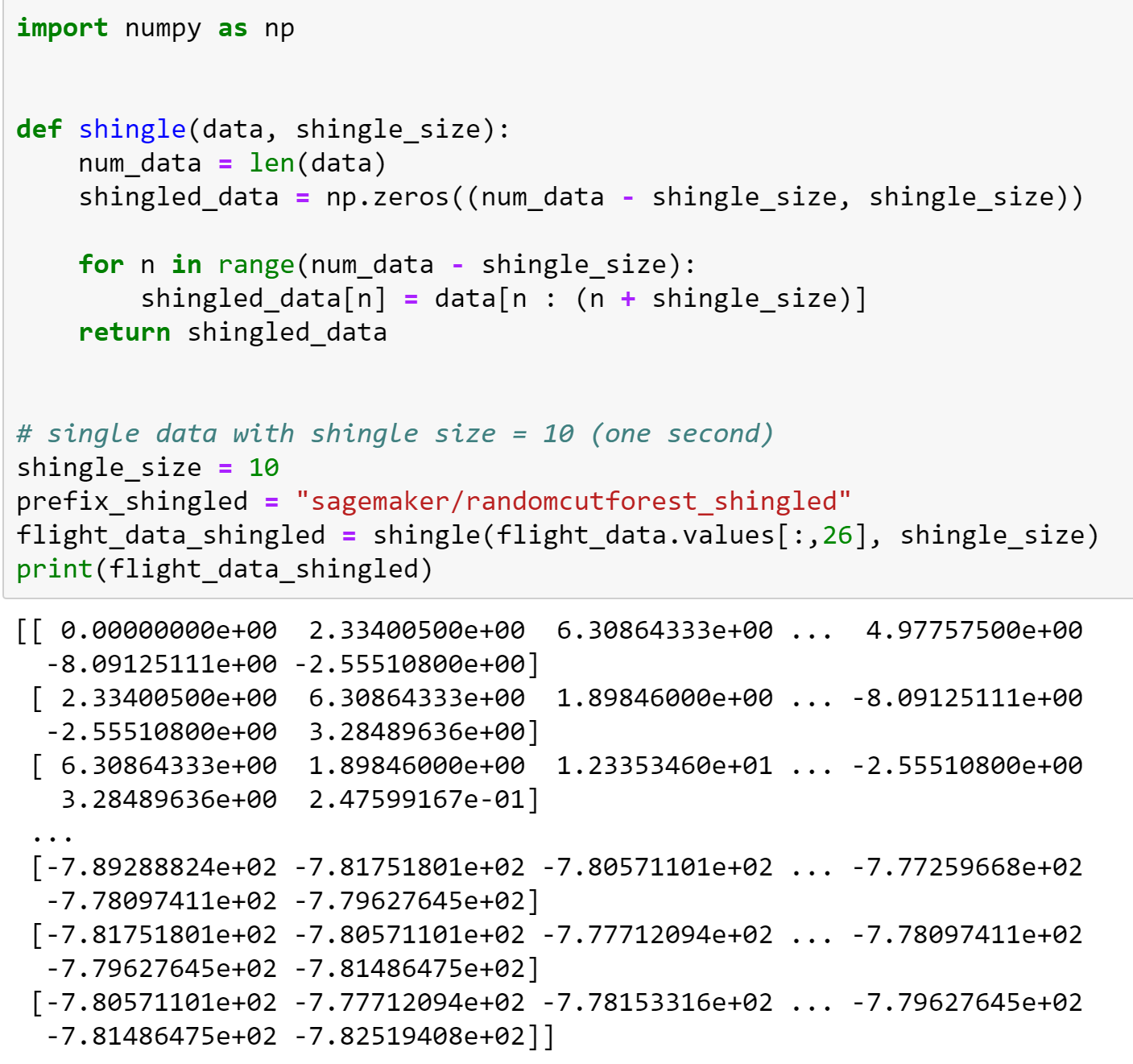
## Random Cut Forest:

In the future with this project, we would like to explore in more depth a technique known as ‘Shingling'' for our RCF model. In this project, the team used a RCF model to detect anomalous data points in our flight dataset. In our dataset, the anomalies occurred when the dependent (or target) flight variables were uncharacteristically high or low. However, the RCF algorithm is also capable of detecting when, for example, data breaks periodicity or uncharacteristically changes global behavior. Shingling is especially useful when working with periodic data with a known time period, like we have in this project. The idea of shingling is to treat a period of **𝑃** data points as a single datapoint of feature length **𝑃** and then run the RCF algorithm on these feature vectors. Figure 7.1 below demonstrates how single dimensional data can be transformed into four-dimensional shingles.



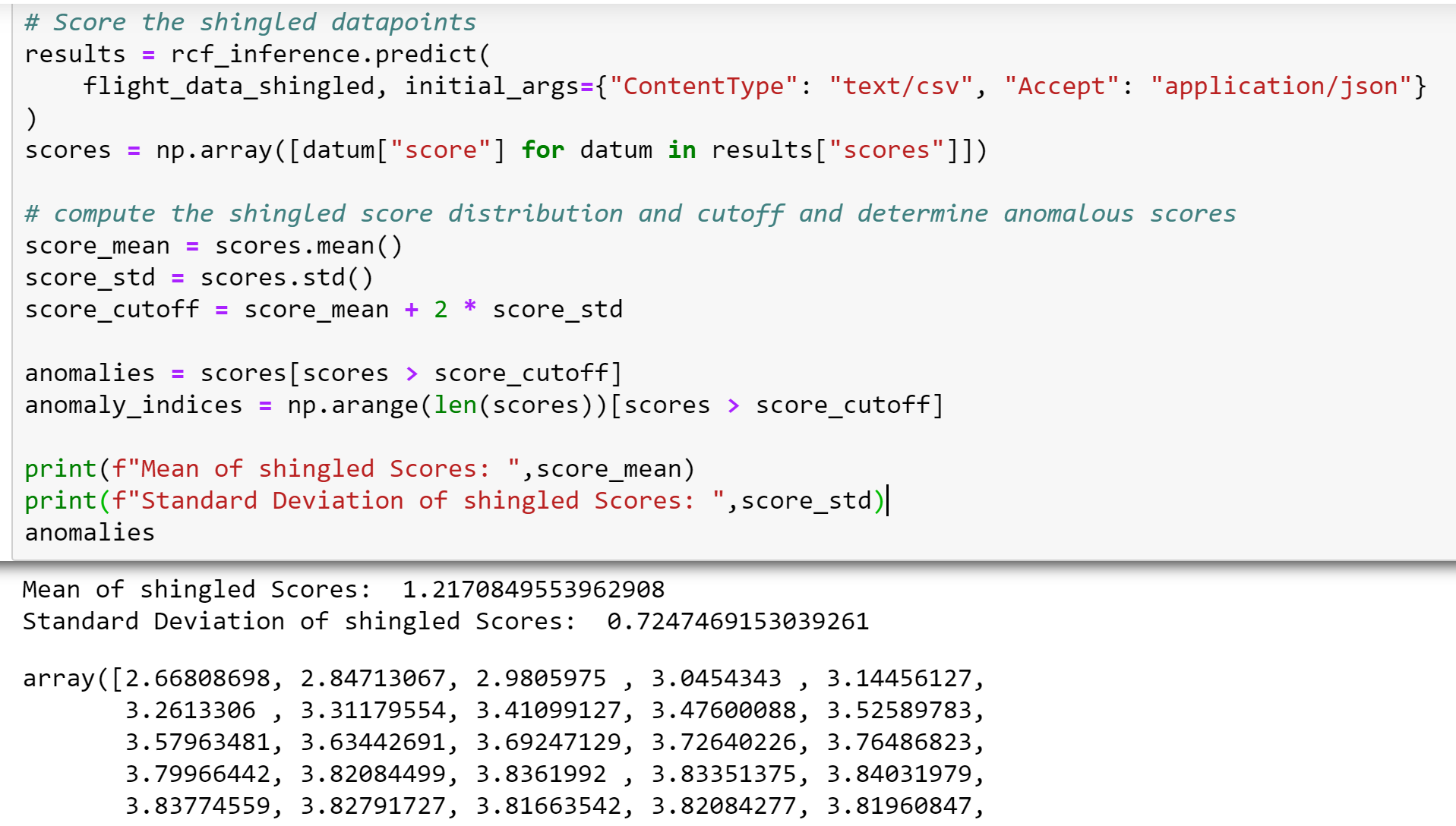
*Figure 7.1 - Example of Shingling a Dataset*

With the limited time and resources the team had for this project we were only able to do a small amount of work with shingling our data. Shingling only seemed to work on our dataset when we specified one variable to shingle at a time. Figure 7.2 below shows how we used shingling on the variable ‘Vertical Speed’, seen as flight\_data.values[:,26], where the 26 indice is the column for ‘Vertical Speed’.



*Figure 7.2 - Shingling One Variable from Flight Simulation (Vertical Speed)*

We then followed the same steps we took in our final RCF model (Figure 5.1.2.2) of calculating the total mean anomaly score and total standard deviation of the anomaly scores for the ‘Vertical Speed’ variable. With all the same methodology in our final RCF algorithm, Figure 7.3 below highlights the mean anomaly score, the standard deviation for the anomaly scores, and the top few records where the anomaly score is larger than our threshold formula. The threshold formula is the same as it was in 5.1.2 above, just with the new mean and standard deviations.



*Figure 7.3 - Shingled Vertical Speed Mean and Standard Deviation*

Finally we see the end visual results of our initial shingling of the “Vertical Speed” variable. We can see below in Figure 7.4 that the first instance (black dot) of an anomaly greater than two standard deviations from the mean score is much sooner compared to the first instance seen in Figure 5.1.2.3. Using the shingling method added an additional 7 seconds of detection time for this variable. That is over a 40% increase in detection time compared to our baseline RCF model, a huge improvement. Future work should explore the foundation of shingling this project has built to see if further shingling and model creation improves detection times as well as it did in the one test case shown in this project.



*Figure 7.4 - Shingled Anomaly Scores for Vertical Speed*

## DeepAR:

DeepAR is a powerful tool to identify trends and forecast time series data. Its ability to train on multiple time series examples, identify common trends, and forecast enables it to handle use cases with large simulations of data. Unfortunately, due to the time limitations of our project, we were unable to further experiment with DeepAR’s tuning parameters and other configuration options. Additionally, the current allowed levels of granularity limits its applicability for detecting aerodynamic stall due to the time steps being too large to detect important fluctuations in data. While the algorithm itself may be limited, recurrent neural networks are a possible avenue for future efforts in this domain.

Algorithms such as Long-Term-Short-Memory (LSTM), which DeepAR references as a means of inspiration for its architecture, might be a valuable algorithm for future teams to consider. Additionally, our end model was configured to train on a single simulation. While the base code does enable some support for multiple simulations of a single time series variable, time limitations in our project prevented us from experimenting with multiple simulations. Future renditions of the algorithm would expand the number of simulations, utilize Sagemaker’s built-in hyperparameter tuning capabilities, and investigate workaround solutions for the time granularity issue. Overall, our experiences with DeepAR was a valuable learning experience in how forecasting methods can apply in detecting stall events.

# 

# Appendix A- GitHub Links

Data Creation:

<https://github.com/ryanjake818/Explainable-Stall-AI/tree/main/FlightGeneration>

Deep AR:

<https://github.com/ryanjake818/Explainable-Stall-AI/tree/main/Algorithms/DeepAR>

Random Cut Forest:

<https://github.com/ryanjake818/Explainable-Stall-AI/tree/main/Algorithms/RCF>

# 

# Appendix B- Risk Analysis and Mitigation

For our Risk Process we used the Department of Defense’s Risk Analysis process. This process uses a 5 by 5 matrix comparing consequence (Table B.1) vs. likelihood (Table B.2) according to the following rubric (Kendall 2017, 23-28).

|  |  |
| --- | --- |
| Table 8.1:Risk Consequence | Table 8.2: Risk Likelihood |
|  |  |

Using this process, we tracked the following risks and mitigations (Figure B.1). Risk #1 was related to the dataset. While missing the full dataset could be bad, with the formulas provided to us by Dr. Sherry this can be mitigated. Risk 2 related to our algorithm can be mostly mitigated by testing/retraining, but there’s still some likelihood it doesn’t get good enough for approval. Risk 3 can be fully mitigated with proper training or creation of our own algorithms.

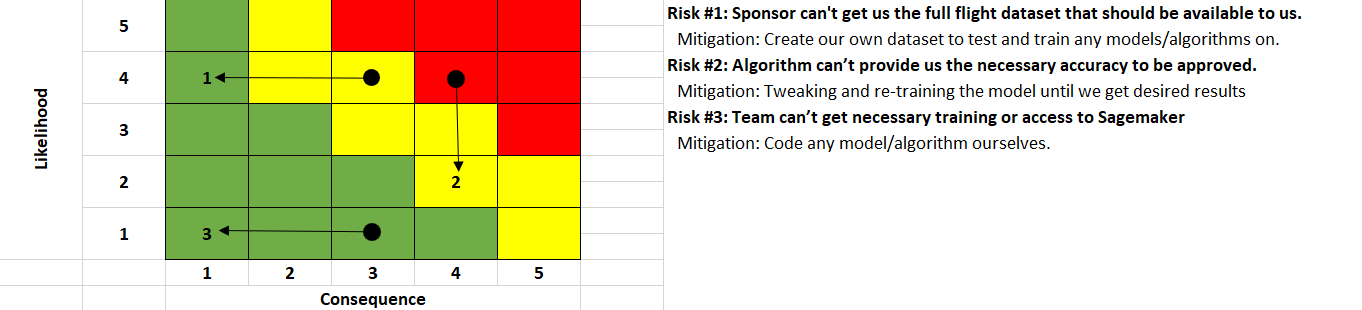
****

Figure B.1: Risk Matrix

# Appendix C- Agile Process

Our team implemented a standard agile methodology throughout the lifespan of this project and each team member had a designated role including scrum master, product owner, and developer. We had two weekly meetings, one with the professor to update and think through next steps and another with our partner Dr. Sherry. Both of these meetings provided feedback to the work we had already completed, provided chances to resolve issues and clarify uncertainty, as well as plan future steps on how we would move forward.

The software used to facilitate this Agile Development was YouTrack. Through this software we created cards for specific tasks, updated our current status, and added notes as we progressed through the tasks. This process allowed our team to have a clear line of communication on who was assigned to each task. We also used these cards to provide updates during our weekly meetings to show what tasks were still pending and which had been completed.

# 

# Appendix D- Definition of Terms

* [**Stall**](https://www.skybrary.aero/index.php/Stall) is defined as a sudden reduction in the lift generated by an aerofoil when the critical angle of attack is reached or exceeded.
* An **aerodynamic twist** can be introduced to the wing with the leading edge near the wing tip twisted downward. This is called **washout** and causes the [wing root](https://en.wikipedia.org/wiki/Wing_root) to stall before the wing tip. This makes the stall gentle and progressive. Since the stall is delayed at the wing tips, where the [ailerons](https://en.wikipedia.org/wiki/Aileron) are, roll control is maintained when the stall begins.
* A [**stall strip**](https://en.wikipedia.org/wiki/Stall_strip) is a small sharp-edged device that, when attached to the leading edge of a wing, encourages the stall to start there in preference to any other location on the wing. If attached close to the wing root, it makes the stall gentle and progressive; if attached near the wing tip, it encourages the aircraft to drop a wing when stalling.
* A [**stall fence**](https://en.wikipedia.org/wiki/Wing_fence) is a flat plate in the direction of the [chord](https://en.wikipedia.org/wiki/Chord_(aircraft)) to stop separated flow progressing out along the wing.
* [**Vortex generators**](https://en.wikipedia.org/wiki/Vortex_generator), tiny strips of metal or plastic placed on top of the wing near the leading edge that protrude past the [boundary layer](https://en.wikipedia.org/wiki/Boundary_layer) into the free stream. As the name implies, they energize the boundary layer by mixing free stream airflow with boundary layer flow thereby creating vortices, this increases the [momentum](https://en.wikipedia.org/wiki/Momentum) in the boundary layer. By increasing the momentum of the boundary layer, airflow separation and the resulting stall may be delayed.
* An **anti-stall strake** is a [leading edge extension](https://en.wikipedia.org/wiki/Leading_edge_extension) that generates a [vortex](https://en.wikipedia.org/wiki/Vortex) on the wing upper surface to postpone the stall.
* A [**stick pusher**](https://en.wikipedia.org/wiki/Stick_pusher) is a mechanical device that prevents the pilot from stalling an aircraft. It pushes the elevator control forward as the stall is approached, causing a reduction in the angle of attack. In generic terms, a stick pusher is known as a *stall identification device* or *stall identification system*.
* A [**stick shaker**](https://en.wikipedia.org/wiki/Stick_shaker) is a mechanical device that shakes the pilot's controls to warn of the onset of stall.
* A **stall warning** is an electronic or mechanical device that sounds an [audible warning](https://en.wikipedia.org/wiki/Buzzer) as the stall speed is approached. The majority of aircraft contain some form of this device that warns the pilot of an impending stall. The simplest such device is a *stall warning horn*, which consists of either a [pressure](https://en.wikipedia.org/wiki/Pressure) [sensor](https://en.wikipedia.org/wiki/Sensor) or a movable metal tab that actuates a [switch](https://en.wikipedia.org/wiki/Switch), and produces an audible warning in response.
* An **angle-of-attack indicator** for light aircraft, the "AlphaSystemsAOA" and a nearly identical "**Lift Reserve Indicator**", are both pressure differential instruments that display margin above stall and/or angle of attack on an instantaneous, continuous readout. The General Technics CYA-100 displays true angle of attack via a magnetically coupled vane. An AOA indicator provides a visual display of the amount of available lift throughout its slow speed envelope regardless of the many variables that act upon an aircraft. This indicator is immediately responsive to changes in speed, angle of attack, and wind conditions, and automatically compensates for aircraft weight, altitude, and temperature.
* An **angle of attack limiter** or an "alpha" limiter is a flight computer that automatically prevents pilot input from causing the plane to rise over the stall angle. Some alpha limiters can be disabled by the pilot.

# Appendix E - Works Cited

Amazon AWS. 2021. Amazon Sagemaker Developer Guide - DeepAR Forecasting Algorithm. Amazon. <https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html>

Arigoni, Alberto. “Paper review and code: Amazon DeepAR”. (2018). Medium. <https://medium.com/@albertoarrigoni/paper-review-code-amazon-deepar-809938a319d9>

Donges, Niklas. 2019. A Guide to RNN: Understanding Recurrent Neural Networks and LSTM. Builtin.com <https://builtin.com/data-science/recurrent-neural-networks-and-lstm>

Engdahl, S. (2008). Blogs. Amazon. <https://aws.amazon.com/blogs/machine-learning/use-the-built-in-amazon-sagemaker-random-cut-forest-algorithm-for-anomaly-detection/>

IBM Cloud Education. 2020. Recurrent Neural Networks. IBM.com <https://www.ibm.com/cloud/learn/recurrent-neural-networks>

Kendall, Frank. 2017. "Department of Defense Risk, Issue, and Opportunity Management Guide for Defense Acquisition Programs“. Washington DC: Office of the Deputy Assistant Secretary of Defense for Systems Engineering, January 9. <https://www.dau.edu/tools/Lists/DAUTools/Attachments/140/RIO-Guide-January2017.pdf>

Mishra, A. (2019). Machine learning in the Aws Cloud: Add intelligence TO Applications with AMAZON SageMaker and Amazon Rekognition. Amazon. <https://docs.aws.amazon.com/sagemaker/latest/dg/rcf_how-it-works.html>

Nadipalli, R. (2017). Effective business intelligence with QUICKSIGHT: From data to actionable business insights using Amazon QuickSight! Amazon. <https://docs.aws.amazon.com/quicksight/latest/user/how-does-rcf-detect-anomalies.html>

Salinas, David, Flunkert, Vaelntin, Gasthaus, Jan. (2017). DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. Amazon Research. <https://arxiv.org/abs/1704.04110>

Wilt, L. Chief Data Scientist, L. (n.d.). Robust random Cut Forest (RRCF): A no math explanation. LinkedIn. <https://www.linkedin.com/pulse/robust-random-cut-forest-rrcf-math-explanation-logan-wilt>