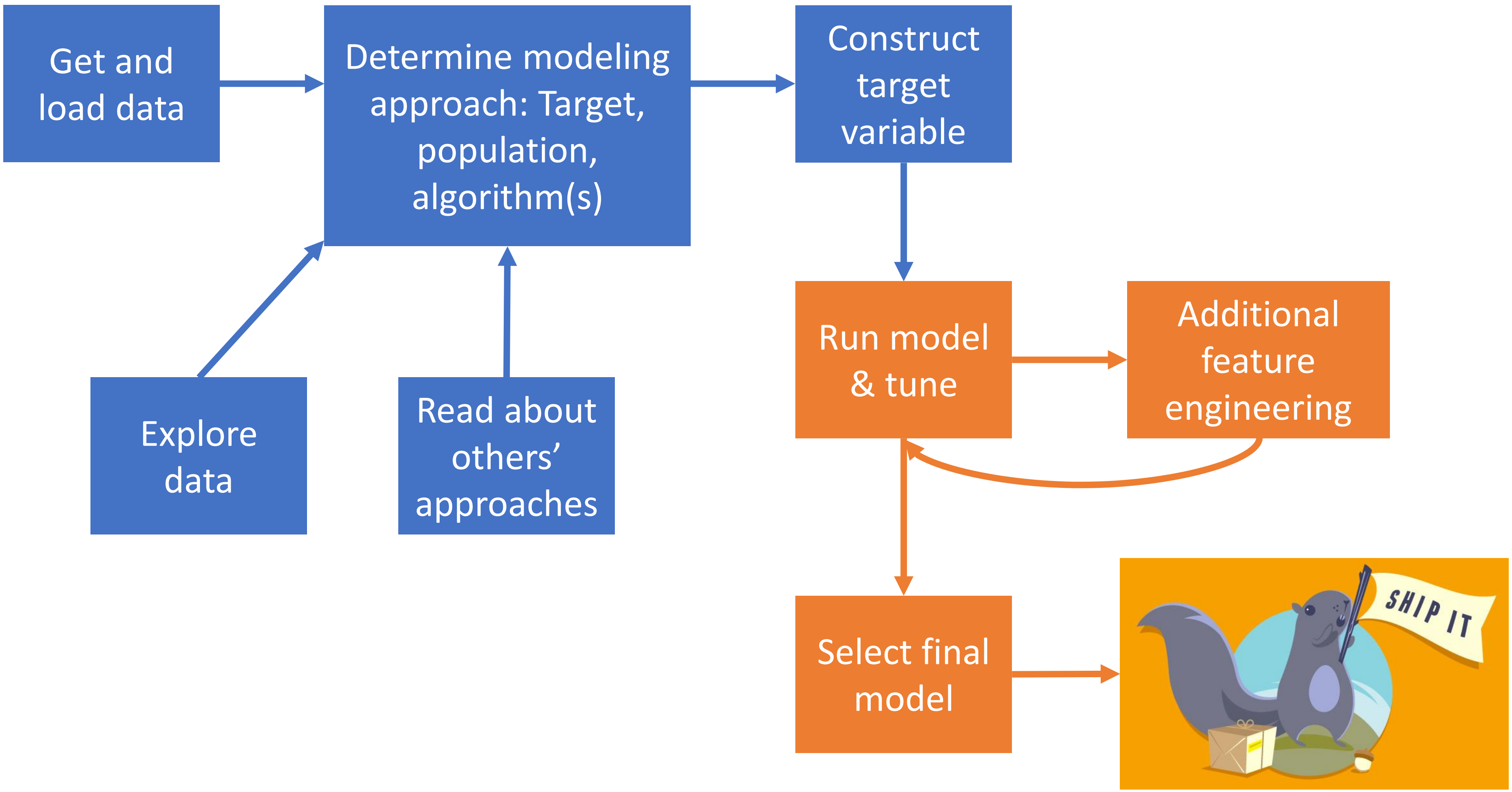
Charles Hammell, Nelson Turbo CEO

Our approach



Final Sagemaker workflow

Load CMAPSS engine data → Data explor-ation and cleansing → Feature engin-eering → Partition data → Define training job and hyper-parameter tuning job → Create endpoint with winning model → Score data via endpoint

Data

Data set overview

Four data sets

	1 Operating Condition	6 Operating Conditions
HPC Degradation Failure	**FD001**	**FD002**
HPC Degradation and Fan Degradation	**FD003**	**FD004**

We chose to focus on HPC Degradation only, so we used data sets FD001 and FD002.

Training data format

Sensor readings per engine/cycle

Engine	Time in Cycles	Sensor 1 Reading	…	Sensor 26 Reading
1	**1**	34.99	…	1.02
1	**2**	41.67	…	1.02
…	**…**	…	…	…
1	**143**	63.12	…	1.26

In Train, there is one record per cycle until failure.

Test set is same format, except data for each unit truncates at a mystery point in time before failure.

Target definition

In Train data, we calculated for engine *i* in time period *t*:

Cycles until Failure$_{it}$
= (Max cycle$_{i}$) - (Current cycle)$_{it}$

Why XGBoost?

Can be used for regression-type prediction

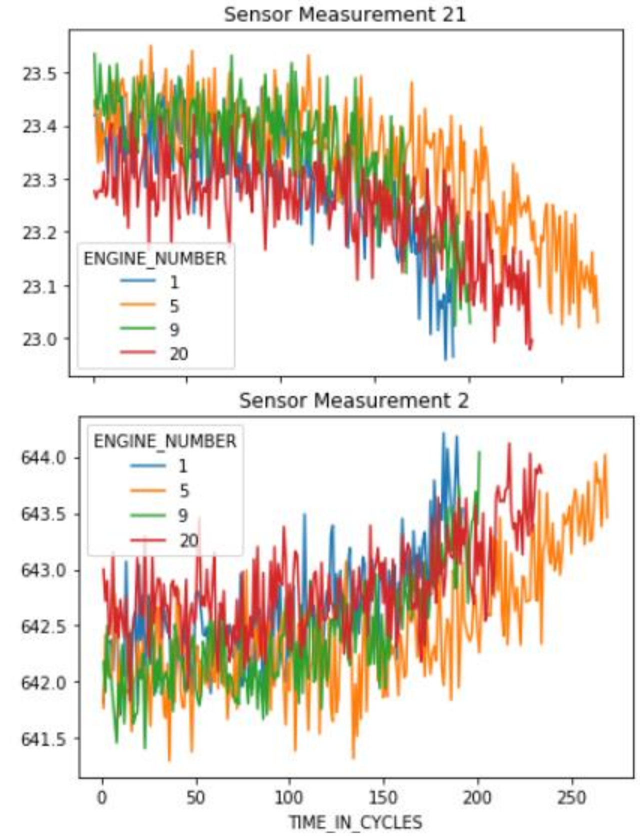
Not much needed for data prep – doesn’t need standardizing, and our data was already all numeric

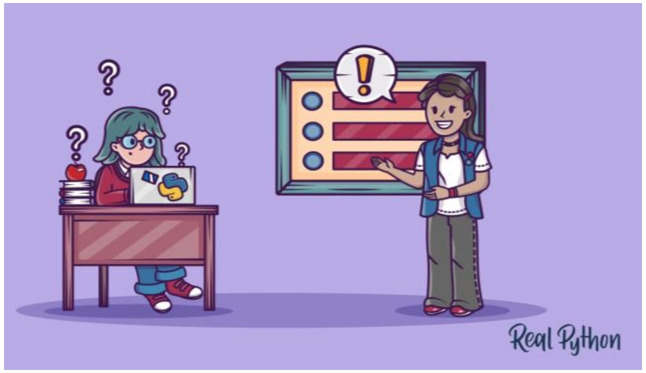
Robust to outliers

Speedy

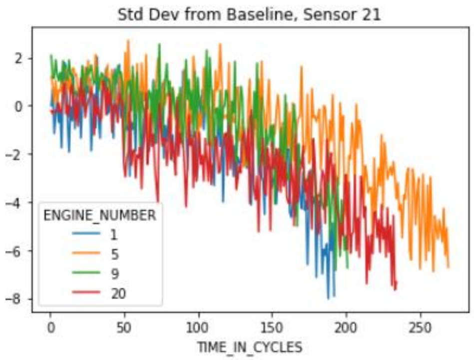
Data exploration

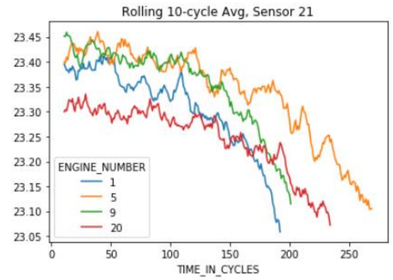
- Sensor readings gradually increase or decrease over time as they approach failure
- The readings are very noisy

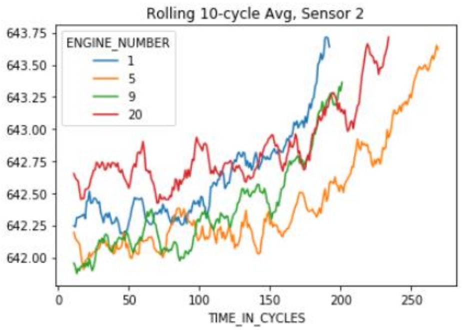


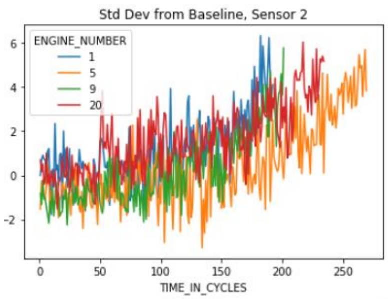


Feature engineering



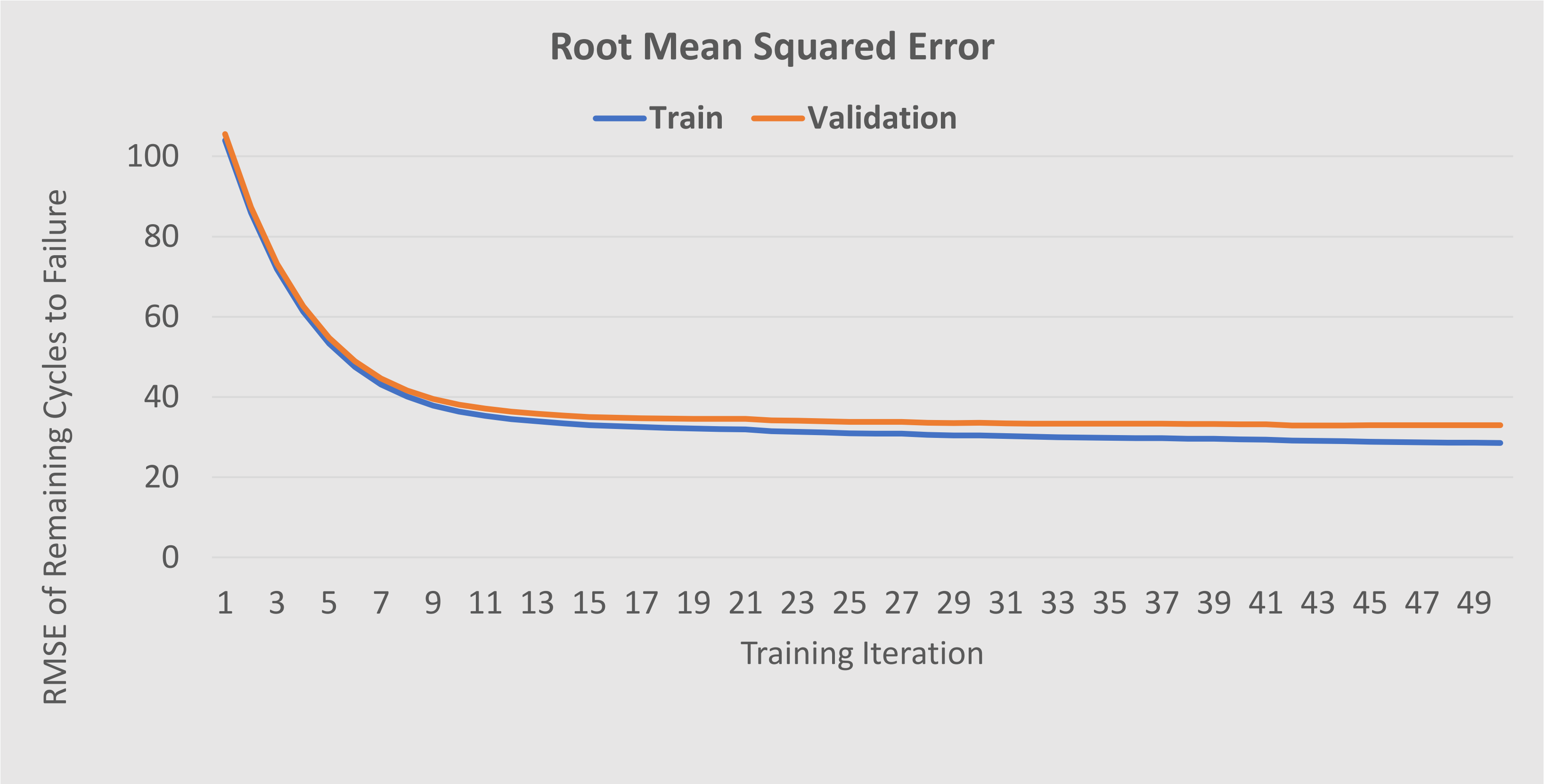






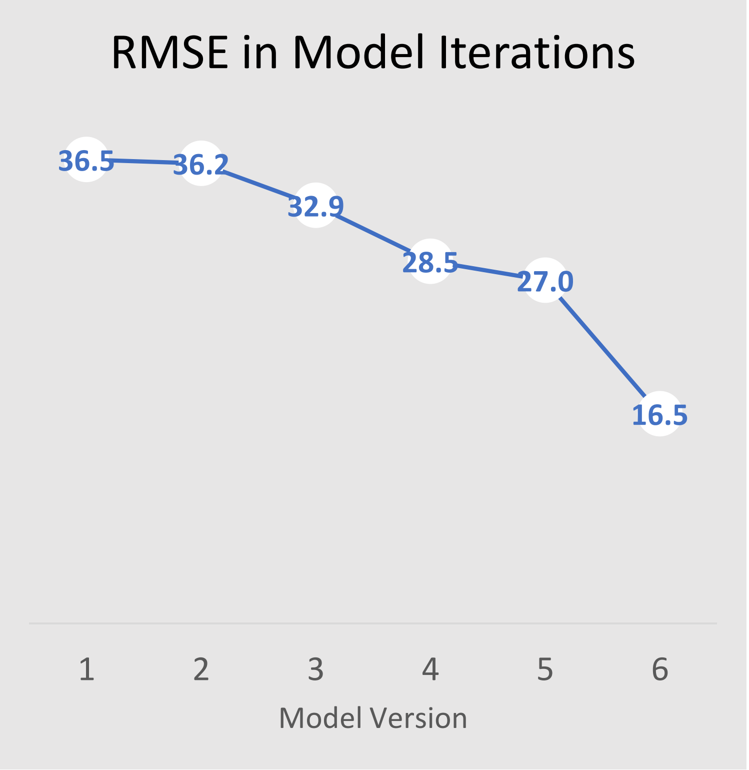
- Rolling averages of sensor readings
- Standard deviations from an initial baseline, specific to each engine
- Rolling averages of standard deviations from initial baseline

XGBoost 1st training evaluation



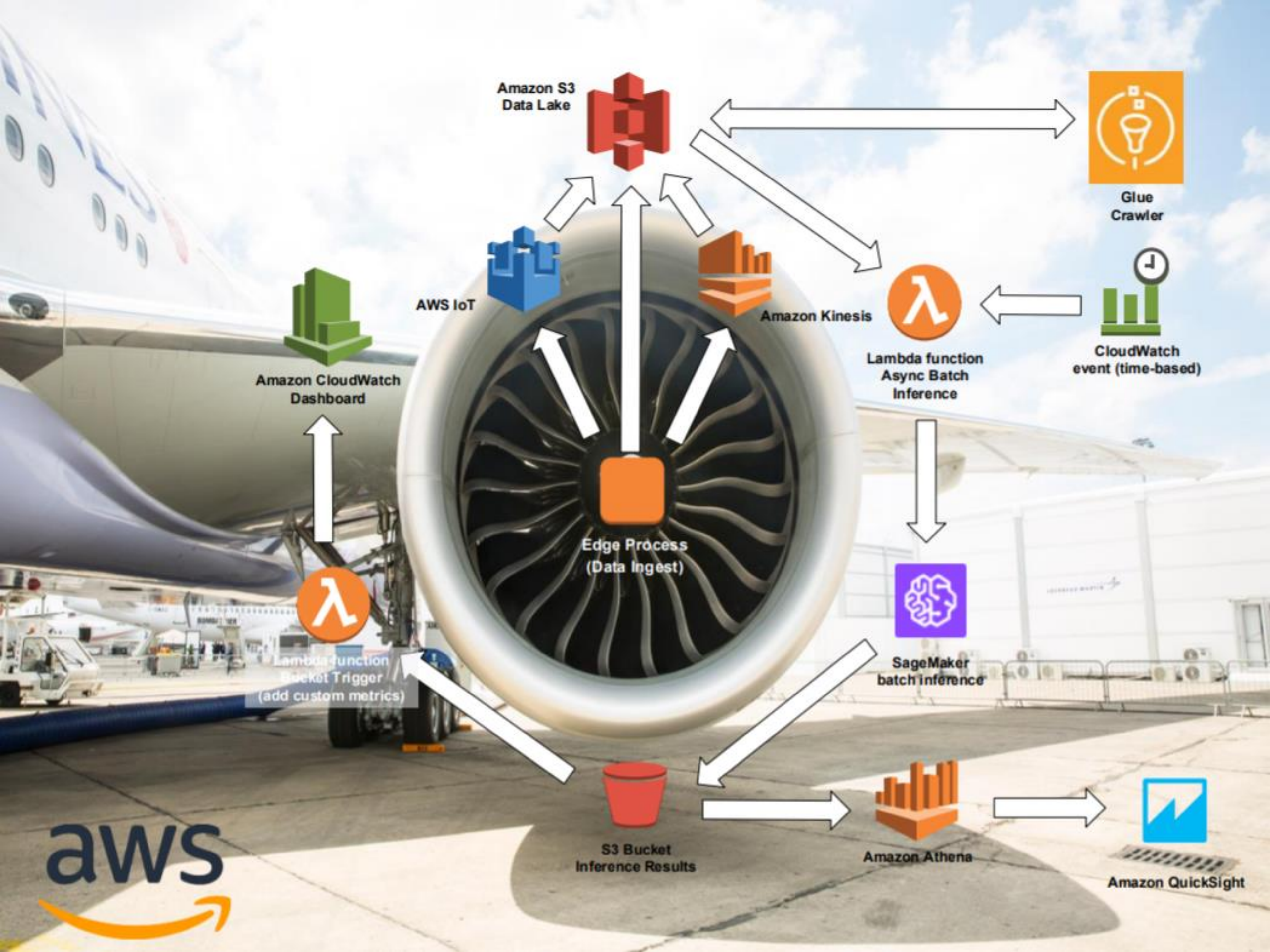
Iterative improvement through feature engineering + hyperparameter tuning

Model Version	Model Description	Validation RMSE for Dataset FD001
1	Limited feature engineering, No hyperparameter tuning	**36.5**
2	Limited feature engineering, With hyperparameter tuning	**36.2**
3	Feature engineering round 1 (std deviation from baseline)	**32.9**
4	Feature engineering round 2 (rolling avg sensor readings)	**28.5**
5	Feature engineering round 3 (rolling avg of std deviation from baseline)	**27.0**
6	With hyperparameter tuning	**16.5**



“If we had time…”

Additional feature engineering	There are papers with some cool feature engineering to identify signals in each engine’s sensor data over time
Clustering of similarly behaving engine units	Can be used to further segment population
	Or as an input in predictive model
Try alternate populations	Can we successfully combine our models for the “one condition” and “six condition” data sets?
Try alternate target definitions	We completely ignored Fan Degradation failure type.
	Try separate models for each failure type, compare to a combined model.
Try alternate modeling algorithms	Linear Learner, etc.
Try alternate evaluation metric	It may make sense to train to minimize Mean Absolute Percent Error (MAPE) instead of RMSE, because as you get closer to failure, a small number of cycles off is more of a problem than when you’re farther from failure.
Further evaluation of our residuals (errors)	To diagnose where we perform well/poorly


Amazon S3
Data Lake
Glue
Crawler
AWS IoT
Amazon Kinesis
Lambda function
Async Batch
Inference
CloudWatch
event (time-based)
Amazon CloudWatch
Dashboard
Edge Process
(Data Ingest)
Lambda function
Bucket Trigger
(add custom metrics)
SageMaker
batch inference
S3 Bucket
Inference Results
Amazon Athena
Amazon QuickSight
aws