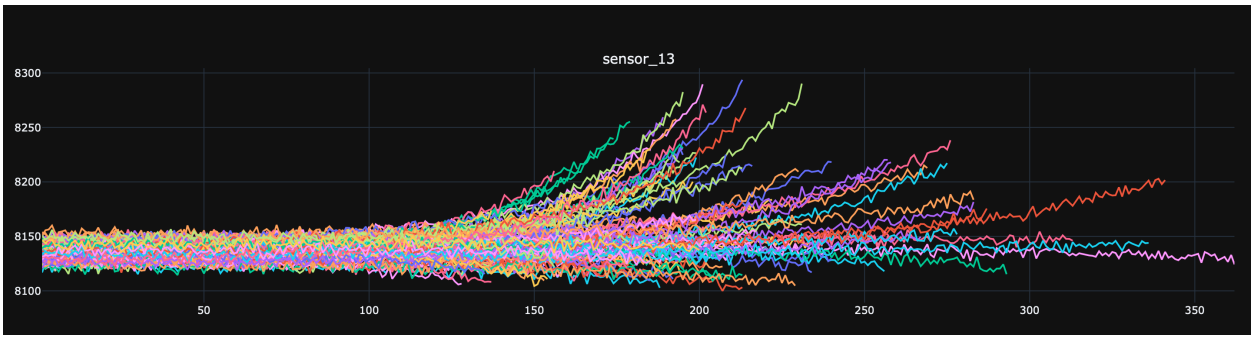
Hot Finishing Mill Days Until Failure Prediction

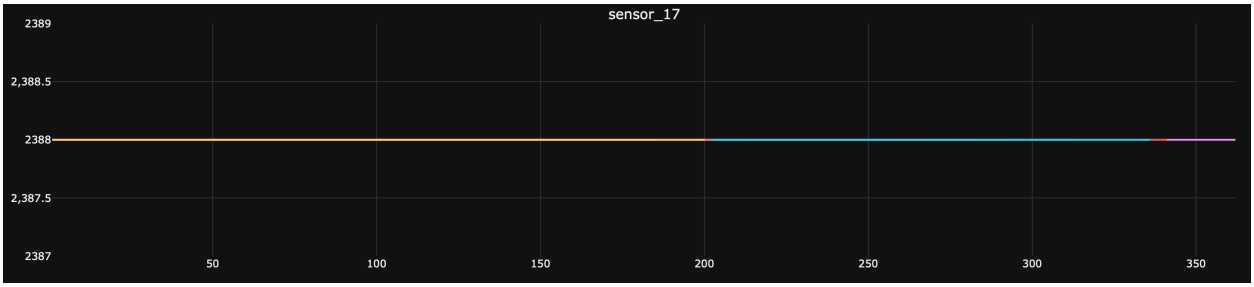
Summary:
Data from 20 total sensors was collected for 100 hot finishing mill (hfm) runs that ended in failure. The sensors were broken up into two groups, one containing 8 sensors and one containing 12 sensors. Initial data exploration revealed 6 sensors that provided no information, and these were omitted for model training and analysis. Three separate datasets were created, one for the 10 column dataset, one for the 14 column dataset, and one that combined the two. Three models were then trained on these three datasets independently creating 9 unique models in total. The three models were a ridge regression, K nearest neighbors model, and a gradient boosted regression tree. Of the tree types of models, the ridge regression generally underperformed compared to the KNN and gradient boosted models, but would be the easiest to operationalize and deploy. Of all 9 models trained the gradient boosted tree trained on the large dataset that combined all sensors had the lowest error and highest coefficient of determination.

Assumptions:

- no_of_days means days up or online not how many days remain until an observed failure happened. I will assume that a failure happened the day after the highest no_of_days value for each given run. For instance, day 192 was the last no_of_days for the first hfm run I will assume a failure happened on day 193 before the reading could be taken.
- Each observation is stand alone and does not show autocorrelation with previous days readings. In short, each observation is a snapshot in time
- The 10 column and 14 column datasets were collected at the same time. The hfm run numbers and duration in days align between the two datasets so we can assume that the two datasets can be combined into one large dataset if necessary.

Data Exploration:
During initial data exploration it was noticed that the hfm runs between the 10 column dataset and 14 column dataset aligned perfectly in both labeling and duration in days. This was the driver for assumption 3 above. Looking at sensor data showed a large range in nominal values for the sensors. For example, sensor 13 trended around 8150 while sensor 14 trended around 8. This large discrepancy in range drove the decision to normalize training data — more on that later.
Viewing plots showed that many sensors sere trending in one direction or the other prior to failure and some sensors also showed no change in value through all data collection, even across hfm runs. The two plots below highlight one sensor, sensor 13, with interesting characteristics and another sensor, sensor 17 that showed no change.





Data Cleaning and Transformation:
The previous visual observation that some sensors never changed value was numerically verified with a min and max across all data for those sensors. Each of the suspect sensor showed 0 or 0.01 units change across all hfm runs. Since this would provide no information to models during training these sensors were removed from the datasets. The sensors removed were 17, 4, 5, 9, 15, and 18.
Based on the observation for the large discrepancy in sensor nominal readings each sensor was normalized. This prevents sensors that have high nominal values from dominating model fitting.
Since we assume that the two datasets can be aligned based on hfm run and no_of_days a third dataset was created that combined all valid sensors from the 10 column dataset and 14 column dataset into one large dataset.
There was also a new variable created that was days_until_failure because that is the target we are striving to predict. This variable is simply the no_of_days reversed for each hfm run since we are assuming that all hfm runs ended in failure the next day.
Once all three datasets were created, they were split 80/20 into a training and testing sets.

Model Training Overview:
The same basic approach was used for developing all three types of models. A parameter space was defined for possible hyperparameters for each model. A grid search was conducted on all combinations of parameters where each model was trained on each unique combination of hyperparameters. Each training instance further split the training data into a training and test set to evaluate performance of each hyperparameter set. This additional split used a stratified k-fold approach to avoid overfitting.

Baseline Ridge Regression Model:
Since the sensor data visually appears to be trending one way or the other a good first model to try would be a linear model. Since these sensors are located on the same piece of equipment and may be monitoring similar characteristics (horizontal and vertical vibration for instance)

multicollinearity is a concern which makes a standard linear regression unsuitable. Ridge regression is more resistant to multicollinearity and thus was selected as the first model. This model underperformed the other two types of models with a mean average error of 34.3 days, average percentage error of 0.78, and R2 value of 0.58.

Improvements Using a More "State Aware" KNN Model:
The next iteration was to try to model the data in a fundamentally different way. Here we selected K nearest neighbors regressor. This model is fundamentally different than linear regression because it is not evaluating an equation for prediction. Rather, it is finding the nearest known state(s) of past operation and predicting current state based on that. This showed approximately a 10% improvement over the Ridge regression model with a mean average error of 30.3 days, mean average percentage error of 0.36, and R2 value of 0.63.

Testing a More Complex Tree Based Model:
The final approach was to try to model the data in a higher dimensional representation. Tree based algorithms are one approach to that, so a gradient boosted ensemble learner was selected. This model seemed like a good fit to the problem because it is an ensemble of smaller trees that are trained on data that previous trees got wrong which makes them less prone to overfitting. This means a much larger and more expressive model can be created without fear of overfitting the data. This model performed on par with the KNN model achieving a mean average error of 29.4 days, mean average percentage error of 0.38, and R2 value of 0.64. One advantage was that the boosted tree trained faster and had a final model saved size that was 13x smaller than the KNN model. This means at scale it will be easier to retrain/update and deploy/store.

Deployment:
Ad-hoc inference can be run using the included app.py program which serves an API on localhost:8080. This is most easily achieved by building and running the docker image from the supplied DOCKERFILE. The API will run inference on all nine models by specifying the json key for the desired model and then an array of sensor data. For more technical documentation on API use consult the included README.md.