Transient State Detection in Machine Data

Advisors:

Dr. Yoav Freund (UCSD)
Dr. Chad Holcomb (Solar Turbines)

Team Members:

Garrett Cheung
Michael Galarnyk
Jared Goldsmith
Jillian Jarrett
Orysya Stus

Executive Summary

Solar Turbines, a gas turbine manufacturing company, offers an online platform to allow customers to remotely monitor and manage their equipment. The goals of this system are to avoid downtime, increase asset reliability and lower lifecycle operating costs. One of the modules intended to facilitate these objectives is the alerting capability. Alerts are generated when a machine parameter goes above a custom set limit. Once an alert is generated, a fleet manager views and responds accordingly. These alerts often allow fleet managers and customers to take preventative measures that avoid unplanned downtime and associated costs. However, alerts are also triggered when a gas turbine is undergoing a transient state. Transient states occur when a machine is shifting operating loads (power output) or recalibrating to maintain a steady state. Their occurrence is often part of healthy and normal machine operation. During transient states, key sensors in the machines spike quickly before returning to their typical values. These quick fluctuations trigger alerts which are not necessarily useful. At best, they are a minor inconvenience to customers. At worst, they can result in unplanned downtime while the machines are undergoing unnecessary diagnostics, which, depending on the customer and use case, can cause millions of dollars of lost revenue. By identifying transient states and the subsequent alerts they generate, Solar Turbines can eliminate non value-added alerts shown to their customers.

How does one classify transient states in Solar's packages? First, a data source was identified. Specifically, machine data was obtained for 72 packages of two engine models for a two year period in both 10 minute and 1 hour resolution. Packages are units composed of compressors, combustors, turbines and other components, depending on their application type. This time series dataset is primarily comprised of, but not limited to, temperature and pressure sensor readings from various positions along the packages. Due to its proprietary nature, using this dataset required extensive preprocessing and sanitization efforts, as well as approval from Solar Turbines. After acquiring the data, it was cleaned such that it could be used to build a model. Because the data set did not include labels or a clear way to distinguish transients from normal operation, modeling was done using a combination of unsupervised learning techniques. For the same reason, domain experts were relied on heavily for validation of results. To facilitate this process, a user interface was constructed to allow domain experts to visualize, interpret, validate and annotate results. The process, from identifying a data source to labelling it, as well as the measures taken at each step, are detailed below.

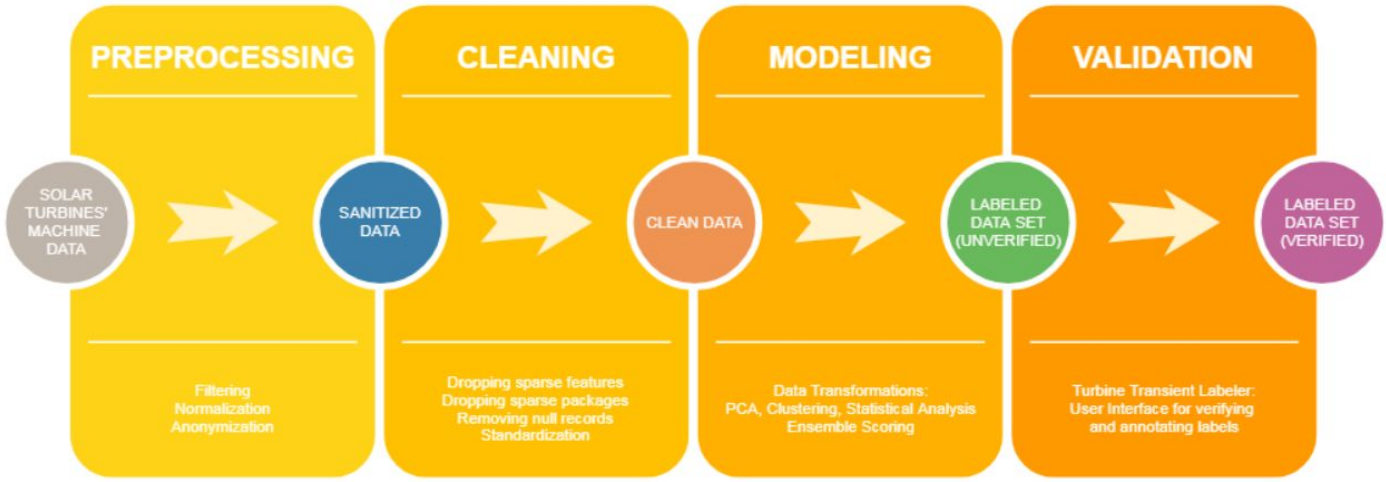

Figure 1. End-to-end data science process

Preprocessing. Data was filtered by a series of conditions to ensure consistency. Additionally, data was anonymized and normalized. All methodologies were reviewed and approved by Solar Turbines prior to movement off their network.

Cleaning. After preprocessing, data quality issues arose. As time series analysis requires consistent and continuous data, difficulties related to sparsity were addressed. Ultimately, given the preference for high temporal resolution and data completeness, this project focused on one engine model's (aliased to "Model2") 10-minute sampled data.

Modeling. Once clean, Model 2's data was standardized and principal component analysis (PCA) was performed. PCA reduced the data from 77 dimensions to 20 dimensions while maintaining 88% of the original dataset's variance. These principal components were analyzed using statistical methodologies, segmented by lengths of time, and fed to unsupervised clustering algorithms. Additionally, the raw feature "power" was analyzed using statistical functions both as a part of our ensemble model and a means to ascertain the accuracy of the other methods. Each of these methodologies resulted in the a binary classification of each datapoint as "transient" or "normal" operation, where 1 = "transient" and 0 = "normal." These scores were weighted and combined into an ensemble score, which was evaluated against a transient threshold value. For each package serial number and timestamp, the ensemble model offers a classification of "transient" or not. Due to the complexity of the dataset and lack of true labels, validation of the results requires domain expertise.

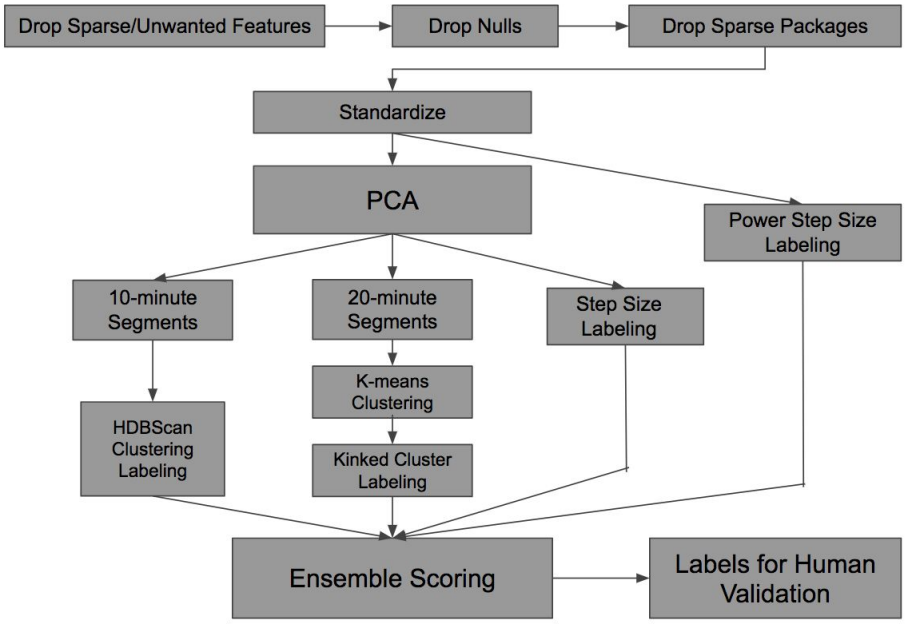

Figure 2. Transient State Classification Model

Validation. Fig 3a shows the user interface that was built to allow domain experts to visualize, validate and annotate results. The interface allows users to map the PCA transformed features to and from raw features. The ensemble labels, as well as the individual scorers that comprise them, can be explored in both the raw and engineered features. To provide context to the data, load profiles, package similarities and feature importance metrics are able to be viewed in the interface. The inclusion of annotation capabilities allow domain experts to validate and correct transient labels, turning these unverified labels into ground truth. With confirmed labels, this dataset could be fed into a supervised machine learning model to predict transient states in gas turbines.

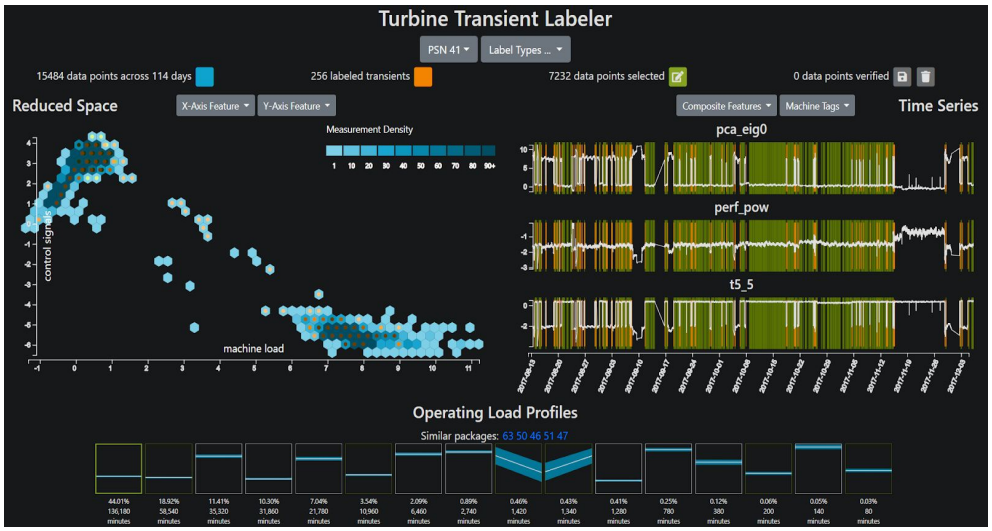

Fig 3a. User Interface Overview

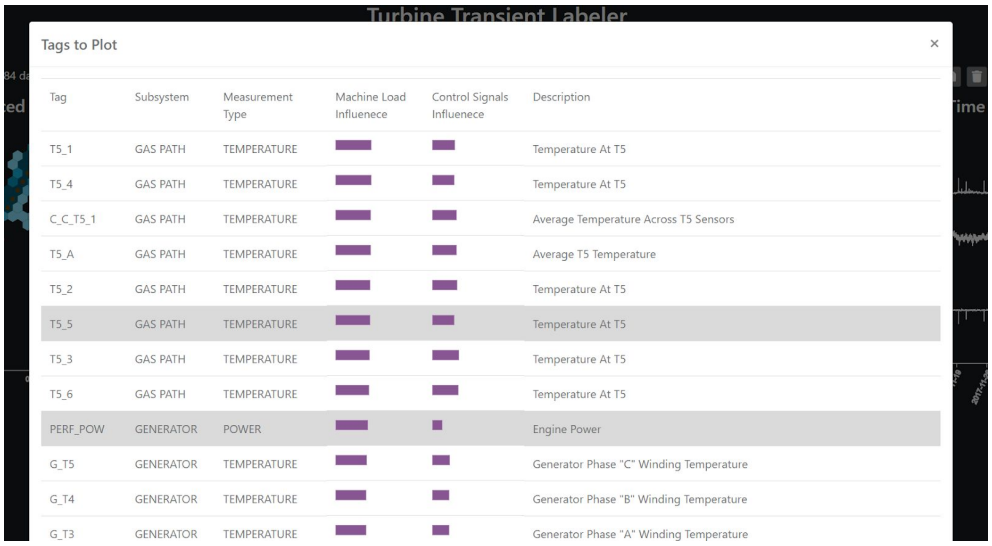

Fig 3b. Raw Features and Their Influence on Engineered Features

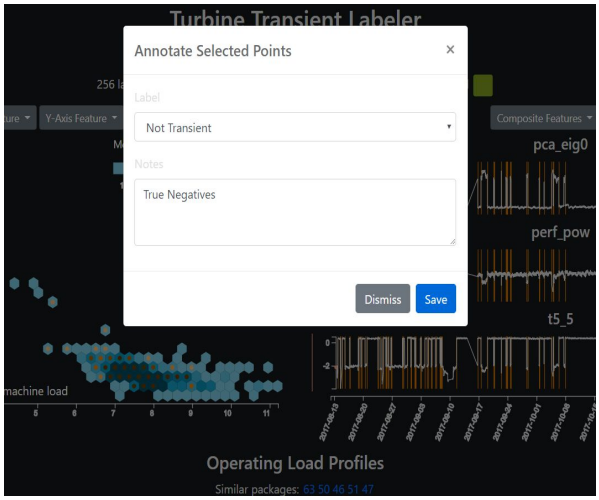

Fig 3c. Annotation Capability of the User Interface

Figures 3a, 3b and 3c highlight the main features and capabilities of the User Interface. Domain experts can explore the raw and engineered features over time in the Time Series component (Figure 3a, right) or in alternative subspaces in the Reduced Space component (Figure 3a, left). They can also gain context and visualize how often a package operates in a given state or find similar packages using the Operating Load Profile section (Figure 3a, bottom). Additionally, to better understand what the raw features are and how they relate to the engineered features, users can explore the Machine Tags pop out window (Figure 3b). Lastly, domain experts can validate or modify transient state classifications using the Annotation capability (Figure 3c).

In summation, using a combination of unsupervised methods, transients can be identified, quantified and distinguished from normal operating behavior. While the complexity and proprietary nature of these findings significantly narrows our target audience, the methodology for approaching an unlabeled, highly dimensional dataset can be applied to variety of other disciplines.

Table of Contents

Background

Solar Turbines

Solar Turbines is one of the world's leading producers of industrial gas turbines, with over 15,000 gas turbine systems installed in over 100 countries across the globe. In addition to manufacturing products, they offer Equipment Health Management (EHM) to their customers. EHM is a technology platform designed to deliver advanced remote capabilities and decision support tools. The objective of EHM is to provide proactive support and optimize availability, reliability and value of their products. Two members of this team, Jillian Jarrett and Garrett Cheung, are employees of Solar Turbines and work to develop new capabilities for EHM with these goals in mind.

The Machinery

Solar Turbines' 'packages' are units comprised of compressors, combustors, turbines and application specific components. They are extremely complex machines with hundreds of moving parts, and are generally used for movement of gas through pipelines and power generation. A brief explanation of gas turbines and their utilities can be found on the company website:

"A gas turbine engine is a type of internal combustion engine. Essentially, the engine can be viewed as an energy conversion device that converts energy stored in the fuel to useful mechanical energy in the form of rotational power. The term "gas" refers to the ambient air that is taken into the engine and used as the working medium in the energy conversion process. This air is first drawn into the engine where it is compressed, mixed with fuel and ignited. The resulting hot gas expands at high velocity through a series of airfoil-shaped blades transferring energy created from combustion to turn an output shaft. The residual thermal energy in the hot exhaust gas can be harnessed for a variety of industrial processes."[1]

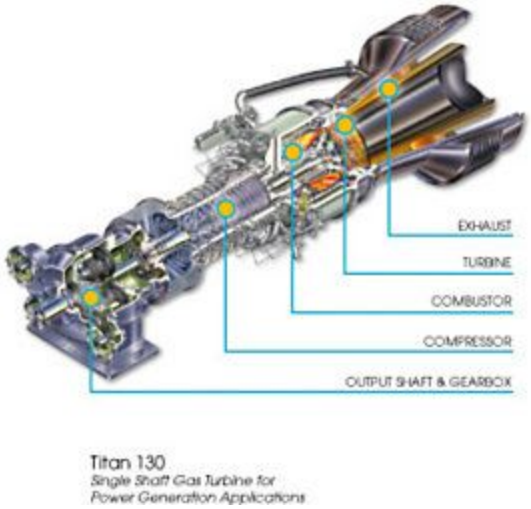

Figure 4. Components of a Gas Turbine for Power Generation Applications[2]

[1] "Gas Turbines - Products | Solar Turbines." https://www.solarturbines.com/en_US/products/gas-turbines.html. Accessed 22 May. 2018.
[2] "Sigma Labs Wins Contract from Solar Turbines to Use ... - 3DPrint.com." 19 Apr. 2017, https://3dprint.com/171698/sigma-labs-solar-turbines-inc/. Accessed 22 May. 2018.

Gas turbines are work producing turbomachinery: the application specific components are driven by the rotating machinery that converts thermal energy to mechanical energy. The maximum rate at which this conversion take place is the maximum power output, which is referred to as 100% load. Load refers to the percentage of the maximum power a package is outputting at a given time.

A single package can have up to 600 machine data features, depending on model, control system, application type, etc. For each package, there exists time series data at various resolutions, events data, alerts data, contract data, etc. While there are over 1,900 connected packages worldwide, domain experts and company policy limited the project to 72 packages of two different engine models.

Transient Definition

Transient states occur when a gas turbine is undergoing some sort of transition: it can be speeding up, slowing down, or attempting to re-achieve an equilibrium at a steady state. They are periods known to be strenuous on the packages. While no universally accepted definition exists in the industry, one domain expert defines it as as “any instance where load (power output) changes by more than 25% in a 10 minute timeframe.”

Business Value

Transient state identification allows Solar Turbines to improve their alerting capabilities. Currently, alerts are generated when parameters move outside of set, static limits. For example, if a temperature moves above a defined limit (set by design engineers), an alert is triggered. The alert is sent to customers or fleet managers who respond accordingly. These alerts can help users identify and prevent malfunctions in their machinery. However, transients are often visible in machine data as quick spikes in alerting parameters, despite being part of normal operation. Thus, often times, when a transient occurs, it triggers alerts that are false positives. They are not helpful to customers, and can misdirect them into thinking a healthy package is malfunctioning. This can result in unnecessary diagnostics and unplanned downtime, which can cost customers millions of dollars in lost revenue. Identifying specific types of transients and the subsequent alerts they generate allows Solar Turbines to reduce the number of non value-added alerts shown to their customers.

Additionally, it is hypothesized that transient states are correlated with machine failures. Proper identification of that transient subtype would allow for employees and customers to take preventative actions and reduce major failures and unplanned downtime. In terms of quantifying the cost of early inspection/repair versus catastrophic failure, Solar Turbines uses an order of magnitude difference as a baseline. A $100,000 spent on early repairs could prevent a $1 million catastrophic failure. Before the relationship between machine failures and transient states can be investigated, transient states must be identified. For these reasons, a transient state detection system would reduce unplanned downtime (and associated costs) and hence be invaluable to Solar Turbines and their customers.

Related Works

Transient detection in the processing industry is not unexplored terrain. Kim et al.[3] developed a physics-based gas-path analysis based model for the transient behavior analysis of a 150-MW stationary

[3] "Model Development and Simulation of Transient Behavior of Heavy ...." http://gasturbinespower.asmedigitalcollection.asme.org/article.aspx?articleid=1421229. Accessed 6 Jun. 2018.

gas turbine to predict time-dependent variations of the performance parameters such as power, shaft speed, fuel consumption, and gas-path temperatures. One issue is the degradation of gas-path parts affects the functional relation between performance parameters. Furthermore, developing a physics-based model has its limitations, especially when little information is available from the gas turbine parameters and component maps[4]. An alternative approach is to develop a quantitative model based on gas turbine measurements. These models often embed knowledge of a proprietary nature such as the performance maps of the turbomachinery components and other empirical correlations derived from extensive testing. Most of the reports of gas-path analysis applied to real data found in the literature rely on publicly available engine models and are used to assess the health condition of a particular gas turbine during its operational life.[5]

Data and Environment

Solar's Data Requirements

Due to the proprietary nature of this dataset, strict guidelines were set forth before data could be moved off the internal network. Requirements are as follows:

- All customer data removed
- Details relating to engine models excluded
- Package serial numbers aliased
- Column names aliased
- Data values normalized
- All columns required approval by the Solar Digital team
- Data must be for machine operation between December 5, 2015 and December 5, 2017
- Data only for 72 packages belonging to two engine models
- Only machine data with 10-minute and 1-hour resolution may be used
- Dataset may not be made publicly available

Data Sources

Due to the restrictions set forth above, machine data is our principal data source. This data comes from the packages themselves, and is mainly comprised of sensor readings. More specifically, they are primarily temperature and pressure readings, but also include vibration and displacement measurements, speeds, programmable logic controller values, etc. Index-type columns, such as package and engine serial numbers, as well as time measurement columns (timestamp, engine hours, engine starts) were also included. These features or, as 'tags' as they are referred to at Solar, are listed and explained in greater detail in our data dictionary, available on our GitHub. Solar currently stores 1 hour resolution data, which is used to generate alerts and support the EHM platform. They also store 10-minute resolution data. One second resolution data also exists, but only for intervals surrounding shutdowns. Unfortunately, the 1 second temporal resolution data was not approved for investigation. Ultimately, we obtained 1 hour and 10

[4] "IEEE Xplore: IEEE Transactions on Reliability - (Popular)." https://ieeexplore.ws/xpl/topAccessedArticles.jsp?punumber=24&topArticlesDate=November+2017. Accessed 6 Jun. 2018.
[5] "A Model-Based Anomaly Detection Approach for ... - ASME Proceedings." http://proceedings.asmedigitalcollection.asme.org/proceeding.aspx?articleid=1908327. Accessed 6 Jun. 2018.

minute data for 72 packages belonging to two different engine models for two years. A summary table of data specification is included below.

Table 1. Data Specifications

Number of Packages	Timeframe	Data Resolution	Features
72 (total) **Model1: 33** **Model2: 39**	**2 years** Start date: 12/5/2015 End date: 12/5/2017	**1 hr** and **10 minute** for both models	**Model1: 145 features,** **Model2: 77 features**, comprised of: **Sensor readings:** pressures, temperatures, speeds, displacement **Package basics:** timestamps, engine hours, package serial numbers, etc **Programmable logic controller (PLC) readings:** signals sent to the machine through the control system

Data was processed on the Solar servers and moved off the network in the form of 144 csv files. There were two csvs for each package: one with the 10 minute data and another with the 1 hour data. The columns for each engine model of all csvs were identical. The source and destination of the machine data are shown in the table below.

Table 2. Raw data Sources and Destinations

Dataset Name	Source	Destination	Acquisition Notebooks, Code, Documents	Data Size & Format	Other Notes
1hr_machine_data	Solar Turbines servers	AWS EC2	Available on our github: https://github.com/mGalarnyk/AnomalyDetectionMachineData	~1gb (421,204 rows) in 72 csv files	Anonymized, confidential (data cannot be made publicly available). Data dictionary available
10min_machine_data	Solar Turbines servers	AWS EC2	Available on our github: https://github.com/mGalarnyk/AnomalyDetectionMachineData	~6gb (3,107,256 rows) in 72 csv files	Anonymized, confidential (data cannot be made publicly available). Data dictionary available.

Data Acquisition

Machine data was pulled from Solar Turbines databases using an internal Solar Turbines Python library. This module allows users to retrieve machine data values for a specified date range in the form of a pandas dataframe. Both 1hr and 10 minute resolution dataframes were acquired for the 72 packages operating in the approved timeframe, December 5, 2015 - December 7, 2015.

Data Preprocessing

Extensive preprocessing was required prior to moving data off of the Solar network. In addition to the requirements set forth by Solar Turbines, issues with data consistency were addressed. The data preprocessing workflow is as follows:

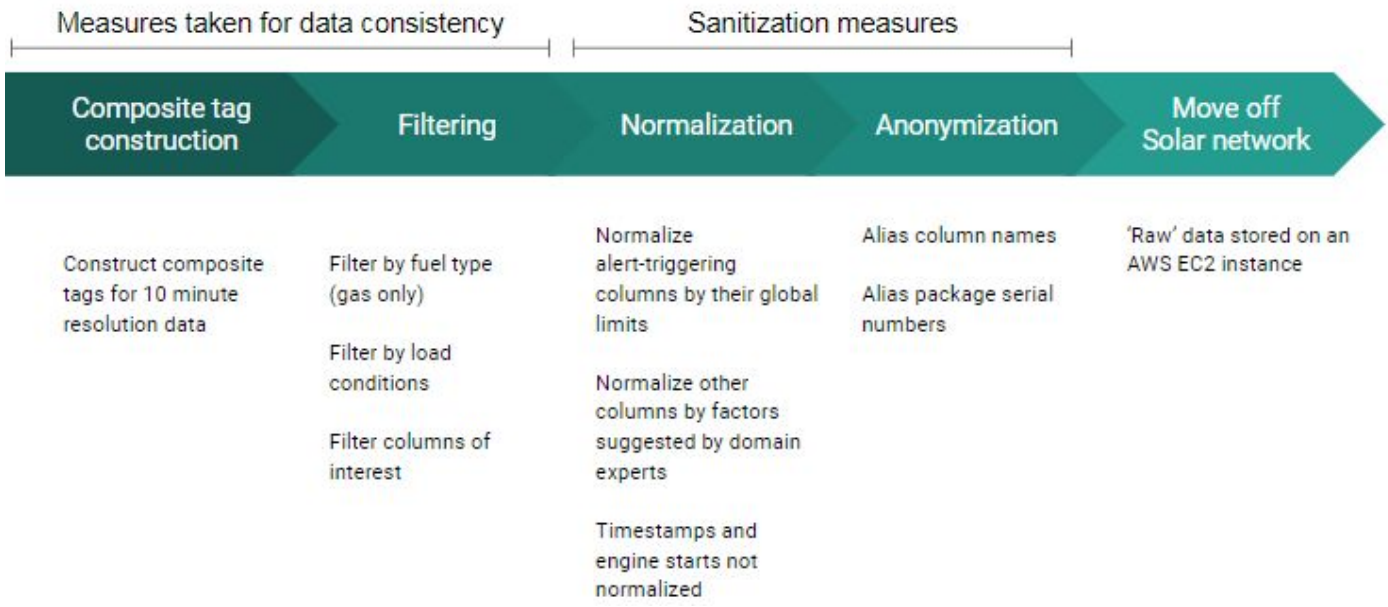

Figure 5. Solar-side Data Preprocessing Workflow

Data Consistency

Individual packages have unique configurations and sensors among other features. For example, packages used for power generation applications have generator related sensors, while packages used for compression of gases do not. Measures were taken to ensure consistent data for individual packages of each engine model at each temporal resolution.

Composite Tag Creation

Composite tags are features that are constructed from the raw sensor data. For example, the T5 spread tag is difference between the highest and lowest T5 sensor readings. It is not a directly measured value, but rather, a calculated one. In Solar's data ingestion pipeline, these composite tags are calculated and stored for the 1hr resolution data, but not the 10 minute data. To account for this, the 10-minute resolution data includes the same features as the 1 hour data, composite tags were generated for the 10 minute dataset using the equations defined by Solar. Neither these calculations nor the code executing them could be made publicly available.

Filtering

The machine data was filtered by a series of conditions, per domain experts' suggestions. These measures were taken to reduce the amount of variation in the original data and reduce the influence of potentially confounding variables. First, we are only considering data where the package is running close to its capacity in terms of power output, which is referred to at Solar as "on_load" conditions. Second, we acknowledged that some of the packages run only on gas fuel, while other can be run on liquid or gas. Fuel

type can cause major differences in turbine operation. To maintain consistency, all data points where the machines are running on liquid fuel were removed. Finally, because not all machines contain the exact same sensors (due to different configurations and use cases), columns which are not common across all packages are removed as well. This also applies to the composite columns mentioned above. For Model 1, that reduces our dataset to 145 features, 163,544 record in 1 hour temporal resolution and 1,501,927 records in 10 minute resolution. For Model2, our dataset was reduced to 77 features, with 257,732 records of 1 hour data and 1,604,987 records of 10 minute data.

Sanitization

Measures taken to satisfy the aforementioned Solar requirements are detailed below.

Normalization

Solar Turbines mandated data be normalized by factors that would remain unknown to any non-Solar employees. Factors were largely influenced by the EHM Alerting capability. As detailed in the Business Value section, many features in our dataset are used to generate alerts (notifications to customers and Solar employees that specific parameters had crossed some sort of limit). These limits are values defined by Solar Turbines' engineers and considered thresholds outside of which packages should be monitored. Not all features are used to generate alerts. Those that do, however, were normalized by their global high limit values. For columns without global limits, domain experts were consulted and data was normalized by factors suggested by them. Two features were not normalized: engine starts (the number of time an engine has started) and timestamps. While the normalization was a requirement of Solar Turbines, it offers some consistency and rules of thumb to our team members. After normalization, for most columns, expected values are generally between 0 and 1.

Anonymization

Column names were aliased based on their subsystem and tagtype, as described in the Understanding Data Format section below. Additionally, package serial number and engine serial numbers were aliased to integers. A data dictionary was also created to provide descriptions and data types of each column.

Caveats

Due to the proprietary nature of our data, our true 'raw' dataset is not publicly available. What is henceforth referred to as 'raw data' is what was moved off of the Solar network after extensive processing, as detailed above. Additionally, the code used to preprocess our data is not included in our github repositories. It is housed and executed exclusively from within the Solar network.

Data Environment

Data Storage

The anonymized data from Solar is stored in a PostgreSQL database. The current volume of data does not require a distributed storage solution, and ease of use was considered paramount. The 1 hour and 10 minute resolution data is currently hosted on an AWS EC2 t2 large instance. Data from the 144 csvs was stored in 4 tables, as detailed below.

Processed Data Location: PostgreSQL, AWS EC2
Import Code: /src/storage/postgres.py (subject to change)
Input: 1hr_machine_data
Destination Tables:
sensor_readings_model1_1hr
sensor_readings_model2_1hr
Input: 10min_machine_data
Destination Tables:
sensor_readings_model1_10min
Sensor_readings_model2_10min

For more detail on our data storage solutions, refer to the Pipeline section.

Exploratory Data Analysis

Understanding the Data Format

A data dictionary was generated to exhaustively document data structure, meaning, and format. In the datasets, the column headers define a tag using the format **SUBSYSTEM_C_TAGTYPECOUNTER** where

- **SUBSYSTEM** refers to the turbine section it is usually associated with
- **C** denoted whether or not a tag is calculated versus measured (C means calculated, as is an optional argument)
- **TAGTYPE** refers to the type of measurement being recorded
- **COUNTER** is an incremental counter to differentiate tags with the same subsystem and tagtype.

For example, C_C_DP1, is a composite tag measuring differential pressure in the combustion system. The first C represents the associated subsystem (Combustion), the second C indicates that the tag was Calculated, and DP refers to the type of measurement (Differential Pressure). It is the first tag in the set with these qualities, as indicated by the 1. This data format will be referenced throughout this project.

Data Quality Issues

As time series analysis requires consistent and continuous data, analysis was conducted to determine data quality and consistency. Specifically, the shape of the data and null counts were noted and feature plots generated. Complete records are defined as timestamps per package with no nulls in any features. With regard to the 1 hour resolution data, Model1 has 163,544 total records and 145 tags with 46.5% (76,833 timestamps) being complete records while Model2 has 257,732 total records and 77 tags with 64.6% (166,500 timestamps) being complete records.

The data completeness on a per-package basis differs greatly between models. To quantify these differences, each package was analyzed individually. Specifically, the first and last timestamps were noted and the theoretical of number of data points required to have a 100% 1-hour data for that time frame was calculated. The actual number of records for each package was divided by this maximum and a raw percentage of percent completeness was generated. For example, a package that only ran for one week in

the approved timeframe should have 168 data points (24 points per day * 7 days). If a package is found to only contain 84 records, it is considered 50% complete (84/168). The distribution of the data completeness metrics between the two engine models is shown below.

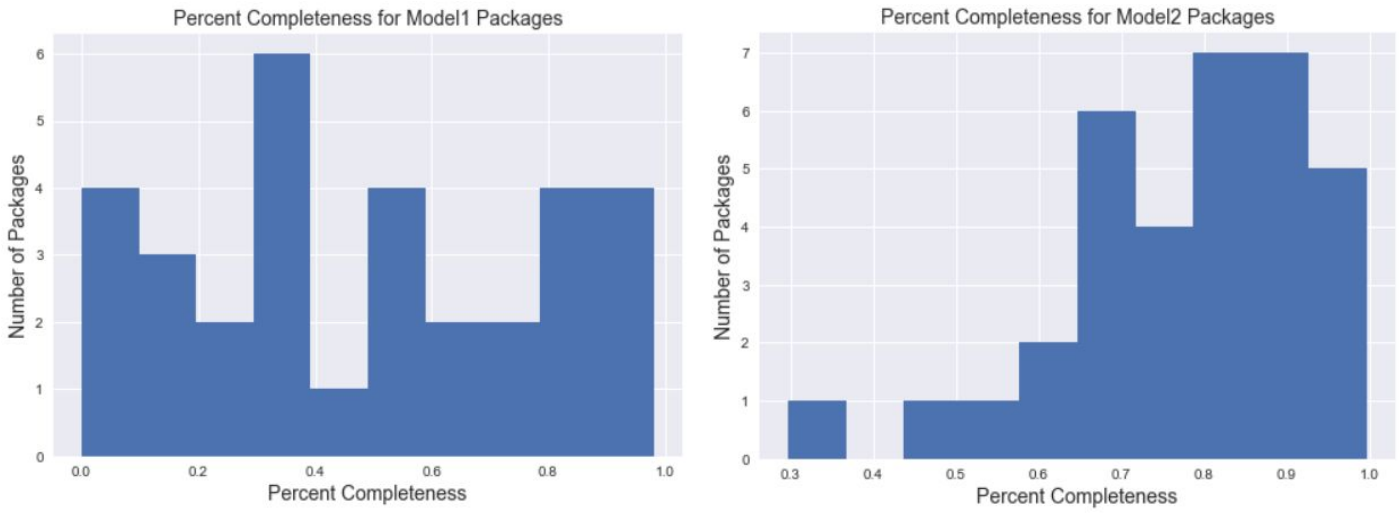

Figure 6. Distribution of Data Completion Percentages for Each Engine Model (Based on 1-hour data)

These distributions show that Model2 has a higher degree of data completeness than Model1. While packages with high data completeness have similar data availability between the two models, the medians for each differ dramatically. Specifically, the average Model1 package has 48.55% data completeness: the average package in the distribution has less than half of it's possible records. For Model2, on the other hand, the mean is 78.20%, and the median package is 82.32% data complete. A summary of findings is shown below.

Table 4. Data Completeness Results (1-hour data)

Data Completeness (Calculated Per PSN)	Model1	Model2
90th percentile	89.25%	94.94%
Mean	48.55%	78.20%
Median	48.60%	82.32%
25th percentile	24.84%	69.23%

Data completeness analysis resulted in Model2 data being used for our final product.

Tools and techniques

The primary tool used for exploring the dataset was using the programming language Python (version 3.6) by way of Jupyter notebooks. Details of the libraries and their functions are as follows:

Table 5. Python Libraries Used

Python Library	Use Case(s)
pandas	Data cleaning, parsing, and filtering. Dataframe manipulation
sqlalchemy	Data import
seaborn	Visualization
matplotlib	Visualization
sklearn	Data standardization, dimensionality reduction, clustering, scoring
datetime	Timestamp manipulation
hdbscan	Unsupervised clustering

Additionally, Tableau and JavaScript dashboards were constructed to allow for more interactivate means of data exploration.

Google Colab

Google Colaboratory was initially utilized to facilitate collaboration between team members and advisors. While this tool allows for individuals to work together on a live notebook, it had several drawbacks. When users are idle on the notebook for 90 minutes, the virtual machine used to host the notebook is reclaimed. Consequently, Python libraries need to be reinstalled every time when returning to the notebook. Luckily, most commonly used modules are preinstalled on the virtual machine. The rest were reinstalled in the first cell of the notebook. Additionally, file I/O ended up being very convoluted and difficult to manage. In terms of benefits, Google Colab works well with the PostgreSQL database on AWS EC2 and allowed for the efficient loading of data. Additionally, there was no need to set up Python environments or worry about different versions of Python or various libraries.

Findings

Exploratory data analysis has shown that there is a huge amount of variability within our dataset, which is supposed to represent normalized, steady state operation of gas turbines at on load or full load conditions. Extensive exploration has elucidated the causes of these inconsistencies. For context, much of the initial exploration was done with the objective of identifying anomalies. However, because transients are extreme values, exploratory data analysis in the pursuit of anomalies successfully helped us identify them. Furthermore, it provided insight and facilitated identification of trends and patterns in an extremely complex, multidimensional time series data set.

Feature Plots

Feature plots were generated for each tag of both engine models, with all the packages considered. For each column, a line graph shows value of that tag over the two year period. Each colored line represents a unique package of that engine model. These feature plots are useful in understanding how

tags move over time, which tags have greater variance which are more stable (low variance) as well as visually seeing spikes/dips in trends signifying potential anomalies. An example feature plot:

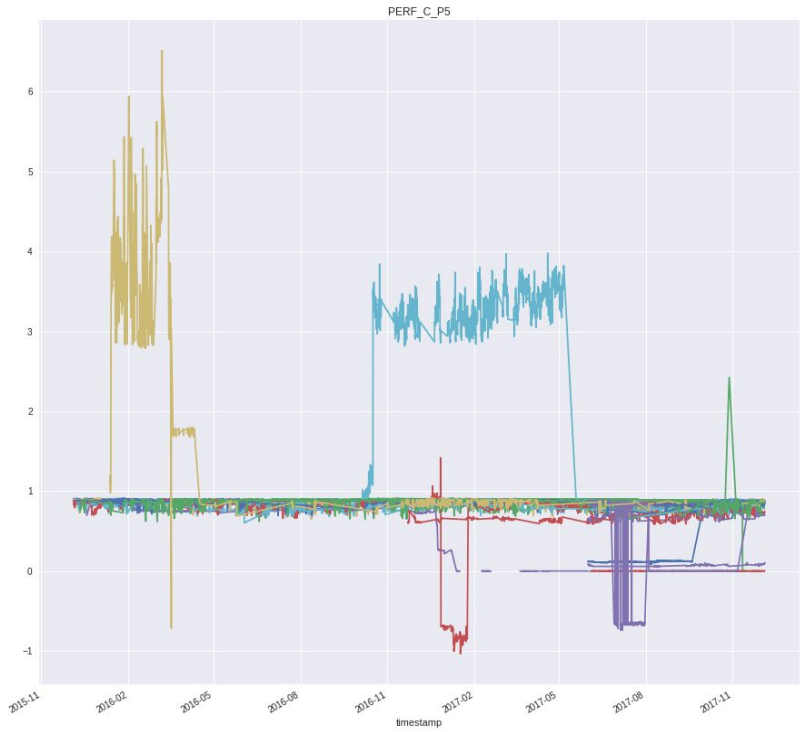

Figure 7. Feature plot of PERF_C_P5 for all Model1 packages, a composite pressure tag related to performance. This highlights that packages generally operate between 0.5 and 1, with some outliers visible outside of these boundaries.

The complete set of feature plots for all tags of both engine models can be found on the Team Drive.

Statistical Analysis

Statistical methods were developed for finding anomalies values in our dataset. Specifically, Python methods were built to identify the following:

- Flatlines (unchanging values may indicate failed sensors)
- Datapoints outside defined thresholds
- Outliers (anything outside the interquartile range, IQR = 1.5*q3-q1)
    - Using single package IQRs
    - Using fleetwide IQRs
- Data points outside several standard deviations of mean or median
    - Using single package means/medians
    - Using fleetwide means/median
- Step-size changes
    - Based on rolling averages and rolling standard deviations (looking for sudden spikes or dips in the data)
- Power Jump
    - Looks for jumps in output power in the dataset. Used for detecting transients
- Global limit deviants

- Looks for data points where any feature is outside the global limit

We evaluated these functions on our raw data. Below is a feature plot, where we look at the feature ‘C_C_DP1’, a tag measuring a differential pressure in the combustion subsystem. Each package is plotted in a different color. Then, we applied one of our outlier functions to this data. Specifically, we used the IQR outliers function that identifies any points outside 1.5*IQR from the 25th and 75th percentiles as outliers. IQR is the interquartile range and represents the range from the 25th percentile to the 75th percentile of a given feature. We replotted the raw data, where blue points show what we consider ‘normal’ data and red points are labelled outliers, as can be seen below:

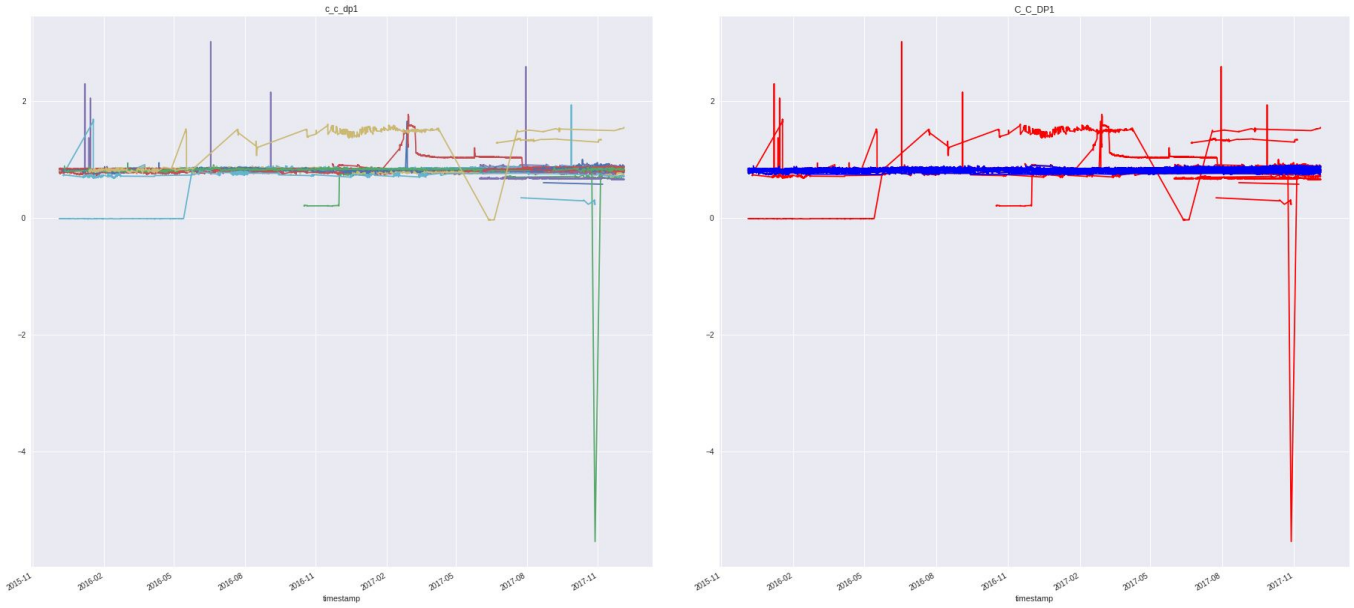

Figure 8. Successful IQR method results (left side: raw data colored by package, right side: anomalous data points colored red and normal data points are shown in blue)

Another statistical function identifies data points several standard deviations outside of a measure of central tendency. The number of standard deviation used (threshold) can be changed as an input parameter, as can the measure of central tendency value (mean or median, by package or by fleet).The figure below shows a pressure ratio related to package performance. Datapoints outside 2.5 standard deviations of the mean are colored red, while points within 2.5 standard deviations are blue. The plots allow domain experts to visually explore cutoffs and data behavior. While a 2.5 standard deviation threshold catches most of the outliers, some are missed.

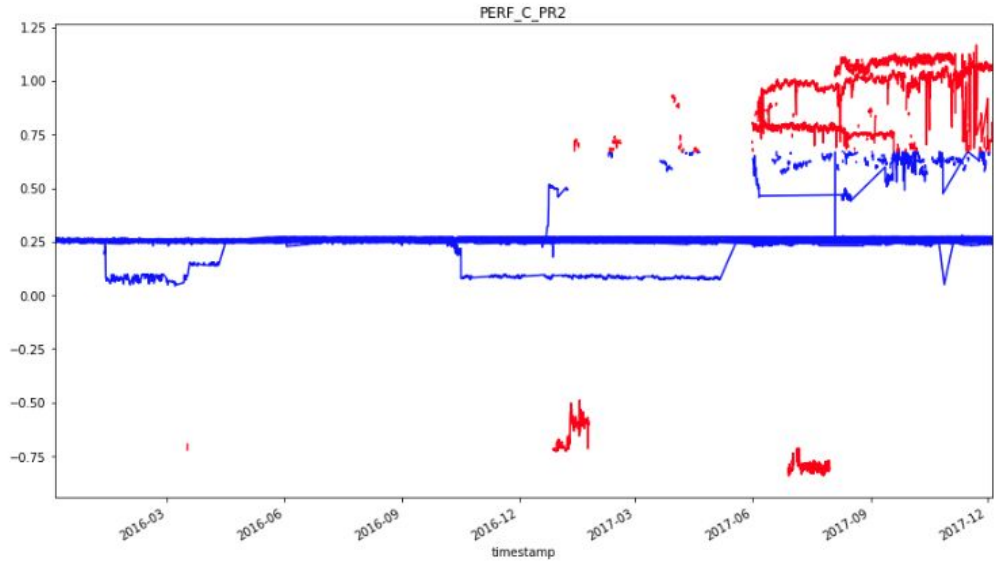

Figure 9. Somewhat successful Stdev method results (red: outlier, blue: normal)

Similar to the standard deviation function, a method to detect step size changes over time was implemented. This step-size function calculates a rolling mean and rolling standard deviation, and identifies points z standard deviations outside of these values where z is a threshold. The length of time for which the rolling averages and rolling standard deviations are calculated, as well as the z-score threshold, are input parameters. In the example below, points were identified where PCD jumps more than 3 standard deviations from a rolling mean. A seven day period was used to calculated the rolling mean and rolling standard deviation. This method allows for investigation of relative step size changes rather than distances from an overall mean. A scatterplot graph was constructed to visualize where the step sizes changes are occurring. Because of the extensive filtering we had to do to obtain the raw data, there are cases where data skips timestamps, resulting in spots where the data could look like a jump but in reality, the data points were simply not continuous:

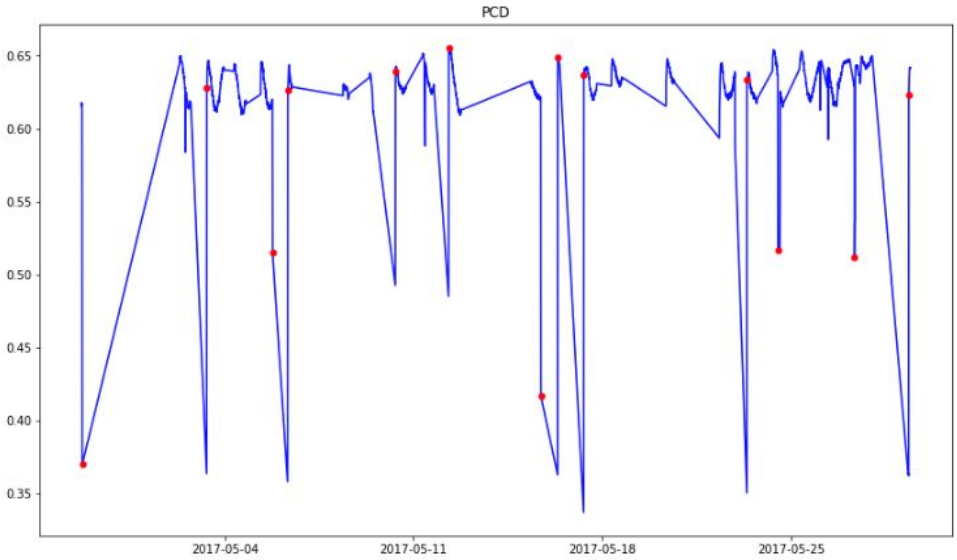

Figure 10. Step-size jumps using a 7-day rolling mean, ignoring data points with gaps in time (blue: data plotted over time, red: step-size jumps detected)

From these statistical functions, a large number of statistical outliers were identified. The following tags were excluded from the analysis: 'timestamp','sum_eng_st','sum_esn','sum_eng_h','psn','id'. Findings are detailed in the summary table.

Table 6. Statistical Analyses Findings (Using 1-hr Resolution Data)

Function	Parameters	Model	Outliers Count	Total Count	% Outliers	Interpretation
Standard deviations outliers	Z =2.5, centerpoint = fleetwide mean per tag, ignore_cols =['timestamp','sum_eng_st','sum_esn','sum_eng_h','psn','id']	1	479,470	11,140,785	4.30%	For Model 1, 479,470 data points outside 2.5 standard deviations of the fleetwide mean for a given parameter.
		2	534,316	12,820,500	4.17%	For Model 2, 534,316 data points are outside the 2.5 standard deviations of the fleetwide mean
IQR outliers	By psn = False, ignore_cols = ['timestamp','sum_eng_st','sum_esn','sum_eng_h','psn','id']	1	417,966	11,140,785	3.752%	Model 1 has 417,966 data points outside the fleetwide interquartile range across all tags
		2	634123	12,820,500	4.964%	Model 2 has 634,123 data points outside the fleetwide interquartile range across all tags
Global limit outlier	ignore_cols = ['timestamp','sum_eng_st','sum_esn','sum_eng_h','psn','id']	1	983,294	11,140,785	8.826%	Model 1 has 983,294 data points outside the global limits across all tags
		2	347,878	12,820,500	2.713%	Model 2 has 347,878 data points outside the global limits across all tags
Power Jump **	Powercol = ‘perf_pow’, jump=0.25, data_res = ‘1hr’	1	284	76,833	0.37%	Power was not available for this model so PCD was used as a substitute
		2	2658	166,500	1.60%	Model 2 has 2,658 jumps in power for our given dataset
Flatline	Threshold = 12	1	246,247	11,140,785	2.21%	Model 1 has 246,247 instances where the data flatlined for a 12 hour period
		2	366,065	12,820,500	2.855%	Model 2 has 366,065 instances where the data flatlined for a 12 hour period

stepsize	threshold=3, ignore_cols = ['timestamp','sum_eng_st','sum_esn','sum_eng_h','psn','id']	1	91673	11,140,785	0.823%	Model 1 has 91,673 step size jumps across all tags
		2	176425	12,820,500	1.376%	Model 2 has 176,425 step size jumps across all tags

** Note: Because transients are defined as 25% jump within a 10 minute period, results from running the Power Jump function on the 1hr dataset is unsuitable for labeling transients and this example is purely for exploration. Transient verification (described later in report) was run on the 10 minute dataset.

Model 1 Findings

Due to the larger number of features, initial EDA was conducted on the Model1 1-hour resolution data.

PCA by Fleet

PCA was performed on 1-hr resolution machine data for 32 distinct Model1 packages. Each record included a package serial number, timestamp, and 142 features. 7 of the 142 features were dropped because of a high number of missing values, detailed below. Each of the dropped features was derived from other features and highly correlated when populated. After removing these sparse tags, data was standardized using scikit-learn's StandardScaler module and PCA was performed.

Table 7. Tags Excluded from Model1 PCA by Fleet (1-hr Resolution Data)

Tag	Number of records missing
sc_pct2	30,561
sc_pct1	31,088
c_dt7_3	50,726
c_dt7_2	50,726
c_dt7_4	50,726
c_dt7_1	50,726
vsc_c_pct_e1	83,730

When evaluating a fleetwide PCA model, we found that the top 20 principal components retain 86.16% of the dataset's variance. 25 principal components retain 90.25% of the dataset's variance.

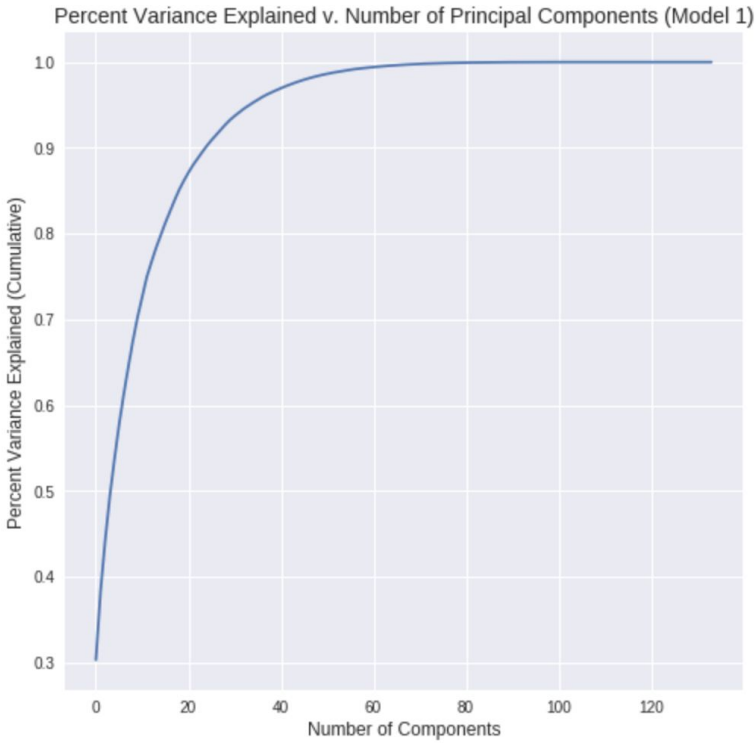

Figure 11. Variance retained versus number of principal components (Model1 1hr data)

The raw data features contributing to the top 5 eigenvectors (which retain 53.66% of the were identified. 53.66% They were analyzed by both measurement and subsystem type. The second principal component, for example, was identified to be primarily temperature readings (shown below on the right in red). The third principal component was heavily influenced by tags related to the fuel subsystem (shown below on the left in yellow).

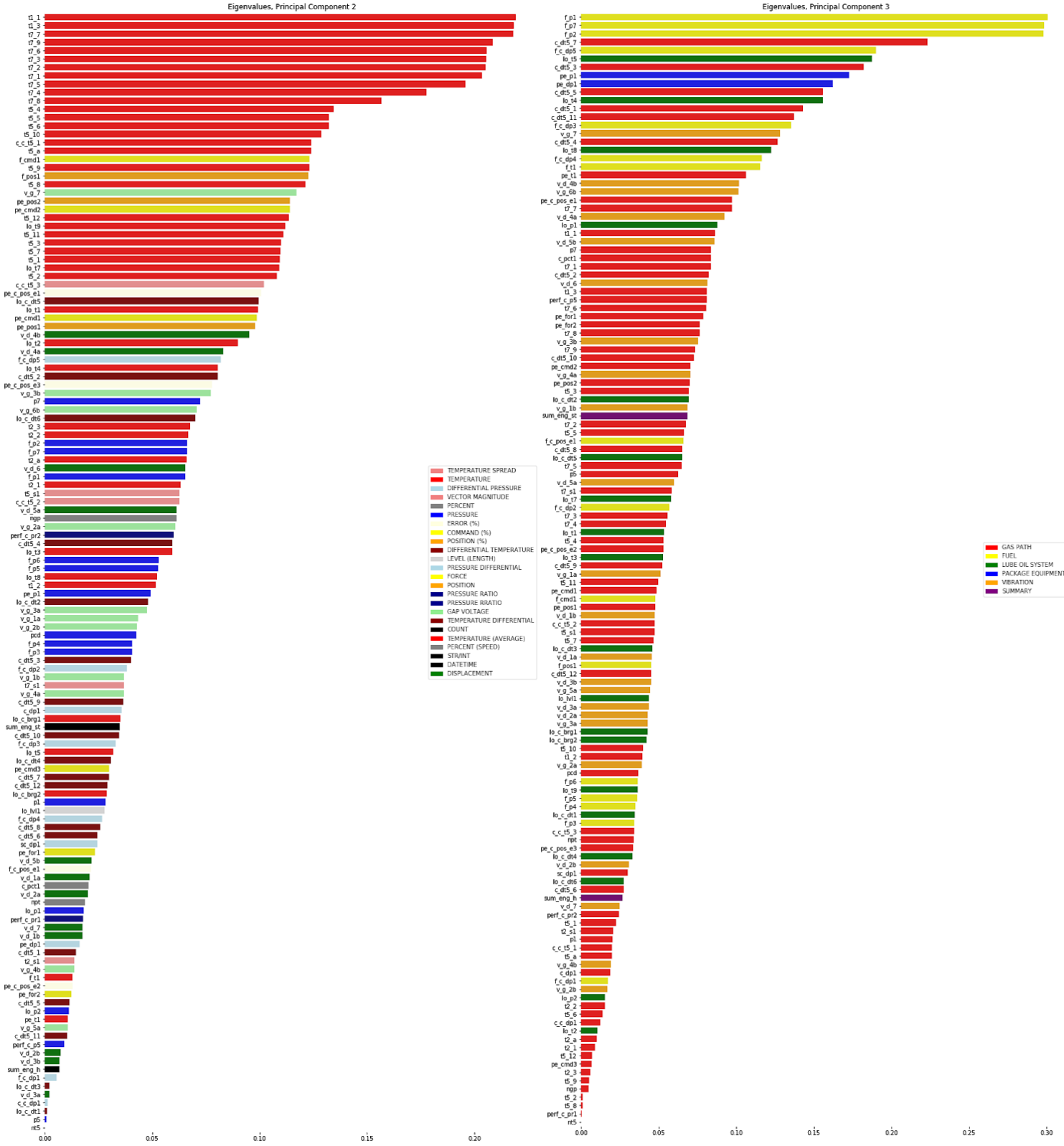

Figure 12. Features and their weights in the principal components (Model 1 Fleetwide PCA). Left: Principal Component 2, colored by measurement type. Right: Principal Component 3, colored by subsystem.

The complete set of these eigenvalue plots, as well as plots of top 5 eigenvectors over time for each package are available on our GitHub.

A summary of our initial Model1 fleetwide PCA findings is detailed below:

Table 8. Detailed Findings of our Principal Component Analysis (Model 1 Fleetwide model)

Principal component	Contributing features
1	Correspond to pressures, some temperatures in fuel system and gas path
2	Correspond to temperatures (distinct from the temperature tags in principal component 1)
3	Mix of pressures, temperatures, differential pressures and differential temperatures (no clear or obvious groupings)
4	Vibration related tags (displacements and gap voltages) plus some temperatures and differential temperatures
5	Mix of gap voltages, temperatures, positions, levels (no clear or obvious groupings)

PCA By Package

In addition to investigating PCA by engine model, principal component analysis was done on an individual package basis, per the suggestion of domain experts. The primary motivation for this approach was see how our packages related to one another and better understand the inner workings of a single package. The same preprocessing steps were taken as the PCA by fleet model (see PCA by Fleet section). The results were informative: the principal components looked very for different packages. The figures below show two distinct Model1 packages with very different features contributing to their first principal component. PSN26's first principal component is most heavily influenced by temperature tags related to the gas path subsystem. PSN7, on the other hand, is dominated by gap voltages related to the vibration subsystem, shown in green, as well as engine history tags (engine hours and engine starts), which colored in black. Additionally, the shape of of the bar charts is noticeably different. PSN 26 is heavily influenced by approximately 30 tags, visible by the steep reduction in values after the 'v_d_7' bar. PSN 7 has large contributions from over 50 tags and a less dramatic shift in values (influence) from each of the contributors.

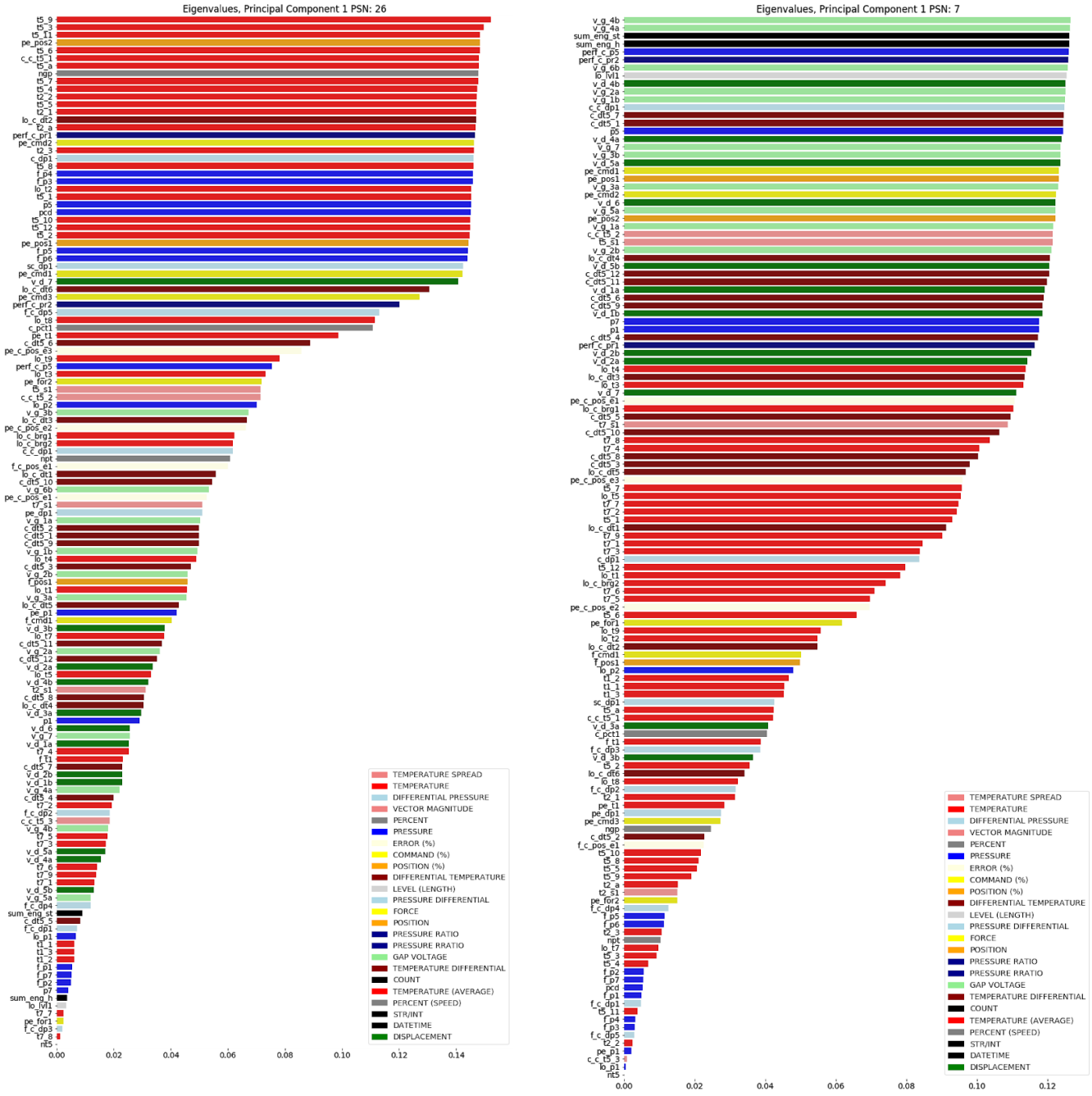

Figure 13. Features contributing the first principal component of two distinct Model 1 packages (Model 1 PCA by package). Higher values imply bigger contributors. Left: PSN26's first principal component is primarily influenced by temperature related features (colored red). Right: PSN7's first principal component is primarily influenced by gap voltages (colored in green) and tags related to engine age (colored in black)

K-means Clustering

Clustering was implemented in efforts to find similarities between data points. Its main advantages are its speed and that it doesn't require labels. Furthermore, it allows for comparisons between our transformed data and our raw data. Clustering on the engine-wide PCA model was difficult to interpret, but the PCA model by package revealed obvious groupings. For example, the plot below shows three distinct clusters for PSN21. The clustering was done with 20 principal components, but the axes in this plot are principal components 1 and 2.

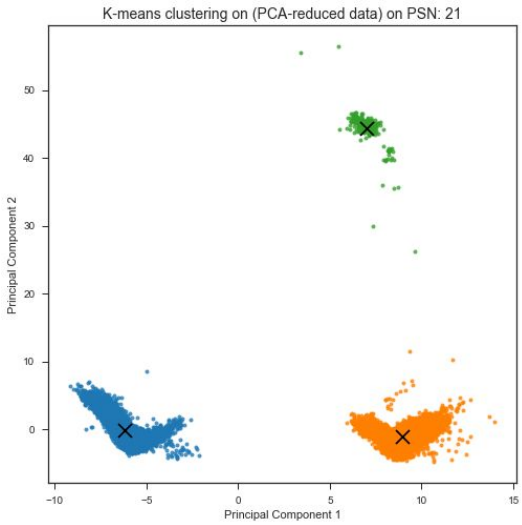

Figure 14. K-means clustering on PSN 21 1hr resolution reduced data set. (PCA model by package)

We mapped these clusters back to the raw data, and found the following distributions:

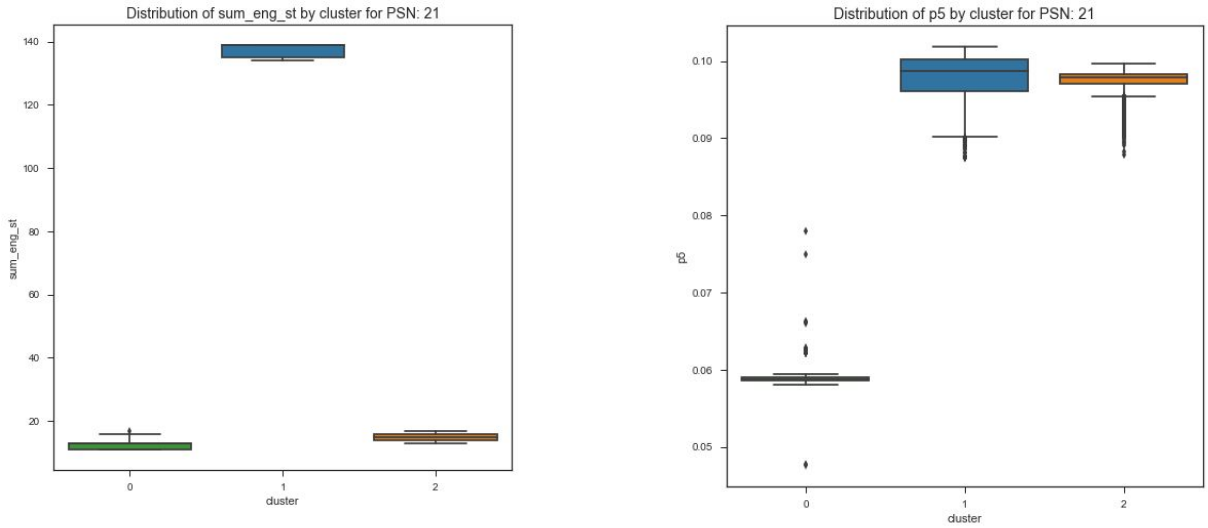

Figure 15. Distribution of values by cluster of raw data features. Left: boxplots of engine starts for each of the three clusters. Right: boxplots of P5 (pressure at turbine inlet) for each of the clusters.

The x-axis shows the distinct clusters and the y-axis represents raw data features. Sum_eng_st is a major contributor to the first principal component and P5 is prevalent in principal component 2. It was discovered and confirmed by our domain experts that the distinct clusters to correspond to operating states of the

package. In the example above, the blue cluster contains data points where the package is operating at full-load conditions, the yellow cluster contains on-load conditions, and green cluster captures the transitions between the two load states.These transitions between the two load states were initially believed to be anomalies but later confirmed to be transients.

Local Outlier Factorization

Local Outlier Factorization, LOF, is a method for finding anomalous data points based on local neighbor distance and density. Its ability to handle high dimensional data and varying cluster density made it a potentially good match for our data set. We explored it on the PCA by package reduced data set, as shown below. Though we considered combining LOF with subspace sampling, the project changed directions before parameter tuning was conducted.

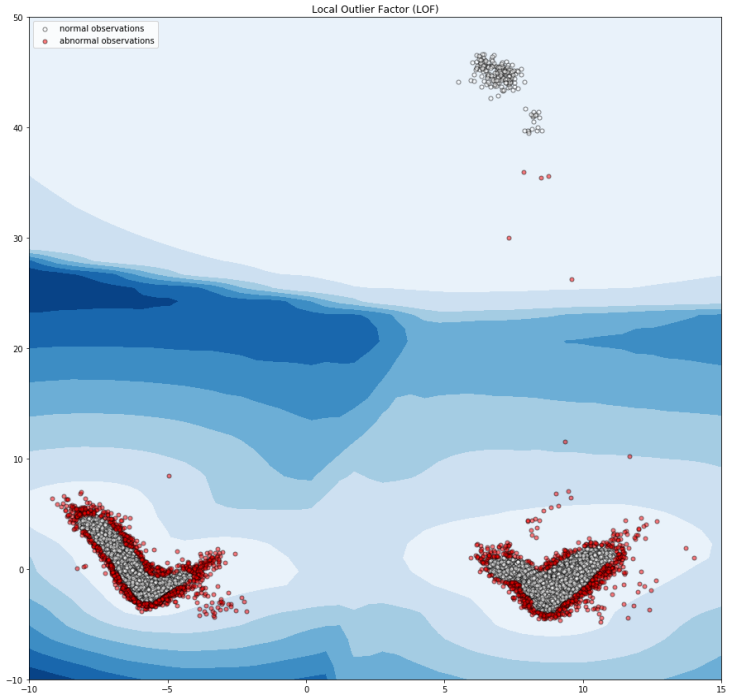

Figure 16. LOF on PSN 21 reduced data set. White data points are normal observations, red data points are hypothesized to be abnormal observations. The blues correspond to probability densities of point being abnormal, where darker blue refers to higher probability of a data point being flagged as abnormal.

Temporal Analysis

The temporal element of our PCA by fleet analysis was investigated at different timescales. As detailed above in the Model1 Findings PCA by Fleet section, the first five eigenvectors are comprised primarily of the following: 1 - pressures, 2 - temperatures, 3 - mixed tags, 4 - vibration tags, 5-mixed tags. 24 hour, 7 day, and 30 day sample timeframes were reviewed. On the 7 day scale, we were able to see regular 24 hour fluctuations in some of the packages. Some showed no fluctuations. Some showed partial sensitivity to time. An example of each are shown below:

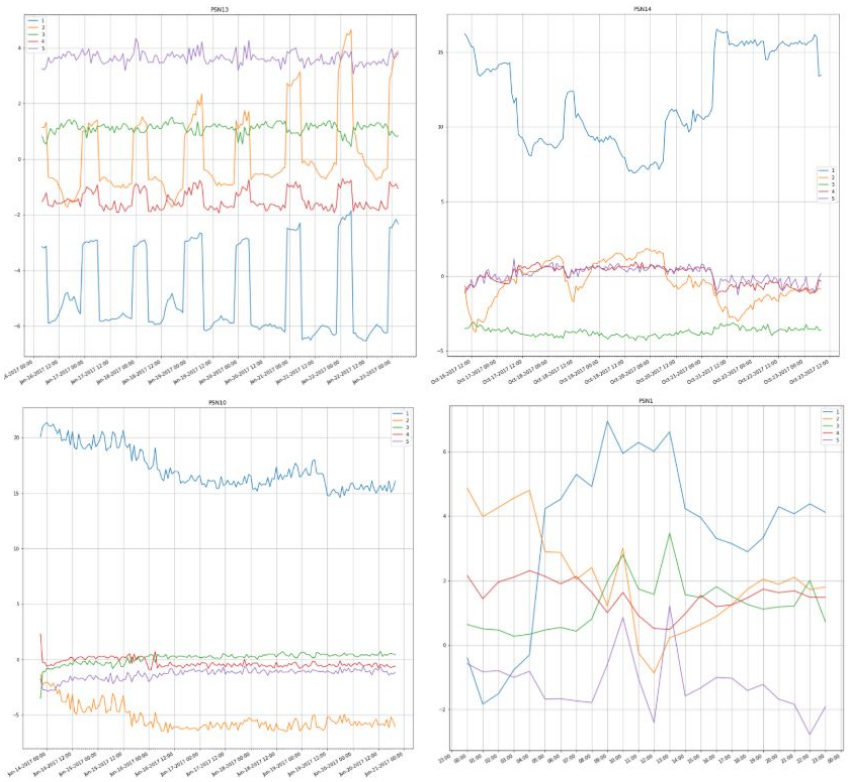

Figure 17. TOP LEFT. PSN 3 shows a strong temporal elements, with 7 peaks over a one week period, particularly in eigenvectors 1, 2, and 4. TOP RIGHT. PSN 14 does not show clear peaks at 24 hours intervals. There are some peaks but not as distinct as in PSN 13 (left) BOTTOM LEFT: PSN 10 shows no clear temporal patterns. BOTTOM RIGHT: PSN 1 for a single day shows strong temporal elements for eigenvectors 1 and 2 as well as weak temporal elements for 3, 4, and 5. Additionally, eigenvectors 1, 3, and 5 peak upwards while 2 and 4 dip downwards during the daytime.

A temporal element was not found in every package nor consistently in a subset of packages. It is unclear why some packages have strong temporal elements, while others do not. We hypothesize that these different results might be due different application types of the individual packages. For Model1, some of the packages are generator sets (to generate electricity) while others are compression sets (used for movement of gases through pipelines).

Table 9. Model1 Temporal Findings (Based on 24 hour analysis)

Temporal Patterning	Count	Package Serial Numbers
Packages with at least 1 of the top 5 eigenvectors showing **strong temporal patterns**	9	3, 6, 7, 13, 17, 21, 22, 26, 28
Packages with at least 1 of the top 5 eigenvectors showing **partial temporal patterns**	10	1, 12, 14, 15, 16, 18, 23, 24, 25, 31
Packages with **none** of the top 5 eigenvectors showing temporal patterns	9	2, 4, 5, 8, 9, 10, 11, 27, 33

Model 2 Findings

Similar to Model 1, principal component and temporal analyses were performed on Model2 data. Due to data quality, the temporal analysis completed was much more in depth and package similarity was also performed.

PCA by fleet

PCA was performed on 1-hr resolution machine data for 38 unique Model2 packages. Each record included a package serial number, timestamp, and 74 features. Of the 74 features, 3 were dropped because of a high number of missing values, detailed below. Each of the dropped features is highly correlated to other features when populated. After removing these sparse tags, records with nulls were dropped, data was standardized using scikit-learn's StandardScaler module and PCA was performed.

Table 10. Model2 Tags removed

Tag	Number of records missing
sc_pct1	56,907
sc_pct2	56,907
sc_c_pct_e1	91,154

When evaluating a fleetwide PCA model, we found that the top 20 principal components retain 88.02% of dataset's variance. 22 principal components are required to surpass the 90% threshold (90.36%), as shown below. The number of principal components and the cumulative percent of the dataset's variance retained is illustrated below.

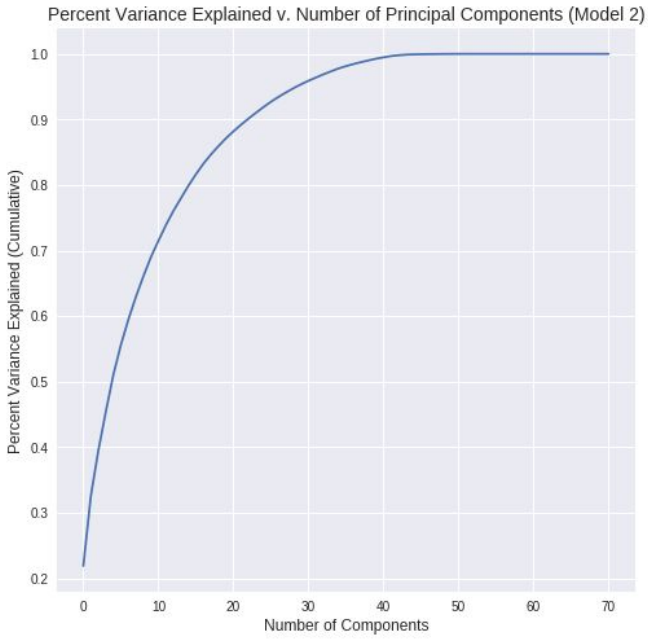

Figure 18. Variance retained versus number of principal components, for Model2 1hr resolution data

The raw data features contributing to the top eigenvectors of both models were identified. They were determined by both measurement type and subsystem type. The first principal component, for example, was identified to be primarily temperature readings. More specifically, they are temperatures related to the load (power output) of the gas turbine. Plots of all eigenvalues corresponding to our first five principal components both color coded by subsystem and measurement type, is available on our GitHub.

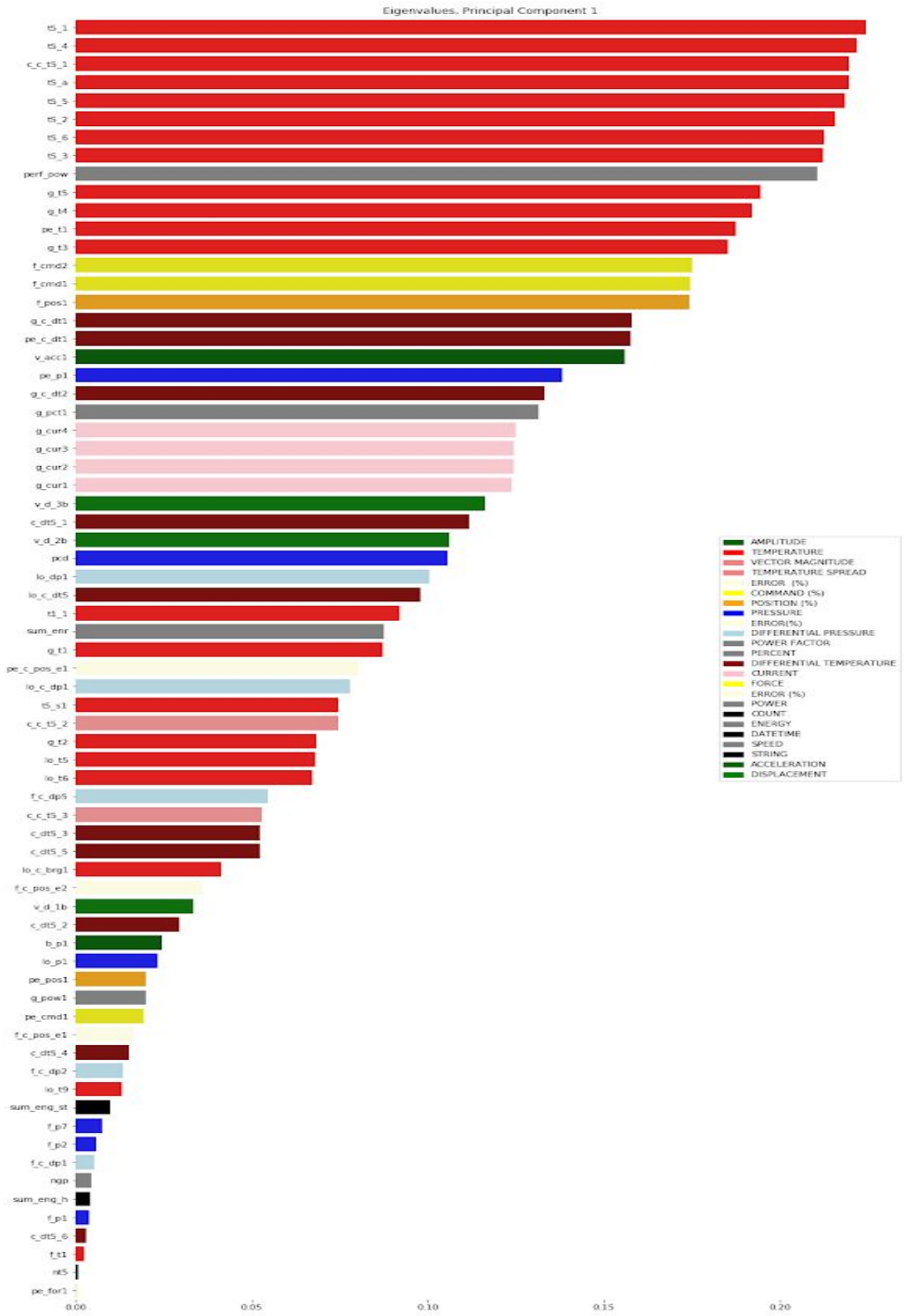

Figure 19. Features contributing the first principal component of Model2 packages (Model2 PCA by fleet) Main contributors are T5 and power, which are tags related to the load (power output) of the packages

A summary of our initial Model1 fleetwide PCA findings is detailed below:

Table 11. Key Findings of Principal Component Analysis on Model2 (Fleetwide model, 1-hr data)

Principal component	Contributing features
1	Predominantly temperatures related to the gas path, fuel system and generator system
2	Controller related tags: command, position
3	Pressure and temperatures related to the fuel and lube oil system
4	Pressure, differential pressures, temperature, differential pressures related to fuel and lube oil systems.
5	Temperatures, differential temperatures, engine hours and starts, currents related to the generator and lube oil systems. Most significantly, the largest contributor is ambient temperature

PCA by Subsystem

Prior to implementing any clustering algorithms on the eigenvectors, principal component analysis by subsystem was investigated. The goal of this exploration was to find a subset of features that would show three distinct groupings when plotted in the reduce space. These three groupings were initially expected to represent the three clusters we saw in the Model1 PCA by package model: one cluster containing points where the package is operating at on-load conditions, another containing points where the package is operating at full load conditions and the third capturing the transients. While the “by package pca” model was informative, it was governed by operating behavior of the single packages. If, for example, a package only operated in one mode, it would be difficult to ascertain whether that mode was on load or full load. For that reason, we investigated subspaces such that three groupings were evident in a PCA by package model. Several different approaches were taken, including PCA on different combinations of subsystems, packages, and select specific tags known to correlate with performance (suggested by domain experts). The figure below shows the first 5 eigenvectors plotted against each other in a scatter matrix, where the PCA model is limited to tags belonging to the generator and package equipment subsystems.

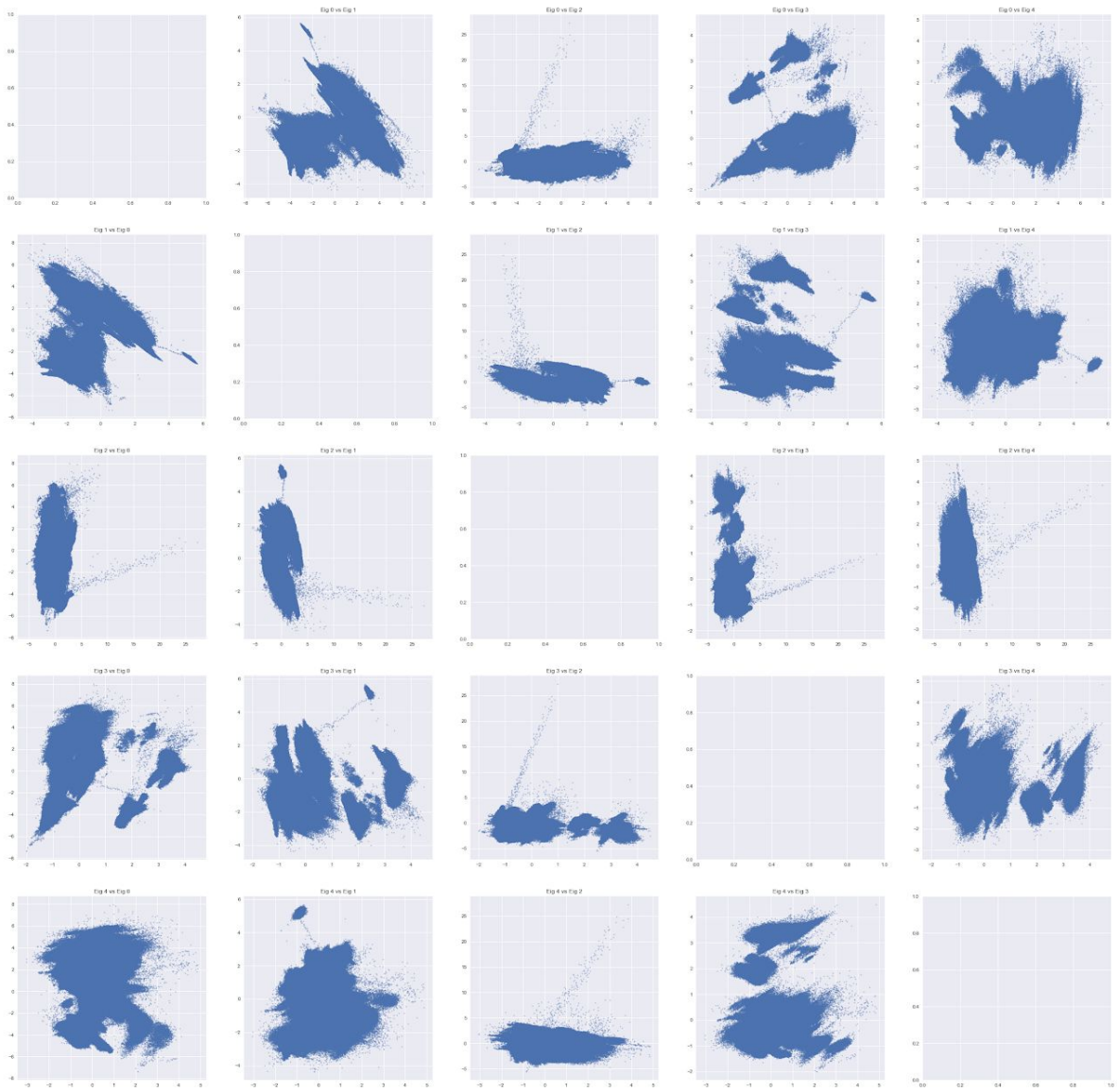

Figure 20. First 5 Eigenvectors on Generator+Package_equipment Subsystems for All Model2 PSNs Scatter matrix. Rows and Columns: Eig0 to Eig4

While the figure above shows some subspaces that have distinct clusters , such as eigenvector 0 by eigenvector 4 (in the first row and fourth column), it was found that these clusters do not correspond to load profile. Employing PCA using only generator and package equipment features change the contributors too much. Specifically, it eliminated the main contributors to the PCA by model “load profile” eigenvector: T5s and power. For this reason, our modeling using principal component analysis with all Model2 subsystems.

Temporal Analysis

The following 24 hour, 7 day, 3 month and 6 month analyses were done using the results from the process:

- Drop sparse packages without any complete days
- Drop engine starts and engine hour tags
- Drop records with any nulls
- Standardize features to 0 mean and unit variance

- PCA on sensor data, each row a point in time with 70 features

The top eigenvectors for Model 2, differing from Model 1, correspond to load profiles (Principal Component 1) and actuator states, which correspond to the positions of the various valves throughout the package (principal component 2). Principal components 3 and 4 are less clear cut, but correspond to pressures.

Similar to Model 1 findings, a consistent temporal pattern was element was not found. Some packages showed regular 24 fluctuations, while others showed weak or none. Examples of each are shown below.

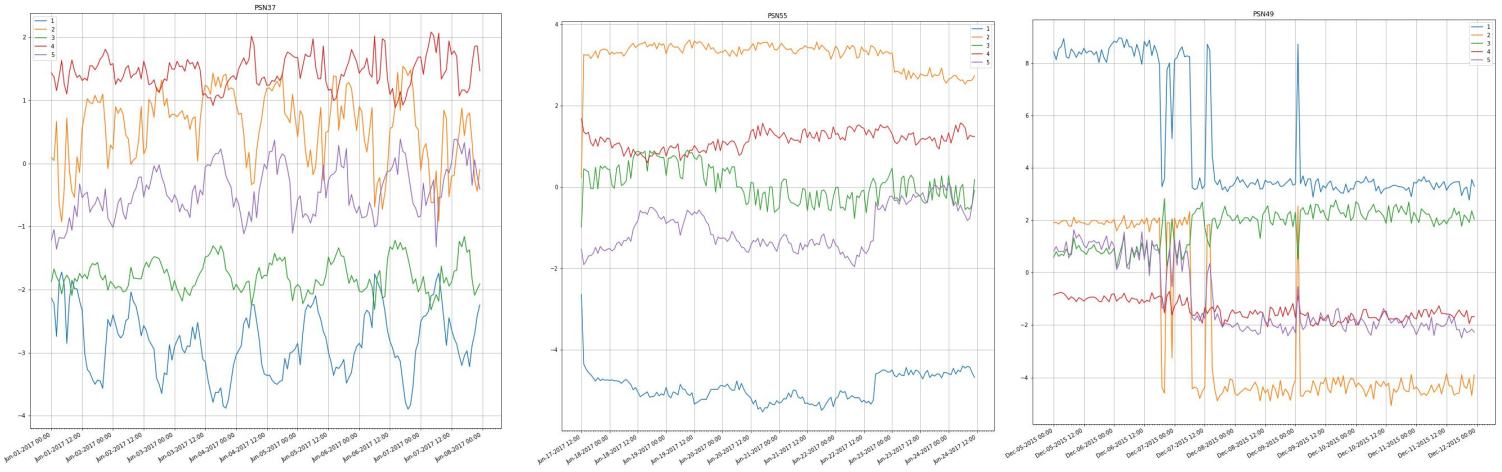

Figure 21. Left: PSN 37 shows clear peaks at 24 hours intervals with 7 peaks over a one week period for all eigenvectors. Center: PSN 55 shows weak temporal influence. Right: PSN 49 does not show obvious temporal influence.

Temporal patterns findings in each Model2 package are summarized below.

Table 12. Model2 Temporal Patterns (Based on 24 hour analysis)

Temporal Patterns	Count	Package Serial Numbers
Packages with at least 1 of the top 5 eigenvectors showing **strong temporal patterns**	15	35, 36, 37, 45, 46, 47, 56, 57, 58, 59, 60, 62, 68, 71, 72
Packages with at least 1 of the top 5 eigenvectors showing **partial temporal patterns**	8	38, 39, 40, 41, 53, 55, 61, 67
Packages with **none** of the top 5 eigenvectors showing temporal patterns	12	34, 42, 48, 49, 50, 51, 52, 63, 64, 65, 66, 69

24 Hour Analysis

For each package and for each of the top 4 eigenvectors, complete 24 hour segments were compiled to track the change in eigenvector coefficients over those periods. When plotted together, these results suggested several types of behavior.

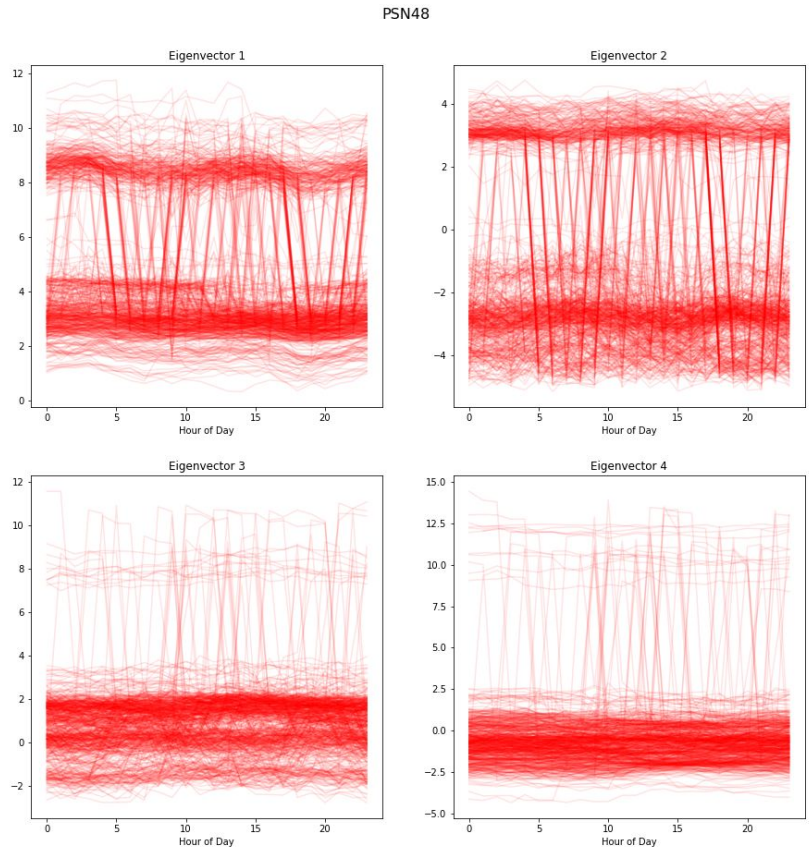

Figure 22. 24 hour analysis of PSN 48 (10 minute data). Each subplot shows a unique eigenvector plotted by hour of day. (top left: eig 1, top right: eig 2, bottom left: eig 3, bottom right: eig 4)

To understand these daily behaviors better, we ran the K-Means clustering algorithm on all 24 hour segments from all packages. For the first eigenvector, there appeared to be approximately 60 clusters. Within those 60 clusters we see variations of 3 different patterns

- A relatively stable daily fluctuation. This pattern represents normal operating and dominates most of the clusters. Each cluster represents a different load on the package. We have found each package typically has a high and low operating modes. (see Figure 24)
- Normal operating interrupted by one or more jumps between operating modes. (see Figure 25)
- Normal operating interrupted by a quick, sharp spike. This pattern represents transient states. (see Figure 24)

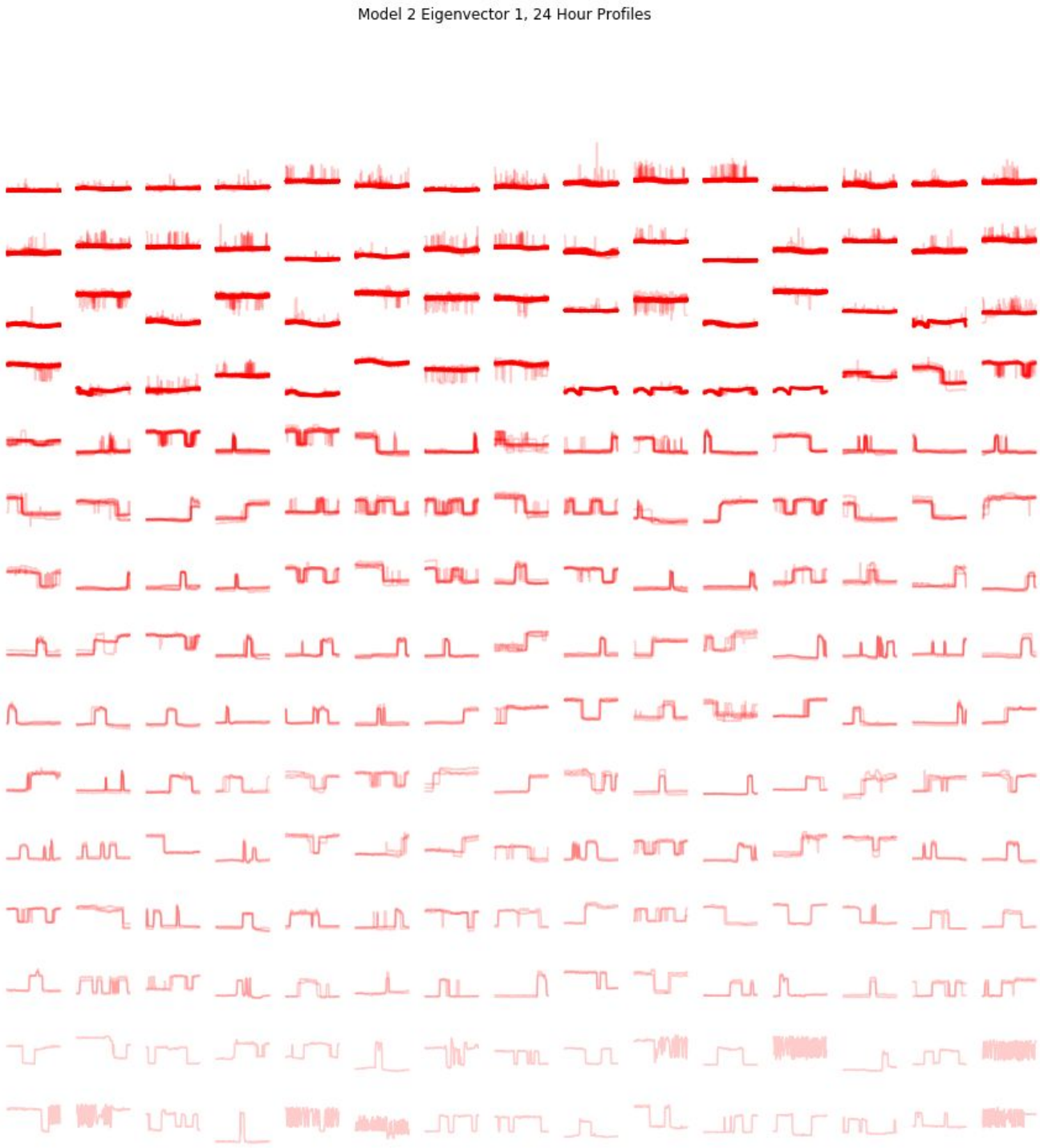

Figure 23. Clusters of 24 hour load profiles (also continuous, Model 2, 10 minute data). The grid is comprised of 225 distinct clusters and is sorted such that the most common clusters (load profiles) are on the top and the less common clusters are towards the bottom. The top rows should steady, single load behavior, punctuated by spikes (transients). The lower rows show step size shifts that correspond to changes in load conditions.

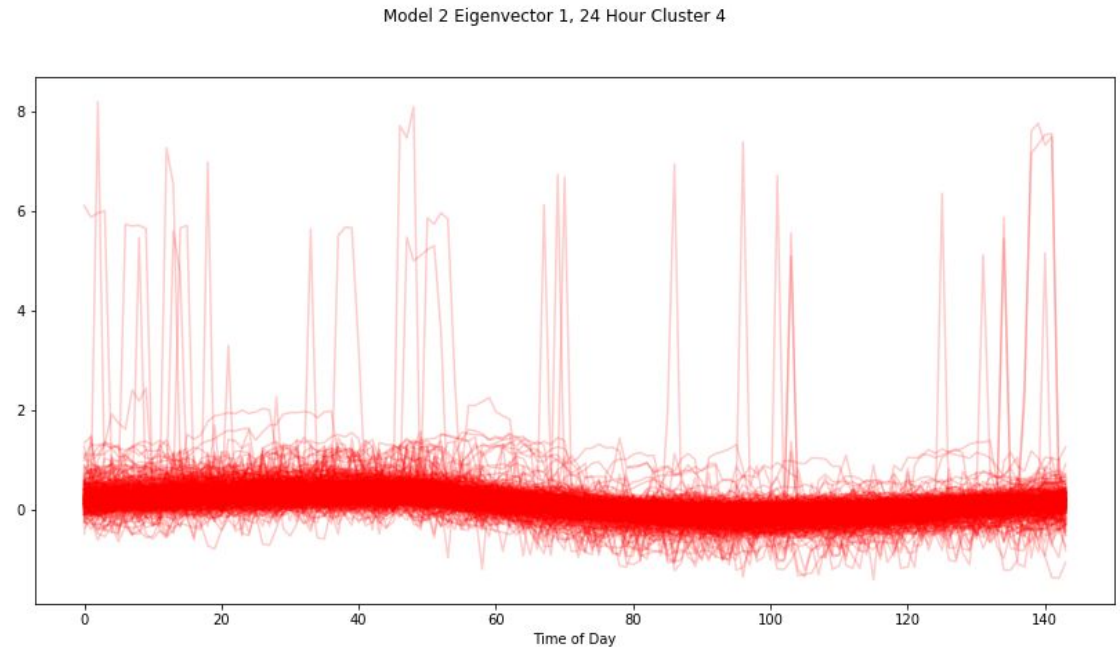

Figure 24. A sample cluster of 24-hour eigenvector 1 coefficients. The mean represents normal operating status at this load. The spikes signify transient states.

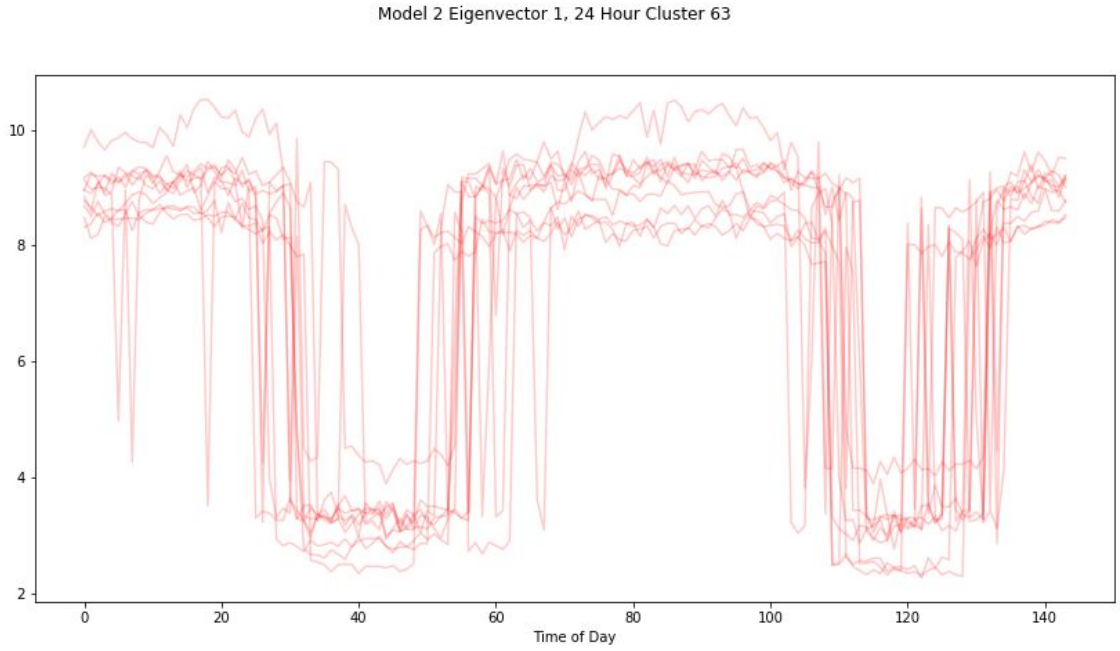

Figure 25. This sample of eigenvector 1 coefficients depicts a group of 24-hour periods with four operating mode switches and many shorter transient states.

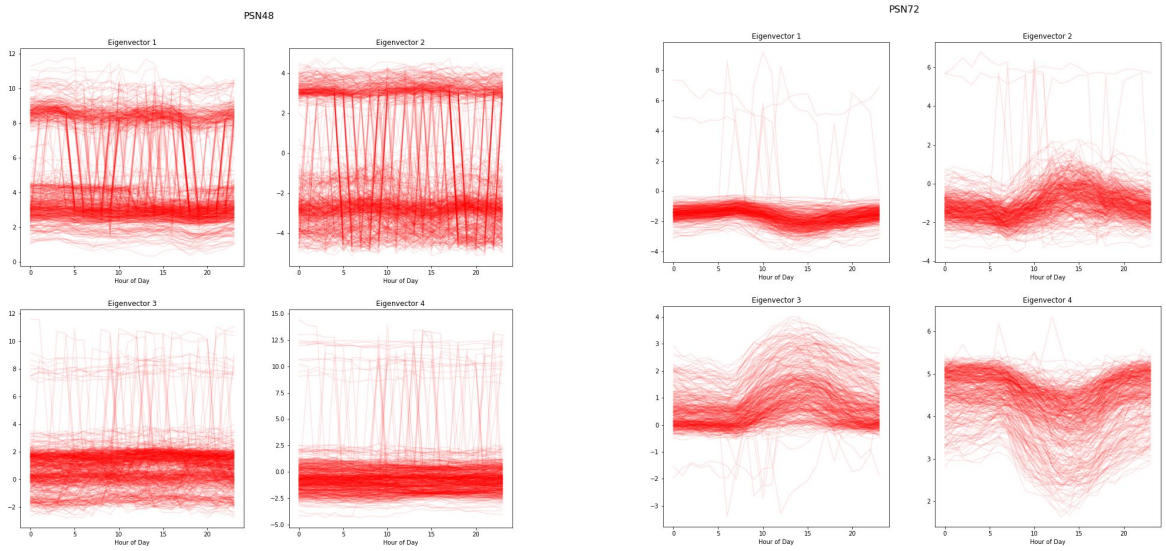

Figure 26. Example distributions of 24-hour load profiles per package for the top 4 eigenvectors. Different PSNs exhibit different operating behaviors.

7 Day Analysis

An in-depth exploration of the first two eigenvectors in seven day increments was completed. It helped visualize normal operating behavior, as well as identify transient events, as shown in the example below (Figure 27). Here, we can see a large shift in both eigenvectors 1 and 2 on Wednesday, November 30th.

3 Month and 6 Month Analysis

Using 10 minute data from Model 2, we plotted 3 and 6 month spans of eigenvector 1 and 2 coefficients, colored by day of week. In addition, a scatter plot of the 2-d space was plotted using the same coloring. In the 3 month plots, the points are connected by edges to show movement over time. In the 6 month plot, each day is averaged to a single point. See Figure 28 for examples of each.

No patterns by day of week are evident, nor are longer term trends or patterns. These initial perceptions were confirmed by our domain experts. However, the three main patterns seen in the 24-hour PCA + K-Means analysis are also seen here.

Package Similarity

This analysis was completed to determine which clusters (30 min segments) and which packages are most similar to one another. We generated cluster maps using euclidean distance as the distance metric. The analysis was completed on the 30 min segment load profile cluster results for eigenvectors 1-4, below the clustermap for eigenvector 1 (n_clusters = 150) is shown below (Figure 29). Cluster 60 & 86 are close to one another based on euclidean distance, thus suggesting that the data shape is similar for the 30 min segments and packages 63, 50, and 51 are close to one another based on euclidean distance, suggesting that these might be at the same site, operated by the same customer for the same application. Python functions were constructed to provide users means of finding the most similar packages based on a selected package, time segment, and number of clusters.

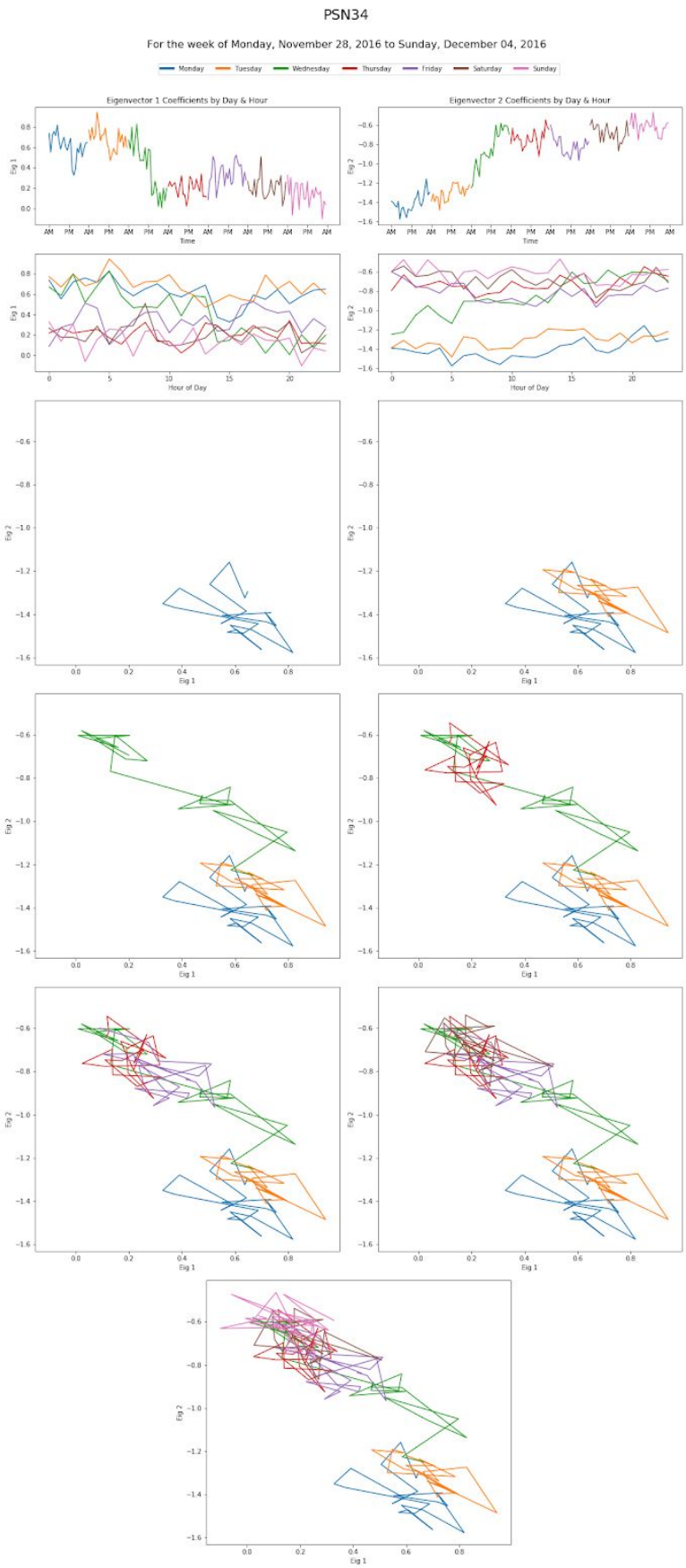

Figure 27. In-depth temporal analysis of PSN 34. All plot color coded by day of week. Top left: eig1 over one week. Top right: eig2 over one week. Middle left: eig1 over one day. Middle right: eig2 over one day. Bottom plots: eig2 over eig1 for day of week (Mon, Mon-Tues, Mon-Weds, etc.)

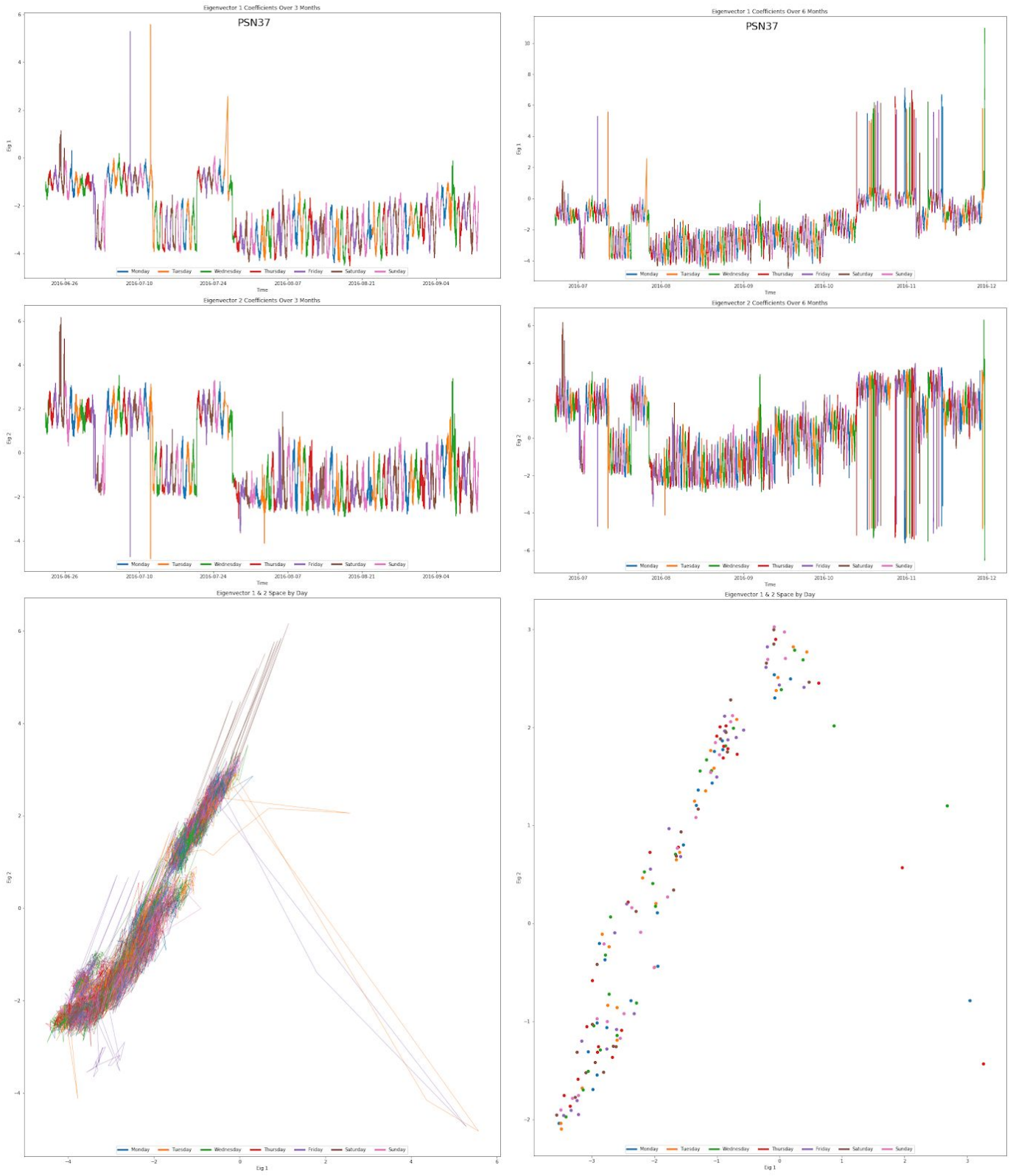

Figure 28. 3 and 6 month analysis of PSN 37 (10 minute data). Each datapoint is color by day of week. Top left: eigenvector 1 over 3 month time span. Middle left: eigenvector 2 over 3 month time span. Bottom left: eigenvector 2 over eigenvector 1 for 3 month time span. Top right: eigenvector 1 over 6 month time span. Middle right: eigenvector 2 over 6 month time span. Bottom right: eigenvector 2 over eigenvector 1 for 6 month time span.

Figure 29. Principal Component 1 Clustermap. X-axis: package serial number. Y-axis: cluster number. The top dendrogram shows hierarchical clustering of the different packages, and the left dendrogram shows hierarchical clustering of the different clusters. Darker blue corresponds to greater density of times segments

Tableau Dashboards

The following Tableau dashboards were developed to allow for exploratory data analysis. Three interactive dashboards were developed to allow team members and domain experts to visualize the data: *Top 5 Principal Components*, *Reduced vs. Raw Datasets, and Clustering of the Reduced Dataset.*

Top 5 Principal Components

In order to identify potential turbine functions associated with each of the principal components, a side by side bar chart was created. The tags for each principal component can be sorted by most to least important for a particular principal component. Furthermore, they are color-coded by measurement type. Using this visualization we were able to determine the features which make each each eigenvector, as noted above.

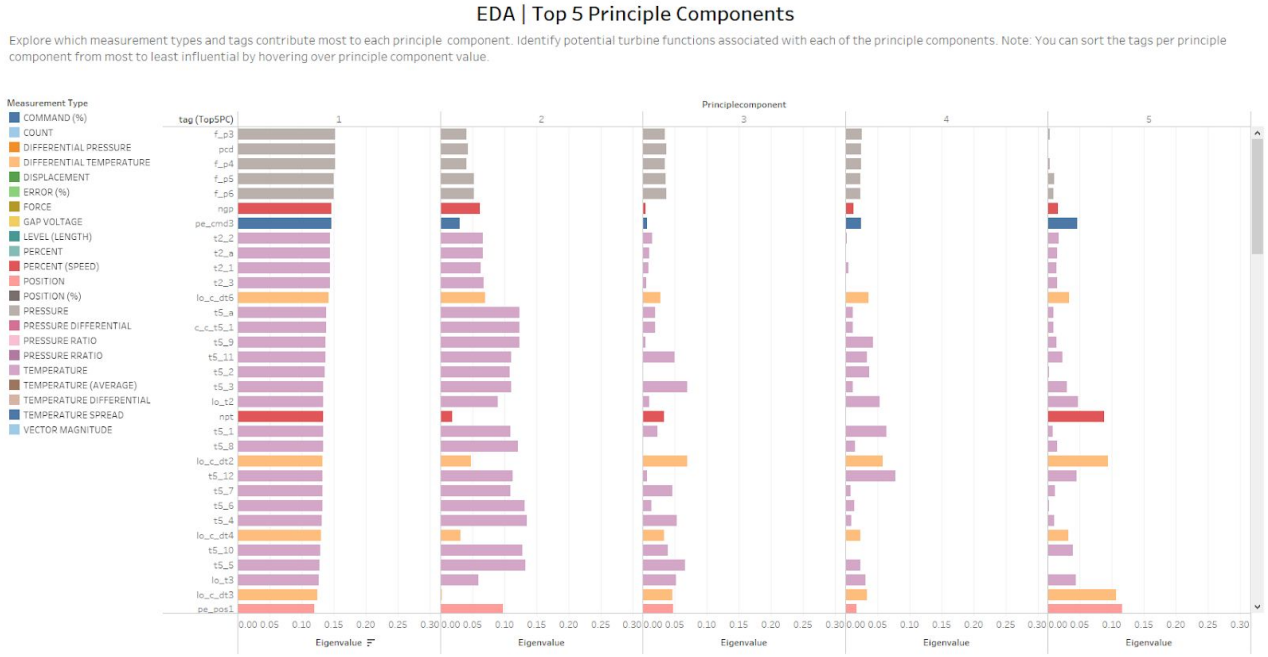

Figure 30. Top 5 Principal Components Visualized through Tableau

Reduced vs. Raw Dataset

In order to identify relationships between tags and eigenvectors, two line charts were developed to provide a view on the top n eigenvectors contributing to a particular package, as well as the raw data showing the trend of tags over time for each package. For deeper exploration, users can first go to the Top 5 Principal Components dashboard, determine which tags they are interested in, select the tags on the Reduced vs. Raw Dataset dashboard and also specify the eigenvector of interest and then have an understanding of how tags vary with time and might affect the eigenvector. Furthermore, by default all timestamps are shown but an individual can explore different daily, weekly, etc. trends of interest by specifying a time range, as can be seen in the figure below. This dashboard was used as an early prototype of what evolved into our JavaScript dashboard (see Exploratory JavaScript Dashboard)

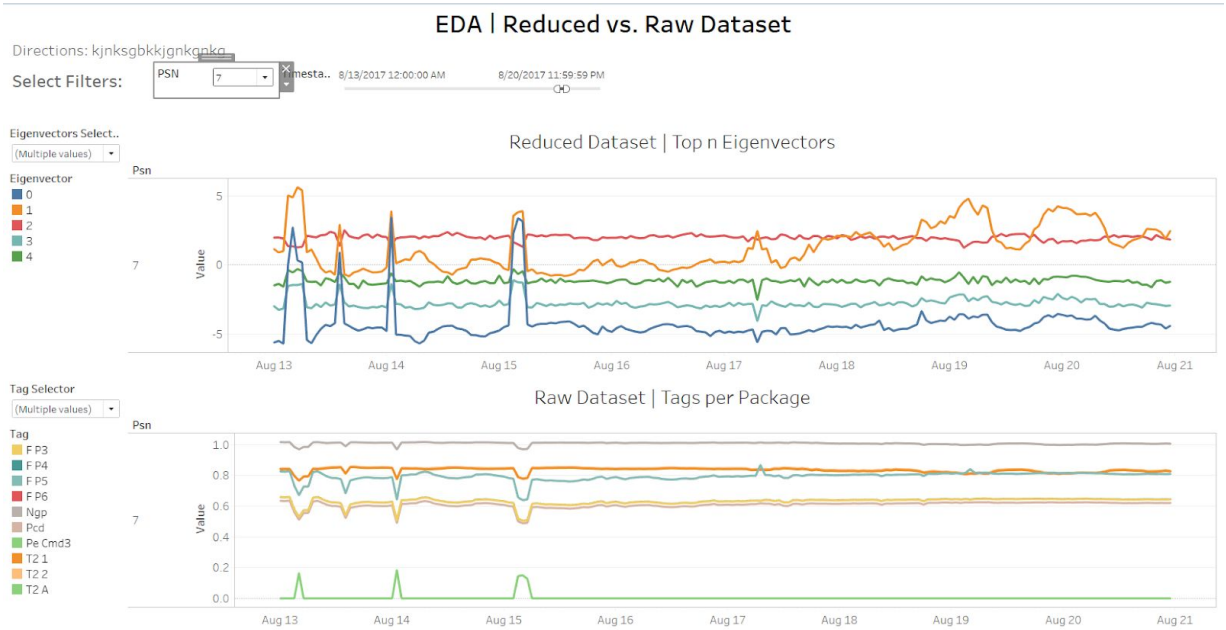

Figure 31. Reduced versus Raw Dataset Visualized through Tableau

Clustering of the Reduced Dataset

In order to test different clustering algorithms, cluster parameters, and validate clustering results, we have developed a dashboard which shows the different quantitative values (top 20 features of the PCA by Fleet Model 1 reduced dataset) within a parallel coordinate visual where each line is associated with a particular timestamp. This visualization intends to provide an quick means of visualizing timestamp clusters within a particular package.

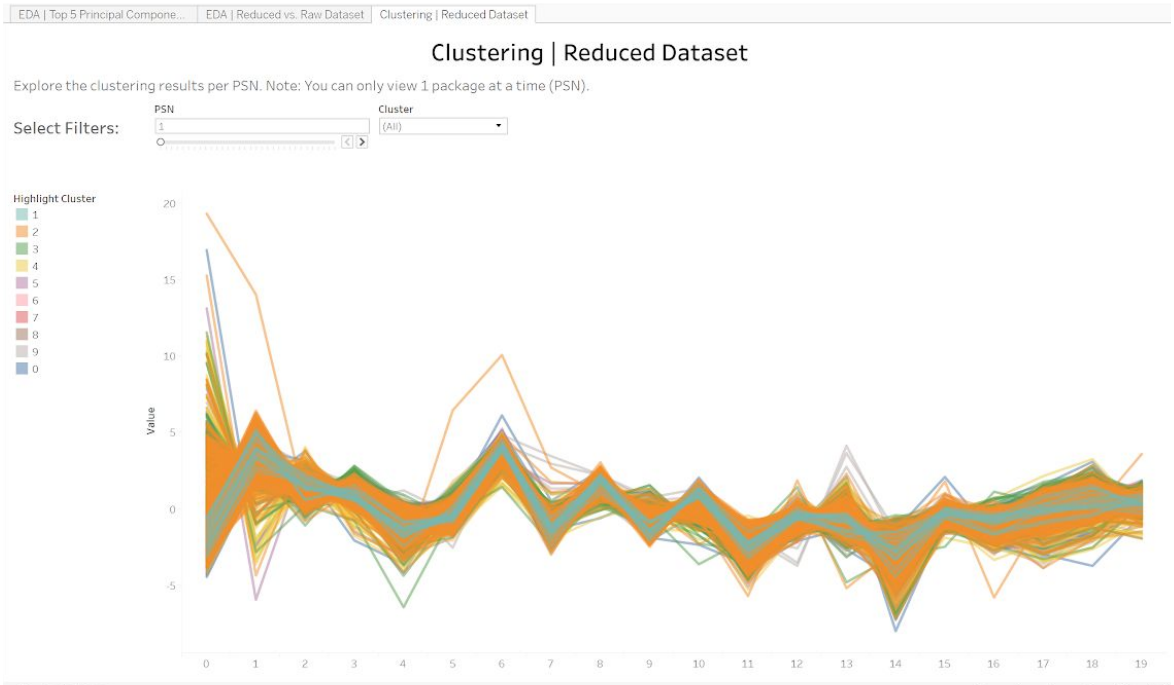

Figure 32. Clustering of the Reduced Dataset Visualized through Tableau

Exploratory JavaScript Dashboard

An interactive JavaScript dashboard was built to allow users to map the raw machine data to and from the PCA reduced space. Users can select packages of interest, raw features of interest (plotted on the top) and eigenvectors as axes to see the reduced data (below). When selecting raw data in the timeline, a green graph animation is created in the reduced space plot to explore changes over time. When one highlights groups of points in the reduced space, the corresponding raw data points are highlighted in red. Additionally, the PCA reduced data set has been colored by time of day, where brown points have evening timestamps and blue/green points were recorded during the day. Examples are included below.

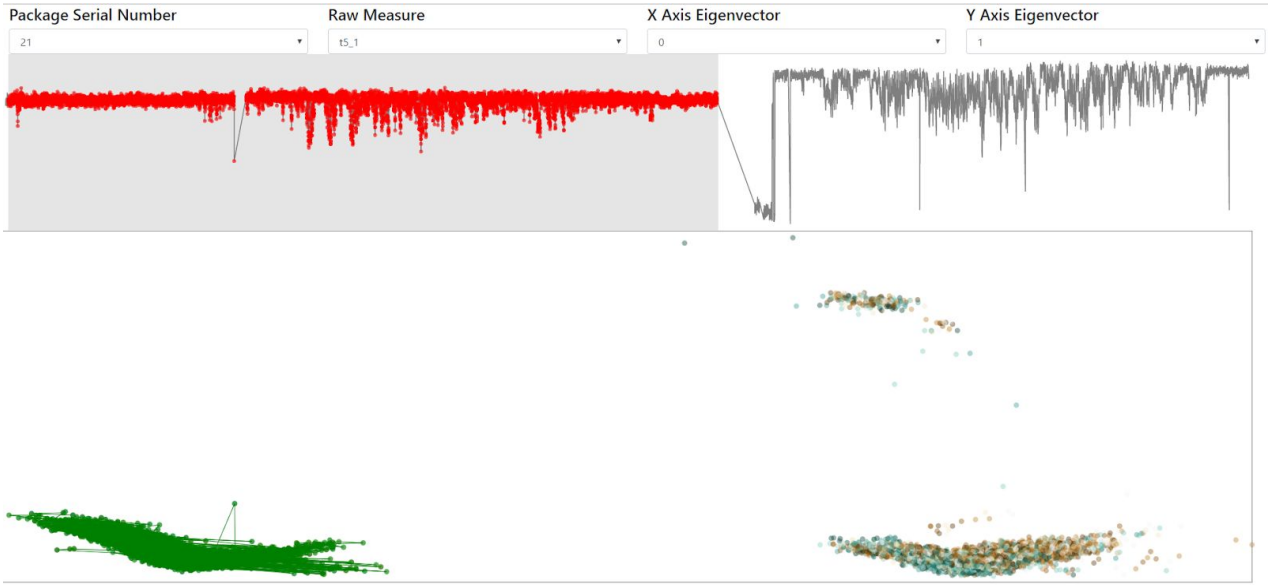

Figure 33. Javascript dashboard. Clusters in the PCA data are shown to correspond to operating modes in the raw data.

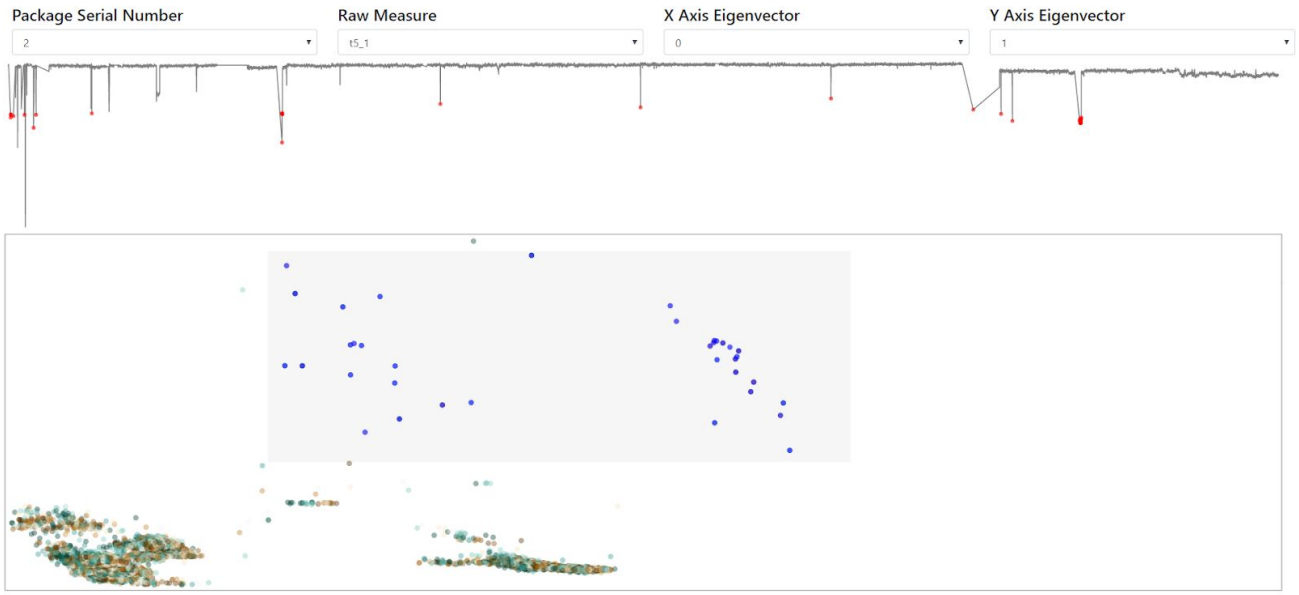

Figure 34. JavaScript dashboard. Outliers in the PCA data are shown, in some cases, to correspond with extreme values in the raw data.

Our Product

Our product is a scalable pipeline for the detection of transients in Solar Turbine's machine data. The product is cyclical because each of the elements form a feedback loop. As domain experts validate our results, the process can be repeated. The components are as follows:

Figure 35. Our Product

Raw Data Query

All data is stored in a PostgreSQL database. The data size did not require a distributed storage solutions, so each of use and cost were primary factors in determining storage solutions. As such, the 1 hour and 10 minute data is currently hosted on an AWS EC2 t2 large instance.
Data is accessible using the following API:

Anonymized Data

Import Interface

import_csvs(directory, destination_table, drop_create_table)

Query Builder

Filtering

.not_null(column_name)
.psn(serial_number)
.model(model_number)
.frequency(f) # '1hr' or '10min'
.min_time(timestamp)
.max_time(timestamp)

Projection

.select(feature_list)

Exporter

.save_df(df, filename)

.save_pkl(obj, filename)

.save_fig(fig, filename)

Programmatic access enables interfaces that abstract away storage details. Coding against the abstractions proposed above allows the underlying storage to change as needed for scalability. For more details on our raw data query process, see the Data Pipeline section.

Data Transformations

Data Transformations refer to all measures taken to manipulate our raw data into formats that can be used for our end goal of identifying transient states in Solar Turbines' packages. This includes data preprocessing, dimensionality reduction, segmentation, clustering, statistical analysis, and label generation, as shown below. For a detailed explanation of all methodologies employed, see the Modeling section. Additionally, scalability measures for transformations are detailed in the Pipeline section.

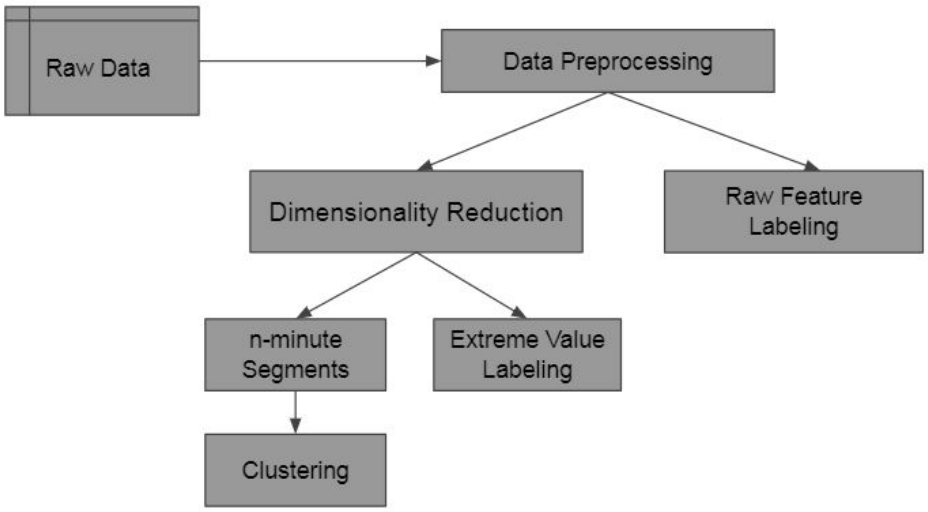

Figure 35. Data Transformation Overview

Derived Feature Cache

Data transformation output can be stored flexibly depending on use case. Local storage and AWS S3 buckets were explored as options. See the Pipeline section for more details.

Transient Ensemble Scores

Labels generated from various data transformations are used to assign transient scores, which can be mapped to the raw data by timestamp and package serial number identifiers. They are combined, weighted and evaluated against a transient threshold, thus generating ensemble labels. See Modeling section.

Visual Validation

Our lack of domain expertise and labelled data requires us to rely heavily on our domain experts for model validation. We built a user interface for domain experts to visualize, interpret, validate and annotate our results. See User Interface section.

Modeling

Model was built using Model2 10-minute data. The generic pipeline is as follows:

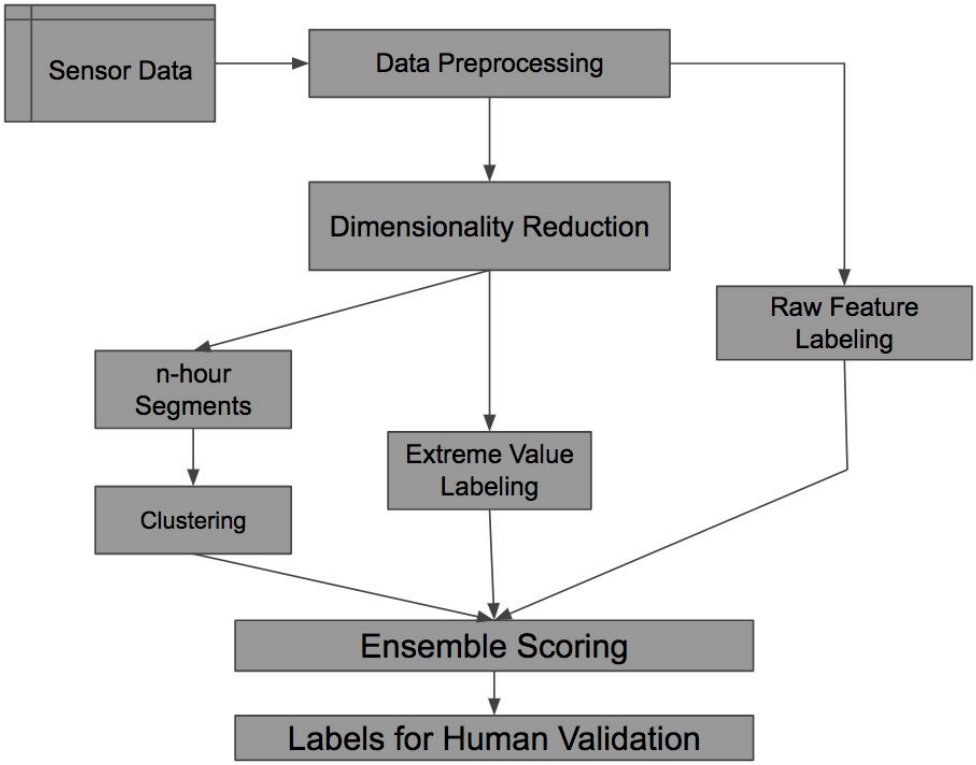

Figure 36. Data Transformation Pipeline (Procedure Used for Modeling)

Data Preprocessing

Preprocessing steps are as follows:

- Columns set as multi-indices: ‘psn’, ‘timestamp’
    - Package serial number and timestamps are unique identifiers to index any record in our dataset
- Ignored columns: ‘id','SUM_ENR','SUM_ENG_ST','SUM_ENG_H','SUM_ESN'
    - ‘SUM_ESN’ and ‘id’ were ignored because they are index-type columns
    - 'SUM_ENR', 'SUM_ENG_ST', and 'SUM_ENG_H' were excluded because they are incrementors and not physical readings
- Drop sparse packages: psns 44, 52, 54, 70
    - These packages were dropped due to low counts of records and data incompleteness

- Drop sparse features: None
    - Any features that had a percentage of nulls greater than a defined threshold are dropped. For the Model2 10 minute dataset, no features were > 10% null.

Raw Feature Labeling: Power Jumps

One of the raw features in our data set is power. Transients can be defined as a sudden change in load, or power output. As such, we used one of our statistical functions, henceforth referred to as “Power Jump”, to identify sudden spikes or dips in this raw feature. Specifically, all points where power changed by more than 25% were labeled transients. For example, the figure below shows power over time for PSN 57 in blue and timestamps where power changed by more than 25% as spikes in yellow.

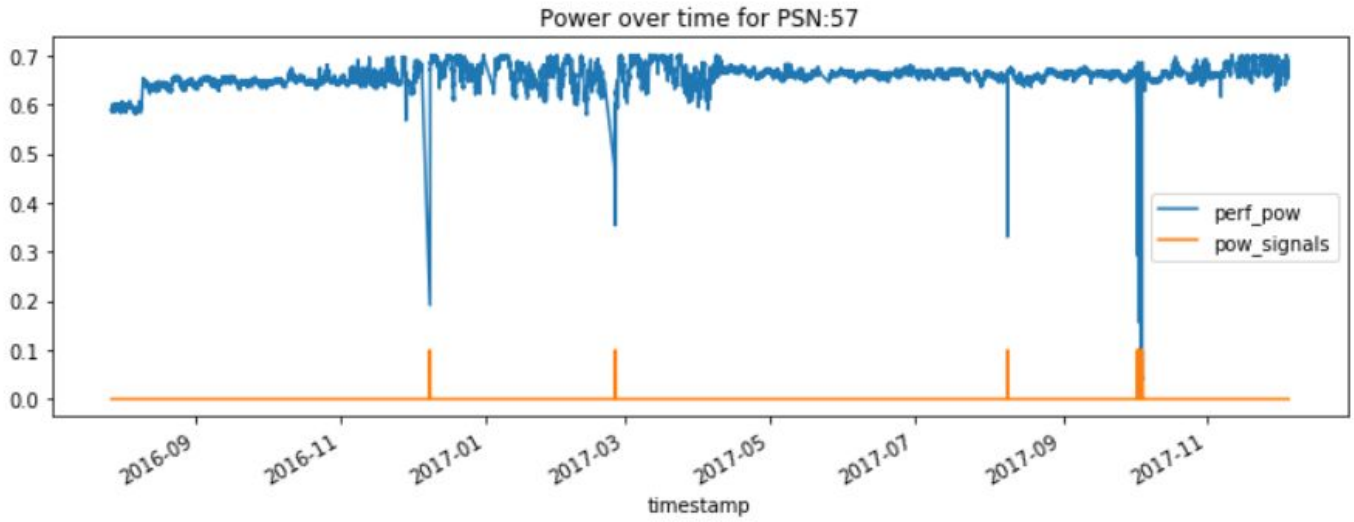

Figure 37. Power (blue) and Power Jump labels (yellow) over time for PSN 57

In addition to providing insight into when transients occurred, this metric provides a baseline for comparison against other transformations. This metric is referred to as “Power Jumps” throughout this report and in our user interface.

Dimensionality Reduction

Principal Component Analysis (PCA) was used both to reduce the dimensionality of our data and to better understand the data. Prior to implementation, each feature was first standardized to have a zero mean and unit variance using the sklearn’s StandardScaler transformation. This was needed as in PCA we are interested in the components that maximize the variance. If one component varied less than another because of their respective scales, PCA might determine the direction of maximal variance more closely corresponds with the feature with bigger scale. After this, PCA was applied using sklearn’s PCA implementation. Essentially the data was projected into an alternative basis to reduce the number of features while maintaining maximum variance. For more information on the underlying math and PCA algorithm, see Jolliffe 2002[6].

[6] "Principal Component Analysis | I.T. Jolliffe | Springer." https://www.springer.com/us/book/9780387954424. Accessed 20 May. 2018.

HDBSCAN Clustering

Model Selection

The team initially considered a variety of unsupervised clustering models, including but not limited to Spectral Clustering, Affinity Propagation, Mean Shift, Agglomerative Clustering, and DBSCAN. However, due to hardware requirements and performance concerns, the team chose to use HDBSCAN (Hierarchical Density-Based Spatial Clustering). The increased capabilities and increased performance compared to other algorithms, aside from K-Means, made it extremely useful. Specifically, HDBSCAN is an improvement upon DBSCAN that converts it to a hierarchical clustering algorithm, then extracts a flat clustering based on the stability of clusters[7]. It also features memory caching to further speed up clustering.

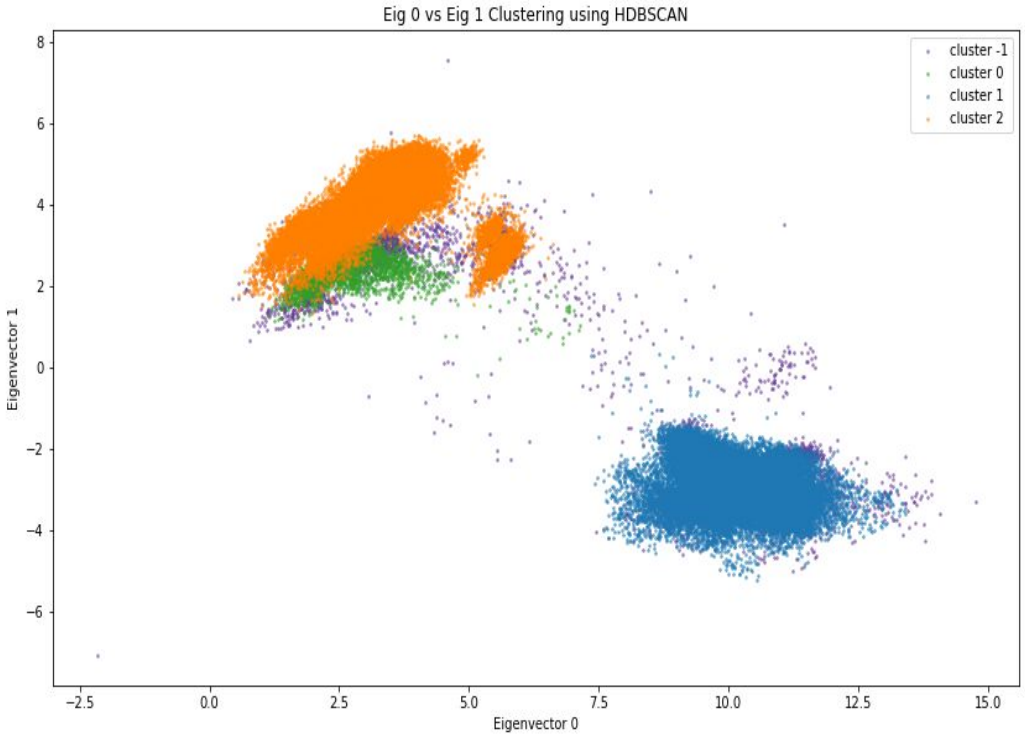

Figure 38.HDBSCAN Clustering. Orange, Green, and Blue clusters are different operating states. Transients do not belong to any operating modes so Purple points, which are labeled “noise” by HDBSCAN, are used for detecting transients

Parameter Tuning

Iterations of HDBSCAN were evaluated using various combinations of input parameters. HDBSCAN has two main parameters which affect the resulting clusters: min_cluster_size and min_samples. Min_cluster_size is a restriction on the size of the cluster whereas min_samples determines the minimum number of datapoints required to be considered a cluster. A feature of HDBSCAN is that if certain datapoints are discovered to not belong to any cluster, they are labeled as “noise”. Because the number of

[7] "How HDBSCAN Works — hdbscan 0.8.1 documentation." http://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html.

normal operation datapoints vastly outweighs the number of transients, we tuned HDBSCAN to cluster operating modes and “noise” labels were called transients.

We validated results from different combinations of parameters against the Power Jump results and then calculated an accuracy score based on the number of non-transients correctly identified. Accuracy scores were originally calculated based on correctly identified transients only (as defined by the Power Jumps metric). However, we immediately noticed cases where certain parameter values would result in every datapoint being labeled as a transient, and thus giving perfect scores. Even scenarios where not all datapoints were labeled transients, the count of transients classified by HDBSCAN would be greater by one or two orders of magnitude. To account for this, we filtered out parameters which resulted in overly large numbers of transients classified and then ranked parameter combinations. Parameter combinations were ranked using the process below:

1. Transients found by the Power Jump function were mapped to the corresponding indices classified by HDBSCAN and an f1 score was calculated.
2. Similarly, transients found by HDBSCAN were mapped to the corresponding indices classified by the Power Jump function and another f1 score was calculated.
3. These two scores were multiplied together and ranked from highest to lowest score.

By filtering and scoring both ways, we account for false positives classified by HDBSCAN and are left with tuning parameters min_cluster_size=20 and min_samples=80. A few packages were required more specialized tuning and so different parameter combinations were derived for those.

Extreme Value Labeling

Model Selection

Transients were first identified during the EDA phase in eigenvector0 and eigenvector1. While they are visible in both reduced features, they are more pronounced in eigenvector0. This intuitively makes sense because the main contributors to eigenvector 0 are related to load (power, t5, etc).To identify sudden changes in this parameter, we relied on our statistical function to find step size jumps in our features over time. This function inputs a rolling window and calculates a rolling mean and rolling standard deviation for that time period. It also requires users specify a z-score threshold, the number of standard deviations outside of the mean used to delineate transients from normal operation.

Parameter Tuning

Both the rolling window and z-score threshold were tuned. Lack of true labels required us to visually assess the accuracy of this portion of the model. For fine tuning the z-score threshold, a rolling window of 24 hours was used. For fine tuning the rolling window parameter, a z-score threshold of 5 was used. In Figure 39, the blue line shows eig0 over time for PSN 55. The yellow line captures transients identified by the step-size function, where the spikes correspond to step size jumps in power, using the various z-score threshold and rolling windows.

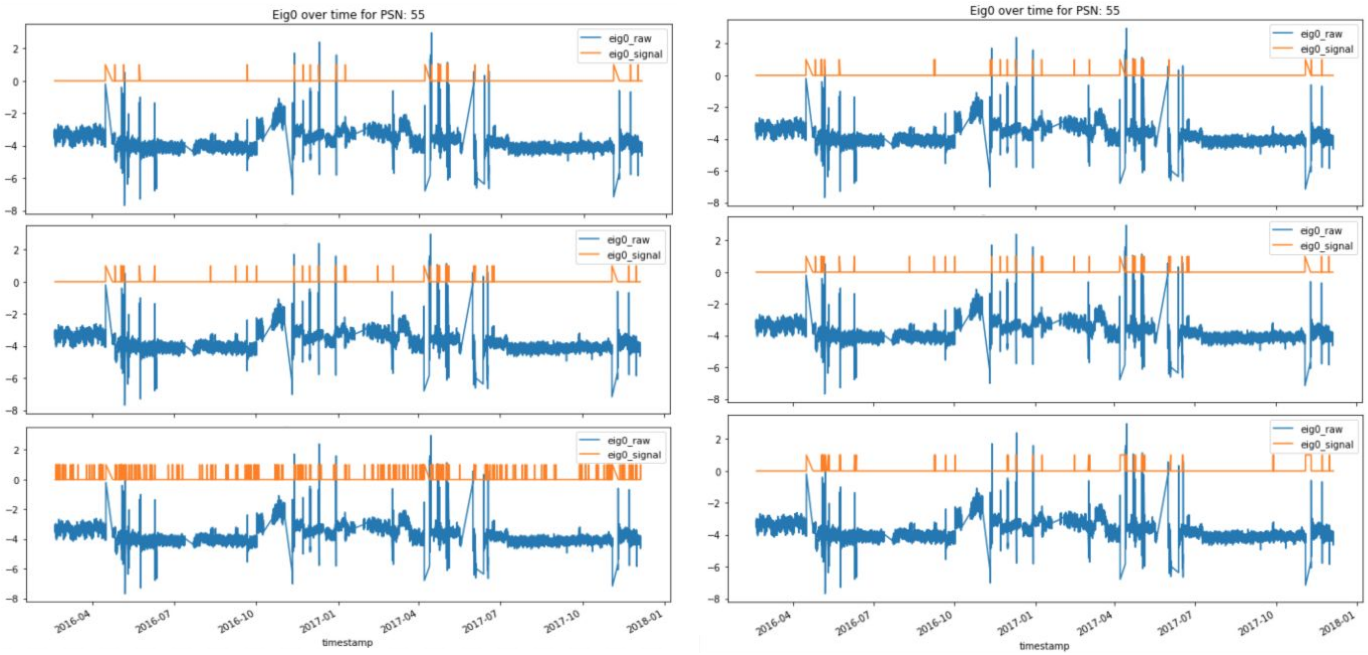

Figure 39. LEFT: Fine tuning the z-score threshold. All figures show PSN 55 with a 24-hour rolling window and share a common x-axis (time). Top: z-score threshold = 8 standard deviations. Some obvious peaks are missed, particularly in 2016-06 and 2017-03. Middle: z-score threshold = 5 std deviations. Slightly sensitive but all peaks are labeled transients. Bottom: z-score threshold = 3 std deviations. Many peaks are incorrectly identified to be transients. RIGHT: Fine tuning the rolling window threshold. All figures show PSN 55 with a 5 std deviation threshold and share a common x-axis (time) Top: rolling window = 10 hours. Some obvious peaks are missed, particularly in 2016-06 and 2017-07. Middle: rolling window = 24 hours. Slightly sensitive but all peaks are labeled transients. Bottom: rolling window = 10 days. Missed peaks in 2016-12.

Analysis was applied to all Model 2 packages, and a rolling window = 24-hours and a z-score threshold of 5 standard deviations was used for our final model. These input parameters are not hard-coded can be tuned based on feedback from domain experts

n-Minute Segmentation & Clustering (Kink Finder Labeling)

A third method of identifying extreme values in the reduced space was developed during EDA in the process of summarizing and categorizing package behavior. The classification occurs in three steps, starting with an eigenvector coefficient over time.

- Partition the coefficient data into 20 minute segments
- Cluster the 20 minute segments
- Apply binary labels to the clusters based on a percent change of the cluster's mean value.

The first step provides a temporal context of comparison. Rather than just looking at a single value at a single point, look at the change in value between points. The second step categorizes that change and the third step uses a simple calculation to label whether each category has relatively stable values or not.

Figure 40. Sample 20-minute cluster plots. The center line is the cluster mean and blue highlights +/- 2 standard deviations. Two distinct normal operating modes are seen (high and low flat lines). Two more variable clusters are labeled as transients (kinked lines)

This model has the added benefit of providing a convenient metric for comparing package operational “modes”. With the change in state across each 20-minute segment categorized, summarizing how a package is behaving becomes easier, both across the lifetime of a package and between packages. See Results and Interpretation section for further work on Package Similarity.

Model Tuning & Variations

Segment Length

The time series segmentation was originally run on 24-hour ranges to get an idea of how packages behave on a daily basis. Once the goal of labeling transients became clear, 60, 30 and 20-min ranges were generated. Both 60 and 30-minute segments were generated starting on the hour and finding groups of 3 and 6 data points as applicable. The 20-minute segments were generated by pairing data points with the subsequent point, if available. During all segmentation processes, only complete time segments were kept. This resulted in the 20-minute segments providing much more data for the clustering algorithm to work on. In addition, since all points in a cluster receive the same label and transients are typically 10 minutes or less in duration, the 20-minute segments were best for labeling transient points as specifically as possible.

Table 13. Model2 10-minute data, Value Counts of Complete Records Segmented into Various Time Intervals

Segmentation	Complete Segments	Timestamps Labeled	Clean Data Set Coverage
12-hour	20,798	1,497,456	93.45%
60-minute	265,741	1,594,446	99.51%
30-minute	533,007	1,599,021	99.79%
20-minute	1,600,590	1,602,238	99.99%

Clustering Algorithm

The time segments were then fed into the K-Means clustering algorithm. The number of clusters was run between 5 and 150 to generate statistics in order to optimize. The plot of the sum of the square distances between points and their cluster centers (inertia) is shown below (Figure 40). Based on those results, n_Clusters = 25 was chosen. Visual validation in the dashboard confirmed those are fairly clean clusters. K-Means was chosen as a computationally efficient algorithm for the task.

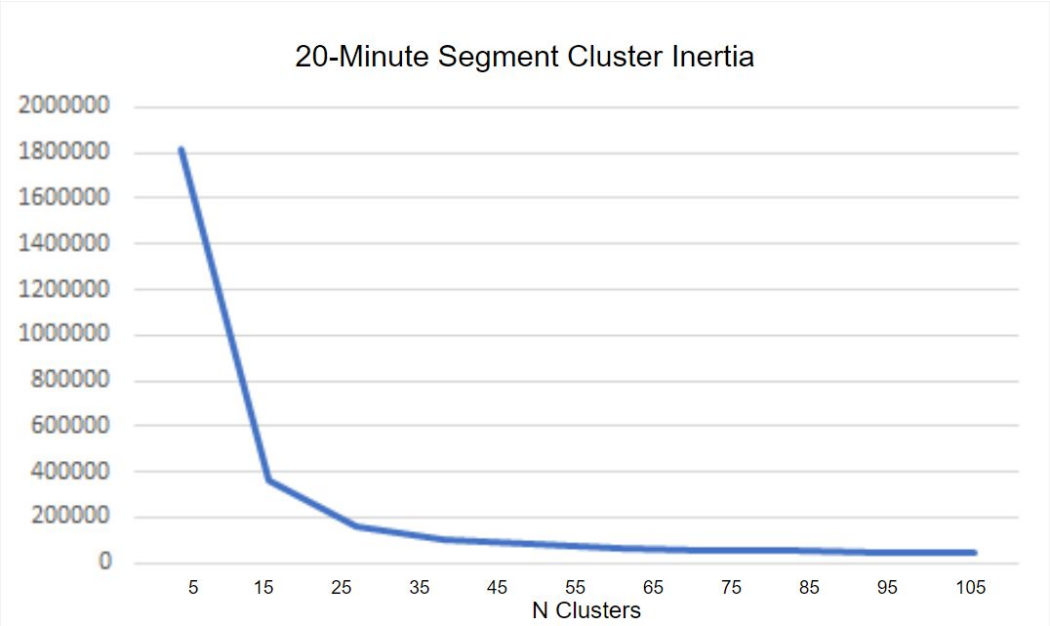

Figure 41. Cluster analysis. Inertia versus number of clusters (x-axis = n_clusters, y-axis = interia)

Kink Threshold

The final step to these “Kink Finder” labels was to determine the percent change between the mean minimum and mean maximum points in each cluster. The transient/not transient labels were generated comparing that change to a static threshold, found to be 0.4 where the change is at least 0.1 of the difference in the global minimum and maximum values. Visual validation also confirms this to be an accurate threshold.

Ensemble Scoring

Each of the approaches assigns boolean transient scores to every datapoint, with 0 being normal and 1 being transient. Boolean scores allow for easier comparison for different combination schemes to create the best ensemble score. Individual scores were combined to create a baseline ensemble score. This score is compared to a transient threshold threshold value. Data points with ensemble scores greater than or equal to the threshold are labelled as transients, and those below the threshold are labelled as normal in our ensemble model. Due to the lack of true labels, model validation requires domain expertise. Additionally, the Power Jump labels are used as both a feature in our ensemble model and a metric by which the labels are evaluated. This is a known limitation in our process, as Power Jumps are not entirely ground truth. Using Power Jumps to fine tune our model would led to the Power Jump feature being weighed to 1 and the other metrics weighted to 0. For this reason, the ensemble labels were not fine tuned: all inputs were equally weighted and a threshold was determined to be 2 (out of a possible 4). However, upon review by domain experts, different thresholds and weighting may be applied to improve results. See the modeling section for current methodology and scores.

Combining all of these methods, our final model used to identify transients is summarized below:

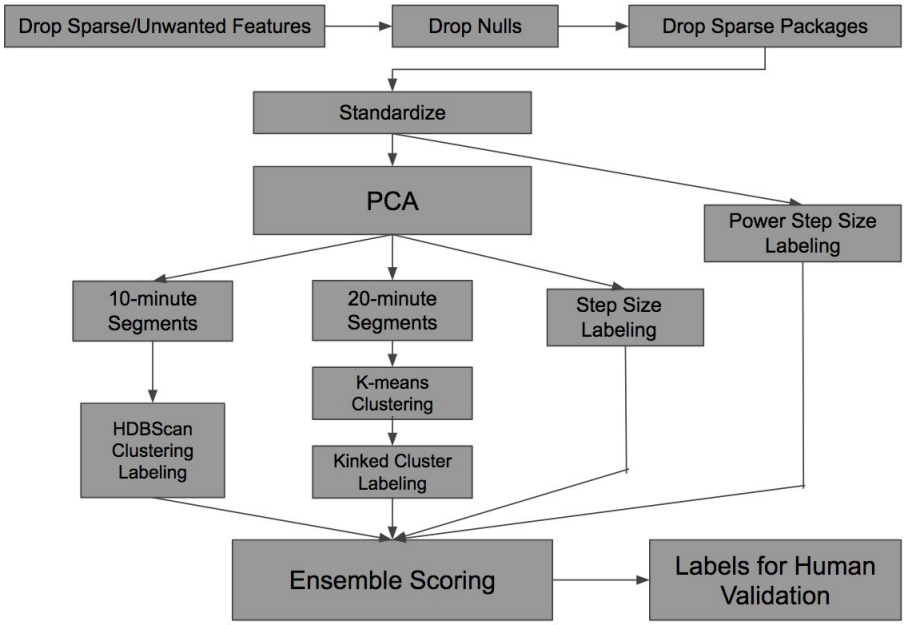

Figure 42. Final Model

User Interface

Overview and Purpose

Our primary goal is to allow users to visualize, interpret and annotate our generated transient labels. Additionally, we've included functionality to explore our transformed datasets in the context the raw data, and provide insight regarding the similarity of operation between packages. The layout and components of the interface is shown below:

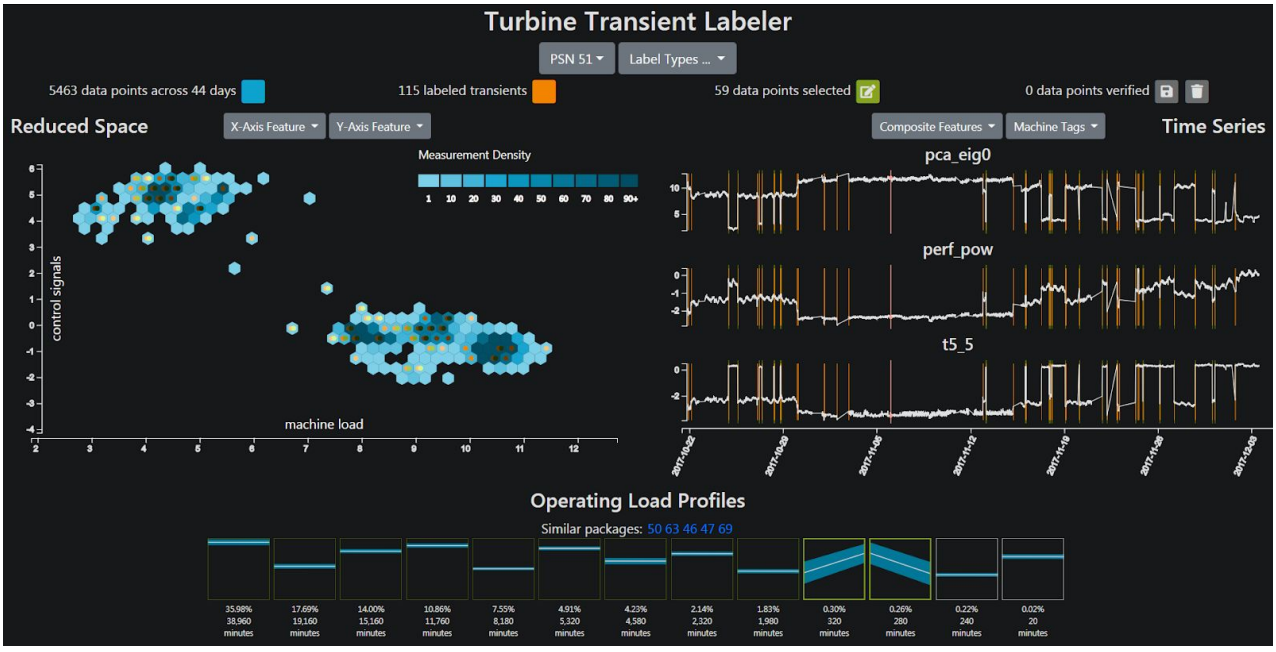

Figure 43. User Interface Screenshot

Libraries Used

Our interface was built using HTML, CSS and the following libraries:

Table 14. Libraries used for User Interface

Language	Name	Use Case
JavaScript	D3.js	data binding
JavaScript	Momentjs	timestamp manipulation
JavaScript	Underscorejs	data filtering
JavaScript	D3-hexbin	scatterplot aggregation
JavaScript	Simplifyjs	svg performance
CSS/JavaScript	Bootstrap	styling
CSS	FontAwesome	icons

Features and Functionality

The user interface is split into 3 main sections - the reduced space, load profiles, and time series plots. Each section interacts with the other two through data selection and highlighting. Presenting our model results in this way allows domain experts to easily select subgroups of data points for annotation.

Reduced Space

The reduced space section is a scatter plot of eigenvector coefficients. Points are binned into hexagons for scalability and hexagons are shaded by the number of points in the bin. The hexbins are colored using a monochromatic blue scale, were darker luminescence implies greater density. Orange dots signify the presence of transients in these bins, and green dots signify that at least one point in these bins is currently selected. Left clicking on a bin selects all data points in the bin. Right clicking anywhere deselects all data points. Hovering over a bin gives details on the makeup of the data points within the bin. Zooming and panning is possible with a mouse. Hexagon radius is scaled with the level of zoom, providing finer grained distributions as a user zooms in. X and Y-axis may be set to any of the top 5 eigenvectors.

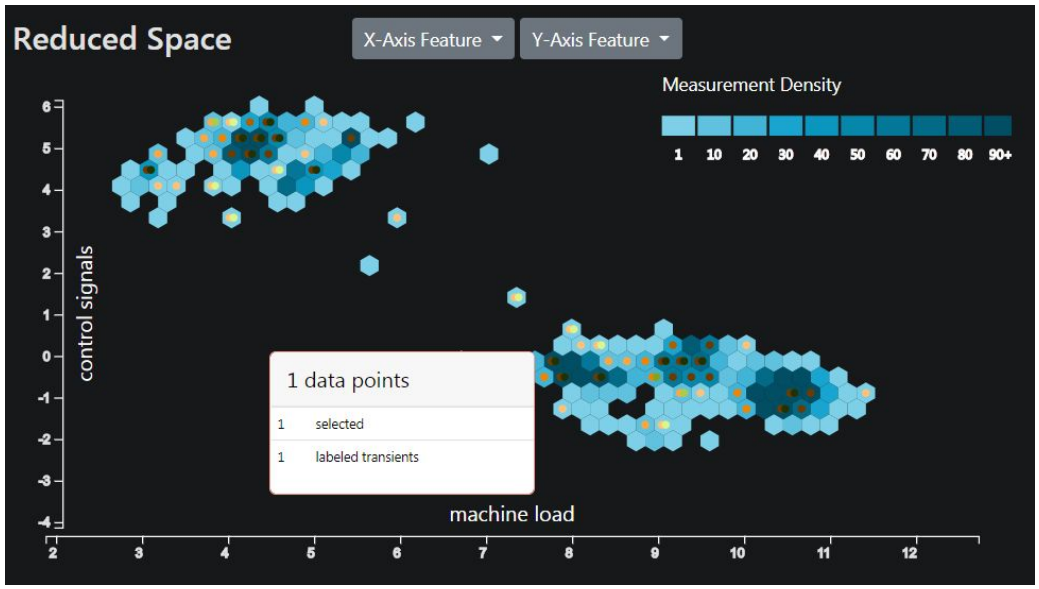

Figure 44. Reduced Space of the UI for PSN 51. Blue shading shows data density. Orange dots show hexbins which include transient labelled points. Green dots show hexbins with currently selected points. Hover tool allows users to see the composition of each hexbin. This example only contains one data point, which is both labelled as a transient and selected.

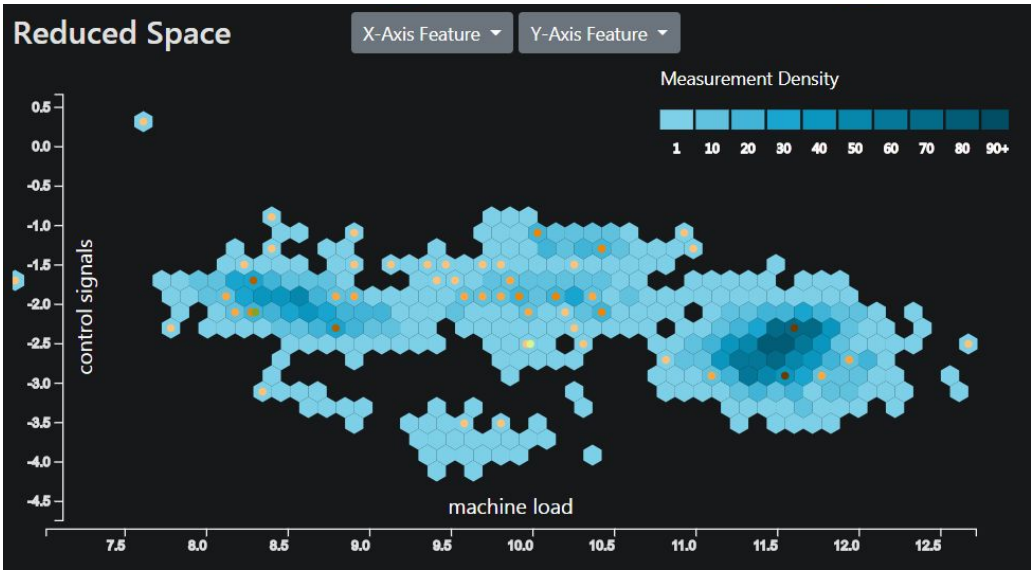

Figure 45. Reduced Space of the UI for PSN 51. Zoomed in view of bottom right cluster in Figure 44

Load Profiles

Load profiles are fleetwide clusters of partitioned eigenvector 0 coefficients. 20-minute clusters and 12-hour clusters are available. Cluster means are plotted in gray and two standard deviations are plotted in blue. Package utilization percentages are printed below each cluster. Left clicking a cluster selects all data points belonging to that cluster. Right clicking anywhere deselects all data points. When a cluster contains a selected data point it is outlined in bright green. Adjacent clusters are outlined in dark green.

Package Similarity

Package similarity rankings were generated based on packages' cluster distributions. When a package is selected, links to the top five most similar packages are listed.

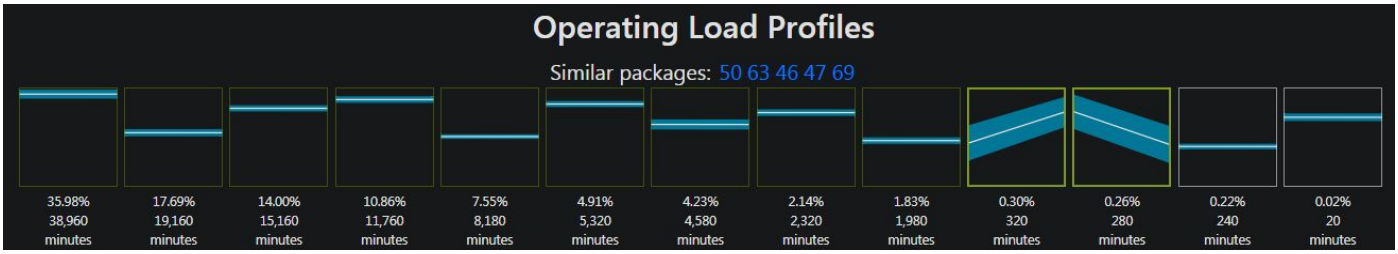

Figure 46. Operating Load Profiles Section of the UI for PSN 51. The two kinked clusters are selected. Similarly operating packages are linked under the heading

Time Series

The time series plots show the raw data and derived eigenvector data across time. Times labeled as transients are highlighted in orange. Selected times are highlighted in green. All plots can be zoomed and panned simultaneously using a mouse. Hovering over the plots gives plot values and any applicable labels. Left clicking selects data points. Right clicking clears all selected data points.

Tag Weights

All of the raw data tags are available for plotting. The ‘machine tags’ selection window includes two bar plots for each tag. These represent the tag’s importance (eigenvalue) to the eigenvectors selected in the reduced space. These plots provide domain experts with hints at what raw data may contain values contributing to eigenvector spikes.

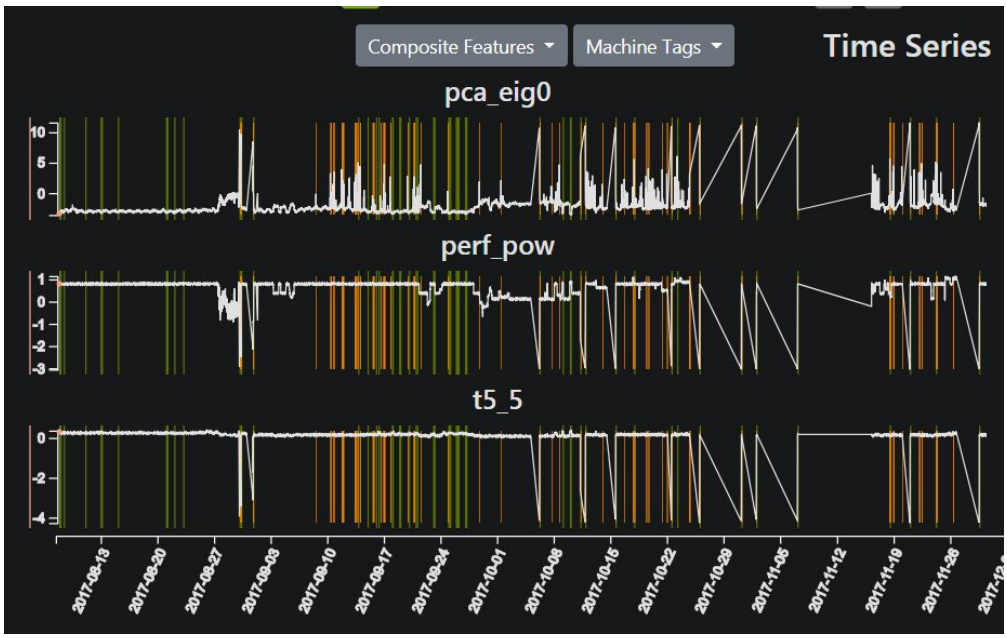

Figure 47. Time Series Section of the UI for PSN 69. Features plotted over time are eig0, power, and t5_5. Transients are highlighted in orange and selected data points and highlighted in green.

Annotations

Each of the sections above provide different ways to select different groups of data points. Once data points are selected, the annotation button can be used to bring up a data entry window. The group of selected points can be assigned a transient/not-transient label along with a short note. Annotations can be exported to a csv file by clicking the annotation download button.

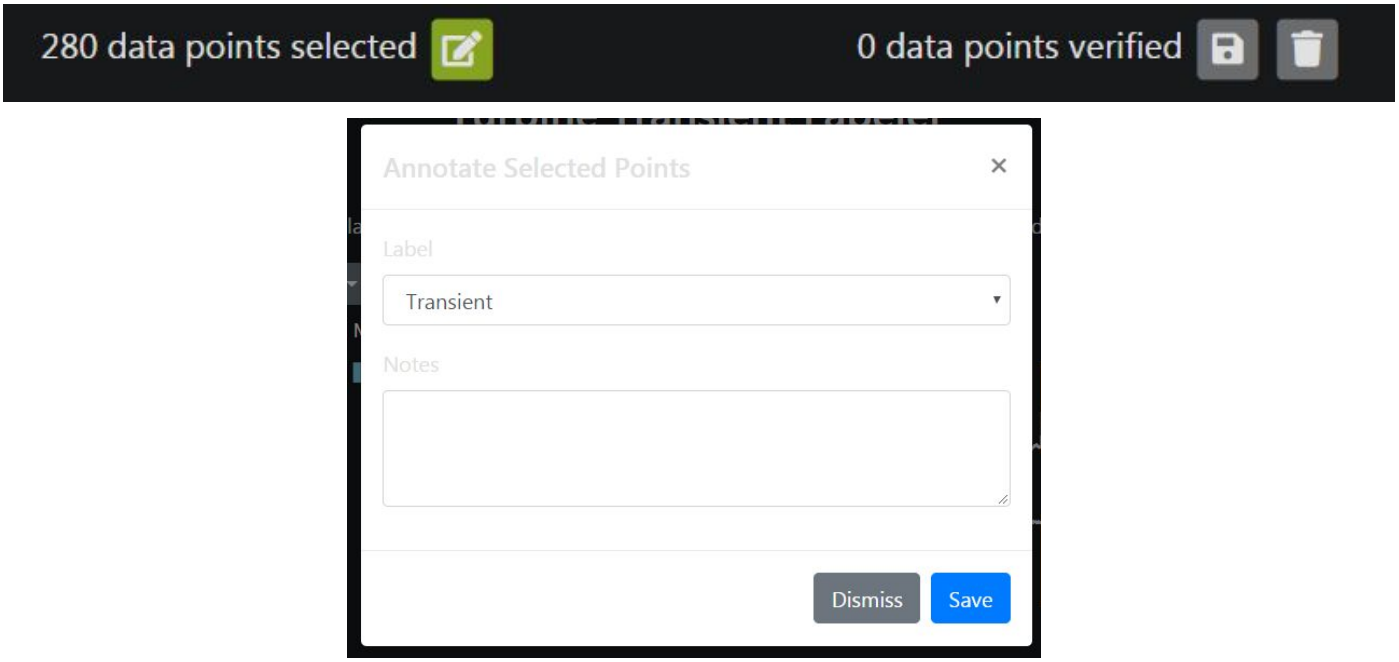

Figure 48. Annotation buttons and windows. Clicking on the green button (top left) allows users to annotate selected data points. A pop out window allows users to select Transient or Not Transient and add notes. Clicking the save button (top right) allows users to save annotated labels to csv.

Audience

Our target audience is currently limited to domain experts. Due to the specificity of the tags and machine configurations, our product is meant to be consumed by Solar Turbines' engineers. Our ultimate goal is to build an interface for domain experts to be able to annotate/validate our findings, so that they could be fed as a training set into a machine learning model to predict transients.

Data Pipeline

Scalability and Robustness Requirements

Our pipeline is built to analyze high dimensional sensor data. It includes a raw data store, in-memory computation, an export store for saving model results and information, as well as a user interface for reviewing results. Each component is comprised of an abstract interface wrapped around

physical or virtualized infrastructure and is designed for that infrastructure to change along with data and computing requirements.

The current 5.75 GB dataset does not require a distributed environment for these components. However, we estimate the data for Solar Turbine's entire fleet is over 100 times the size of our current data set. Thus, a plan to scale each of the pipeline's components is necessary to consider productizing our model.

Our test environment includes a t2.xlarge (4 vCPU, 16GB RAM) AWS EC2 instance for raw data storage and local storage for pipeline exports. Model training has been done on various on-premises machines, from consumer laptops to a 54-core, 256GB server. Given the proprietary nature of the data and timeframe of the project, the user interface is hosted locally for security.

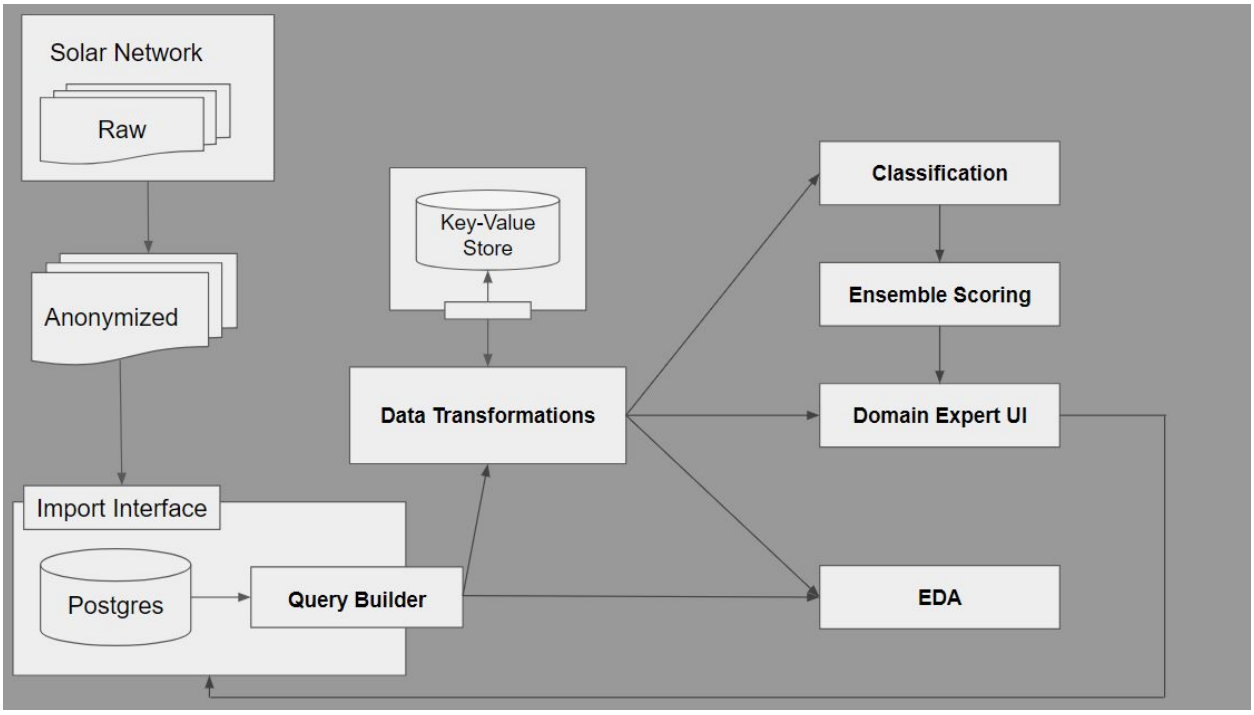

Figure 49. Data Pipeline

Raw Data Storage

The current data set includes two years of data for 72 turbines and totals 5.75 GB. Any production system would need to handle Solar's entire fleet of approximately 2000 connected packages with ten years of data, or an estimated 800 GB.

A MachineDataStore interface to interact with the raw data storage. Simple filters, such as package serial number and timestamps are supported. Generally, however, our model requires the entire raw data set be returned.

The current environment uses PostgreSQL on a t2.xlarge EC2 instance. Postgres has been shown to handle multiple terabytes on a single server when limited transactional access is needed. t2.xlarge instances have .74 Gb/s network speeds off of AWS networks and 5 Gb/s speeds to other EC2 instances.

While we believe Postgres on EC2 handles the scale of the data appropriately, it provides a single point of failure for our pipeline. To improve robustness, a distributed data store should be used. Solar Turbines uses HDFS internally, so that is a natural solution. Implementing the MachineDataStore interface for HDFS would allow it to be used as a replacement, although requiring HDFS-capable data

transformations. Alternatively, AWS RDS could be used as an expensive replacement if a scalable relational database solution is required.

Data Transformations

Data Transformations are the heart of the pipeline and represent the greatest threat of bottlenecks. Each step of the pipeline is defined as a Transformation object that takes an input dataframe and outputs a new, transformed dataframe. Passed dataframes all share Package Serial Number (PSN), and timestamp indexes.

Many of the data transformations we use suffer from high algorithmic complexity. Eigenvector decomposition is $O(n^3)$, K-means has a worst case complexity of $O(n^{(k+2/n\_features)})$, and HDBSCAN performs just under $O(n^2)$. While HDBSCAN does perform slower than K-Means, the difference only becomes significant in very large datasets and even then it is only second to k-means and outperforms notable clustering algorithms such as DBSCAN (which it is derived from) and Fastcluster.

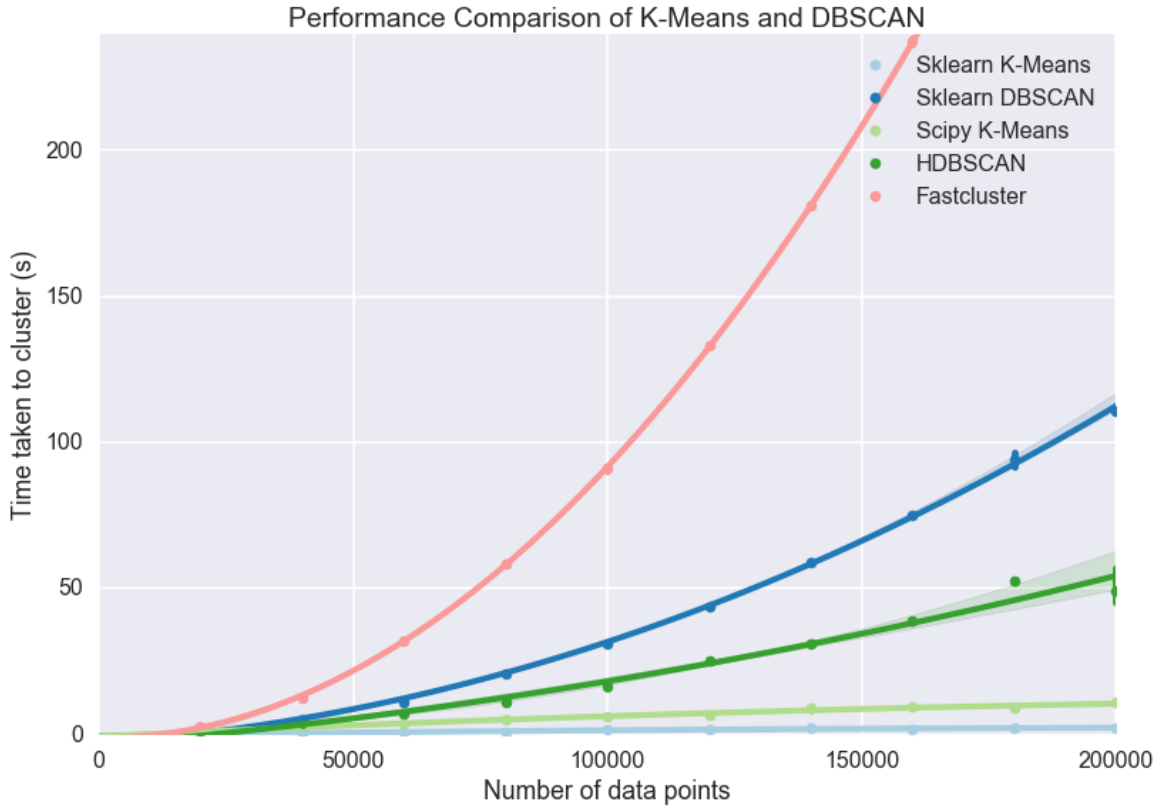

Figure 50. Performance comparison of clustering algorithms[8]

In a single-machine deployment, increasing the memory will allow larger datasets to be run through the pipeline, though, because Python is generally single threaded, performance will slowly degrade. With this in mind, we've developed several approaches for improving performance and scalability. Data transformations are organized in a tree structure. Each branch can be parallelized using multiple cores on the same machine or on different machines using a distributed message queue. Given a large increase in data or a faster runtime requirement, individual transformations could be parallelized further using Spark.

[8] "Benchmarking Performance and Scaling of Python ... - hdbscan." http://hdbscan.readthedocs.io/en/latest/performance_and_scalability.html. Accessed 8 May. 2018.

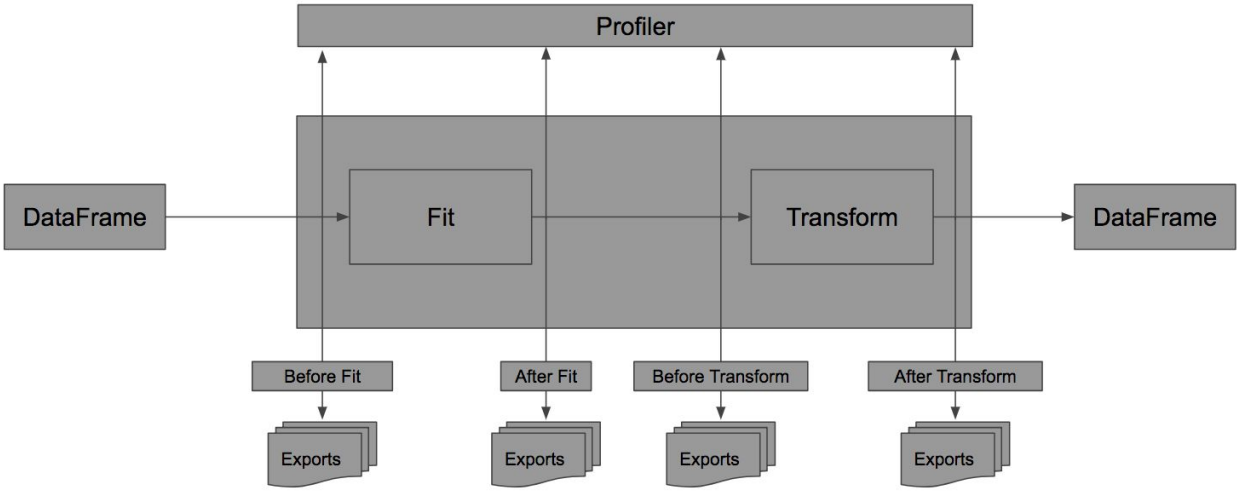

Figure 51. The pipeline is a tree-structure of these Transformation objects

Derived Feature Cache

In addition to computation time, each transformation has 4 events where result exports can be triggered. By default Python executes in a synchronous environment so any disk writes will hold up the rest of the pipeline. For our current data set and export requirements, this is a manageable lag (outlined in Evaluations and Results section). However, asynchronous exports would need to be implemented for a growing dataset. The current strategy utilizing local storage for exports has been best for development, ease of use, and speed. AWS S3 would be used as the export store when scalability is needed. As mentioned earlier, Solar Turbines uses HDFS internally, likely the best fit for their production use. Similar to the MachineDataStore interface, exports are handled using a simple key/value ExportStore interface.

User Interface

Our end product includes a user interface for annotating labels generated by our model. The interface is designed for domain expert use and efficiently reviewing large amounts of data easily. We are currently implementing this as a proof of concept, with all data stored locally. A robust solution would include a horizontally scalable and secure web api to serve raw and derived features from the data set in an on-demand capacity. This would provide the possibility of data aggregation near the data source to reduce the amount of memory needed on the client. Although the number of domain experts requiring access to the user interface is expected to be limited, scalable frameworks include AWS Elastic Beanstalk and Kubernetes. Due to our dataset size and use restrictions, these implementations are outside the scope of this project.

Evaluation Strategy and Results

Our model utilizes in-memory computation as well as heavy disk access for writing model output. Performance will be measured by total run time and available virtual memory. For each transformation, profile records are created before the fit function, after the fit function, before the transformation function,

and after the transformation function. We are assuming the raw data fed into the model is cached locally, the default behavior for MachineDataStore.

A sample pipeline was used and included data preprocessing->PCA->30min partitioning->KMeans->Flatten partitions. Profiling was done on a consumer-grade laptop with an Intel i7-6700HQ chip, 4 cores, and 16GB RAM.

K-Means labeling was the most resource intensive step in this sample pipeline, followed by the time partitioning transformation. Similar profiles were seen on higher RAM servers. Disk writes proved to be less costly than initially estimated.

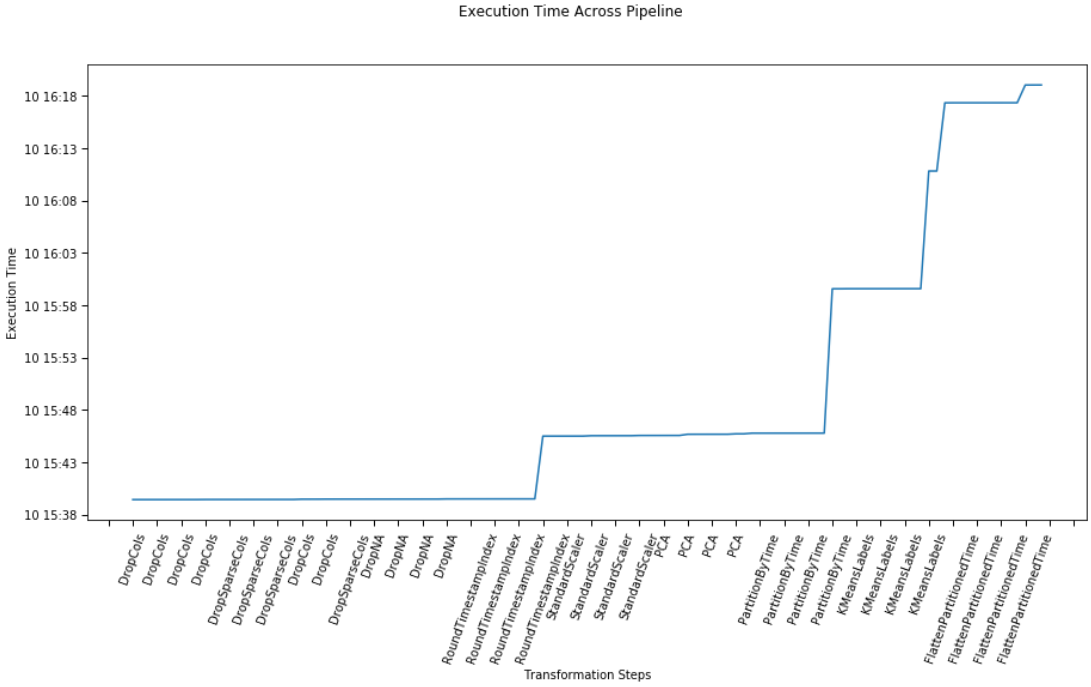

Figure 52. Execution Time Across Pipeline

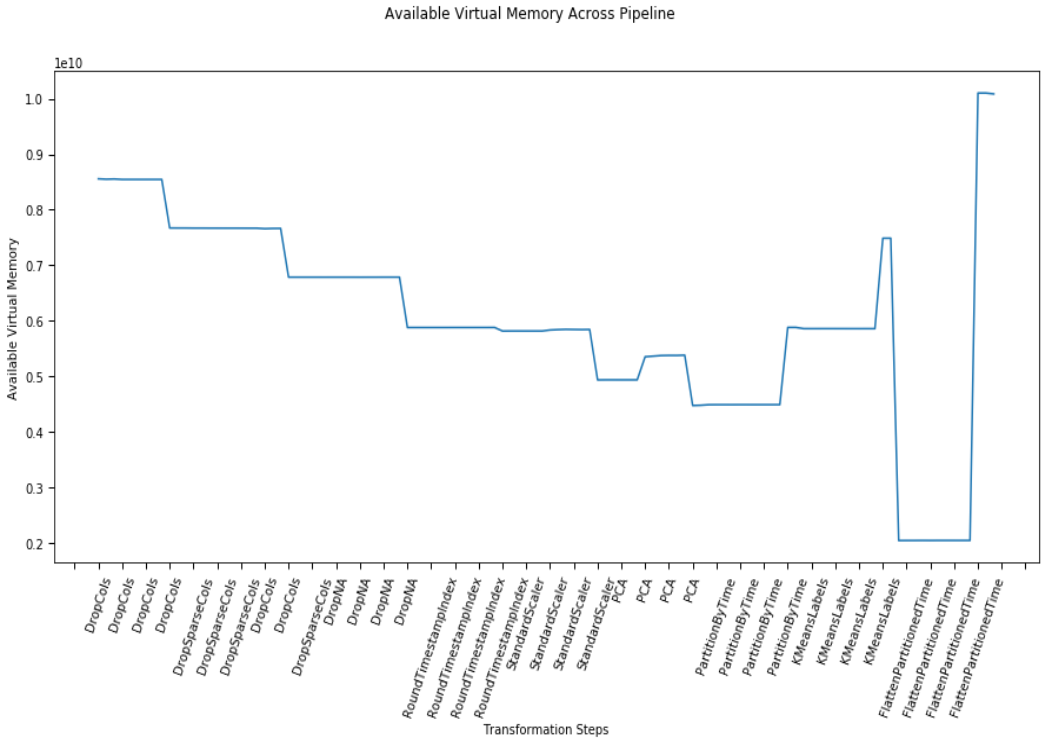

Figure 53. Execution Time Across Pipeline

Results and Interpretation

Our results offer insights into Solar Turbines' machine data. While much of what we uncovered during exploratory data analysis went unused, we narrows our focus on three avenues through which our product offers insight into the dataset: transients, principal components and package operational similarity.

Transients

The primary objective of our project has been to identify transient states in our dataset. We've explored the efficacy of each our transformations compared to a known method for detecting transients, 25% changes in power. However, it must be noted that these power jump 'labels' are not ground truth. Domain experts emphasized that no one method is universally correct. The different methodologies have unique benefits and drawbacks. While a change in power is a simple model, it is limited and does not catch all transients. It is also subject to sensor errors. The methodologies that use our eigenvectors, on the other hand, and more complex and difficult to interpret. However, because they're comprised of multiple features, they're more like to catch changes that a single feature might miss.

Additionally, when a package is changing from conventional to low emissions mode, a transient state occurs. However, because the power output is not changing, the transient event is not captured by the power jump labels. An example of this behavior is shown below.

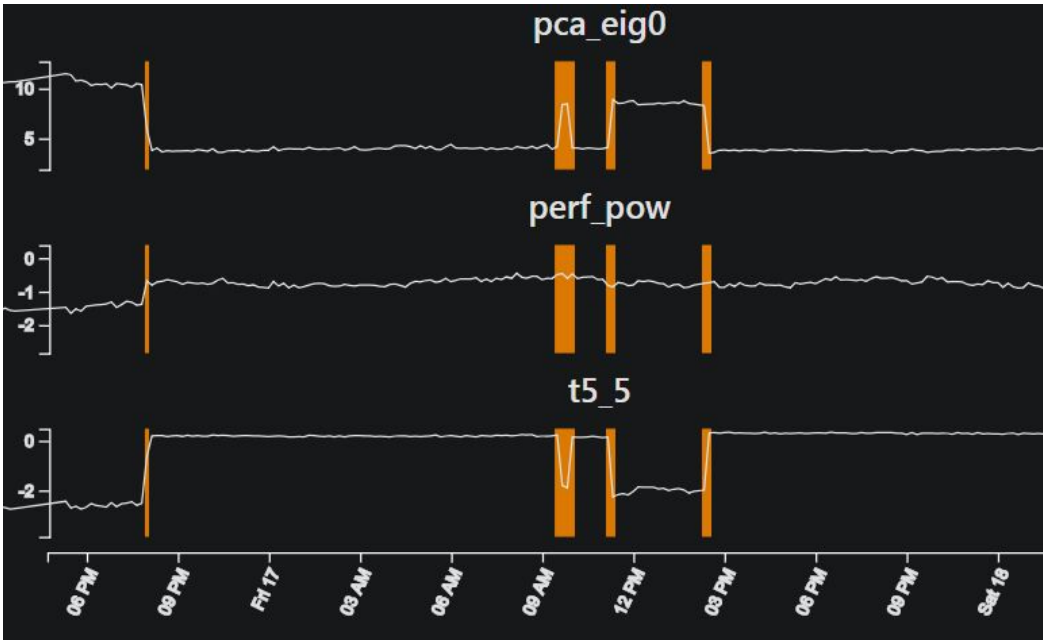

Figure 54. Transient state occurence without a jump in power.

Given this known limitation, results inconsistent with power jump labels do not necessarily mean that our model's labels are wrong. If a transient label does not align with power jumps, is unclear whether it is a false positive (not a transient) or a true positive (a transient that power jump did not capture). For this reasons, we built a UI for domain experts to validate results and assess accuracy.

Results comparing each metric to power jumps are as follows:

HDBSCAN Cluster Labels

As mentioned in the modeling section, accurately clustering datapoints using HDBSCAN primarily relies on tuning two different parameters: min_cluster_size and min_sample size. We explored combinations of these two parameters and discovered that in general, both larger min_cluster_sizes and min_samples led to HDBSCAN clustering too many datapoints as transients. Due to the requirement that they validate results, domain experts requested we minimize false positives and parameters were tuned with that consideration. On average, we achieved an f1 score of 0.10 in detecting transients with HDBSCAN and a f1 score of 1.00 in detecting normal operation.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1595415
1	0.12	0.09	0.10	6866
avg / total	0.99	0.99	0.99	1602281

Figure 55. Classification report for StepSize function on Eig0

We split these scores since the amount of normal operation outweighs the number of transients. We believe our low f1 score in classifying transients is due to our method of verification against Power Jumps and HDBSCAN is simply capturing different types of transients not detectable by simply just measuring Power Jumps.

Eigenvector0 Step Size Labels

Because our first principal component is comprised of features related to load, we applied a step-size function to find spikes or dips in eig0. We then compared the labels we generated to the 25% changes in power. As with the HDBSCAN clustering results, we found high accuracy with our normal operating data points. Our transients, again, do not line as well with our Power Jump labels.

	precision	recall	f1-score	support
normal	0.99598	0.99639	0.99619	1594648
transient	0.17543	0.16034	0.16755	7634
avg / total	0.99207	0.99241	0.99224	1602282

Figure 56. Classification report for StepSize function on Eig0

The reason for this low f1-score score for transient classification is not that our StepSize function is incorrect. Rather, it flags transient states that the Power Jumps metric misses. The example below shows PSN 49. Here, a transient state is clearly visible in t5_5 (bottom) that is mirrored in eigenvector 0 (top) but not in power (middle). This was verified by domain experts to be a transient event where the package was shifting emissions modes.

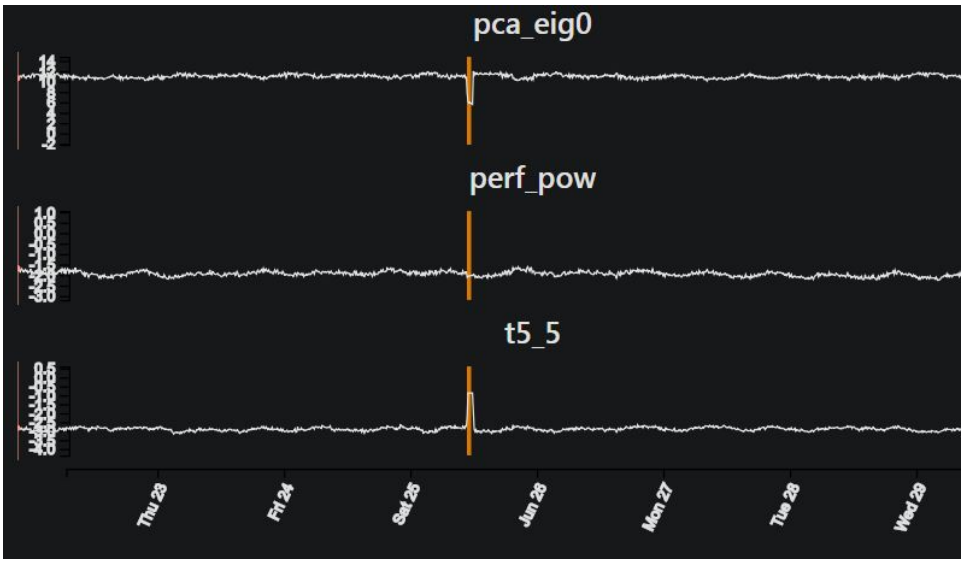

Figure 57. Eig0 StepSize labels (orange) show a transient event (verified by the t5 tag and domain experts) that is not captured by Power Jump labels.

Kink finder labels

We are quantifying the results of our PCA -> segmentation by time -> k-means clustering -> statistical analysis (kink finder). Below is a classification report for the 20 minute clusters. While it performs well for our normal operating class, we are working to improvement accuracy on our transient class.

	precision	recall	f1-score	support
normal	0.99620	0.99734	0.99677	1594648
transient	0.27025	0.20540	0.23340	7634
avg / total	0.99274	0.99357	0.99314	1602282

Figure 58. Classification report for KinkFinder on 20 minute k-means clusters

Power jumps labels

As Power Jumps labels were both a feature in our ensemble model and our best metric for comparison to ground truth, we did not quantify accuracy. It is known a limitation in our ability to evaluate the model, and would show 100% accuracy when compared to itself.

Ensemble labels

Our ensemble model labeled points where two or more of scorers identified transients.

	precision	recall	f1-score	support
normal	0.99598	0.99979	0.99788	1594648
transient	0.78184	0.15680	0.26121	7634
avg / total	0.99496	0.99577	0.99437	1602282

Figure 59. Classification Report for Ensemble Labels

While ensemble modeling has outperformed its individual components, true accuracy is unknown. Review with Solar Turbines' domain experts has validated our results. Points not mapped to changes in power has been positively identified as transients, suggesting higher accuracy than baseline metrics shown above.

Principal Components

Many of our methods for detecting transients utilized principal component analysis on our raw features. Doing so gave us insight into our dataset by allowing us to decipher which raw features have the most variance and thus influence on our model. Findings are as follows:

Table 15. Principal Components Key Findings (Model 2)

Principal Component	Primary Contributing Features	Percentage of Variance Explained	Cumulative Percentage of Variance Explained
1	**Load (Power output): temperatures, power**	**21.35%**	**21.35%**
2	**Controller signals: command, position**	**10.59%**	**31.94%**
3	**Pressure**	**7.36%**	**39.30%**
4	**Unclear**	**6.45%**	**45.75%**
5	**Ambient Temperature**	**5.37%**	**51.12%**

This information is built into our user interface. When users click on the "Machine Tags" button, a pop up window appears. This window contains, tag names, their measurement type and associate subsystem, as well as brief descriptions. Additionally, for each raw feature, there is a bar chart corresponding to it's eigenvalue for each of the eigenvectors, where larger values correspond to higher influence. Users can select and plot these raw features with some context as to how they correspond to the eigenvectors visualized in the reduce space portion of the dashboard.

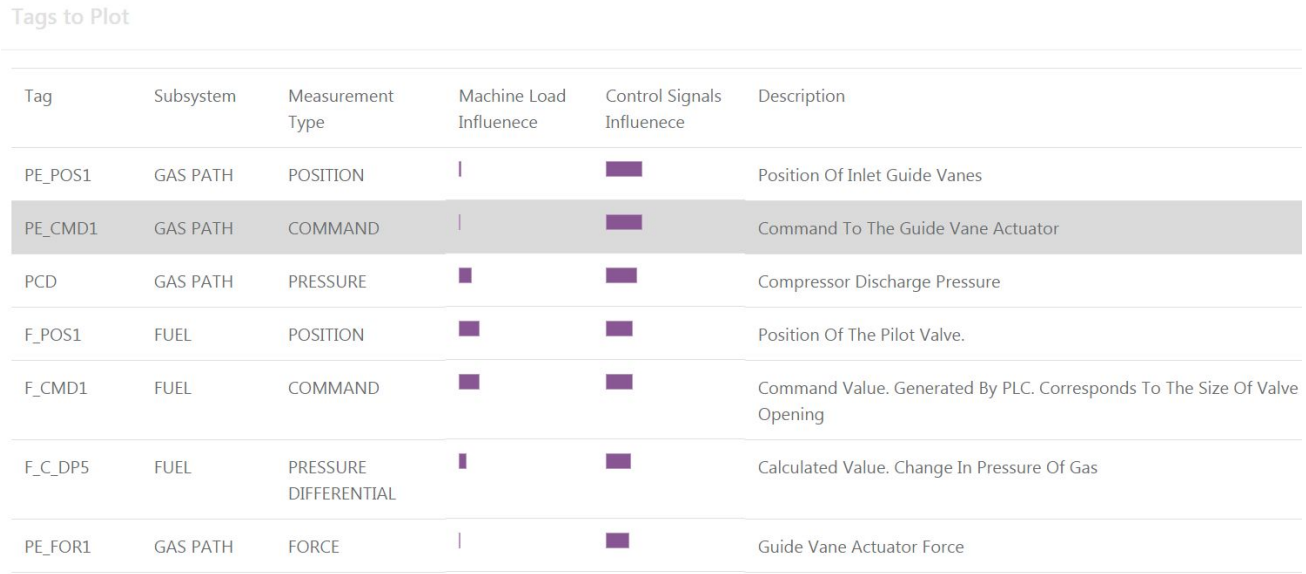

Figure 60. UI showing raw features' details and contributions to principal components.

Package Similarity

In our JavaScript dashboard, we display package-to-package similarities using cluster results based on 20 min segments, where n_clusters = 25. Using % of total usage (or count of 20 minutes segments per PSN in each cluster divided by total number of segments in the PSN), we generate similarity matrices based on euclidean distance. Similar packages suggest that the packages were operated by the same customer for the same application. Below we are showing the top 3 most similar packages to a given package using this approach.

Table 16. Top 3 most similar package for each package using 20 min segment, n_cluster = 25 cluster results for model 2. Most similar 1 denotes euclidean distance between the packages is smallest.

Package	34	35	36	37	38	39	40	41	42	45	46	47	48	49	50	51	53	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	71	72
Most similar 1	65	36	35	36	56	56	60	66	68	58	47	46	49	48	51	50	57	39	39	53	45	57	40	60	51	68	67	34	67	64	42	45	68	35
Most similar 2	66	58	58	35	39	38	61	34	37	36	48	48	46	51	62	49	37	38	53	59	35	53	61	40	61	65	66	64	64	66	71	59	42	58
Most similar 3	64	72	37	42	53	55	46	64	35	35	61	40	51	46	49	48	56	56	38	69	36	69	46	46	49	42	65	67	41	65	37	57	35	36

Obstacles

Pivoting

Our initial pursuit was to detect and differentiate anomalies in Solar Turbines' machine data. Despite finding hundreds of thousands of statistical outliers, we had difficulty identifying true anomalies due to the lack of labels. Additionally, we had no ability to differentiate anomaly subtypes. We pivoted to transient states, which, though a normal part of gas turbine operations, were visible as extreme values in our features. Unfortunately, this change in direction occured two thirds of the way into the project. Much of our exploratory data analysis and approach was structured around anomaly detection. Given the time and data limitations, rather than building a workflow to identify transient states, we had to adapt our work to this different target.

Labels and Data Interpretation

The biggest obstacle faced while trying to find meaning in this high dimension time series data set was our lack of labels. Without ground truth, we could not establish a foundation from which to build a predictive model. Additionally, we faced difficulties interpreting our findings and exploratory analysis, as gas turbines are extremely complex machines and no team members have background expertise in mechanical engineering. Some findings and correlations throughout this project which we believed to be important discoveries were later found to be already well known to domain experts.

Future Work

Transient Prediction

As mentioned, we generated a labelled dataset that classifies normal operating from transients in turbine machine data. With verification from domain experts, we would then have a sufficiently large set of labels to then begin work to predict transients. By being able to predict transients, it would be possible for

Solar Turbines to better understand load patterns and wear of the turbines without having to shut them down and manually inspect, thus saving hundreds of thousands if not millions of dollars in unplanned downtime.

User Interface Gamification

In order to encourage use of the transient labeler, we would explore the idea of making the dashboard into a minigame. By presenting itself as a more of a game than a task, the labeler would become more engaging and fun to use. Incorporation of a leaderboard and hidden easter eggs would further improve enjoyment and increase user consumptions and thus verified labels.

Shutdown Analysis

Unplanned shutdowns occur when gas turbine's programmable logic controller is triggered to turn off the package. They are due to various combinations of operating conditions within the package. While they are not directly mappable to the machine data and difficult to interpret (i.e. shutdowns can happen for any numbers of reasons, not just engine failure), we hypothesize that they can be mapped to transient states. That is, as transient states are known to be strenuous and volatile events, they can trigger shutdowns. We intended to investigate whether or not there was a correlation between shutdowns and transient states. Unfortunately, unplanned shutdown data was not authorized by Solar Turbines. However, we attempted to make our own, using the engine starts feature of our Model 2 raw data. The ENG_ST tag is an incrementer that counts the number of times an engine was turned on. We took the data points where the ENG_ST value increased and assumed that the timestamp immediately prior was linked to a shutdown. We then compared these shutdown labels with our delta power transients, eigenvector 0 step size jump transients, and our first 11 eigenvectors, shown below.

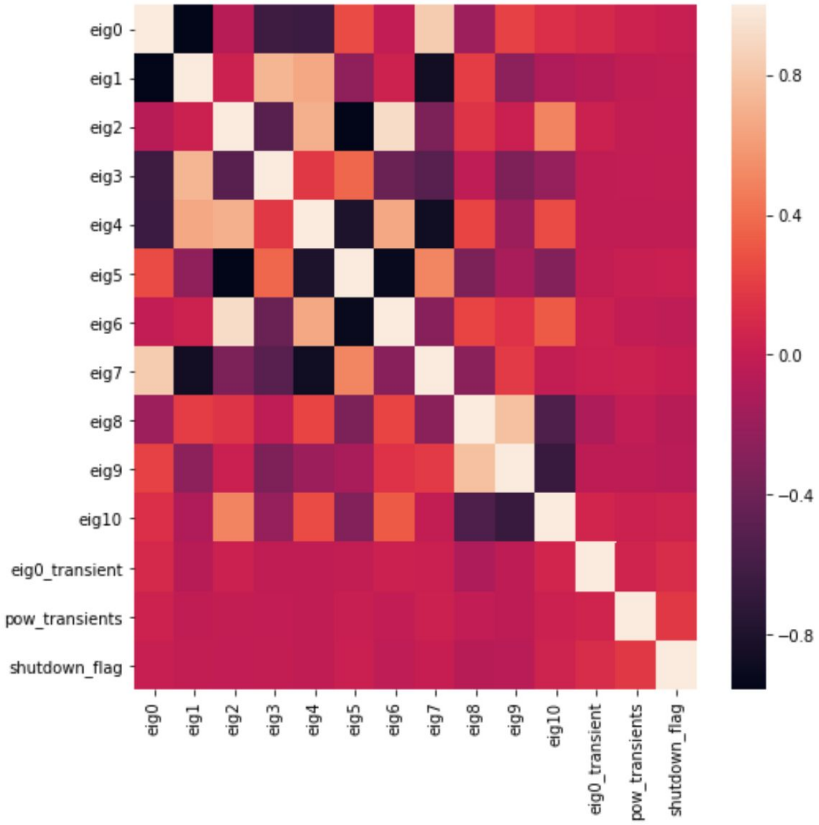

Figure 61. Heatmap showing correlation between eigenvectors, transient labels and shutdown flags

This above does not show a strong correlation between delta power transients (pow_transients) and shutdowns (shutdown_flag). However, we believe the following confounded these results:

- Missing data
    - Shutdowns may have occured while package was operating below onload conditions
    - Shutdowns in dual fuel engines may have occured while the package was running on liquid fuel
    - Shutdowns may have occurred during time intervals where data is missing for other reasons (connectivity issues, etc)
- Planned versus unplanned shutdowns
    - Using starts as an indirect measure of shutdowns does not allow us to differentiate planned versus unplanned shutdowns. Users may have simply turned off the machines any number of reasons (planned) as opposed to the machine shutting itself off (unplanned).

In the future, given a clean set of unplanned shutdowns labels for a packages, we could re-evaluate the correlation between transients and shutdowns and ultimately, assuming there is a correlation, differentiate transients that trigger shutdowns from those that do not.

Integration at Solar Turbines

Detailed in Data and Environment Data Preprocessing, extensive efforts were taken to sanitize the data before it was moved off of the Solar network. The overall process and measures taken at each step is detailed below:

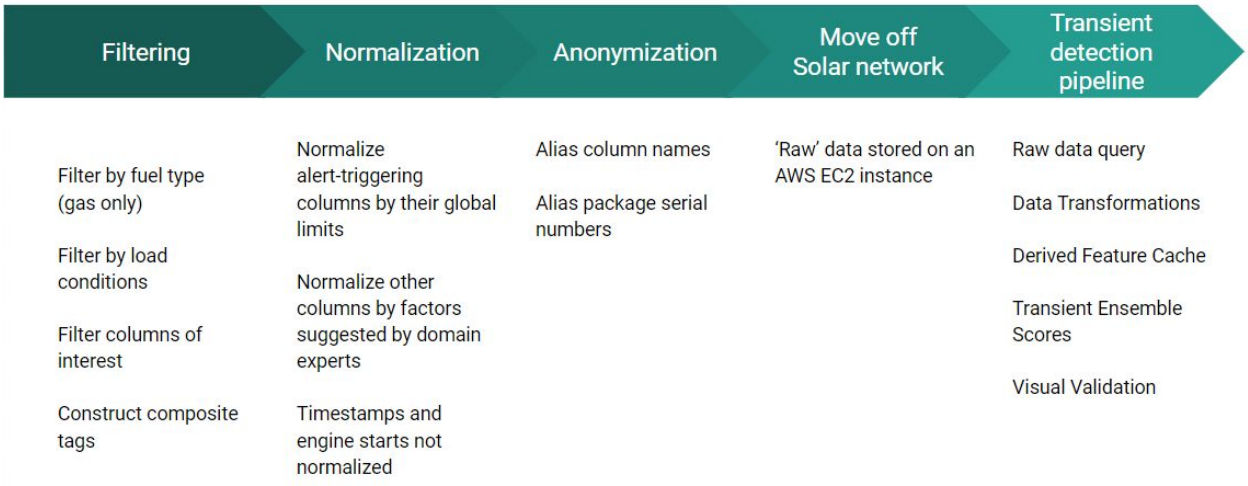

Figure 62. Current non-integrated process.

If we were to integrate our pipeline with Solar's, the anonymization processes would not be necessary. Nor would we need to move data off of the Solar network. Built with integration and scalability in mind, the transient detection pipeline is robust enough to work on the non-sanitized versions of the same data. Additionally, to ensure the same data input quality and outputs, the pipeline would require the same filtering and normalization procedures. An integrated product with Solar Turbines would dramatically simplify the process, as shown below:

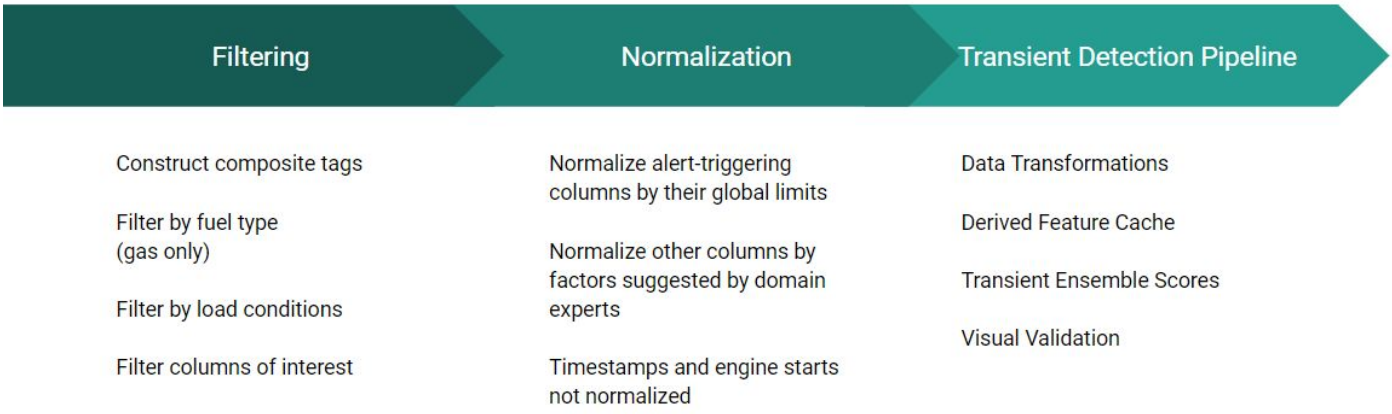

Figure 63. Integrated process.

This is a simplified version of the work and coordination efforts required to integrate our product with Solar's data ingestion and storage processes.

Conclusion

Our product serves as a first pass at labeling transients in Solar Turbines' machines. In addition to identifying transients, we add value by providing insight into the data (which features are most indicative of transition states, etc) and context to our dataset (which packages are operated similarly, etc). With these objectives in mind, we've built a dashboard that effectively communicates our findings and allows domain experts to annotate our labels. This creates a feedback loop to the model to for improving future accuracy. Additionally, our pipeline can be integrated with Solar's network with minimal required changes. Such information can be invaluable to our audience, Solar Turbines engineers, and can ultimately influence Solar's alerting policy so that customers can benefit as well.

Team Roles and Responsibilities

- Garrett Cheung (Data Engineer)
    - Solar-side coder (anonymization/data preprocessing and publishing)
    - Statistical Methodologies
    - HDBSCAN clustering
    - Gap analysis
    - Modeling
- Michael Galarnyk (Bookkeeper)
    - Kept track of expenses (AWS fees, etc)
- Jared Goldsmith (Record keeper)
    - Maintained our github repositories
    - Exploratory Data Analysis
    - Data Pipeline
    - User Interface
- Jillian Jarrett (External Team Coordinator)
    - Liaison between Solar Turbines and UCSD/SDSC
    - Reported progress and correspondence with advisors
    - Exploratory data analysis
    - Modeling
    - Preliminary Shutdown Analysis
- Orysya Stus (Internal Team Coordinator)
    - EDA Tableau Dashboards
    - Cluster analysis
    - Package similarity analysis

References

1. "Gas Turbines - Products | Solar Turbines." https://www.solarturbines.com/en_US/products/gas-turbines.html. Accessed 22 May. 2018.

2. "Sigma Labs Wins Contract from Solar Turbines to Use ... - 3DPrint.com." 19 Apr. 2017, https://3dprint.com/171698/sigma-labs-solar-turbines-inc/. Accessed 22 May. 2018.

3. Kim, J. H., et al. “Model Development and Simulation of Transient Behavior of Heavy Duty Gas Turbines.” *Journal of Engineering for Gas Turbines and Power*, vol. 123, no. 3, 2001, p. 589., doi:10.1115/1.1370973.

4. Hanachi, et al. “Performance-Based Gas Turbine Health Monitoring, Diagnostics, and Prognostics: A Survey.” IEEE Transactions on Reliability, 2018.

5. Simon, D. L., et al, A. W. , 2014 “A Model-Based Anomaly Detection Approach for Analyzing Streaming Aircraft Engine Measurement Data.” ASME Paper No. GT2014-27172. 2014.

6. HDBSCAN http://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

7. Jolliffe I.T. “Principal Component Analysis”. Springer. 2002

Appendices

Appendix A: DSE Knowledge

1. DSE 200: Python for Data Analysis
    1.1. Principal Component Analysis
    1.2. Native Python
    1.3. Use of Python libraries
        1.3.1. Matplotlib
        1.3.2. Pandas
2. DSE 220: Machine Learning
    2.1. Data cleaning
    2.2. K-means Clustering
    2.3. Hierarchical clustering
    2.4. Model Fine Tuning
3. DSE 210: Probability and Statistics using Python
    3.1. Statistical Methodologies
    3.2. Accuracy metrics
4. DSE 201: Data Management Systems
    4.1. PostgreSQL
    4.2. Structured Query Language (SQL)
5. DSE 230 Data Analysis Using Hadoop and Spark
    5.1. Approach to scalability
6. DSE 203 Data Integration and ETL
    6.1. Integration of 10 minute and 1 hour data sets
        6.1.1. Composite tag creation
        6.1.2. Feature matching
7. Data Visualization
    7.1. Visual idioms
    7.2. Expressiveness and effectiveness
    7.3. Tableau Dashboards
    7.4. JavaScript Dashboard

Appendix B: Data and Software Archive

A. GitHub repositories
    a. Pipeline: https://github.com/j-goldsmith/TurbineTimeSeries
    b. User Interface: https://github.com/j-goldsmith/TurbineTimeSeries.UI
    c. EDA (Working): https://github.com/mGalarnyk/AnomalyDetectionMachineData/
        i. Reports
        ii. EDA notebooks
B. Team Drive: https://drive.google.com/drive/folders/0AKP5YIrmOxVVUk9PVA
    a. Reports
    b. Presentations
    c. Data
    d. Visualizations
    e. Google Colab Jupyter Notebooks
C. Dataset
    a. Access to raw data requires password and authorization from Solar Turbines
    b. Data required for UI: https://drive.google.com/open?id=1MM0GsgNS1N5Ur7FoocBmVdwn-PN53uYz