Anomaly Detection of Energy Consumption Data - Final Report

Team

Fei Wang
Yumeng Ding
Gautam Moogimane

Sponsor

Linnea Paton
Jones Lang LaSalle Americas, Inc. (JLL)

Table of Contents

1. Introduction

In recent years, natural resource consumption and conservation have become major areas of focus across academic, industry and political debates. One of the big components of natural resource consumption comes from energy usage in commercial buildings. Therefore, the monitoring of energy usage trends and detection of abnormal activities in these public buildings are essential to a more efficient usage of these resources.

Anomaly detection of energy consumption is an important component in multi-family apartments' and commercial buildings' management process. Comprehensive and automated anomaly detection rules can help management companies identify potential overbilling and energy insufficiency efficiently and effectively. However, the differentiation between normal fluctuations in energy consumption due to seasonal variations and actual anomalies are not always easy and the incorrect detections can cause unnecessary investigations. In order to help our capstone project sponsor, Jones Lang LaSalle Americas, Inc. (JLL), correctly identify actual energy consumption anomalies, we propose the development of an automatic rule-based anomaly detection system.

Energy usage anomaly can come from many different sources, for example, errors in manual data entry, broken infrastructures, seasonality of energy consumption and so on. In this project, we aim to leverage the historical data from JLL and New York City Housing Authority (NYCHA) public data to model the trend in energy consumption across buildings in New York and help JLL successfully differentiate between anomalies caused by unknown factors such as those listed above and known factors like seasonality.

Ultimately, this project will include (1) performing extensive data cleaning and data munging to understand the potential causes for billing issues and manual data entry errors, (2) and building predictive models with confidence intervals to detect anomalies in observed data entries.

2. Problem Statement and Motivation

Our capstone project sponsor, Jones Lang LaSalle Americas, Inc. (JLL) is in charge of collecting energy usage data for properties under their management, to ensure the clients' energy usage are compliant with local energy disclosure laws and measure progress towards sustainability goals. The key to success of this task relies on the accuracy of the monthly-reported utility data. More specifically, we are tasked to output potential abnormal data entries due to either data quality issues or other factors like broken infrastructures for the team to investigate. This will help increase the productivity of JLL's analysts (i.e. time will be saved by targeting only the sites with anomalies for audits), lead to real cost savings (e.g. fixing building operation issues that are causing energy or water waste), and increase confidence in greenhouse gas sustainability reporting data.

JLL has created a set of Data Quality Checking rules based on their experience with the data. Our team's objective will be to clean and aggregate the data to bring it to a form on which we can test different modeling techniques, also flagging accounts that have non continuous or incomplete data along the way, and finally come up with three forecasting approaches, based on data analysis that we feel work best in the current scenario and potentially make suggestions on how to pipeline this detection process into their current environment.

As far as motivation is concerned, one of our primary goals is to help conserve resources, and the fact that the team has never previously worked with forecasting and time series data was an added incentive.

3. Data Summary

We were unable to obtain real client data from our sponsor JLL due to delays in its IT processes. Per sponsor's suggestion, we worked on a publicly available dataset from the New York City Housing Authority (NYCHA) instead. This dataset contains historical electricity billing information of ~ 3,000 residences in the New York City area, which according to our sponsor, has similar energy consumption patterns and characteristics as JLL's clients.

The raw data file from NYCHA's website includes monthly electricity usage and cost data for residence buildings in NYC area from Dec 2009 to June 2018. The complete dataset has 313,147 rows and 27 columns. There are three sets of variables in the dataset: 13 variables describing building information such as Development_Name, Borough, Account_Name and Meter_Number; 8 variables describing the individual billing information such as Bill ID, Revenue Month, Service Start and End Date, and Rate Class; lastly, there are 6 variables used to identify energy consumption and cost information associated with each bill, which include total charges, electricity consumption in kilowatt hour (KWH), maximum electricity demand capacity in kilowatt (KW), KWH charges, KW charges and other charges.

Our initial goal was to detect the anomalous data values in electricity usage, both KWH consumption and KW demand capacity, at individual account level, by comparing the their values across multiple years. However, in the NYCHA dataset, KWH and KW values were logged within billings months of various start/end dates, lengths and intermittent gaps. This poses difficulties to make fair comparisons of the historical values month over month. After a few iterations of communications with the sponsor, we decided to adhere to the industry standard practice of prorating electricity usage values to calendar months. Since KW value represents the maximum amount of electricity demand during a service period and is therefore non-proratable, we narrowed down the project's scope to detecting anomalies in the consumption values (KWH) only.

In order to accomplish this goal, we conducted two phases of data processing to prepare for modeling and statistical analyses:

I. Clean and validate the dataset to ensure it aligns well with known business rules and logics
II. Prorate the KWH values from billing months to calendar months and aggregate the data to the account level, which is uniquely determined by a combination of Building_ID and Meter_Number

In phase I, we took the following steps to clean, validate and transform the data using Python scripts:

1. Applied general data cleaning to exclude duplicate rows and rows with null data only and convert data type from string to datetime or float for relevant columns
2. Created and validated a primary key for the dataset, with a combination of the columns "TDS #", "Location", "Meter Number", "Service Start Date" and "Service End Date". There were ~0.2% of rows removed during the process as those entries were caused by either rebilling or invalid entries
3. Merged related Meter_Number's under the same account, since oftentimes we found that the Meter_Number of an account may switch year over year. It's also frequent that an account may have separate Meter_Number's to log KWH and KW values. After a few rounds of data cleaning and transformation, we reached a point where only a very low percentage of entries doesn't meet our assumptions of the logic of the dataset

In phase II, we prorated and imputed the KWH values from billing month to calendar month. In the original dataset, each row represents an electricity bill for a billing month with a service start date and end date. We split the KWH value of each row into its associated calendar months and for each calendar month calculated a weighted sum of prorated KWH values as its imputed KWH value. For instance, suppose the calendar month of 2017-03 contains prorated KWH values from both billing months of 2017-02 (a kwh) and 2017-03 (b kwh), covering 15 days and 10 days respectively. We calculated the weighted daily average KWH value and then multiply it with the number of days in the calendar month. Therefore the imputed KWH value for 2017-03 is calculated as $(a+b)/(15 +10) * 31$. Our process will also automatically detect the gap months which is not covered by any billing periods and thus have no KWH value at all.

The resulting dataset consists of 178,108 rows, each representing the KWH consumption value for a household account in a given calendar month. Looking at the account level, we got a time series of 9 years monthly data for ~3,000 accounts. The variables of building attributes such as Borough were not included since they are not informative for analyzing the KWH values. The lack of account or building level attributes that may influence the KWH consumptions in the dataset also limits our choices of modeling the later steps.

4. Background

4.1 Domain Knowledge Background

The energy industry in general is undergoing a lot of advancements in terms of technology use. Artificial Intelligence and machine learning are no longer just buzzwords, but strategies that are being applied widely to improve efficiencies and reduce overall costs. In this project we are mainly concerned with anomaly detection, a remote facilities management tool, that detects outliers in a given data set that do not conform to established normal behavior. These errors in reported energy figures could be due to faulty equipment, irregular readings or manual entry mistakes and being able to automatically identify these erroneous inputs, could be extremely beneficial in reducing energy wastage.

An example of a tool that is currently being used across the industry is PEERS, a linear multivariate regression model, that takes in details like month, year, different property details as well as historical data, and comes up with a calculated model, that predicts the estimated value of an account for the current time period. Comparing this value, against the actual values that are reported, is a good way to check for variation and detect anomalies.

Another widely used strategy is to visualize the data as a time series and then split this signal into 3 different components, seasonal, trend and residual data. This way, it can automatically account for seasonality and cyclical trends in the data, and adjust for them accordingly. The residuals can then be tested with statistical tests like the Generalized ESD (Extreme Studentized Deviate) test to identify abnormal data points. There are several inbuilt libraries that can do this in Python and R. We ended up using Prophet package in Python and Anomalize package in R.

For the NYCHA dataset of electricity consumption and cost, we mainly focus on the energy consumption (KWH). A client needs to pay for not only the amount of electricity consumed, but also the maximum capacity of electricity delivered since the utility company need resources to offer it that capacity during its peak energy demand period. We prorated and imputed KWH for all accounts to prepare for methods described below.

4.2 Anomaly Detection Background

In academic literatures and researches, anomaly detection is defined as the discovery of unusual and unexpected patterns which happen in datasets surprisingly [9]. There are three broad categories of anomaly detection techniques [4]: Unsupervised Anomaly Detection, Supervised Anomaly Detection and Semi-supervised Anomaly Detection. Unsupervised Anomaly Detection is built upon unlabeled dataset by assuming the majority of the data points are normal and then looking for instances that fit the least with regards to the rest of the dataset. Supervised Anomaly Detection techniques require a set of labeled data points with 'normal' or 'abnormal' and then training a classifier based on features from either labels. Semi-supervised Anomaly Detection techniques involve constructing a model with training dataset labeled 'normal' first to represent normal behavior and then test the likelihood of test instances to be generated by this model.

For our capstone project, we relied heavily on the Unsupervised approaches since we do not have labeled dataset. Below we will list the literature review for methods we employed and other methods we have tried during the exploration phase.

4.3 Literature Review

4.3.1 Decomposition

As mentioned in section 4.1, a potent method to detect outliers in time series is by decomposing a time series signal into 3 components - seasonal, trend and residual, and then analyze the residual component to identify months with anomalous value. A seasonal component could be associated with any cyclical pattern that causes variations in the electricity consumption, for example weather, temperature or even if the building has some unique characteristics that cause regular ups and downs. The trend component, represents the overall trend in the data across the observed time window. The residual component is the detrend and deseasonalized signal left in the time series after subtracting the previous two. The following figure shows an example of the original time series together with its three components:

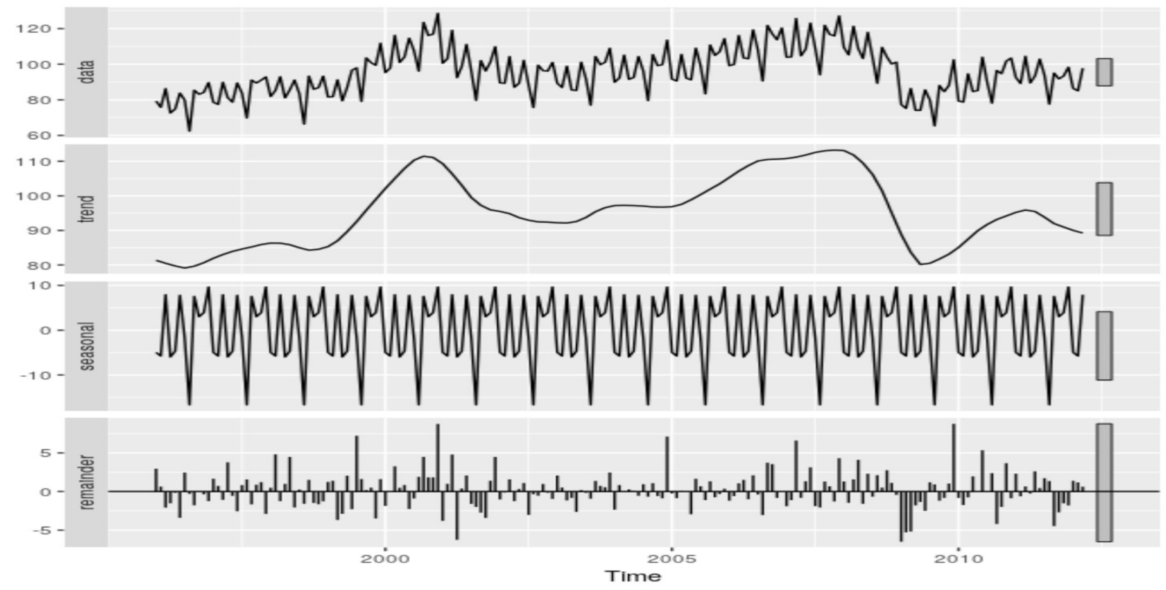

Figure 1

The rationale behind this method is that if we can assume the trend and seasonal components represent the normal behavior of the time series, then any signal that deviates from the normal values will be remained in the residual part, which can then be analyzed with various statistical methods to identify the outliers. Among all time series decomposition algorithms available, we found the STL (Seasonal-Trend Decomposition by Loess) algorithm developed by Cleveland et al.[5] in 1990 fits our purpose the best. This algorithm is very robust to anomalous values, meaning that the abnormal values will not affect the estimate of trend and seasonal components, but will remain in the residual component. It also allows the seasonal component to vary over time and this is suitable for energy consumption data whose pattern may indeed change over years. Besides the STL functions in both Python and R packages (*statsmodel* and *stldecompose* for Python and *stats* for R, Twitter recently published a paper on detecting near real-time anomaly detection using a variant of the STL algorithm[6], together with its implementation package *AnomalyDetection* in both R and Python. The Twitter algorithm calculates the seasonal component identically as the classical version, but replaced the trend component by a step function that represent the moving medians of the data instead of fitting a smoother. The median values work better when a long-term trend is less dominant, in cases like there are multiple level shifts over the years. There is also an *Anomalize* package in R which builds upon on the *AnomalyDetection* package and enables a tidy workflow with neat visualizations.

4.3.2 Prophet

Prophet is an open source forecasting tool that has been developed by Facebook, and is available to implement in either Python or R. At its heart, it is also an additive time series decomposition model, that splits the signal into four main components. This equation can be represented as

$$y(t) <- g(t) + s(t) + h(t) + e$$

- **g(t)**: piecewise linear or logistic growth curve for modelling non-periodic changes in time series
- **s(t)**: periodic changes (e.g. weekly/yearly seasonality) using Fourier Series
- **h(t)**: effects of holidays (user provided) with irregular schedules
- **εt**: error term accounts for any unusual changes not accommodated by the model

Some of the reasons why Prophet is useful in the current application are listed below

- **Prophet makes it much more straightforward to create a reasonable, accurate forecast, that can be easily automated -** Prophet's default settings produce forecasts that are often accurate as those produced by skilled forecasters, with much less effort.

With Prophet, you are not stuck with the results of a completely automatic procedure if the forecast is not satisfactory — an analyst with no training in time series methods can improve or tweak forecasts using a variety of easily-interpretable parameters. Also since these forecasts are generated with minimum human intervention, it is easy to automate this procedure to forecast for multiple time series simultaneously.

- **Prophet is fairly robust to outliers and missing data -** Many of the other models tested like SARIMA and Holt-Winters require complete data for each time series, which can be difficult to ensure in the current setting, resulting in unwanted errors.
- **Prophet automatically generates an uncertainty interval with every forecast -** Since the ultimate goal of this application is identifying outliers, the fact that Prophet can generate an uncertainty interval using Monte Carlo simulations, for each forecasted value, makes detecting outliers fairly straightforward.
- **Prophet's parameters are easy to understand and tune -** There are smoothing parameters for seasonality that allow you to adjust how closely to fit historical cycles, as well as smoothing parameters for trends that allow you to adjust how aggressively to follow historical trend changes. For growth curves, you can manually specify "capacities" or the upper limit of the growth curve, allowing you to inject your own prior information about how your forecast will grow (or decline). Finally, you can specify irregular holidays to model like the dates of the Super Bowl, Thanksgiving and Black Friday.

4.3.3 Clustering

From academic literatures, clustering time series is usually tackled twofold [10]:

(a) *feature-based* or *model-based*, *i.e.*, previously summarizing or transforming raw data by means of feature extraction or parametric models, e.g., dynamic regression, ARIMA, neural networks; so the problem is moved to a space where clustering works more easily. Beyond the obvious loss of information due to *feature-based* or *model-based* techniques, they can also present additional drawbacks; for instance, the application-dependence of the feature selection, or problems associated to parametric modeling.

(b) *raw-data-based*, where clustering is directly applied over time series vectors without any space-transformation previous to the clustering phase. Characteristic drawbacks of *raw-data-based* approaches are: working with high-dimensional spaces (*curse of dimensionality*), and being sensitive to noisy input data.

The main challenge in time series clustering is the definition of similarity measures used to make the clusters [9]. We considered similarity as the measure that establishes an absolute value of resemblance between two vectors, in principle isolated from the rest of the vectors and without assessing the location inside the solution space. We decided to use Squared Euclidean

Distance as the similarity measure because it will run faster than the Euclidean Distance and we have about 1,916 accounts to run through each iteration. We have also explored Dynamic Time Warping (DTW) distance [10], this measure allows a non-linear mapping of two vectors by minimizing the distance between them. The main drawback of the measure remains in the effort dedicated to the calculation of the path of minimal cost, in addition to the fact that, actually, it cannot be considered as a metric, i.e., it does not accomplish the triangular inequality.

4.3.4 Other Methods

- **Classification and regression trees (CART)**
  - Classify anomalous/normal points (need labels)
  - Predict the next points with confidence interval —> compare with real data (using G-ESD test or Grubb's test)
  - Implementation example: XGBoost Library
  - Pros: Easily scalable and flexible.
  - Cons: need careful feature selection, more data points and does not take seasonality of a time series into account.

- **Neural Networks (supervised and unsupervised)**
  - RNN such as LSTM
  - Pros: Easily scalable and flexible.
  - Cons: needs more data points to build a reliable neural network than we could provide, also does not take seasonality of a time series into account.

- **ARIMA and SARIMA**
  - Forecast using time series decomposition, by splitting the signal into its trend and seasonality components
  - Pros: Scalable and intuitive, gives good results without much tweaking
  - Cons: need parameter selection, signal must be independent of time, hard to automate, is not robust to missing data and outliers.
  - Implementation example: tsoutliers R package

- **Exponential Smoothing**
  - Holt-Winter's Seasonal Method
  - Pros: Flexible and easy to automate.
  - Cons: Can only have one seasonal period (such as one from weekly, monthly, yearly), is not robust to missing data.

5. Approach

5.1 Decomposition

The time series decomposition method assumes that the variation of data values are mostly captured in the residual component of the time series after decomposition. There are two steps involved:

1) Leverage Python or R packages to decompose the time series of the electricity consumption (KWH values) monthly data for a given account
2) Apply statistical tests to identify anomalous points in the residual component. Methods we considered include the Generalized ESD test, the Interquartile Range method and the Individuals and Moving-Range charts (XmR Chart) from the statistical control literature

For step 1), we chose the STL decomposition algorithm as it serves our purpose the best by being robust to anomalous values and allow variation in the seasonal component. Since the Twitter version of STL is increasingly popular in the open-source community, we test it along with the classical version. However, since the patterns of most of our monthly billing data are quite smooth, with rare level shifts or short term jumps, we found that the classical STL performed better in terms of assigning data variations into the residual component.

For step 2), we tested out the following three methods:

First, we applied the Generalized Extreme Studentized Deviate (G-ESD) test to identify outliers. However, the method turned out being overly sensitive, detecting very high percentages of data points as outliers (~10% on average). This contradicts with our sponsor's expectation of at most 5% of outliers per account. Reason behind this turned out to be the limited amount of data we have. For each account, we got 9 years of monthly data, around 103 data points in each time series. The G-ESD method assumes the data values to roughly follow a normal distribution, which is violated when the data size is small.

Second, we applied the XmR chart method, an idea from the statistical control literature, which takes into account of the sequential variations among the values via calculating the moving ranges between adjacent data points. The method first calculates an average value for a given set of data points, then it calculates the upper and lower limits of variation by adding and subtracting from the average value the adjusted average moving range values. The biggest advantage of this method is its robustness to data distribution, meaning that the underlying data is not assumed to follow a normal distribution. The model's sensitivity to outliers reduced dramatically when compared to the G-ESD method, however when consulted with our sponsor, it still seems capturing more outliers than she expected. For instance, in the Figure 2 below, the point near end of 2012 is obviously a global anomaly, which our sponsor agrees that it should be detected. The other points beyond the limits though, are not of her concern since they might be local anomalies caused by various legitimate reasons such as temperature or occupation changes. The primary goal here for us it to detect the global anomalies only that might be caused by data entry error or broker infrastructures.

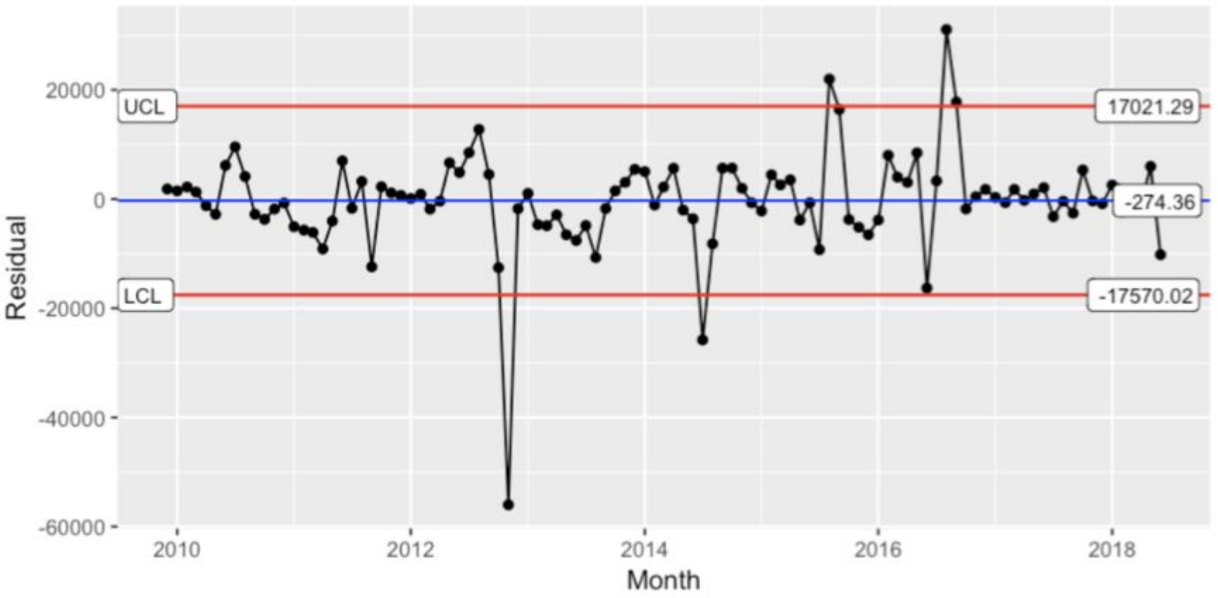

Figure 2

Lastly, we tried the Interquartile Range (IQR) method, an idea originated from Tukey's Boxplots. The biggest advantage of IQR method is its interpretability and ease of calculation. Given a set of residual values, we can easily calculate its quartiles, together with the upper and lower limits, to visualize the anomalous points. The method allows a user to manually choose the level of tolerance when checking outliers. The default setting of boundary limits is 3 times the interquartile range away from the 25th and 75th percentile, as shown in Figure 3. In case the user want to reduce the threshold, one can extend the boundary limits to be 6 times of IQR from the quartiles. These two methods resulted in further reduced sensitivities to outliers. However, the IQR-6X method oftentimes are too relaxed to detect any outlier.

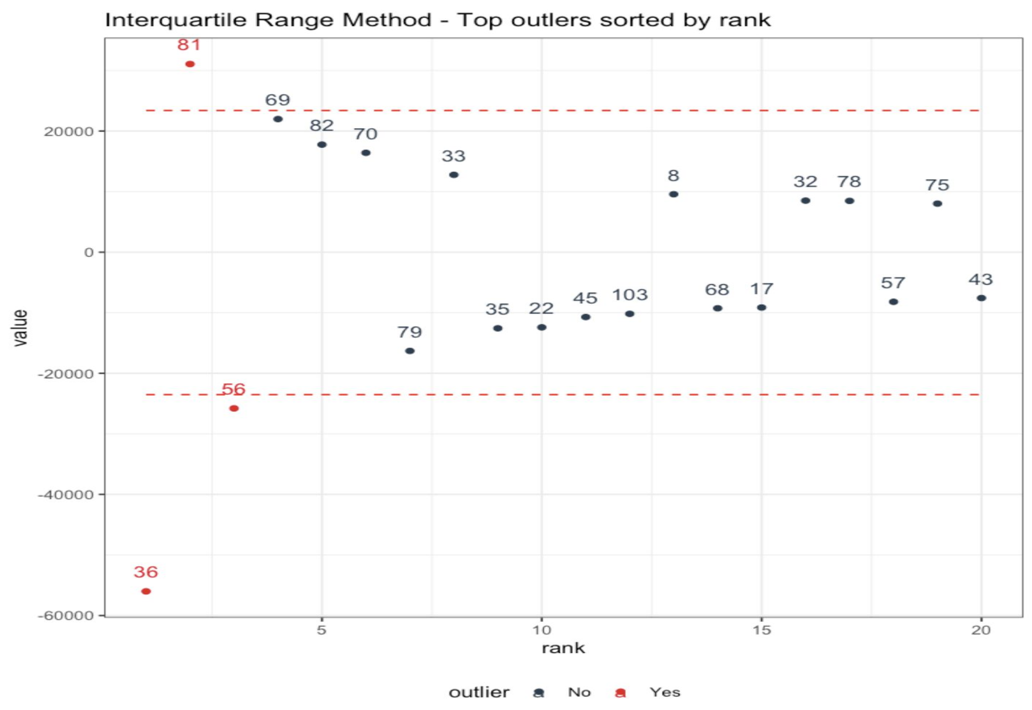

Figure 3

To reach a balance between the voice of the data and the expectations of our domain knowledge expert, we ended up with an ensemble approach of the last two methods. Specifically, if a data point is identified as outlier by the IQR-6X method, we label it as an outlier. If not, we check the IQR-3X and XmR methods, and treat it as an outlier when both these two methods consider it anomalous. Finally, we calculate a weighted rank for each outlier based on its distances from the limit values in all three methods. This will help our client to prioritize the inspection of outliers detected.

5.2 Prophet

At its core, the Prophet procedure is an additive regression model with four main components:

- A piecewise linear or logistic growth curve trend. Prophet automatically detects changes in trends by selecting changepoints from the data.
- A yearly seasonal component modeled using Fourier series.
- A weekly seasonal component using dummy variables.
- A user-provided list of important holidays.

The first step to using Prophet is to get the data into the specified format that it requires. If being implemented in Python, the data should be in a dataframe and must have exactly two columns

with the name ‘ds’ and ‘y’. The column ‘ds’ stands for the date (which could be monthly, daily or weekly) and ‘y’ denotes the quantity that we are interested in forecasting.
Once the data is in the required format, fitting the model is fairly straightforward, using a simple command like

model = Prophet(yearly_seasonality=True, weekly_seasonality=False, daily_seasonality=False, interval_width=0.95, mcmc_samples=50)

Let’s look at each of the parameters in this statement.

‘Yearly_seasonality’ - informs Prophet that the data is in a monthly format, and periodically cycles every 12 months, as is true in our case. Depending on the data Prophet also supports weekly and daily seasonality, which have been marked as False in this instance.

‘Interval_Width’ - By default Prophet uses 80% of the random samples generated in the Monte Carlo simulations to define what the uncertainty interval must be for each forecasted value. This value has been changed to 95% in our case, which helps broaden the lower and upper boundaries and increase the width of this threshold.

‘Mcmc_samples’ - Using the 'mcmc_samples' parameter, which stands for Markov Chains Monte Carlo, Prophet will also take into consideration the uncertainty in seasonality. By default, Prophet only considers the uncertainty in the trend component. Including this parameter results in a slightly longer duration for execution, since full Bayesian sampling is performed. Also the value is set at 50 in our case, since we have restricted our data to only those accounts that have at least 50 rows.

Once the model has been fit, we can use it to predict future and past values. The individual trend and seasonality components can be visualized using a basic plot and a sample has been shown below

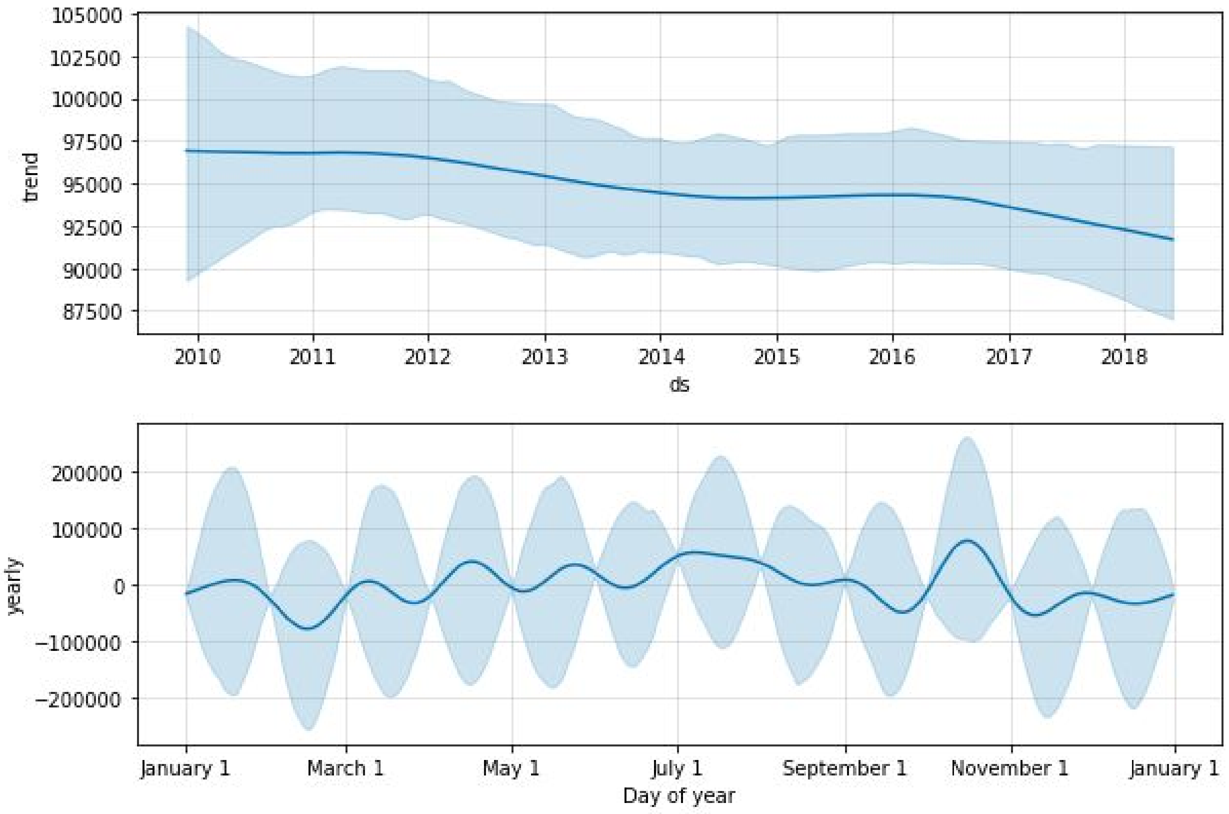

Figure 4

The first plot highlights a slightly decreasing trend for this account over the years and as far as seasonality is concerned, we can see a slight increase during the summer months, but is otherwise fairly constant in this case.

The forecasted values generated by Prophet, can be visualized against the original values and this is depicted in the plot below

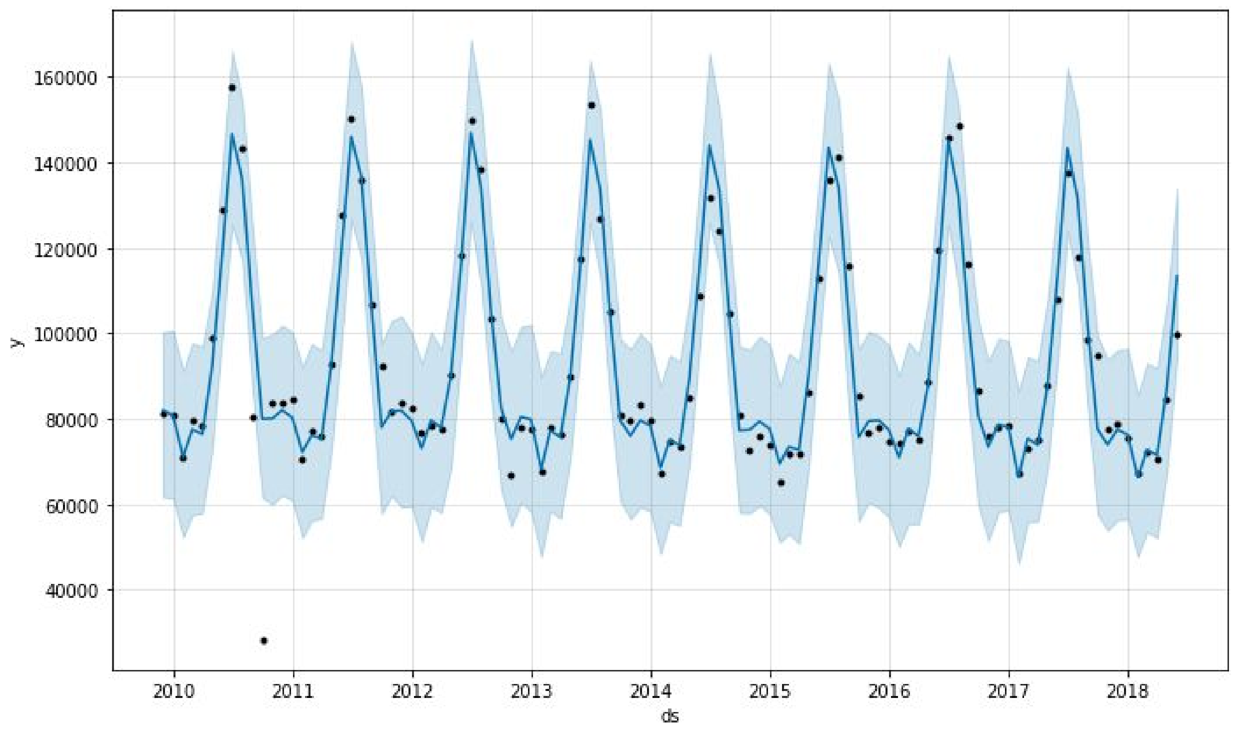

Figure 5

If we analyze the plot above, the shaded blue area is the uncertainty interval generated by Prophet using Monte Carlo simulations, the solid blue line is the forecasted value generated by the model that we fit above, and the black dots are the original values for this account.
It is relatively easy to determine outliers by looking at black dots that do not lie in this uncertainty interval and flagging them.

The uncertainty interval can be tweaked using the parameters 'interval_width' and 'mcmc_samples' described above, to increase or decrease the number of outliers detected.

5.3 Clustering

The time series clustering method leverages the assumption that majority of the observed data points are normal to build out reconstruction for the time series trend using cluster centroids for each of the observed points. There are four major steps in the clustering method:

1) On each individual account level, time series trend is sliced into 8 data point segments with sliding window of 1 step. The graph below shows an example of 9 waveforms, each waveform is 2 steps forward from the previous one.

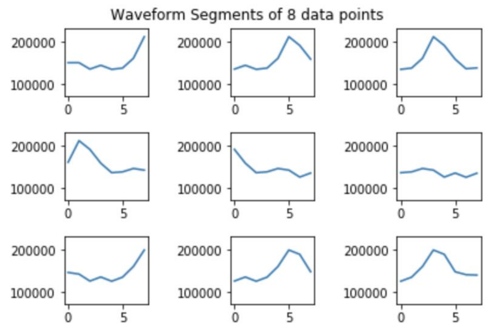

Figure 6

2) Perform KMeans clustering on segments from each account into 12 clusters. Each cluster should capture the segment waveform on a monthly level. The graph below shows an example of nearest centroid for a particular month.

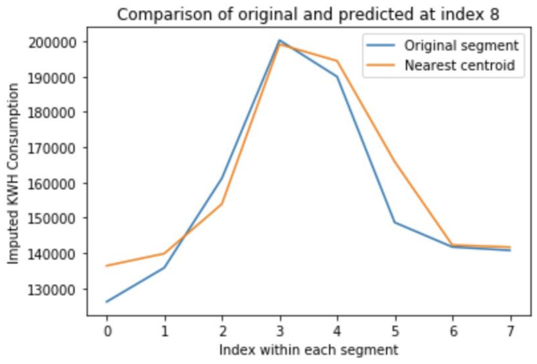

Figure 7

3) With the nearest centroids that pass each data points, reconstruct the time series trend with the mean of centroids at each point. This is built upon the assumption that majority of the data points are normal observations and the clusters should capture the majority waveforms. Below graph shows the 8 nearest centroids that pass through one datapoint.

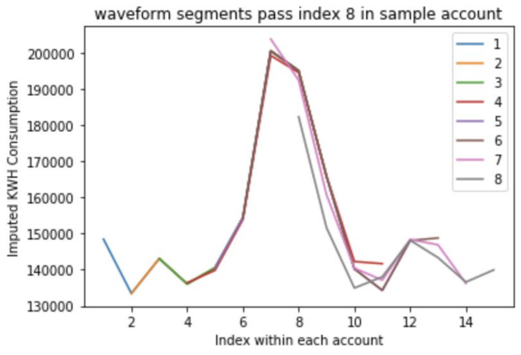

Figure 8

4) Lastly, the reconstructed trend is compared with the original trend. Reconstruction error is calculated by taking the absolute difference between the reconstructed value and original value of each point. Based on the frequency of anomaly occurrence, a threshold can be set on the error terms to detect anomalous observations.

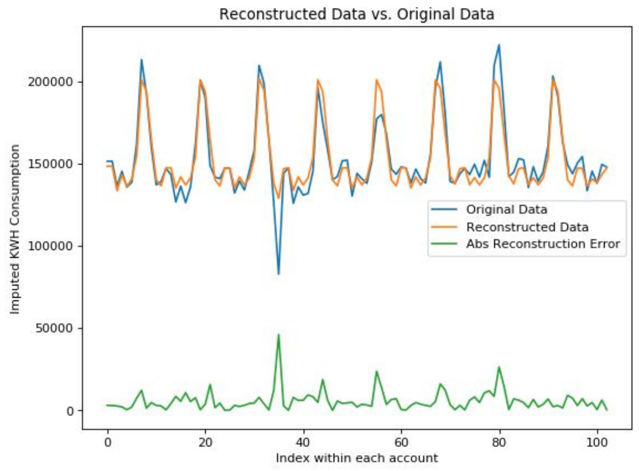

Figure 9

The final results of the clustering methods consist of predicted labels for anomalous observations, reconstruction value based on nearest centroids mean, and reconstruction error. Users are able to adjust the threshold of error term to fit domain knowledge expectations.

6. Results

6.1 Histogram comparison

We first evaluated the results from all three methods using a histogram distribution of the percentages of anomalies detected. Based on domain expert knowledge, the anomaly rate in electricity consumption should be around 2-5% on average, so we used 5% as our benchmark line.

From the graphs below, we can see all three methods have majority of the percentage of anomaly detected in the 2-5% range. For Prophet and Clustering methods, more than 20% of the accounts have 2% anomaly detected; for Decomposition method, the percentage of anomaly detected across all accounts are more uniformly distributed, with more than 50% fall into the 2-5% range.

All three methods have threshold that can be relaxed or tightened based on end users' need.

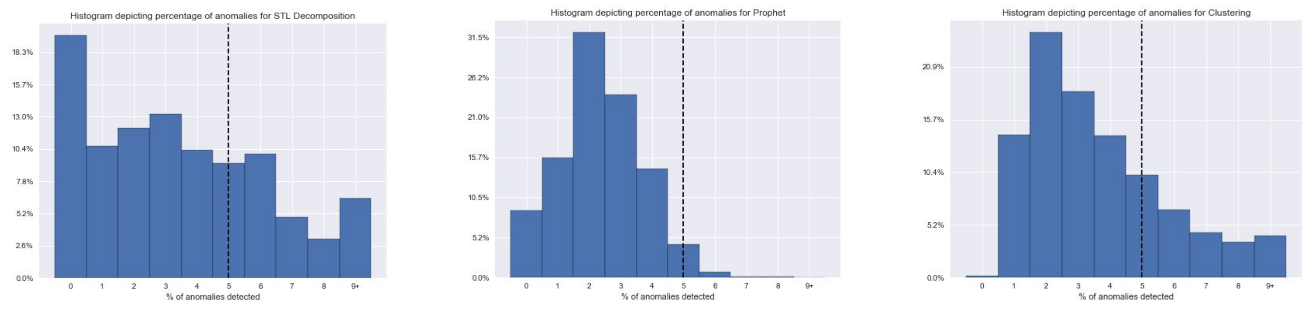

Figure 10

6.2 Usage recommendation

Since we developed three methods in parallel, we want to provide not only the quantitative metrics evaluation of the methods but also the qualitative usage comparison amongst three methods. End users can leverage this usage comparison to choose the best method that fit their specific use case. We run all three methods through 1,916 accounts and all three finished within 6 minutes. If the use case contains large amount of accounts, users are advised to choose either Decomposition or Prophet methods. All three methods are relatively easy to implement and there are parameters users can tune to cater towards a set benchmark percentage of anomalous observations detected. Because Prophet package is commercially developed by Facebook and easy to implement, we lose some of the interpretability aspect of the method. However, both Decomposition and Clustering methods have step by step implementation codes and are highly interpretable.

Please refer to the below table for usage comparison across all three methods:

Methods	**Runtime**	**Interpretability**	**Robustness to Missing Values**	**Implementation Ease**
Decomposition	Low(~0.1s)	High	High	Low
Prophet	Low(~0.1s)	Low	High	High
Clustering	Medium(~0.2s)	High	Medium	Medium

Table 1

Specifically, we come up with the following rules for a user to choose between these methods. (also illustrated in the flow chart of Figure 11)

1) If there is not enough data points (e.g. less than 50), consider IQR test (default setting) or the XmR chart method to detect outliers

2) If there are enough data points (e.g. more than 50)
    a) If the user prefers auto-tuning of model and high scalability, use Prophet package
    b) If the user prefers high model interpretability or would like to manually set up outlier detection threshold, choose the STL decomposition or clustering methods

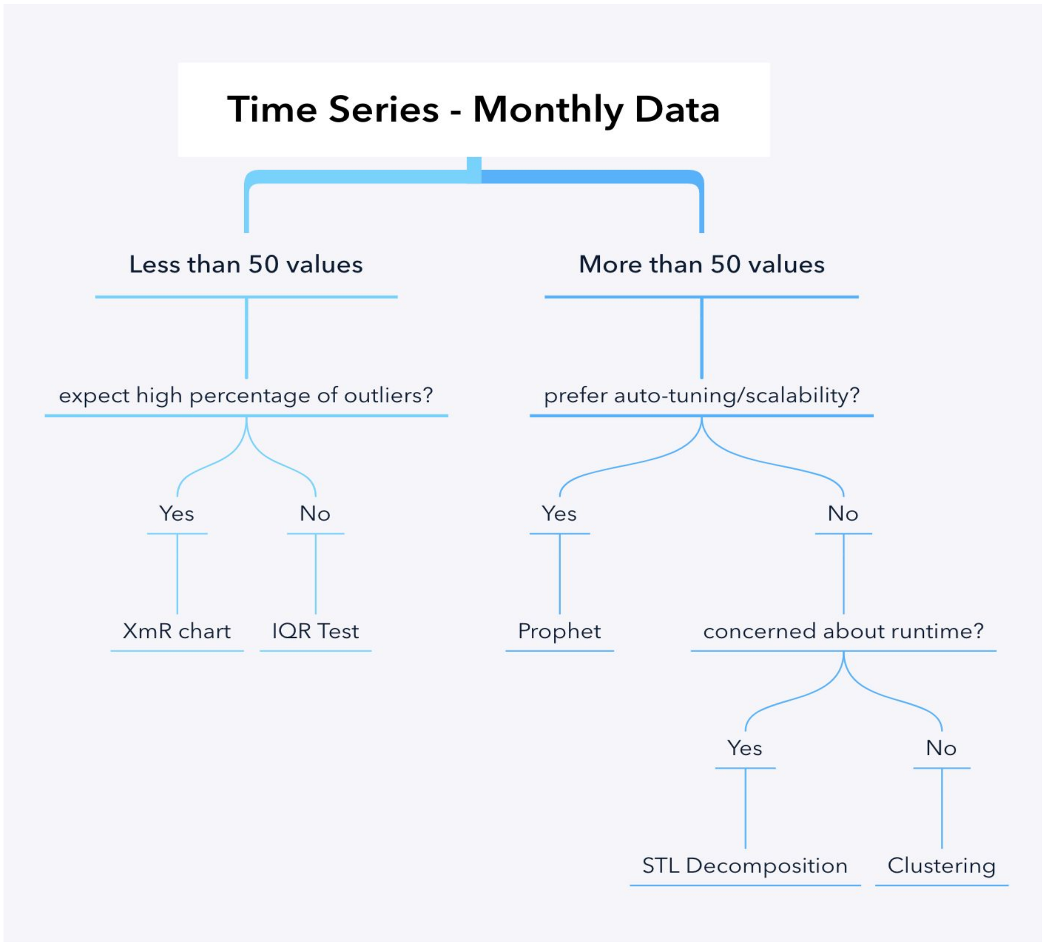

Figure 11

7. Conclusion

This project has successfully developed three methods to detect anomalies in energy consumption time series data. These three methods are not only useful to automate the process of detecting potential anomalous entries in data, but also helpful to provide information and explanations to the PEERS regression model used by our sponsor.

We built these methods upon evaluation of the NYCHA dataset, which has similar seasonal patterns and characteristics compared to our client dataset. However, these methods are robust to most time series based anomaly detection. Users can leverage all three methods to build an ensemble output or consult the recommendations we provided above for individual methods usage.

8. References

[1] New York City Housing Authority Dataset: https://opendata.cityofnewyork.us
[2] Time Series Anomaly Detection Algorithms
https://blog.statsbot.co/time-series-anomaly-detection-algorithms-1cef5519aef2
[3] A closer look at time series anomaly detection
https://www.anodot.com/blog/closer-look-time-series-anomaly-detection/
[4] Chandola, V.; Banerjee, A.; Kumar, V. (2009). "Anomaly detection: A survey". ACM Computing Surveys. 41 (3): 1–58.
[5] *Robert B. Cleveland, William S. Cleveland, Jean E. McRae, and Irma Terpenning. STL: a seasonal-trend decomposition procedure based on loess. Journal of Official Statistics, 6(1):3–73, 1990.*
[6] J. Hochenbaum, O. S. Vallis, A. Kejariwal, "Automatic anomaly detection in the cloud via statistical learning", CoRR, vol. abs/1704. 07706, 2017.
[7] *Taylor SJ, Letham B.* Prophet: *2017. Forecasting at scale.*
[8] *F. Gullo, G. Ponti, A. Tagarelli, G. Tradigo, P. Veltri, A time series approach for clustering mass spectrometry data, J. Comput. Sci. 3 (5) (2011) 344–355. 2010*.
[9] P.K. Chan, M.V. Mahoney, Modeling multiple time series for anomaly detection, in: Proceedings of Fifth IEEE International Conference on Data Mining, 2005, pp. 90–97.
[10] F. Iglesias, W. Kastner, Analysis of similarity measures in times series clustering for the discovery of building energy patterns, Energies 6 (2) (2013) 579–597.