Problem Identification

Problem Statement

XYZ company operates a manufacturing plant and makes household cleaning supplies such as detergents, shampoo etc. for over 50 customers across 4 different countries with the revenue of $800 million per year. Their operations heavily depend on over 30 industrial pumps across their manufacturing plant to make their products. In the last 6 months, one of their pumps failed 7 times unexpectedly resulting in an estimated $500,000 in production loss and additional $300,000 in environmental damage due to the spilled toxic chemicals. To avoid this problem in the future, they want to be able to detect anomalies in the pump behaviour and be able to stop the pump before it goes down hard.

Context

Manufacturing industry is considered a heavy industry in which they tend to utilize various types of heavy machinery such as giant motors, pumps, pipes, furnaces, conveyor belts, haul trucks, dozers, graders, and electric shovels etc. These are often considered as the most critical assets for their operations. Therefore, the integrity and reliability of these equipment is often the core focus of their Asset Management programs.

The prime reason why they care so much about these assets is that the failure of these equipment often results in production loss that could consequently lead to loss of hundreds of thousands of dollars if not millions depending on the size and scale of the operations. So this is a pretty serious deal for a Maintenance Manager of a manufacturing plant to run a robust Asset Management framework with highly skilled Reliability Engineers to ensure the reliability and availability of these critical assets.

Therefore, the ability to detect anomalies in advance and be able to mitigate risks is a very valuable capability which further allows to prevent unplanned downtime, unnecessary maintenance (condition based vs mandatory maintenance) and will also enable more effective way of managing critical components for these assets. The production loss from unplanned downtime, the cost of unnecessary maintenance and having excess or shortage of critical components translate into serious magnitudes in terms of dollar amount.

Criteria For Success

A model that detects the anomalies with at least 75% accuracy and generalizes well on other samples with zero rework.

Scope of The Solution Space

The scope of this project is limited to detecting anomalies in the 53 sensors of the selected pump and excludes other pumps and predicting the failures of the pump.

Constraints

Data set is limited to the sensor readings from a single pump hence may not be the best representation of all the pumps. Computing power might become a constraint for effectively visualizing all 53 features at the same time.

Stakeholders

Samwell Tarly - Maintenance Manager of the XYZ Company

Data

The data set is sourced from https://www.kaggle.com/nphantawee/pump-sensor-data and consists of 53 numerical features and a categorical label that represent the state of the pump. 53 numerical features contain raw sensor readings from 53 different sensors that monitor various attributes of the pump.The label contains string values that represent normal, broken and recovering operational conditions of the pump. The data set represents 219,521 readings from 53 sensors.

Solution Approach

I first built a benchmark model using IQR technique and then I implemented two other unsupervised learning algorithms to compare their resulting performances and accuracies. In doing so, I followed the following steps:

- Data sourcing and loading
- Data wrangling
- Exploratory Data Analysis (EDA)
- Pre-Processing and Feature Engineering
- Modeling
- Model Evaluation

Project Deliverables

- Final Project Report
- Final Presentation
- Jupyter Notebook that contains all of the Python code
- Published article on TowardsDataScience

Data Wrangling

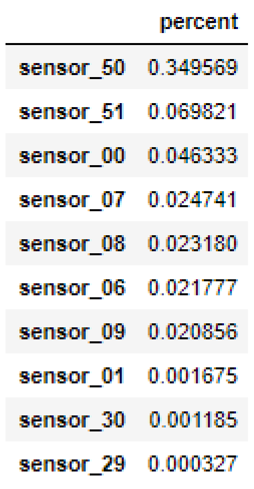
The data set has 55 columns in total including 52 sensors, timestamp, machine status and Unnamed column. The information about the data set is as follows.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220320 entries, 0 to 220319
Data columns (total 55 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   Unnamed: 0      220320 non-null  int64  
 1   timestamp       220320 non-null  object 
 2   sensor_00       210112 non-null  float64
 3   sensor_01       219951 non-null  float64
 4   sensor_02       220301 non-null  float64
 5   sensor_03       220301 non-null  float64
 6   sensor_04       220301 non-null  float64
 7   sensor_05       220301 non-null  float64
 8   sensor_06       215522 non-null  float64
 9   sensor_07       214869 non-null  float64
 10  sensor_08       215213 non-null  float64
 11  sensor_09       215725 non-null  float64
 12  sensor_10       220301 non-null  float64
 13  sensor_11       220301 non-null  float64
 14  sensor_12       220301 non-null  float64
 15  sensor_13       220301 non-null  float64
 16  sensor_14       220299 non-null  float64
 17  sensor_15       0 non-null       float64
 18  sensor_16       220289 non-null  float64
 19  sensor_17       220274 non-null  float64
 20  sensor_18       220274 non-null  float64
 21  sensor_19       220304 non-null  float64
 22  sensor_20       220304 non-null  float64
 23  sensor_21       220304 non-null  float64
 24  sensor_22       220279 non-null  float64
 25  sensor_23       220304 non-null  float64
 26  sensor_24       220304 non-null  float64
 27  sensor_25       220284 non-null  float64
 28  sensor_26       220300 non-null  float64
 29  sensor_27       220304 non-null  float64
 30  sensor_28       220304 non-null  float64
 31  sensor_29       220248 non-null  float64
 32  sensor_30       220059 non-null  float64
 33  sensor_31       220304 non-null  float64
 34  sensor_32       220252 non-null  float64
 35  sensor_33       220304 non-null  float64
 36  sensor_34       220304 non-null  float64
 37  sensor_35       220304 non-null  float64
 38  sensor_36       220304 non-null  float64
 39  sensor_37       220304 non-null  float64
 40  sensor_38       220293 non-null  float64
 41  sensor_39       220293 non-null  float64
 42  sensor_40       220293 non-null  float64
 43  sensor_41       220293 non-null  float64
 44  sensor_42       220293 non-null  float64
 45  sensor_43       220293 non-null  float64
 46  sensor_44       220293 non-null  float64
 47  sensor_45       220293 non-null  float64
 48  sensor_46       220293 non-null  float64
 49  sensor_47       220293 non-null  float64
 50  sensor_48       220293 non-null  float64
 51  sensor_49       220293 non-null  float64
 52  sensor_50       143303 non-null  float64
 53  sensor_51       204937 non-null  float64
 54  machine_status  220320 non-null  object
dtypes: float64(52), int64(1), object(2)
memory usage: 92.5+ MB
```

As seen from above, clearly, this data set requires some cleaning before it can be analyzed further.

- There is a column with missing values in the entire column
- The column “Unnamed: 0“ appears to be a duplicated index column
- There are some missing values in the other columns.
- Timestamp is of type object, not datetime.

The percentage of missing values are as follows:

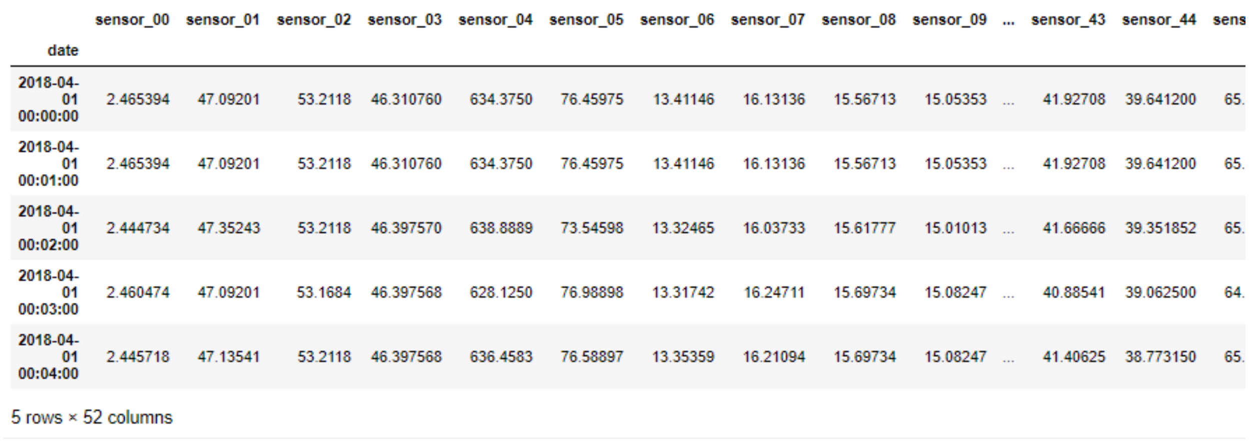


	percent
sensor_50	0.349569
sensor_51	0.069821
sensor_00	0.046333
sensor_07	0.024741
sensor_08	0.023180
sensor_06	0.021777
sensor_09	0.020856
sensor_01	0.001675
sensor_30	0.001185
sensor_29	0.000327

To clean-up the data, the following steps were applied.

- Remove redundant column (sensor_15)
- Remove duplicated column (Unnamed: 0)
- Handle missing values
- Convert the type of timestamp to pandas datetime

After removing the entire column with the missing values and the duplicated column, I looked at the data deeper to think about how to handle the missing values in the remaining columns in the data set. I decided to impute some of the missing values with their mean and to drop the rest of the missing values. After the data wrangling process, my final tidy data set includes 52 sensors/features, datetime and machine status column that contains three classes that represent the NORMAL, BROKEN, or RECOVERING operating conditions of the pump. The first 5 rows of the tidy data looks like as follows.



	sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	sensor_05	sensor_06	sensor_07	sensor_08	sensor_09	...	sensor_43	sensor_44	sens
date														
2018-04-01 00:00:00	2.465394	47.09201	53.2118	46.310760	634.3750	76.45975	13.41146	16.13136	15.56713	15.05353	...	41.92708	39.641200	65.
2018-04-01 00:01:00	2.465394	47.09201	53.2118	46.310760	634.3750	76.45975	13.41146	16.13136	15.56713	15.05353	...	41.92708	39.641200	65.
2018-04-01 00:02:00	2.444734	47.35243	53.2118	46.397570	638.8889	73.54598	13.32465	16.03733	15.61777	15.01013	...	41.66666	39.351852	65.
2018-04-01 00:03:00	2.460474	47.09201	53.1684	46.397568	628.1250	76.98898	13.31742	16.24711	15.69734	15.08247	...	40.88541	39.062500	64.
2018-04-01 00:04:00	2.445718	47.13541	53.2118	46.397568	636.4583	76.58897	13.35359	16.21094	15.69734	15.08247	...	41.40625	38.773150	65.

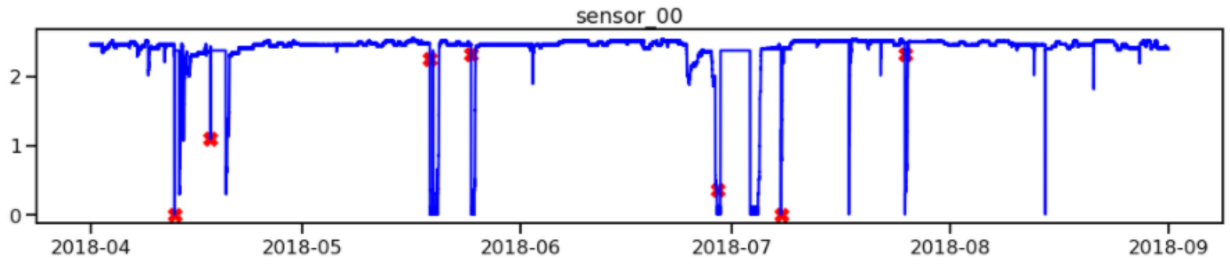
5 rows × 52 columns

Exploratory Data Analysis

In addition to having looked at descriptive statistics of numerical columns, I performed various graphical EDA to look for trends and any odd behaviors. In particular, it is interesting to see the sensor readings plotted over time with the machine status of “BROKEN” marked up on the same graph in red color. That way, I could clearly see when the pump breaks down and how that reflects in the sensor readings. The following code plots the mentioned graph for each of the sensors, but let’s take a look at the graph for the sensor_00.

```
# Extract the readings from the BROKEN state of the pump
broken = df[df['machine_status']=='BROKEN']
# Extract the names of the numerical columns
df2 = df.drop(['machine_status'], axis=1)
names=df2.columns
```

```
# Plot time series for each sensor with BROKEN state marked with
X in red color
for name in names:
    _ = plt.figure(figsize=(18,3))
    _ = plt.plot(broken[name], linestyle='none', marker='X',
color='red', markersize=12)
    _ = plt.plot(df[name], color='blue')
    _ = plt.title(name)
    plt.show()
```

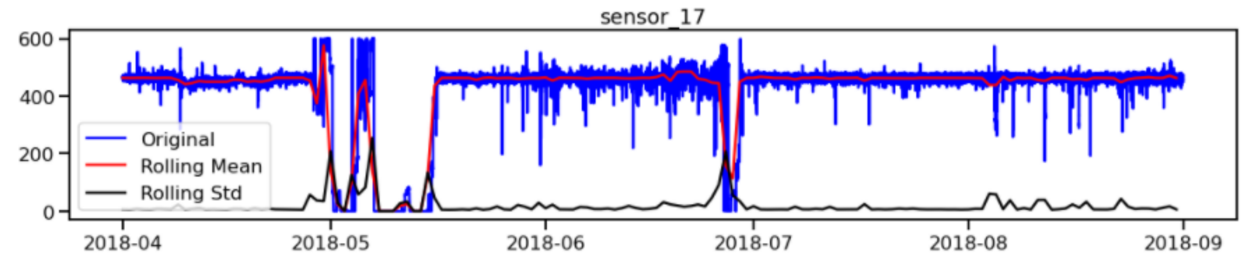


As seen clearly from the above plot, the red marks, which represent the broken state of the pump, perfectly overlaps with the observed disturbances of the sensor reading. After having looked at the similar plots for the rest of the sensors, I have a pretty good intuition about how each sensor reading behaves when the pump is broken vs operating normally.

Stationarity and Autocorrelation

In time series analysis, it is important that the data is stationary with no autocorrelation. Stationarity refers to the behavior where the mean and the standard deviation of the data changes over time, the data with such behavior is said to be non-stationary. On the other hand, autocorrelation refers to the behavior of the data where the data is correlated with itself in a different time period. In the next step, I visually inspected the stationarity of each feature in the data set. To do that, I had to first resample the data by daily average. Later, I performed the Dickey Fuller test to quantitatively verify the observed stationarity. In addition, I inspected the autocorrelation of the features before feeding them into the clustering algorithms to detect anomalies.

```
# Resample the entire dataset by daily average
rollmean = df.resample(rule='D').mean()
rollstd = df.resample(rule='D').std()
# Plot time series for each sensor with its mean and standard
deviation
for name in names:
    _ = plt.figure(figsize=(18,3))
    _ = plt.plot(df[name], color='blue', label='Original')
    _ = plt.plot(rollmean[name], color='red', label='Rolling
Mean')
    _ = plt.plot(rollstd[name], color='black', label='Rolling
Std' )
    _ = plt.legend(loc='best')
    _ = plt.title(name)
    plt.show()
```

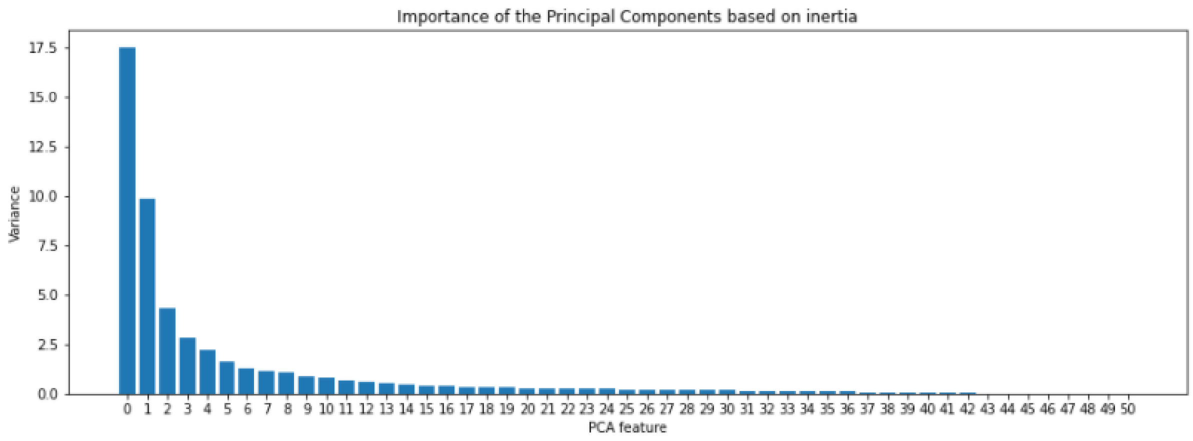


Looking at the readings from one of the sensors, sensor_17 in this case, notice that the data actually looks pretty stationary where the rolling mean and standard deviation don't seem to change over time except during the downtime of the pump which is expected. This was the case for most of the sensors in this data set but it may not always be the case in which situations various transformation methods must be applied to make the data stationary before training the data.

Pre-Processing and Dimensionality Reduction

It is pretty computationally expensive to train models with all of the 52 sensors/features and it is not efficient. Therefore, I used Principal Component Analysis (PCA) technique to extract new features to be used for the modeling. In order to properly apply PCA, the data must be scaled and standardized. This is because PCA and most of the learning algorithms are distance based algorithms. If noticed from the first 10 rows of the tidy data, the magnitude of the values from each feature is not consistent. Some are very small while some others are really large values. Hence, the following steps were applied using the Pipeline library.

1. Scale the data
2. Perform PCA and look at the most important principal components based on inertia



It appears that the first two principal components are the most important as shown by the importance plot. As such, I performed PCA with 2 components which are the features to be used in the training models.

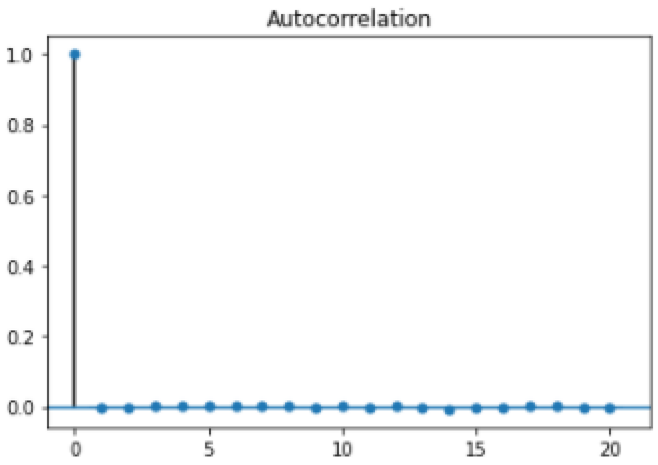
```
# Calculate PCA with 2 components
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents, columns =
['pc1', 'pc2'])
```

Then it is important to check again the stationarity and autocorrelation of the new derived principal components just to be sure they are stationary and not autocorrelated.

```
from statsmodels.tsa.stattools import adfuller
# Run Augmented Dickey Fuller Test
result = adfuller(principalDf['pc1'])
# Print p-value
print(result[1])
```

Running the Dickey Fuller test on the 1st principal component, I get a p-value of 5.4536849418486247e-05 which is a very small number (much smaller than 0.05). Thus, I rejected the Null Hypothesis and said the data is stationary. I performed the same on the 2nd component and got a similar result. So both of the principal components are stationary which is what I wanted.

Then I checked for autocorrelation of the principal components. It can be done one of the two ways; either with the pandas autocorr() method or ACF plot. I used the latter in this case to quickly visually verify that there is no autocorrelation.



Given the new features (principal components) from PCA are stationary and not autocorrelated, the section demonstrates the modeling with these principal components.

Modeling

In this step, the following learning algorithms were implemented to detect anomalies.

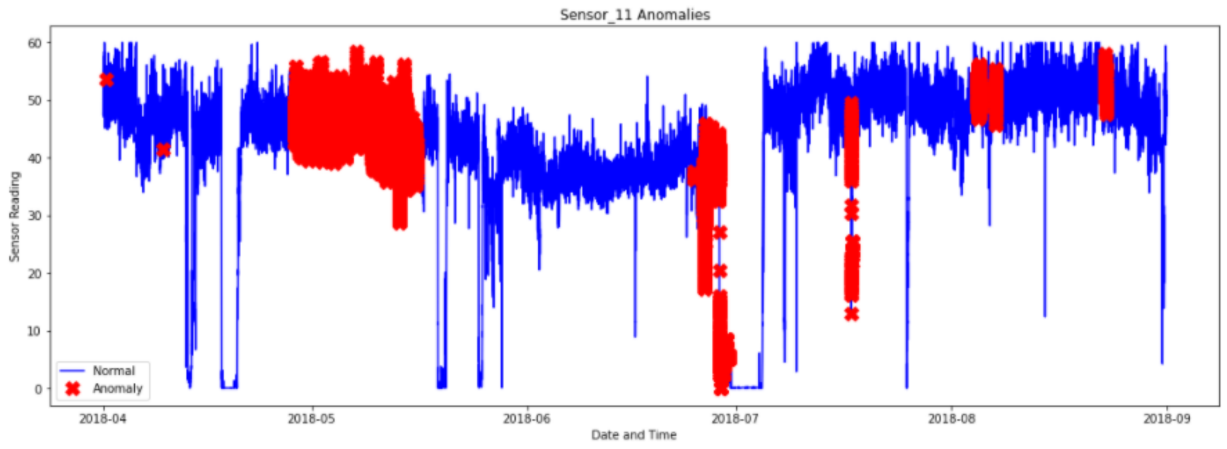
1. Benchmark model: Interquartile Range (IQR)
2. K-Means clustering
3. Isolation Forest

The idea of implementing the IQR is to use it as a benchmark against the other two unsupervised learning algorithms.

Interquartile Range

The following strategy was followed to implement the benchmark model with the Interquartile Range.

1. Calculate IQR which is the difference between 75th (Q3)and 25th (Q1) percentiles.
2. Calculate upper and lower bounds for the outlier.
3. Filter the data points that fall outside the upper and lower bounds and flag them as outliers.
4. Finally, plot the outliers on top of the time series data (the readings from sensor_11 in this case)

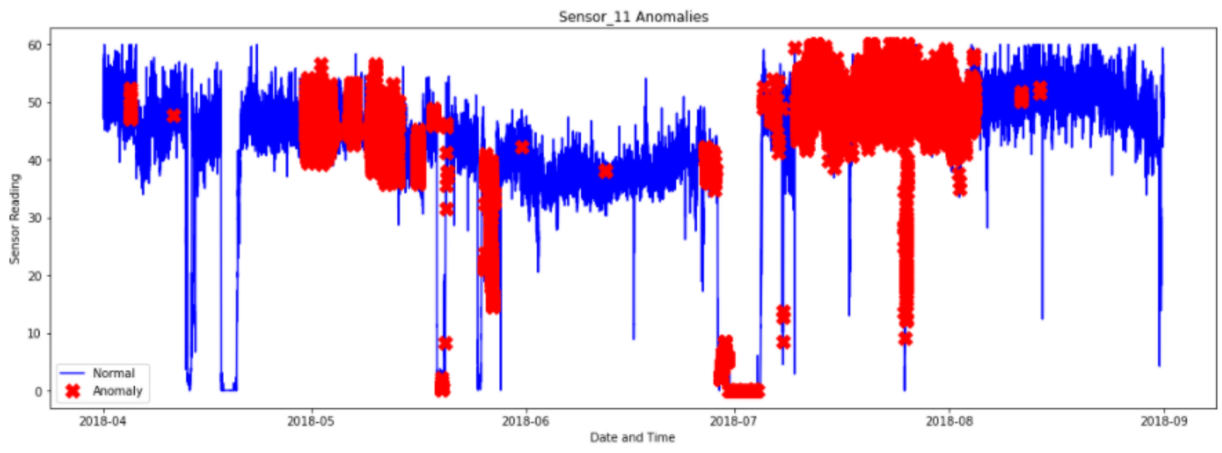


As seen above, the anomalies were detected right before the pump broke down. This could be very valuable information for an operator to see and be able to shut down the pump properly before it actually goes down hard. Let's see if we detect similar patterns in anomalies from the next two algorithms.

K-Means Clustering

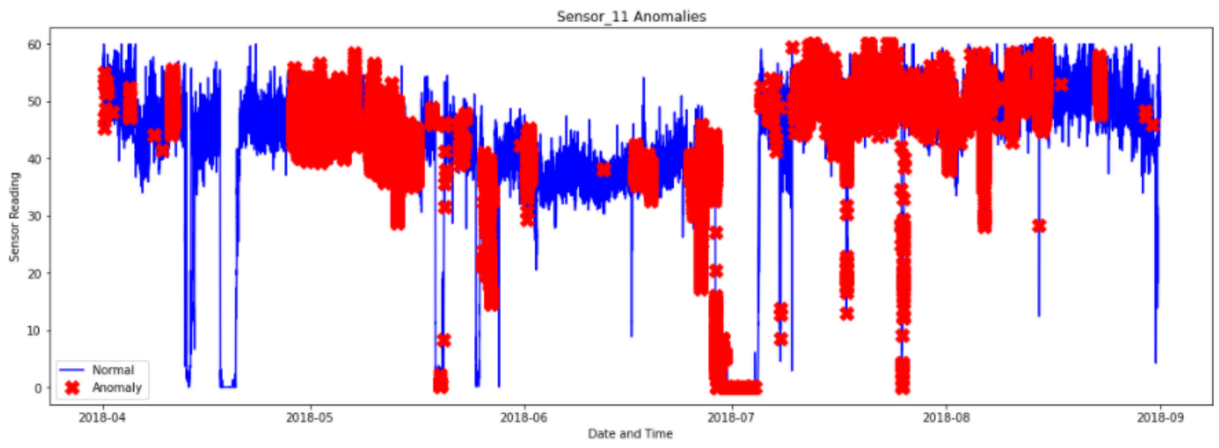
The following strategy was followed to implement the K-Means Clustering algorithm.

1. Calculate the distance between each point and its nearest centroid. The biggest distances are considered anomalies.
2. We use outliers_fraction to provide information to the algorithm about the proportion of the outliers present in our data set. Situations may vary from data set to data set. However, as a starting figure, I estimate outliers_fraction=0.13 (13% of df are outliers as depicted).
3. Calculate number_of_outliers using outliers_fraction.
4. Set the threshold as the minimum distance of these outliers.
5. The anomaly result of anomaly1 contains the above method Cluster (0:normal, 1:anomaly).
6. Visualize anomalies with Time Series view.



As seen above, K-Means clustering appears to have detected far more anomalies than that detected by the IQR. The trick here was to manipulate the parameter “outliers_fraction”. Changing the value of the parameter “outliers_fraction” resulted in different outcomes.

Isolation Forest



Model Evaluation

It is interesting to see that all three models detected a lot of the similar anomalies but actually IQR has detected more anomalies than the other two algorithms as per the following table. It appears the K-Means Clustering and Isolation Forest have produced very similar results. Based on this I conclude that IQR is the best of these three in detecting more anomalies. However, the performance of the K-Means and Isolation Forest based models could be further improved significantly by applying more advanced hyperparameter tuning.

Anomalies detected by IQR:

0 (NORMAL)	189,644
1 (ANOMALY)	29,877

Anomalies detected by K-Means Clustering:

0 (NORMAL)	190,984
1 (ANOMALY)	28,537

Anomalies detected by Isolation Forest:

0 (NORMAL)	190,983
1 (ANOMALY)	28,538

Challenges

One of the challenges I faced during this project is that training anomaly detection models with unsupervised learning algorithms with such a large data set can be computationally very expensive. For example, I couldn't properly train SVM on this data as it was taking a very long time to train the model with no success.

Next steps:

I suggest the following next steps in which the first 3 steps are focused on improving the model and the last two are about making things real:

1. Feature selection with more advanced Feature engineering techniques
2. Advanced hyperparameter tuning
3. Implement other learning algorithms such as SVM and DBSCAN
4. Predict the machine status with the best model given a test set
5. Deploy the best model into production

The Jupyter notebook can be found at this link https://github.com/bauyrjanj/Anomaly_Detection