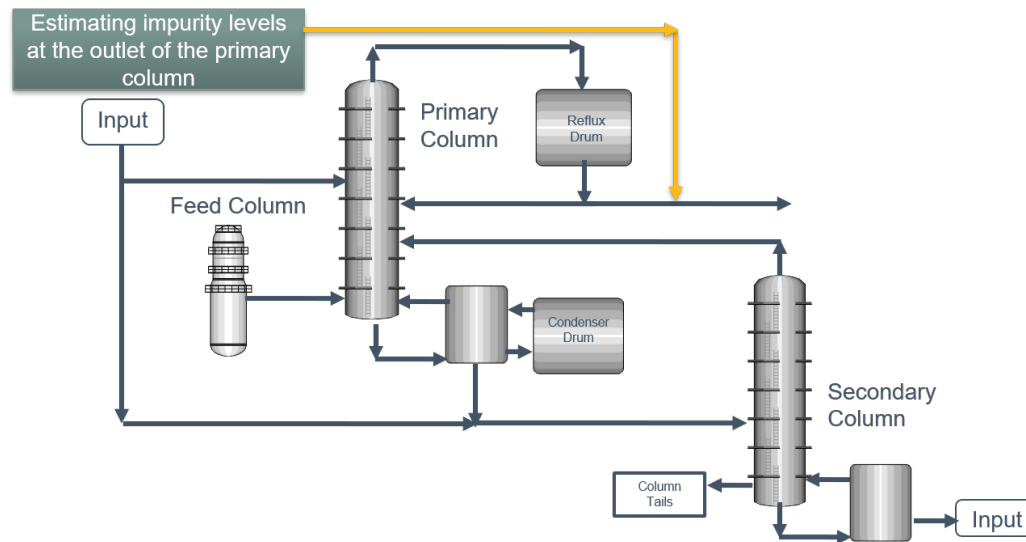


Lecture 9: Identifying Outliers

Goals for Today:

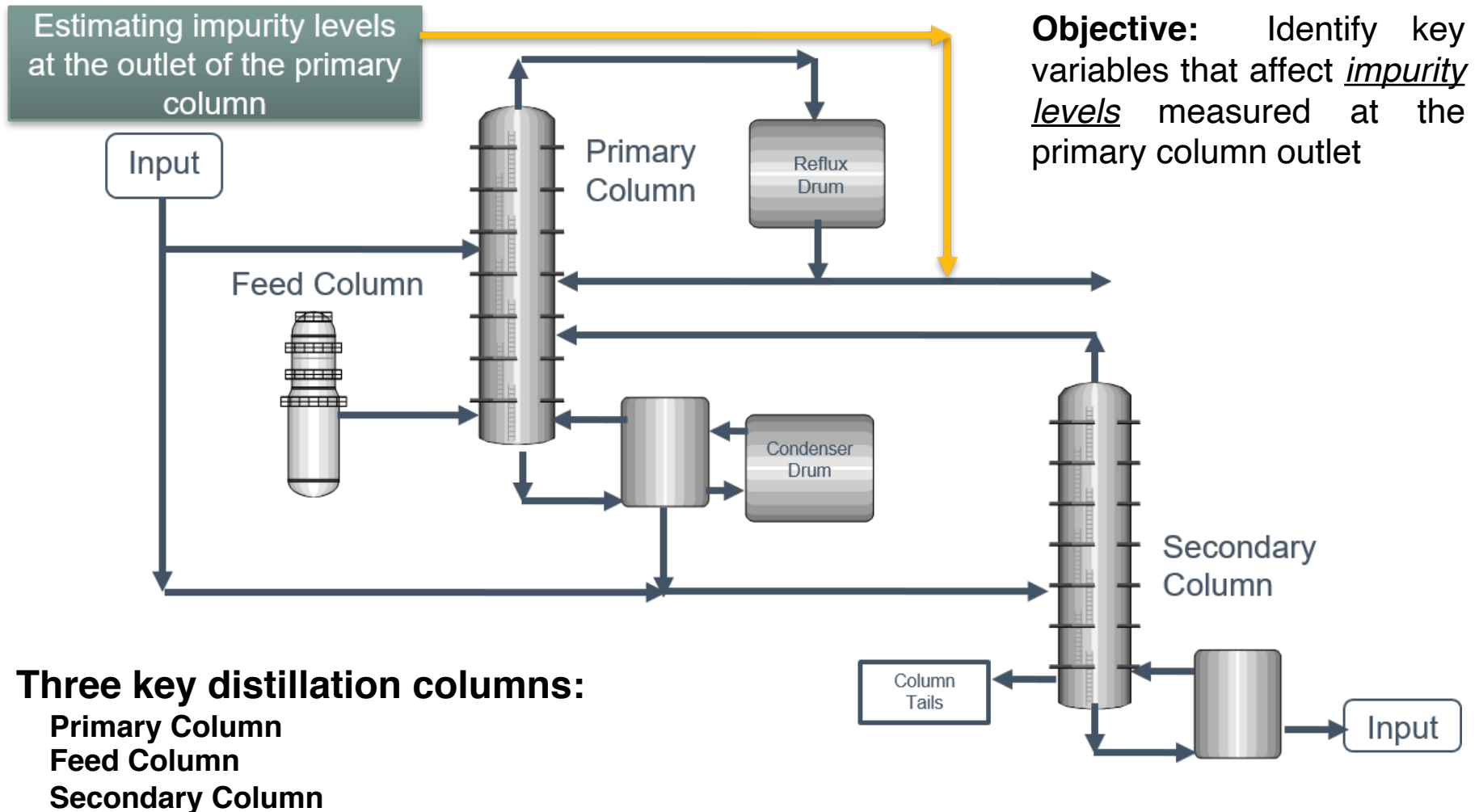
Identifying Outliers:

- There is no Black Box
- Robust Z-Score (Hampel Filter)
- Principal Component Analysis (PCA)



Dow Dataset - Overview

Dow has generously given us operating data for a plant:



Dow Dataset - Overview

Summary of Variables

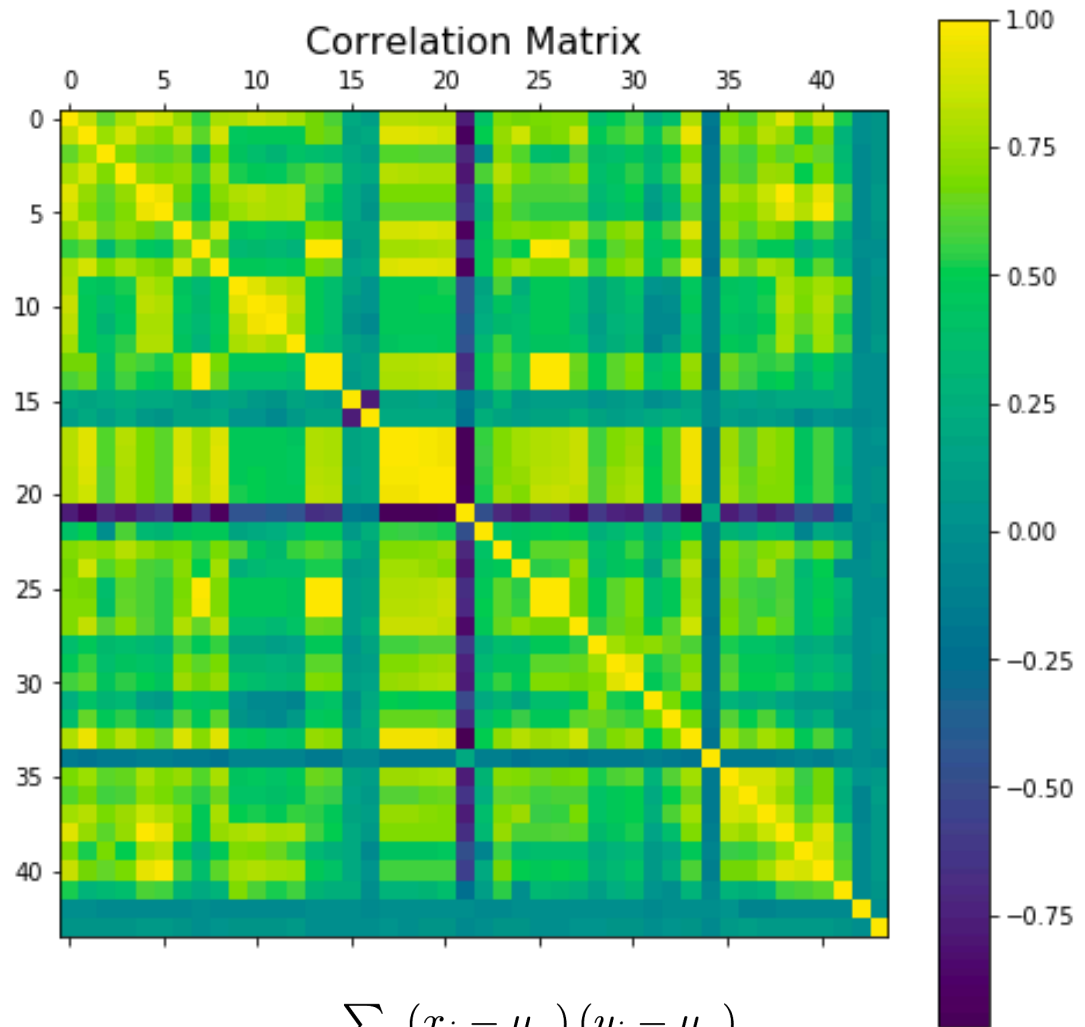
Primary Column Variables	Secondary Column Variables	Feed Column Variables
x1:Primary Column Reflux Flow x2:Primary Column Tails Flow x3:Input to Primary Column Bed 3 Flow x4:Input to Primary Column Bed 2 Flow x5:Primary Column Feed Flow from Feed Column x6:Primary Column Make Flow x7:Primary Column Base Level x8:Primary Column Reflux Drum Pressure x9:Primary Column Condenser Reflux Drum Level x10:Primary Column Bed1 DP x11:Primary Column Bed2 DP x12:Primary Column Bed3 DP x13:Primary Column Bed4 DP x14:Primary Column Base Pressure x15:Primary Column Head Pressure x16:Primary Column Tails Temperature x17:Primary Column Tails Temperature 1 x18:Primary Column Bed 4 Temperature x19:Primary Column Bed 3 Temperature x20:Primary Column Bed 2 Temperature x21:Primary Column Bed 1 Temperature Avg_Reactor_Outlet_Impurity Avg_Delta_Composition Primary Column y:Impurity Primary Column Reflux/Feed Ratio Primary Column Make/Reflux Ratio	x22: Secondary Column Base Concentration x23: Flow from Input to Secondary Column x24: Secondary Column Tails Flow x25: Secondary Column Tray DP x26: Secondary Column Head Pressure x27: Secondary Column Base Pressure x28: Secondary Column Base Temperature x29: Secondary Column Tray 3 Temperature x30: Secondary Column Bed 1 Temperature x31: Secondary Column Bed 2 Temperature x32: Secondary Column Tray 2 Temperature x33: Secondary Column Tray 1 Temperature x34: Secondary Column Tails Temperature x35: Secondary Column Tails Concentration	x36: Feed Column Recycle Flow x37: Feed Column Tails Flow to Primary Column x38: Feed Column Calculated DP x39: Feed Column Steam Flow x40: Feed Column Tails Flow

The variables associated with the primary column are of higher importance than the variables associated with the other columns

Dow Dataset – Review from Last Time

Correlation Matrix:

- Only captures linear correlations (*not more complicated relationships like, cos or exp*)
- Nevertheless, we see that many of the variables exhibit strong positive or negative linear correlation (*maybe they aren't all important*).

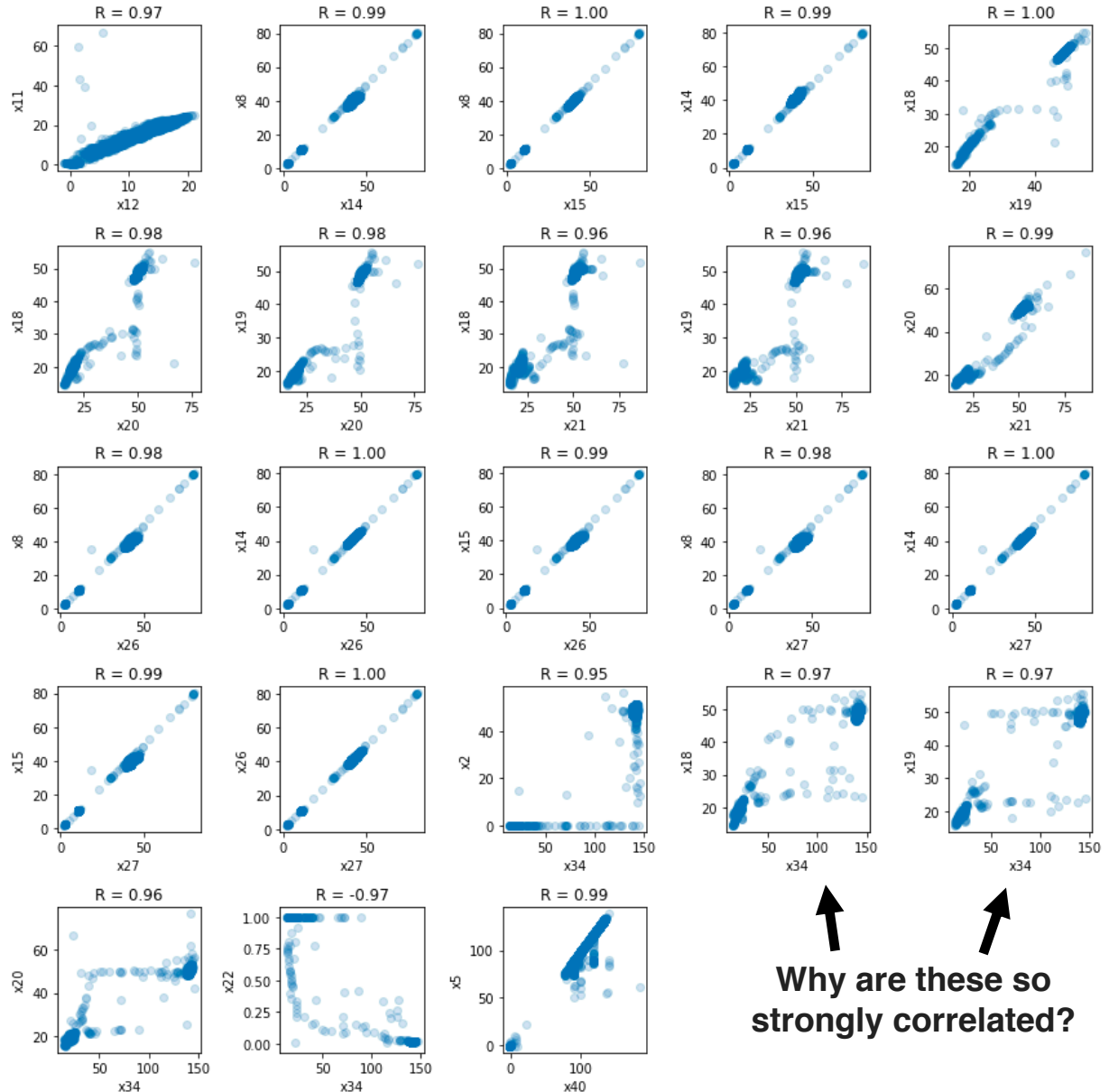


$$r_{xy} = \frac{\sum_i (x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_i (x_i - \mu_x)^2} \sqrt{\sum_i (y_i - \mu_y)^2}}$$

Dow Dataset – Review from Last Time

Parity Plots:

- We looked at the parity plots for all correlated features with $r \geq 0.95$
- We see evidence of “on” and “off” behavior of unit operations
- Correlations result from this two state behavior, rather than correlation during operation.



Outlier Basics

Outliers: samples that we consider unrepresentative or not a true member of the population we are investigating.

Classifying outliers is highly problem specific and there are no black-box methods that should be universally used.

Classifying outliers depends on:

- (1) your domain knowledge of what the data represents
- (2) what you want to predict from the data
- (3) your model of the data

Outlier Basics – Domain Knowledge

Domain knowledge plays an important role in understanding if something is an outlier.

- In the Dow example, we have sensor data on many variables, including things like pressure, concentration, and temperature, as a function of time.
- Our domain knowledge includes our physical understanding that pressure can change very quickly, while large temperature changes will be limited by the heat capacities and thermal conductivities of the components.
- We would consult this knowledge when considering whether a sudden large fluctuation in temperature or pressure were "real" or an "outlier" that should be removed from our dataset.

Outlier Basics – Prediction Objective

What we want to predict affects how we classify outliers

- Suppose we only want to predict seasonal trends. In that case, we might discard large fluctuations in certain variables that only occur over short time periods before trying to build our model.
- In this case, we would be classifying samples as outliers or replacing them, *not because they are unphysical*, but because they are unimportant with respect to our prediction target.
- The reference to our prediction objective is another reason why a one-size-fits-all approach does not exist.

Outlier Basics – Data Model

Outlier classification often relies on an implicit model of the data or population.

- For example, typical outlier classification procedures depend on calculating a z-score (i.e., a normalized magnitude of the deviation from normal/gaussian behavior) for each observation and removing values that are larger than some threshold.
- In such a procedure, *we are assuming* that our variable should obey normal/gaussian statistics.
- This assumption may be justified or just convenient. In either case, we would be classifying outliers based on our model of the data, not because we necessarily had evidence that the sample was unphysical or incorrect.

Outlier Basics – Data Model

Outlier classification often relies on an implicit model of the data or population.

• For example, typical outlier classification procedures

Today we'll compare two common ways of identifying outliers. The first will be applied to find outliers in our dependent variable/output feature (y), the second will be applied to find outliers with respect to our independent variables/input features (x).

Neither is a black-box and you will see that domain-knowledge is still required to apply them in practice.

either case, we would be classifying outliers based on our model of the data, not because we necessarily had evidence that the sample was unphysical or incorrect.

Z-Score

When assuming normal statistics, a typical approach for classifying outliers is based on whether a sample **deviates from the mean** beyond a given threshold.

The **z-score** of a sample is the deviation from the mean normalized by the standard deviation:

$$z_i = \frac{x_i - \mu_X}{\sigma_X}$$

Under the assumption that the data obeys normal statistics, the z-score can be related to the probability of observation using the cumulative normal distribution function.

Under the assumption that the data obeys normal statistics, 68% of samples would have a z-score between -1 and 1, 95% between -2 and 2, and 99.7% between -3 and 3. If you observe a sample with a z-score of 4 in a dataset of 100 values, you might reasonably conclude that this is an outlier and would want to remove it or replace it

Z-Score

When assuming normal statistics, a typical approach for classifying outliers is based on whether a sample **deviates from the mean** beyond a given threshold.

This sounds simple: we just need to calculate the z-scores, compare with a threshold, and then replace or remove the outliers.

$$z_i = \frac{x_i - \mu_X}{\sigma_X}$$

However, *if outliers are in the sample*, then they will affect the calculation of the mean and standard deviation. How can we calculate the mean from a sample that contains outliers?

Robust Z-Score

The motivation for the robust z-score, is to use an estimators for the mean and standard deviation that are relatively insensitive to outliers.

The fundamental observation is that the **median** of a sample is an **unbiased estimator** for the **mean**. Although the sample mean is typically the **most efficient estimator** of the population mean, the **median** is much less affected by outliers. Similarly for the standard deviation.

We can calculate a (robust) **z-score** based on the sample median instead of the mean as:

$$z_i = \frac{|x_i - X|}{k \cdot \text{MAD}} \quad \text{MAD} = \text{med}(|x - X|)$$

X is the **median** of the variable x, MAD is the **median absolute deviation from the median**, and the factor k is a scale factor that relates the MAD to the standard deviation ($k=1.4826$ in the case of a gaussian distribution, for other distributions a different scale factor would be used).

Robust Z-Score – Hampel Filter

When utilizing the robust z-score for time-series data, it makes the most physical sense to apply it to a running window of values.

For each sample, one calculates the robust z-score for the center of the window. If the robust z-score exceeds a threshold then the value is replaced by the median of the window. This strategy for removing outliers from time-series data is known as a **Hampel Filter**:

```
def hampel(x,w,n_sig=3.0,k=1.4826):
    # Check that the length is at least 2w
    assert len(x) > 2*w, "x isn't long enough to filter"
    # Copy array so that updates don't conflict with median calculation
    y = x.copy()
    # Calculate length and initialize lists
    N = len(y)
    z = [0.0]*N      # list for holding z-scores
    o = [False]*N    # list of holding True/False outlier spec
    for i in range(w,N-w):
        med = np.median(x[i-w:i+w+1])
        mad = k*np.median(np.abs(x[i-w:i+w+1]-med))
        z[i] = np.abs(x[i]-med)/mad
        if z[i] > n_sig:
            y[i] = med
            o[i] = True

    return y,z,o
```

Robust Z-Score – Hampel Filter

Let's apply the filter to the Dow dataset with a few values of threshold and see which values it identifies as outliers:

```
for i in np.arange(1.0,5.0):
    y,z,o = hampel(dow["y"].to_numpy(),w=100,n_sig=i)
    o_inds = [ count for count,_ in enumerate(o) if _ is True ]
    i_inds = [ count for count,_ in enumerate(o) if _ is False ]

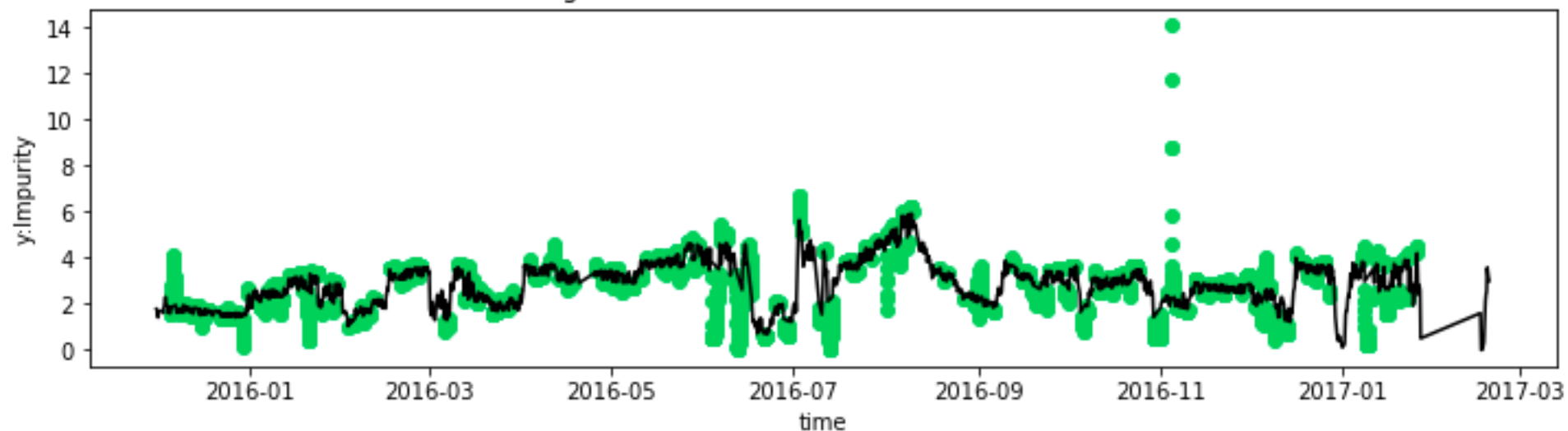
    # Plot original data with outliers marked
    plt.figure(figsize=(12,3))
    plt.plot(dow.iloc[o_inds,0],dow.iloc[o_inds,-3],marker='o',\
             color=(0.1,0.8,0.4),linestyle='None')
    plt.plot(dow.iloc[i_inds,0],dow.iloc[i_inds,-3],color="k")
    plt.xlabel("time")
    plt.ylabel("y:Impurity")
    plt.title("original with outliers ({:d}, {:<4.2f}%) marked".format(\
              sum(o),sum(o)/float(len(o))*100.0))

    # Plot original data with outliers replaced by median
    plt.figure(figsize=(12,3))
    plt.plot(dow.iloc[:,0],y,color="k")
    plt.xlabel("time")
    plt.ylabel("y:Impurity")
    plt.title("outliers ({:d}, {:<4.2f}%) replaced by median".format(\
              sum(o),sum(o)/float(len(o))*100.0))
    plt.show()
```

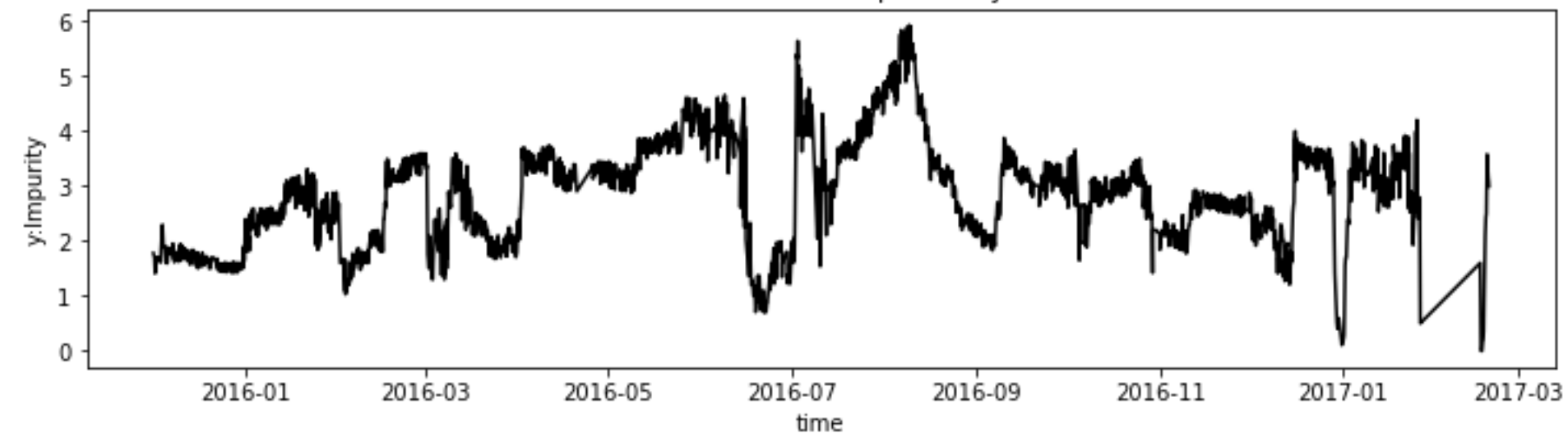
Robust Z-Score – Hampel Filter

$n_sig = 1:$

original with outliers (2157, 20.95%) marked



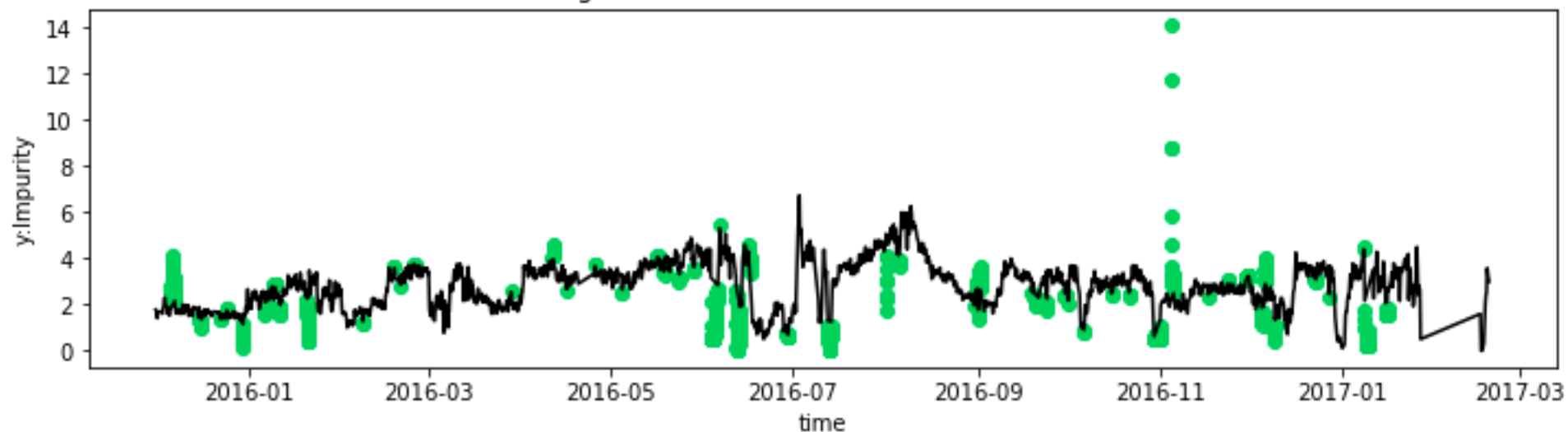
outliers (2157, 20.95%) replaced by median



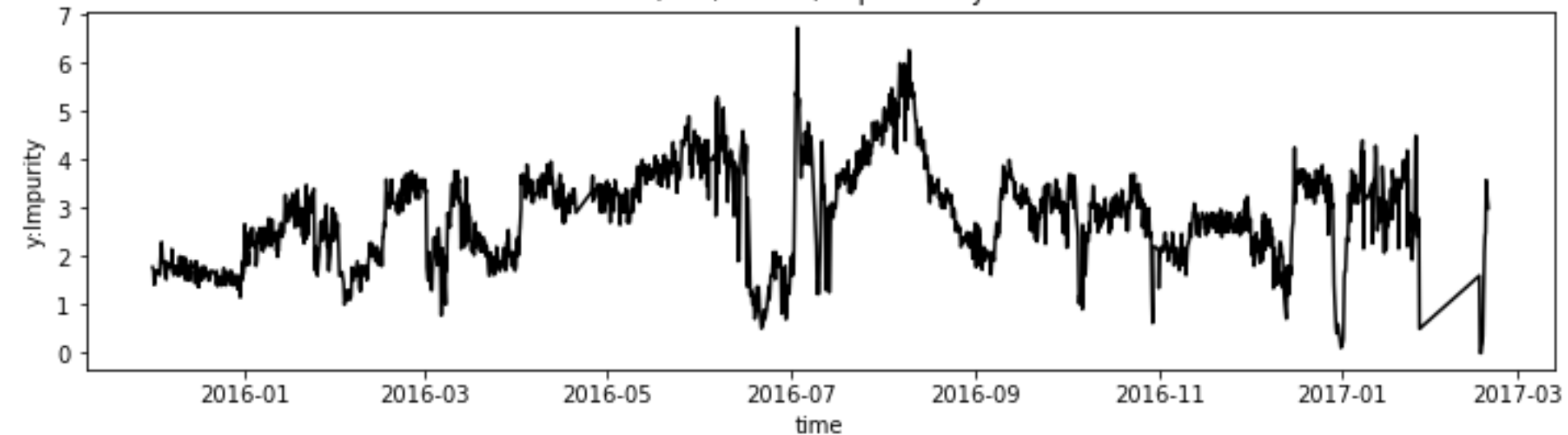
Robust Z-Score – Hampel Filter

$n_sig = 2:$

original with outliers (545, 5.29%) marked



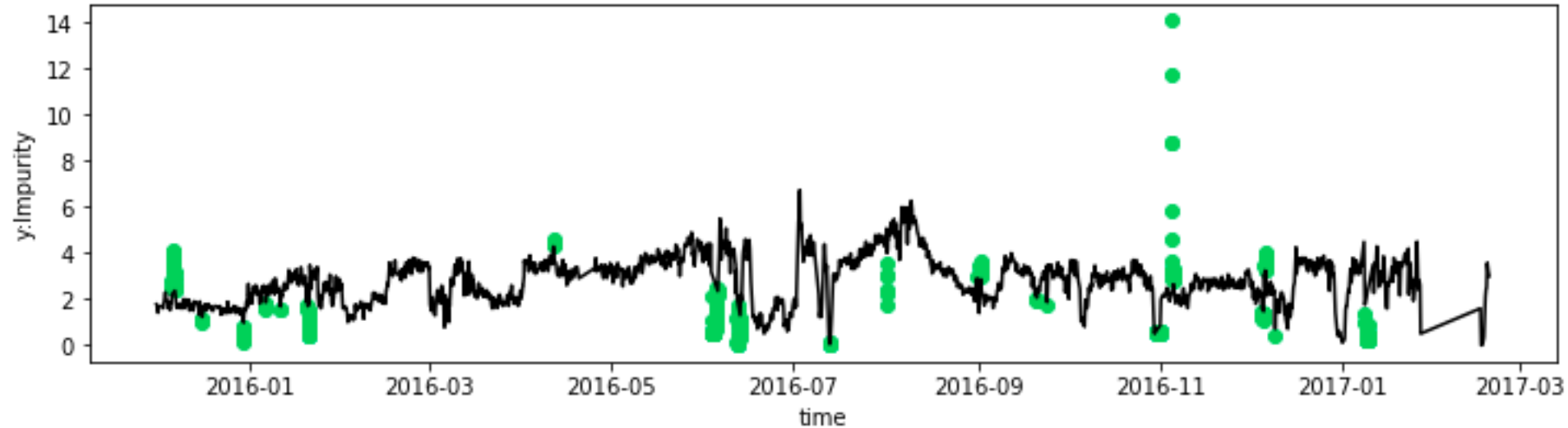
outliers (545, 5.29%) replaced by median



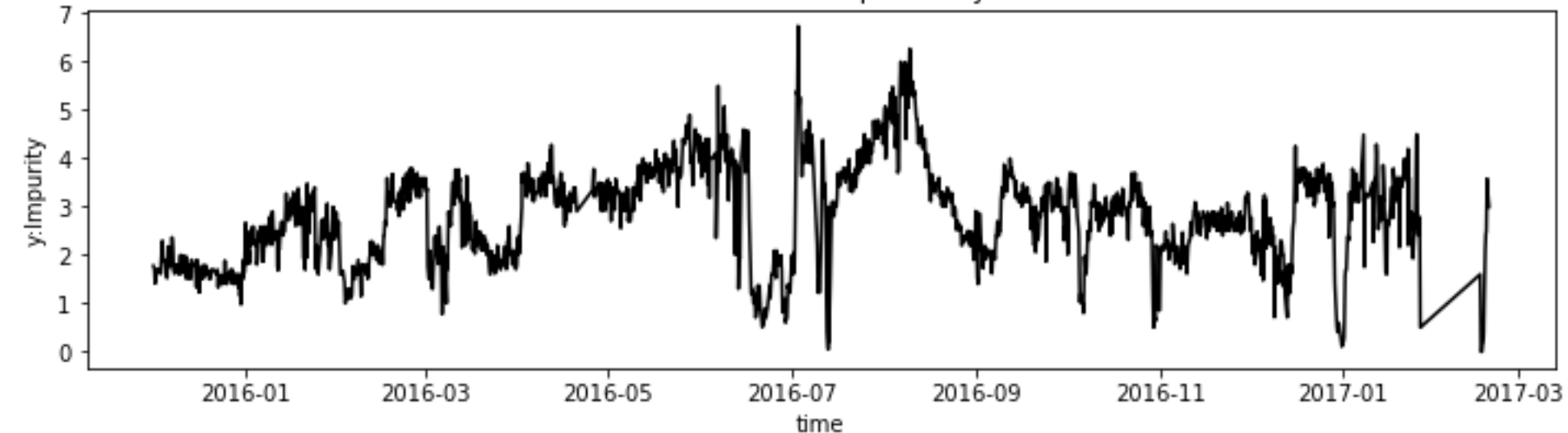
Robust Z-Score – Hampel Filter

$n_sig = 3$:

original with outliers (263, 2.55%) marked



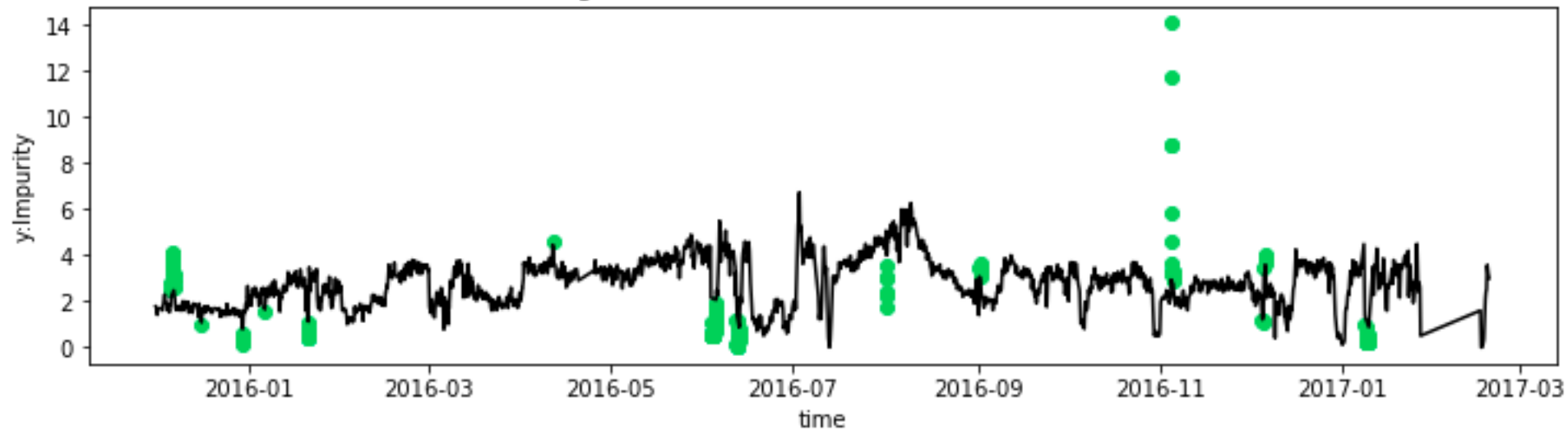
outliers (263, 2.55%) replaced by median



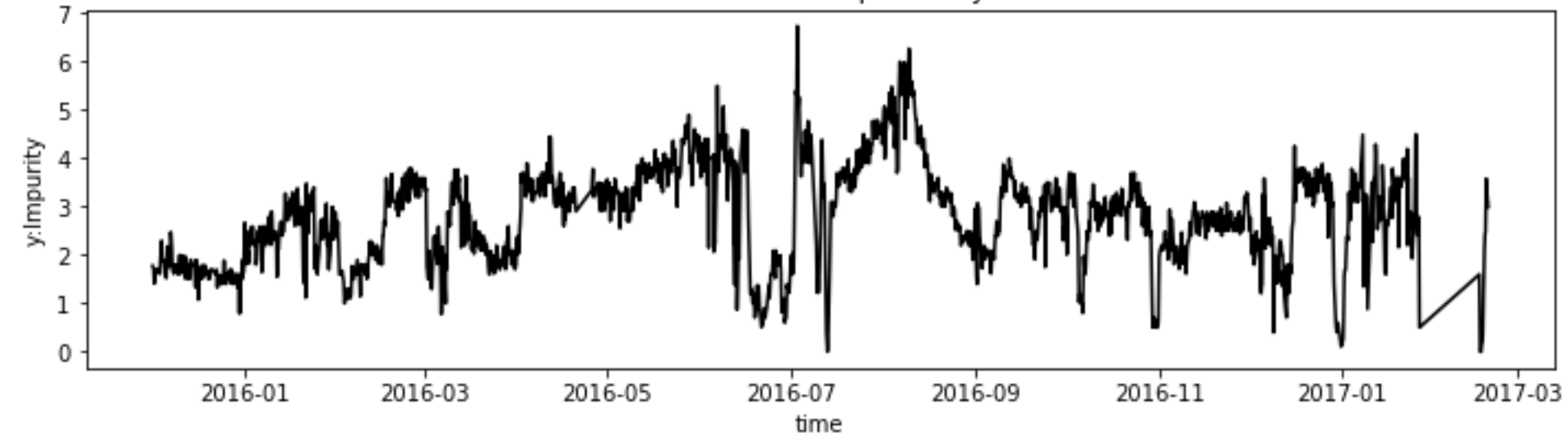
Robust Z-Score – Hampel Filter

$n_sig = 4$:

original with outliers (160, 1.55%) marked



outliers (160, 1.55%) replaced by median



Robust Z-Score – Hampel Filter

$n_sig = 4$:

original with outliers (160, 1.55%) marked

The Hampel filter successfully identifies large fluctuations and replaces them with the median.

As the n_sig parameter decreases, more data is removed. This behavior is intuitive and is often exactly what we want.

outliers (160, 1.55%) replaced by median

Classifying large but real fluctuations as outliers is a danger of the Hampel filter. It is possible that we could still build a predictive model, even without this data, but we might be leaving a critical source of information out.

Principal Component Analysis (PCA)

PCA is an important method for identifying "important" dimensions in a high dimensional dataset.

Specifically, PCA analysis performs a rotation of the data into a coordinate frame where the greatest variance occurs along the first dimension (PC1), the next greatest variance occurs along the second principle dimension (PC2), and so on.

Here we will use PCA to visualize our large dimensional Dow dataset along a new set of dimensions that capture the most variance in the data. Although, PCA isn't itself a method for identifying outliers, in a case like the Dow dataset where we expect the data to be contaminated by periods where units are off, we can potentially use PCA to identify where this occurs in the feature (" x : ") space.

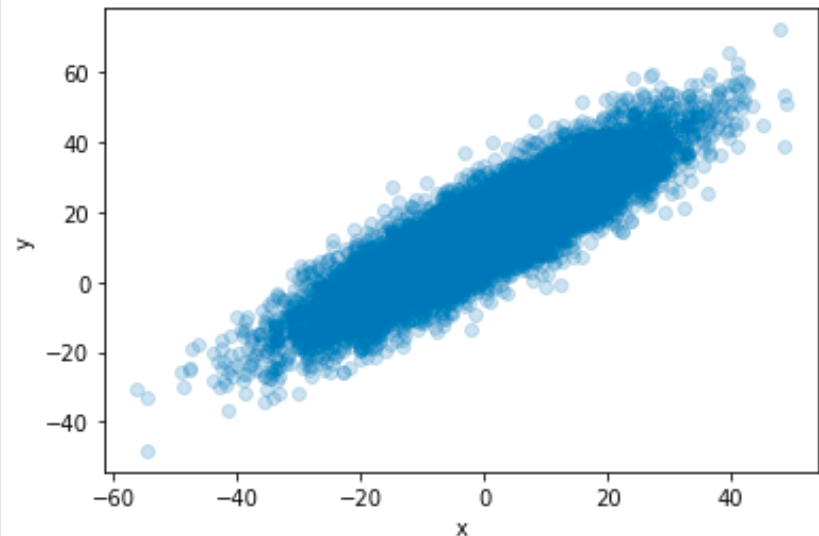
PCA - Example

Before getting into specifics, let's look at an illustration of what PCA does to a two-dimensional dataset based on gaussian distributed x and y values:

```
# Generate the data
np.random.seed(92425) # for reproducibility
x = np.random.normal(loc=10, scale=20, size=10000)
y = np.random.normal(loc=10, scale=5, size=10000)
sample = np.vstack([x, y]).T

# Rotate the data
rad = -np.pi/4.0
rot = np.array([[np.cos(rad), -np.sin(rad)], \
                 [np.sin(rad), np.cos(rad)]])
sample = np.dot(sample, rot)

# Make a scatterplot
plt.scatter(sample[:,0], sample[:,1], alpha=0.2)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



The resulting dataset is normally distributed along the two diagonals.

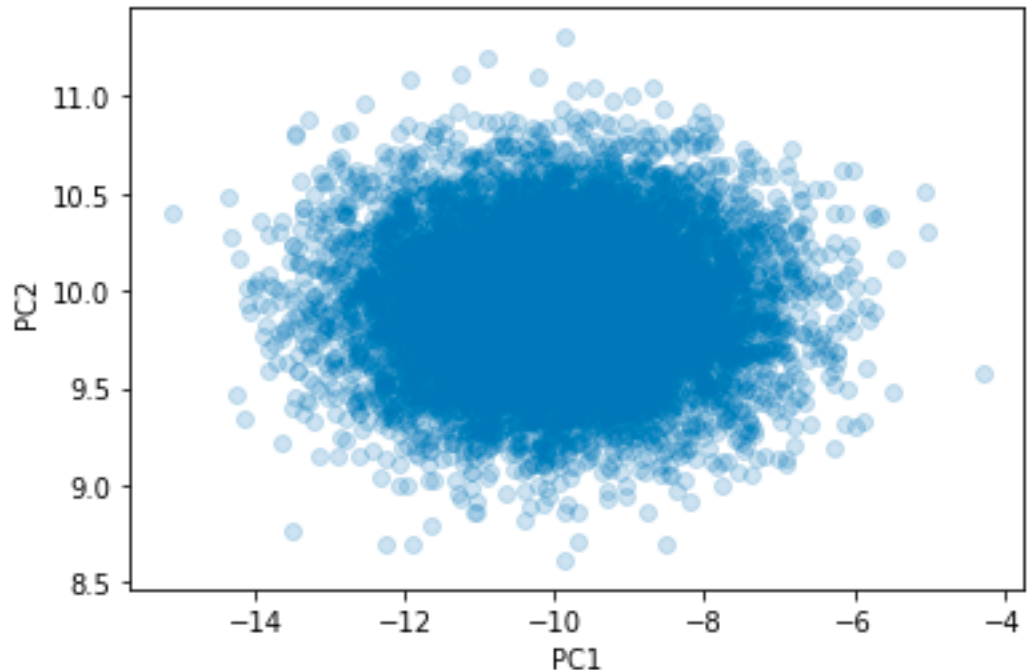
PCA finds the vector that captures the most variance and assigns it to first "principle component" (PC1), and so on. For the current dataset, we would expect PC1 to be the unit vector along (1,1) and PC2 to be the unit vector along (1,-1) (the sign is ambiguous, each can be multiplied by -1)

PCA - Example

Now let's plot the results of the transformed data and the principle components (I'll discuss the `pca` function next):

```
sample_pca, PCs, vars = pca(sample)
# Make a 3D scatterplot
plt.scatter(sample_pca[:,0], sample_pca[:,1], alpha=0.2)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

Here the `pca()` function that we've written returns the original data, but rotated to align with the principle components (`sample_pca`). We see that the largest variance lies along PC1 and the second largest variance lies along PC2.



Note: only relative variance is retained not absolute by default (see function)

PCA - Algorithm

I've told you *what* PCA does, but not *how* it does it.

In brief, consider a dataset with M samples and N variables (e.g., N sensor readings at M times). After centering the data about $\mathbf{0}$, the covariance matrix

$$\mathbf{C} = \frac{\mathbf{X}^T \mathbf{X}}{(N - 1)(M)}$$

holds the covariance* (i.e., unscaled correlation) of the variables i and j at the position C_{ij} , where C_{ii} is the variance along is specific direction i .

It can be shown that the eigenvectors of \mathbf{C} , sorted by their eigenvalue, are the **PCs** we are after. After the eigendecomposition of \mathbf{C}

$$\mathbf{C} = \mathbf{W} \mathbf{\Lambda} \mathbf{W}^T$$

the matrix \mathbf{W} ($N \times N$) holds the eigenvectors of \mathbf{C} along its columns and $\mathbf{\Lambda}$ ($N \times N$) is a matrix with the eigenvalues of \mathbf{C} along its diagonal and zeros elsewhere.

*Using the correlation is also common depending on whether you care about absolute vs relative magnitudes.

PCA –Implementation

Here is the PCA function that we used in the previous example:

```
# Our own PCA function
def pca(X, std=True):

    # remove mean from each column
    Xmean = X.mean(axis=0)
    TMP = X - Xmean

    # Normalize columns by standard deviation
    if std:
        TMP = TMP/TMP.std(axis=0)

    # Compute the covariance matrix
    C = np.dot(TMP.T, TMP) / ((len(TMP[0])-1)*len(TMP))

    # Perform the Eigen decomposition
    evals, evects = np.linalg.eig(C)
    ind = np.argsort(evals)[::-1] # sort descending
    evals = evals[ind] # sort evals
    evects = evects[:,ind] # sort evects

    # Project X onto PC space and add back the mean
    X_pca = np.dot(TMP, evects)
    X_pca = X_pca + np.dot(Xmean, evects)

    return X_pca, evects, evals
```

Besides the mean-centering and normalization, it is just an eigendecomposition and projection.

Note that we typically normalize the columns by their variance to control for differences in units.

PCA –Implementation

Why do we bother returning the eigenvalues? These are actually informative, since they hold the variance along each PC after the transformation:

```
sample_pca, PCs, vars = pca(sample)
print("PCs (along columns):\n{}\n".format(PCs))
print("evals:\n{}\n".format(vars))
print("variance:\n{}\n".format(sample_pca.std(axis=0)**(2.0)))
```

```
PCs (along columns):
[[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]

evals:
[1.88180043  0.11819957]

variance:
[1.88180043  0.11819957]
```

An alternative way of using the eigenvalues is to report each normalized by the sum of eigenvalues, which tells you what fraction of the total variance in the data is described by each PC:

```
tot = sum(vars)/100.0
for count_i, i in enumerate(vars):
    print("PC{:}: {: 6.2f} ({:4.2f}%)".format(count_i+1, i, i/tot))
```

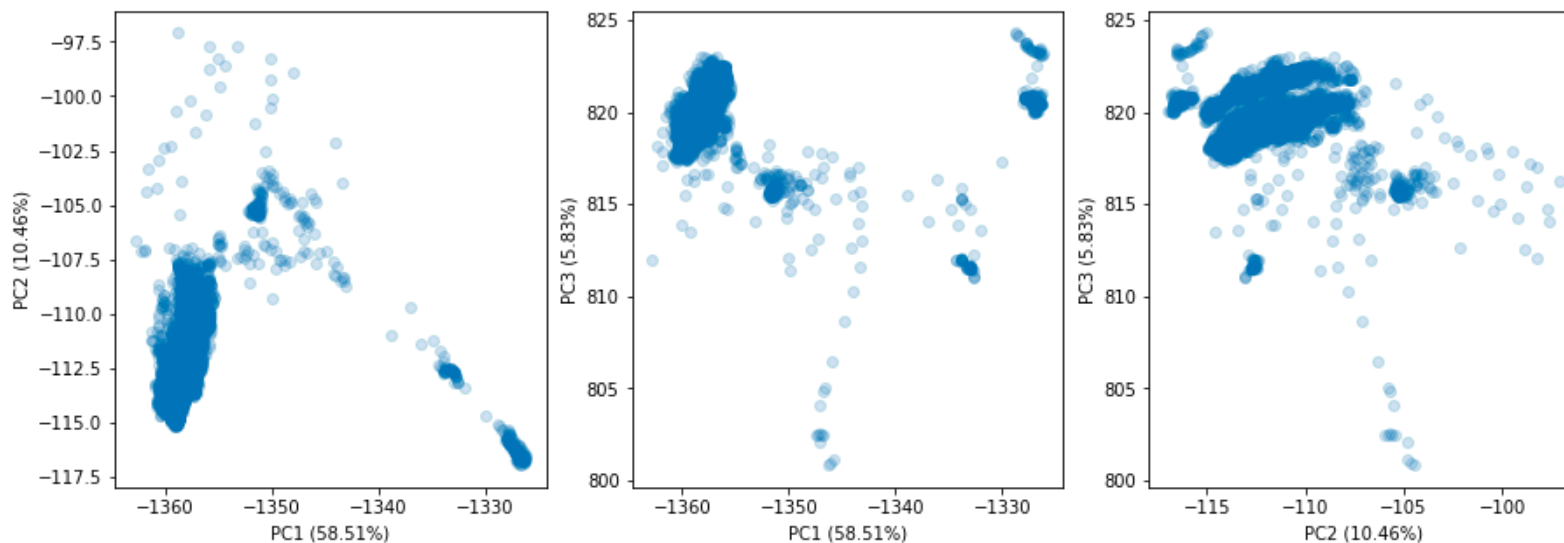
```
PC1: 1.88 (94.09%)
PC2: 0.12 (5.91%)
```

Dow Dataset – PCA

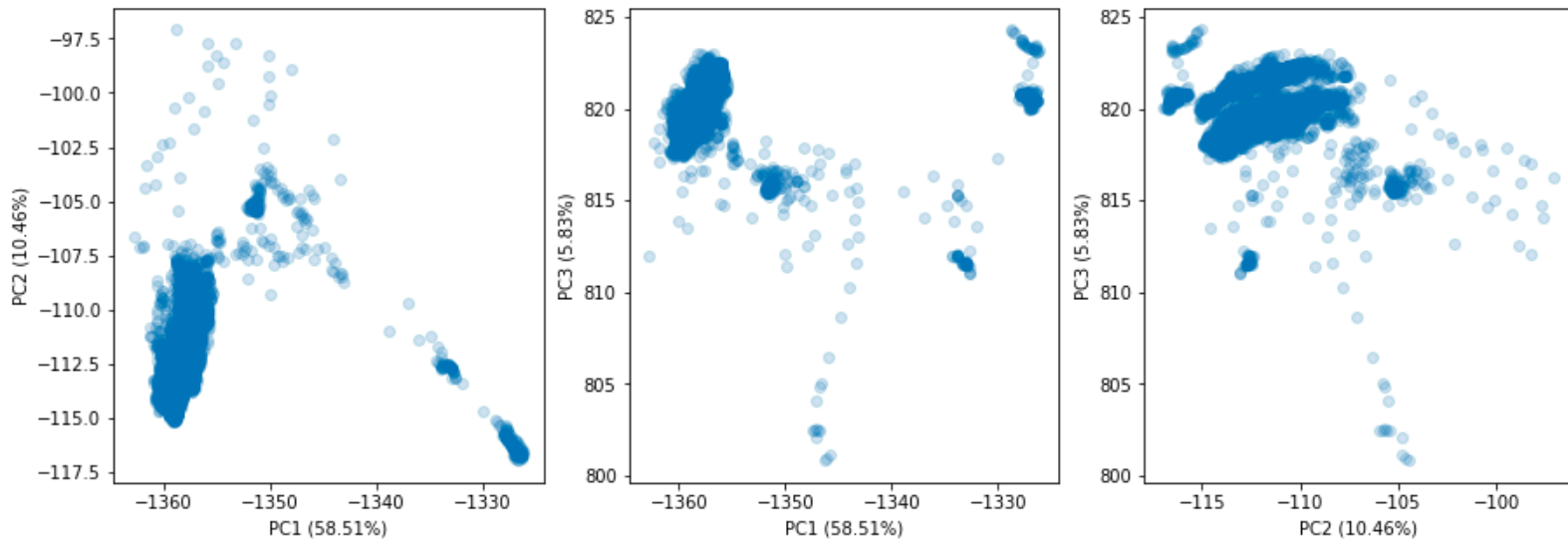
Now that we've gone over the basics of PCA let's use it to help analyze the Dow dataset. In particular, this is a relatively large dataset, with many descriptor variables (x :)

By transforming the data from a coordinate frame defined by the x : variables into a coordinate frame of PCs, we can focus on the dimensions that explain most of the variance in the data:

```
dow_x = dow.iloc[:,1:41].to_numpy() # grab the x-data
dow_x_pca, PCs, vars = pca(dow_x) # PCA analysis
### Make scatter plots (commands not shown) ###
```



Dow Dataset – PCA



We can see from the variances that PC1 alone captures a majority of the variability in the values. In combination, PC1-PC3 capture 75% of the variance.

Looking at these numbers, the PC analysis has automatically ascertained what we observed earlier in the correlation analysis: a lot of these variables are correlated and aren't providing much independent information. Using only 3 variables we can describe a supermajority of the variance.

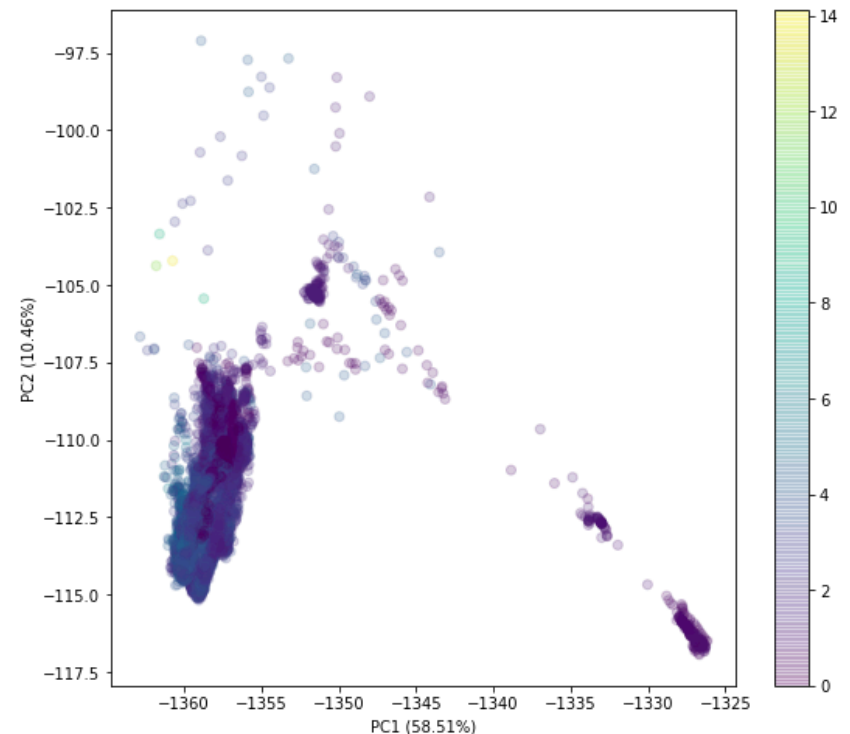
Dow Dataset – PCA

Let's look closer at the PC1 vs PC2 plot and color the data according to impurity:

```
plt.figure(figsize=(9,8))
plt.scatter(dow_x_pca[:,0],dow_x_pca[:,1],alpha=0.2,c=dow["y"],cmap="viridis")
plt.xlabel("PC1 ({:<4.2f}%)".format(vars[0]/sum(vars)*100.0))
plt.ylabel("PC2 ({:<4.2f}%)".format(vars[1]/sum(vars)*100.0))
plt.colorbar()
plt.show()
```

Looking at this figure, we can see a clear separation between values when the plant is operating with $PC1 < -1353$ and $PC2 < -106$.

Along the diagonal there are relatively fewer points.

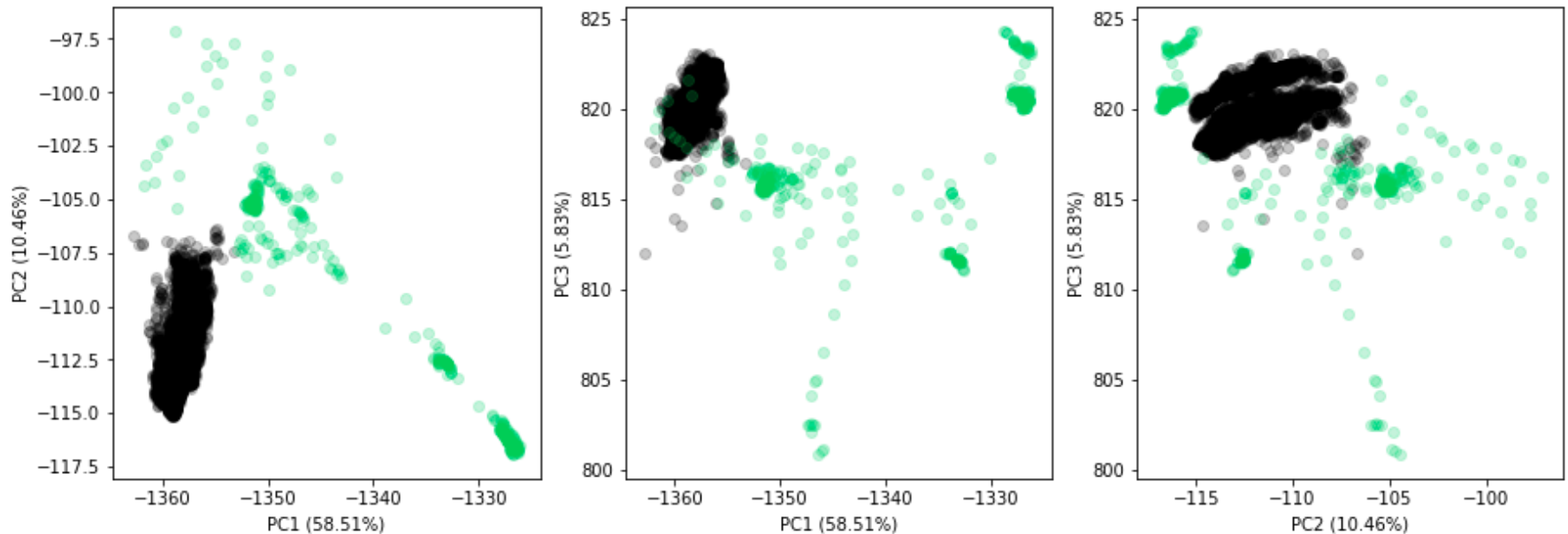


Dow Dataset – PCA

We can experiment with using the PC values in this way for identifying outliers:

```
i_inds = [ _ for _ in range(len(dow_x_pca)) if dow_x_pca[_ ,0] < 1353 \
          and dow_x_pca[_ ,1] < -106 ]
o_inds = [ _ for _ in range(len(dow_x_pca)) if _ not in i_inds ]
print("N outliers: {} ({:<4.2f}%)".format(len(o_inds), len(o_inds)/len(dow_x_pca)*100.0))
```

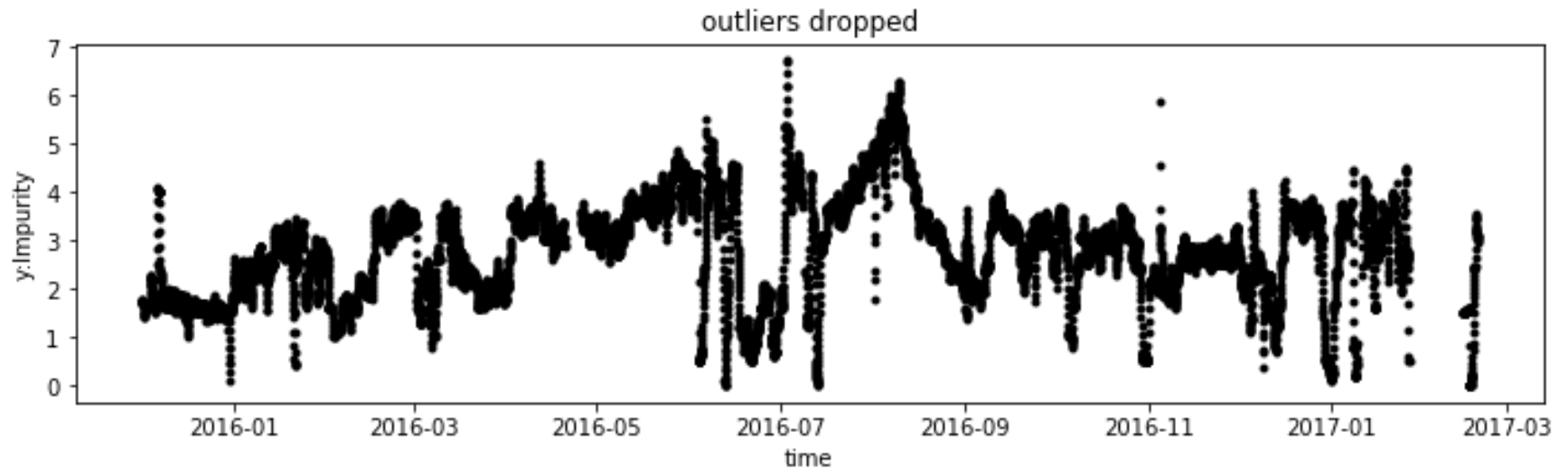
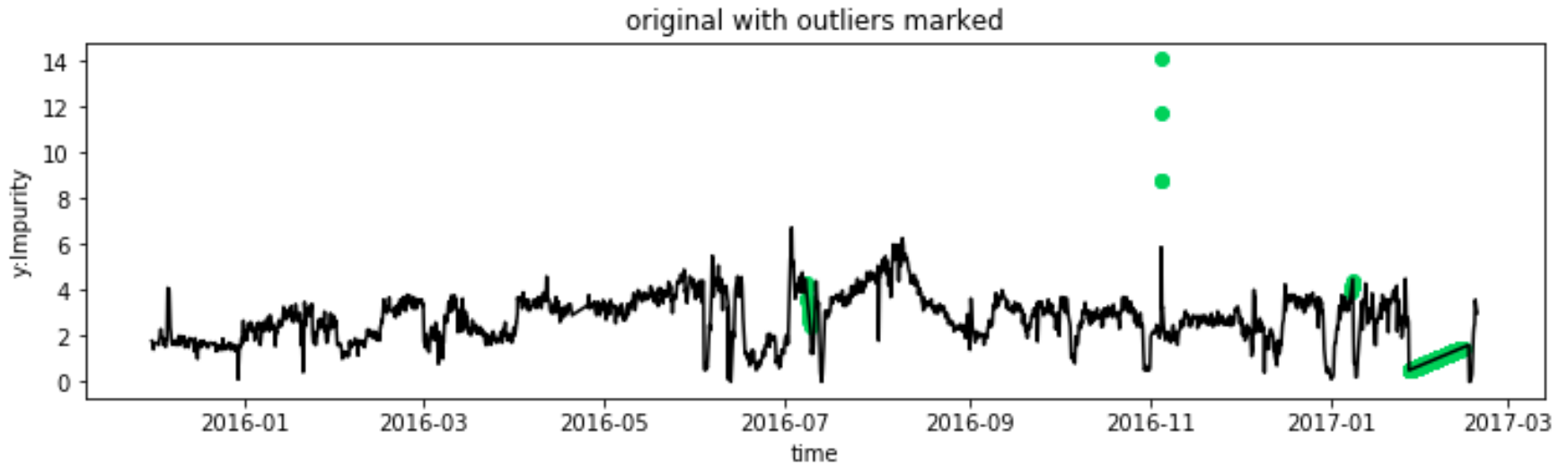
N outliers: 450 (4.37%)



The diagonal in PC1/PC2 only consists of 4.37% of the points in our dataset.

Dow Dataset – PCA

Look at where these outliers occur in the time vs impurity series:

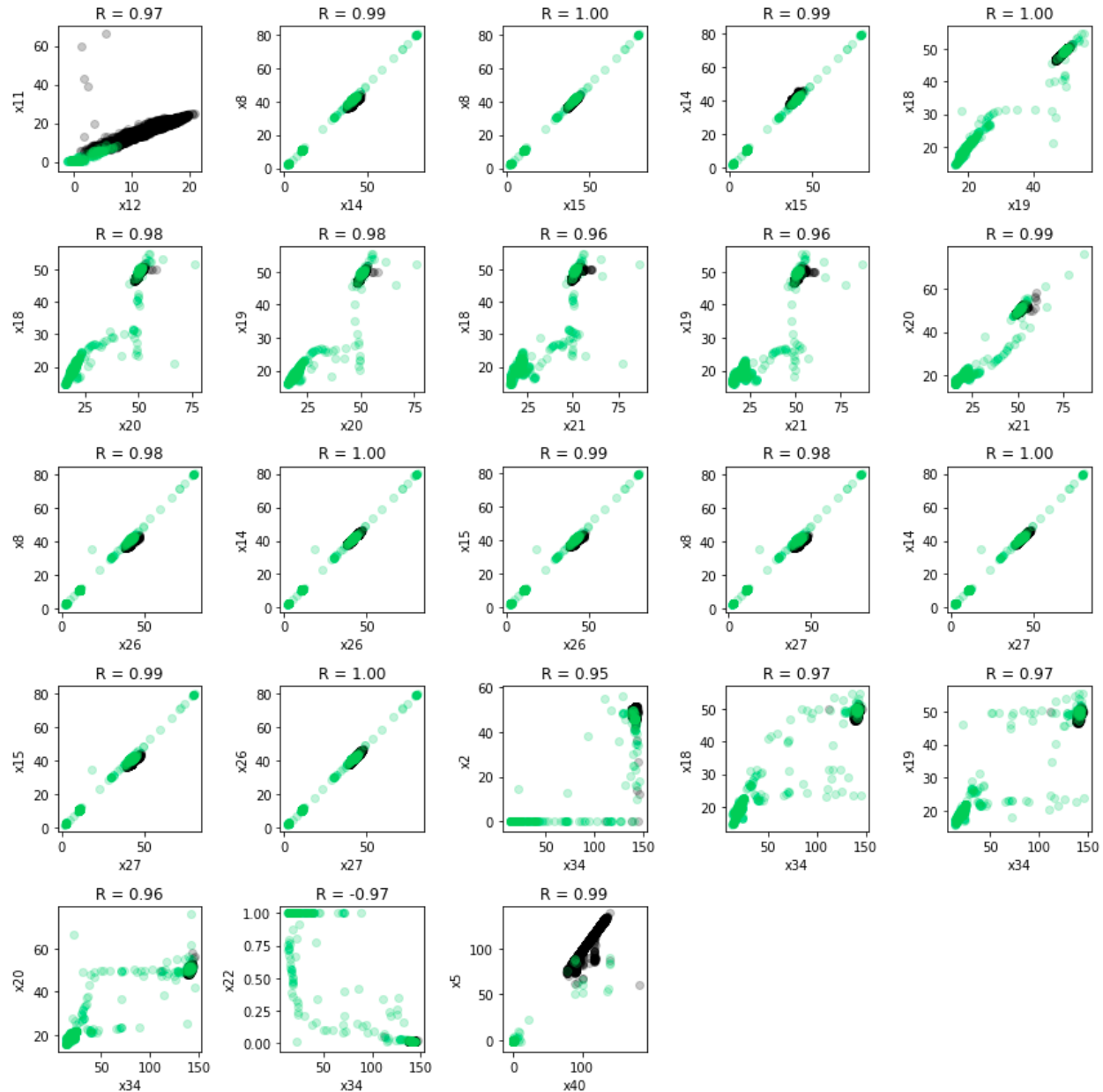


Dow Dataset – PCA

PCA observed the same thing we did, the plant seems to exhibit two states, "on" and "off".

Looking at the temperature variables, x_{34} , x_{18} , and x_{19} , we see that the low temperature regions have been identified as outliers (green).

Only 450 samples (4% of the total) are identified as outliers using our PC1 and PC2 criteria. Thus, the overwhelming majority of the data is bunched in the tiny black clusters shown in the correlation plots.



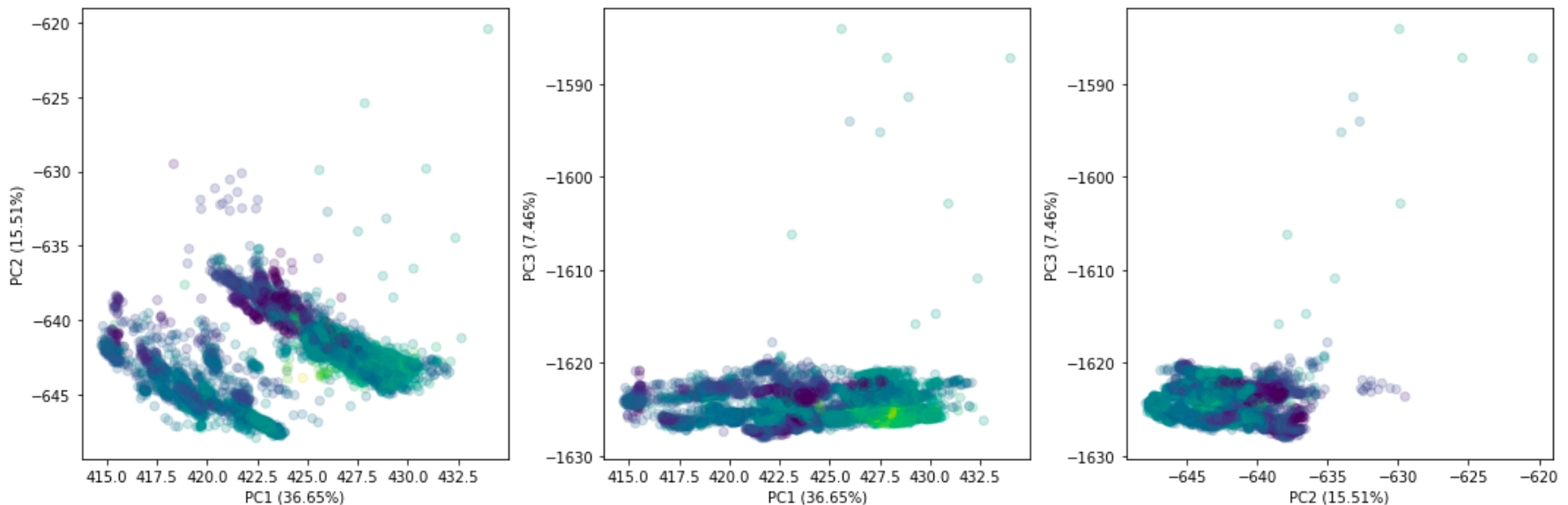
Dow Dataset – PCA Again

Now that we've identified outliers (based on **PCA and our domain knowledge** about what is occurring) we can remove the outliers and perform PCA again to find important variables and/or identify further anomalies.

```
# Create a clean dataframe
dow_clean = dow.iloc[i_inds,:]

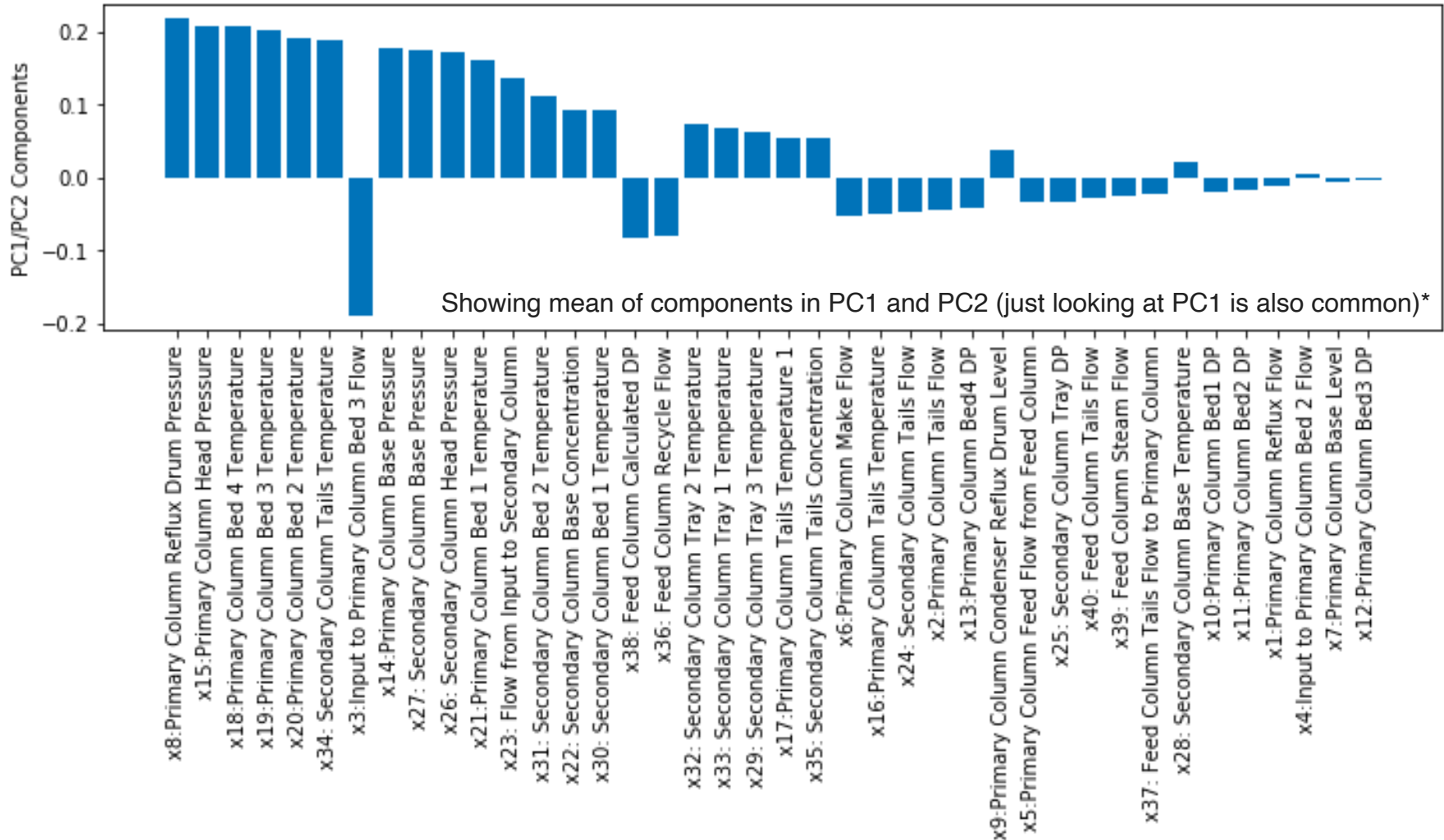
# Create a clean x-only dataframe
dow_x_clean = dow_clean.iloc[:,1:41].to_numpy() # grab the x-data

# Perform PCA
dow_x_clean_pca, PCs, vars = pca(dow_x_clean) # PCA analysis
```

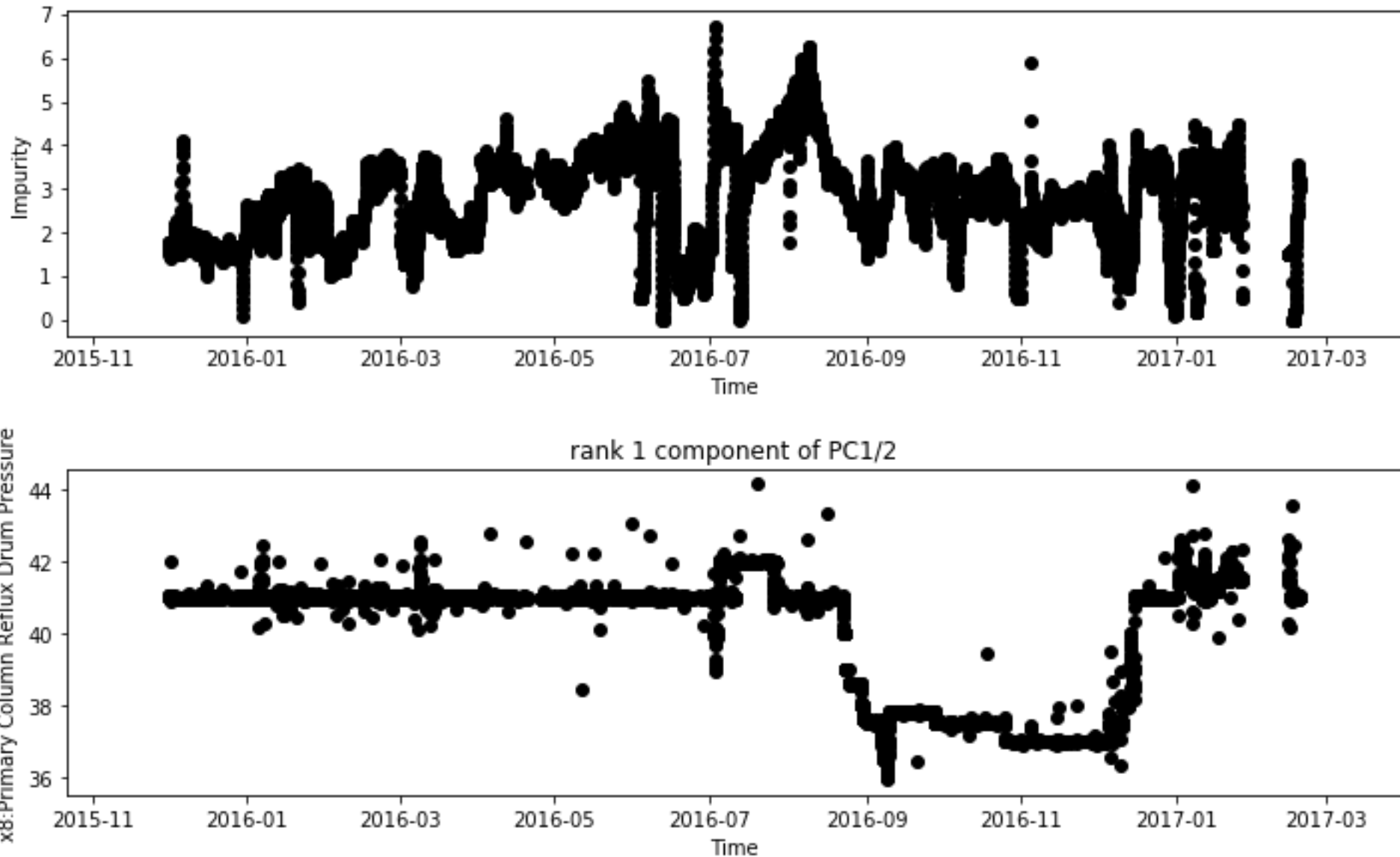


Dow Dataset – PC Component Plot

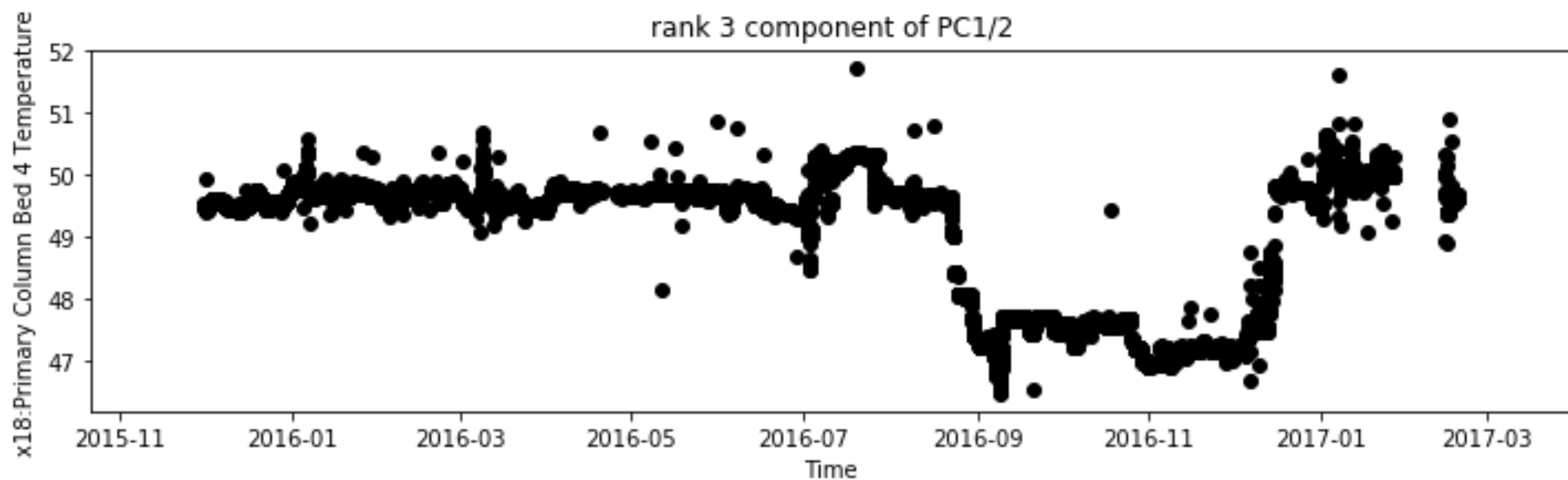
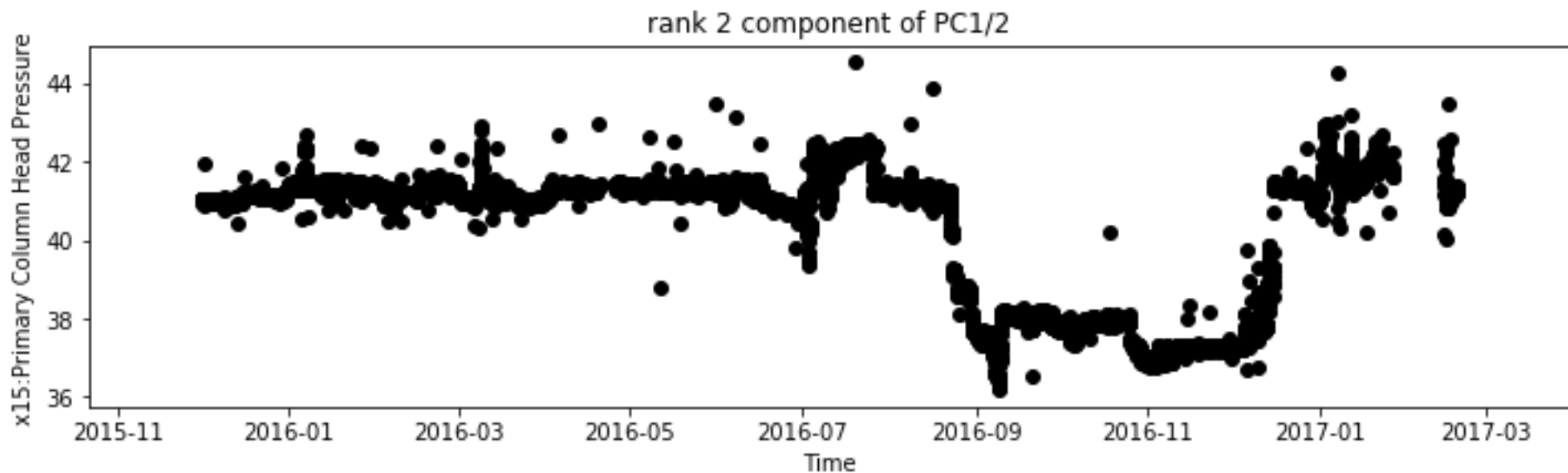
The PCs are the “directions” with greatest variance in feature-space, correspondingly the largest components of the PCs are potentially important explanatory variables.



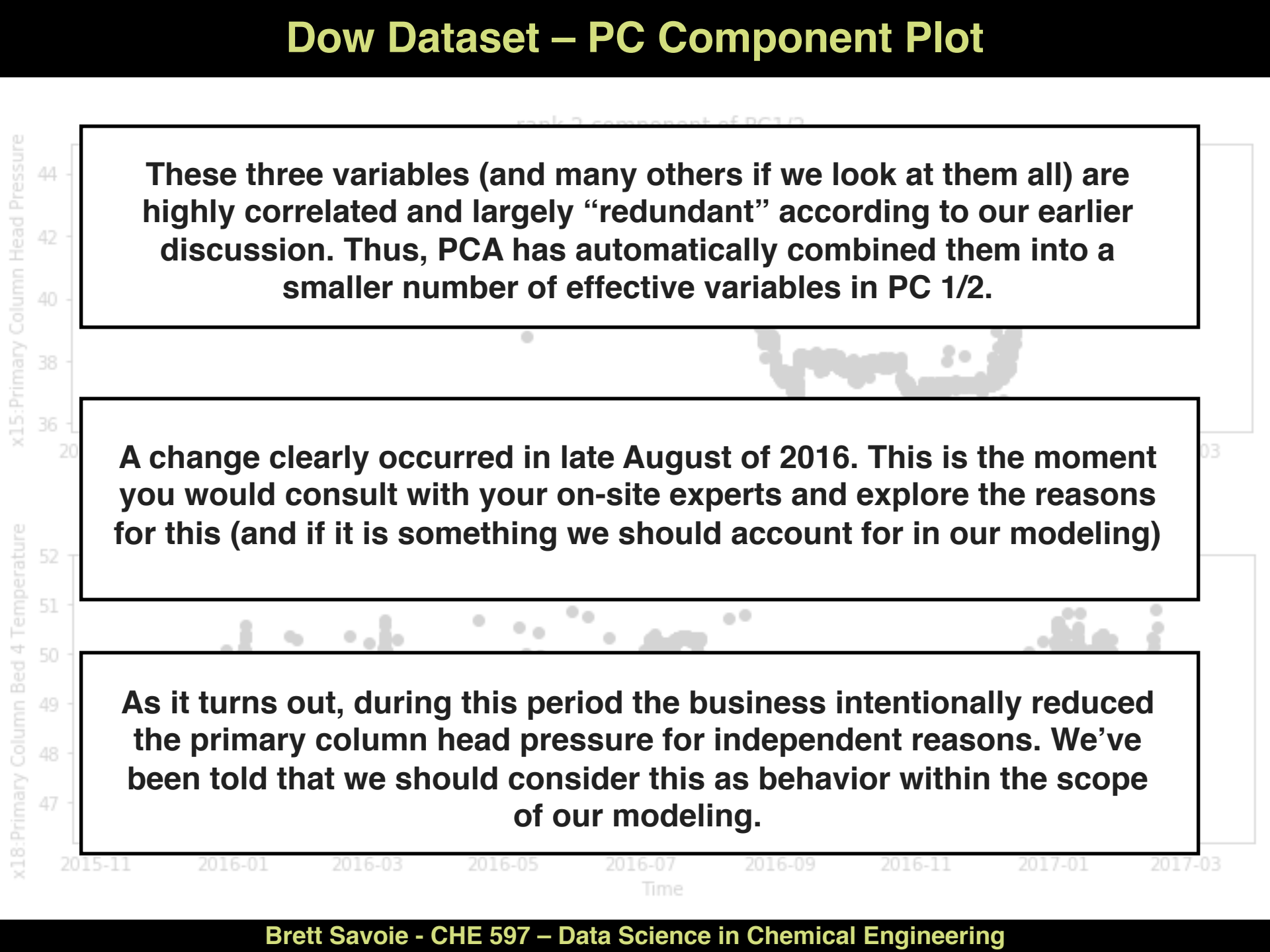
Dow Dataset – PC Component Plot



Dow Dataset – PC Component Plot



Dow Dataset – PC Component Plot



A scatter plot showing the first two principal components (PC1 and PC2) of the Dow dataset. The x-axis is labeled 'rank 1 component of PC1' and the y-axis is labeled 'rank 2 component of PC2'. The plot shows a dense cluster of points with a distinct change in the distribution around late August 2016. Three text boxes are overlaid on the plot, providing context for the data.

These three variables (and many others if we look at them all) are highly correlated and largely “redundant” according to our earlier discussion. Thus, PCA has automatically combined them into a smaller number of effective variables in PC 1/2.

A change clearly occurred in late August of 2016. This is the moment you would consult with your on-site experts and explore the reasons for this (and if it is something we should account for in our modeling)

As it turns out, during this period the business intentionally reduced the primary column head pressure for independent reasons. We’ve been told that we should consider this as behavior within the scope of our modeling.

Dow Dataset – PC Component Plot

What do the correlation plots look like now that we've removed the outliers?

We don't see any of the “two-state” behavior that previously led to spurious correlations.

We also see a lot of strongly correlated variables.

