README: Please install statsmodels, itertools, seaborn, collections using pip. We also have a fully labelled training data 2 and test dataset which we have attached to our submission. All that is needed is to type in the correct paths indicated by the comments in the code under "Reading in the data".

**- `BATADAL\_dataset03.csv` as the training data**

**- `BATADAL\_dataset04.csv` as the validation data**

**- `BATADAL\_test\_dataset.csv` as the test data (labels derived from the attacks listed on the website)**

We observe that we have 4 main types of signals in the dataset, namely

\* Water Levels associated with the water tankers denoted by "L\_TankID". (7 variables)

\* Inlet and outlet pressure associated with the actuated valves and pumping stations denoted as "P\_JunctionID". (12 variables).

\* Flow through the actuated valves and pumps denoted using "F\_actuatorID" (12 Variables)

\* Status of the actuated valves and pumps denoted by "S\_actuatorID". (12 Variables)

1) Firstly based on the above visualization which shows the 4 different signals discussed in the previous question, we do see signs of correlation. More specifically, we see that the flow of water through pump 2 is positively correlated with the status. Moreover, we can also start to see similar temporal patterns for the water level at tank 2 and the pressure at junction 289. ( More generally, we find that the flow and status of actuated valves and pumps are always positively correlated. )

2) We can also spot cyclic behaviour for some of the signals. If we look at the box plots made for the different signals with respect to their hourly values, we can clearly see a cyclic pattern especially for water levels for tank 3 (L\_T3).

Note: Here we use a sliding window approach to predict the next value of a series. The dataset is created by choosing a signal i.e `L\_T1` in our case and using a sliding window of 23 timesteps. Therefore 23 values preceeding the 24th value of the series is considered to be the features with the 24th value being the associated label for our dataset. We then train a random forest regressor model on the first training dataset and use a randomized grid search to find the best paramaters for the model based on the second training dataset. We finally generate predictions using the best parameters we found through the validation set for the test set. We conclude with our findings. Therefore based on the plots and the mean squared error which was quite low as well, we can say predicting the next value in the water levels for tank 1 i.e L\_T1 using a sliding window of 23 time steps is relatively straightforward.

Computing and the avg lof\_scores for different number of nearest-neighbours.

The Iterative code takes a while to run. We experiment with several values of k (num of neighbours considered) but we decide to use `k = 100` as nearest neighbours by looking at the true positives generated by the values as presented in the plots below.

NOTE: We follow the notion of not having the test data at train time and hence ONLY use the Validation data to tune the nearest neighbour value which is optimal at `k = 100`. In our experimentation we saw that a value of `k = 2,10,50` give higher true-positives for the test data.

Visualization: LOF: We can use the scatter-plot below to visualize the outlier scores intuitively based on the local density of surrounding points (neighbours). We plot samples using two signals (2-d representation using L\_T1 and L\_T2) and identify points with the `LOF scores > 1.5`. Through this plot we can visualize how the LOF\_score allows us to segregate the anomalous instances based on local sample density. The encircled points are identified as outliers by our LOF implementation; points with a high LOF\_score (more likely outliers) will have a smaller radius than the point with a lower LOF score (likely to be inliers). We can hence see that the Local Outlier Factor assigns scores to identify anomalous \*\*point-instances\*\* based on the \*\*local density\*\* of samples using `k` neighbours.

An account of Abnormality:

The `dataset03.csv` (training data) doesn't actually capture any anomalous behaviour (all values of `ATT\_FLAG = 0`). Considering `dataset04.csv`, we see that there are 219 anomalous samples out of 4177 total instances. It is hence clear that that bulk of the data form a SCADA system is going to be normal i.e. \_non-anomalous\_. This anomalous behaviour can be observed due to attacks or due to malfunctioning devices and is hard to model for due to the the imbalance of sample-size in terms of anomalous/non-anomalous data.

To understand the concept behind using LOF scores as a method for anomaly detection, we first look at its components which have been implemented as helper functions:

- k-distance: Given a point, the k-distance is the distance of given point to its kth neighbour.

- Reachability distance: The reachability distance is the maximum out of the distance between the two given points and the k-distance of the second point. It acts as a “smoothing factor” in a sense that if point a is within the k nearest-neighbors of point b, the reach\_dist(a,b) will be the k-distance of b. Otherwise, it is the actual distance between a and b.

- Local Reachability Density (LRD): the LRD is the inverse of the avg. reachability distance of a point 'a' with its k neighbours. By intuition it tells us how far we have to travel from 'a' to reach next point or cluster of points.

Finally, the LOF is a score (for each data point) that computes the ratio of the average densities of the point's neighbors to the density around the point itself. It is the average ratio of the LRD of a point with that of its neighbours.

Describe the kind of anomalies captured using LOF: Point Anomalies

The LOF score is roughly indicative of the ratio between the density around a point and the density of its neighbours. It thus allows us to find anomalies based on the notion that the density (of data samples) around an outlier object will be different (significantly) than that of the density around its neighbours (which should be the inliers). We are hence able to identify point-instances that occur in regions of similar density with an \*\*LOF score of 1 (or less than 1) that corresponds to the point having a similar density as its neighbours (or higher density) and segregate those point-instances that have substantially lower density than its neighbours \*\*(LOF scores > 1 which are considered as outliers).

Note- The steps performed to detect anomalies using the PCA approach are described here :-

1) I remove irrelevant columns created previously for visualization purposes and mean centre the data before applying PCA.

2) I apply PCA and visualize the explained variance ratio of the principle components.

3) I then plot projections of the first training dataset i.e Training Data 1 on the different components.

4) Based on the paper titled "Diagnosing Network-Wide Traffic Anomalies", I similarly use a heuristic which is used to select the optimal number of components to be assigned to the normal and anomolous subspaces for training data1.

5) I then project the data points of training data 1 on the normal and anomolous subspaces.

6) I plot the squared norms or the SPE of the data points projected on the anomolous as well as the squared norms of data projected on the full space (anomalous+normal).

7) In order to calculate the threshold to identify anomalies, I use the same Q-statistic which was used in the paper titled "Diagnosing Network-Wide Traffic Anomalies."

8) As mentioned in the PCA assignment, I remove the abnormalities for training data 1 as we wish to extract normal behaviour and repeat the steps 1-7 but now i transform training data 2 test data and similarly detect anomalies.

9) I present a confusion matrix for my results and conclude with the answers.

This graph shows us that a few principle components capture the most amount of variance in the data meaning that the data can be effectively encapsulated in a lower dimensional space.

Based on these plots we can see that there is a clear difference between the normal and anomolous projections. The data represented through the 1st and 2nd principle components show more deterministic and cyclic behaviour as compared to the 10th and 11th components which are largely homogenous but contain anomalous spikes.

Similar to the visualisations made previously, we see here that the squared prediction error (SPE) or the squared norms of the data points projected on the anomolous space contains suspicious spikes which are not present in the squared norm of the data points projected on the full space(normal+anomalous).

3b-I) Why do we see anomolies in the residuals for training data 1?

This method works by capturing the significant temporal variations of our dataset in a lower dimension through the principal component analysis. Based on our plots, we find that the first few principle components capture the majority of the temporal variance and are an effective means of representing the normal subspace whereas the remaining principal components are useful in studying anomolous behaviour. This is because the bulk of the data which is normal will be mainly represented by the normal subspace however it will not be able to adequately account for the variance caused by the few anomalous data points and this is what is captured through the anomalous subspace. And this is why we can see these suspicious spikes in the residuals or the squared norms of the data points projected in the anomalous subspace.

Note- As the question asks to investigate each individual sensor, I shall do the ARMA analysis for the following signals:

1) L\_T1

2) F\_PU7

3) P\_J289

4) S\_PU2

\* Incase the residuals gives a dataframe error, please re-run the code from under 4a and it should fix the error, don't know why it happens sometimes\*

On the basis of these partial and full autocorrelation plots, we can select the correct orders for the AR and MA parts of the ARMA model respectively. From the partial autocorrelation plots, we see that 1 is the correct order for the models for all signals (ignoring lag values after 10 as it would correspond to too many variables in the model) except for F\_PU7 for which the order seems to be 0. From the autocorrelation plots, we see that the order is 2 for signals S\_PU2 & L\_T1 and the order is 1 for P\_J289. For F\_PU7 the order from the autocorrelation plot also seems to zero.

### NOTE:

Running this code takes a while due to forecast being quite inaccurate after 24 timesteps. I've optimized the time as much as possible but it will take 15 minutes roughly. Here I actually train on training data 1 and fit on training data 2 and test set to predict anomalies using the L\_T3 signal only(doing it for more would take infinite time...). This is just an extra effort to understand ARMA by undergoing the full motions, if you wish to run it you can.

### 4c. I) What kind of anomalies can you detect using ARMA models?

Based on the ARMA models we can detect contextual anomalies. This is because ARMA models regress on previous values and make next step predictions based on them. Therefore if a high residual (large difference between ARMA's predicted value and current actual value) is found, it means that the particular data point is anamolous in context to the previous values that came before it as it differs from the predictions made by the ARMA model which represent the normal behaviour of the system given those same previous data points.

4c. II) Which sensors can be modeled effectively using ARMA?

Based on my results, the sensors that correspond to flow and status were not effectivel modelled by ARMA and had the worst overall performance. The sensors that correspond to pressure and water level were more amenable to this approach i.e L\_T1 and P\_J289 as they displayed cyclic behaviour and were largely continous throughout. However status and flow sensor data which can be constant or oscillate between discrete values are not modelled well by regression. This can be confirmed by looking at the residuals for F\_PU7 which were pretty bad. I also found through my analysis that the very cyclic signal L\_T3 was predicted quite well. However I couldn't yet find alot of anomalies even with a good fit with ARMA. However the L\_T1 signal provided alot of anomolies in the residuals. I suspect applying arma to all the signals and combining the anomalies they find will maximise performance further. \

Step1: Implement discretization of the signal values: We discretize the signals by grouping them together based on their values as per the percentile levels. We do this by splitting the signals into 10 levels based on quantiles calculated for each column.

The implementation of the N-Grams detection is as follows (Step 2):

- Discretize the data using percentiles (10 percentiles lead divides the data with equal samples)

- Generate ngram map from the validation data and create table by calculating n-gram frequencies. window\_size = 10, ngram\_length = 3 based on experiments.

- Apply similar logic to generate ngram table for the test data.

- Compute cosine similarity between the two dataframes and predict labels (for the test set) based on the label of the closest window in the validation set.

(Step 3) N-gram Detection using Cosine similarity

- Compare the N-Gram frequency tables from test and train set using cosine similarity

- Assign labels to the test windows based on the label of the most similar window (instance) in the train set

- Predict the labels for all windows in the test set.

- Finally, unpack the window labels to assign values to every point of the said window.

5b. Analysis and answers to the questions. Also provide relevant plots.

Analysing the N-grams: The anomaly detection approach using N-grams is a \*\*signature-based\*\* approach. Elaborating further, in `N-grams` we generate signatures (n-gram sequences for example) of benign (or non-anomalous) and anomalous collections from given input signals. It is a supervised detection method in a sense that it learns to classify n-gram patterns (signatures) as benign/anomalous based on labels that identify given signal sequence as such. Based on our experiment, we do see that n-grams approach is able to classify unseen (predict) sequences as well (we predict the test data labels and present the accuracy score)

Types of Anomalies: The N-grams approach, by the way of using \_'n'\_ related data instances as \_'signatures'\_ for a given profile (or a window of time) effectively identifies \*\*collective anomalies\*\*. In our implementation, by comparing the cosine distances we assign predictions to a collection of points (classifying an entire window as anomalous/non-anomalous) where the point instances may or may not be anomalous themselves but get classified as such by the way of being a part of an anomalous/non-anomalous collection. Additionally, we believe that it also considers a notion of context in a way that an n-gram sequence is anomalous or non-anomalous within the given context using which it is modelled i.e. it picks up this contextual information through the `n-gram` sequences of the data that it is trained

Comment on Modelling of Signals: Based on our results, the sensors that are best modelled using the this approach are the ones that display \*\*cyclic behaviour\*\* with peculiar signature sequences which can be modelled and eventually classified using n-grams. The `FLOW` and `STATUS` sensors which are either ON or OFF throughout (eg: `'F\_PU5', 'S\_PU5', 'F\_PU9', 'S\_PU9'`) were not effectively modelled and additionally had no effect on the overall performance i.e. the model performance remains unchanged even on dropping these signals for the prediction task. Hence, we conclude that status and flow related data which can be \*\*constant\*\* or \*\*doesn't oscillate\*\* between discrete values is \*\*not modelled well\*\* by using n-grams approach. We attribute this to the fact that in order to successfully model signature sequences, there should be visible change in the signal over time; it is effectively modelled further if these sequential changes are repetitive (cyclic). Furthermore, we see that the cyclic and continuous signal `'L\_T1'` was able to predict the sequence quite well as can be seen by the high `f1\_score` and a high number of true positives that it models-> `column 0`, which further supports our inference.

A Comparative Study of the implemented methods: In this section, we tabulate the results we obtained by running our implementations on the TEST set. We look at the following metrics:

- `S\_TTD\_SCORE`: The \*\*TTD\*\* or \*\*time-to-detection\*\* score is a representation of the time it takes (for the detection system) to detect the anomaly when it occurs. It is computed by calculating the differences between the time of detection (the time slice when the anomaly was detected) and the time of occurrence (the time slice when the anomaly first occurred or started) and then averaging over the different attacks observed in the test data. We feel that it is a reasonable metric for evaluating our detection systems as it captures the quality of system-response by looking at the time it takes for detecting an anomaly. A higher S\_TTD score suggests faster system response.

- `Sensitivity` or `TPR`: We tabulate the \*\*Sensitivity\*\* as a metric to demonstrate the system's ability to correctly predict the anomalous instances.

- `Fall-out` or `FPR`: We tabulate the \*\*Fall-out\*\* as a metric to get an idea of the number of False Alarms that are raised by the system.

We've already discussed the different types of anomalies that the four systems can detect in their independent sections (previously). We now apply the said evaluation metrics to compare their performance. We see that `ARMA` has the highest time-to-detection score meaning that it is able to respond quicker to the anomalies that might occur in a system. This is no surprise as it is a contextual approach that seems to model the data well. `PCA` and `LOF` have comparable response scores. The negative score for collective approach like `N-grams` is a result of 'windowing' our inputs which results in the entire collection (or window) being assigned as anomalous or non-anomalous. When the windows are unpacked, the labels get assigned to all the points in the window which causes the system to assign anomalous labels even \*\*before\*\* the anomaly occured! This is more of an implementation detail than a question mark on the performance of the N-grams.

Additionally, talking about the `TPR` and `FPR`, `ARMA` seems to raise relatively more number of false alarms (high FPR) which is a trade-off that the analyst has to make in deploying a detection system. It then depends on the cost of analysing an instance which gets flagged as anomalous because `ARMA` (as mentioned previously) models the detection task relatively well with a high TTD score. Out of `PCA` and `LOF`, `PCA` seems to fare better due to its high `TPR` and low `FPR` scores. NOTE: The LOF can be further optimized by tuning the nearest neighbour parameter.