2022 PRACTICUM

CHANGE POINT DETECTION OF TIME SERIES DATA

FOR COMNET MOBILE DATA PING TIME

BY

ALEX WALTON

(GT-ID: 903398136)

OMSA PRACTICUM

# Abstract

The idea behind this project is to detect significant change points in ping response time from cell towers around the US. The data is unlabelled, and presented to us in raw form. The data is also extremely noisy, and of the 363 cell towers, there are many different magnitudes of signal, that the final method must be able to deal with. The approach taken needs to be able to deal with significantly noisy data, and be able to distinguish between noise and real underling changes in the time-series data. A range of methods are explored, from unsupervised detection to a Neural Net based approach, to see if I can improve on the current accuracy.

The data is from 300+ cell towers, with 4 variables. In this project, identifying that there is a change in mean, and that when there is a delay in ping time, we prioritise voice data over data at all times. The algorithm not only has to be accurate in detecting change points within the time series data for each variable on each ping site, but also has a time constraint component. The data is pulled every 20 minutes from 2 servers, therefore we need this algorithm to be able to scan all the data within the time frame, as well as be able to show results, to allow engineers to highlight issues in reasonable time.

In the following project, I will discuss the methods I used, the difficulties, and also successes that allowed a final approach to be recommended, one that should improve the detection of change points as well as improve the accuracy of false positives.

# Table of Contents

[Abstract 1](#_Toc108983189)

[Table of Contents 2](#_Toc108983190)

[1. Introduction 3](#_Toc108983191)

[2. Problem Statement 5](#_Toc108983192)

[3. Methods 5](#_Toc108983193)

[1.1. PELT 5](#_Toc108983194)

[1.2. Bottom up 5](#_Toc108983195)

[1.3. Window 5](#_Toc108983196)

[4. CPDE Approach 6](#_Toc108983197)

[5. Neural Net Approach 10](#_Toc108983198)

[6. Results and Discussion 12](#_Toc108983199)

[7. Conclusion 12](#_Toc108983200)

[References 13](#_Toc108983201)

# Introduction

For this project, data from a plethora of mobile sites around the US, each with a timeseries lookback of ping times (in Ms) for 4 variables (EFlarge, EFsmall, BElarge, BEsmall). Each of these variables represents a type of data packet, and the response time of each of these packets of data, for every sample interval, is represented on a time series graph as shown below (Fig1.1)

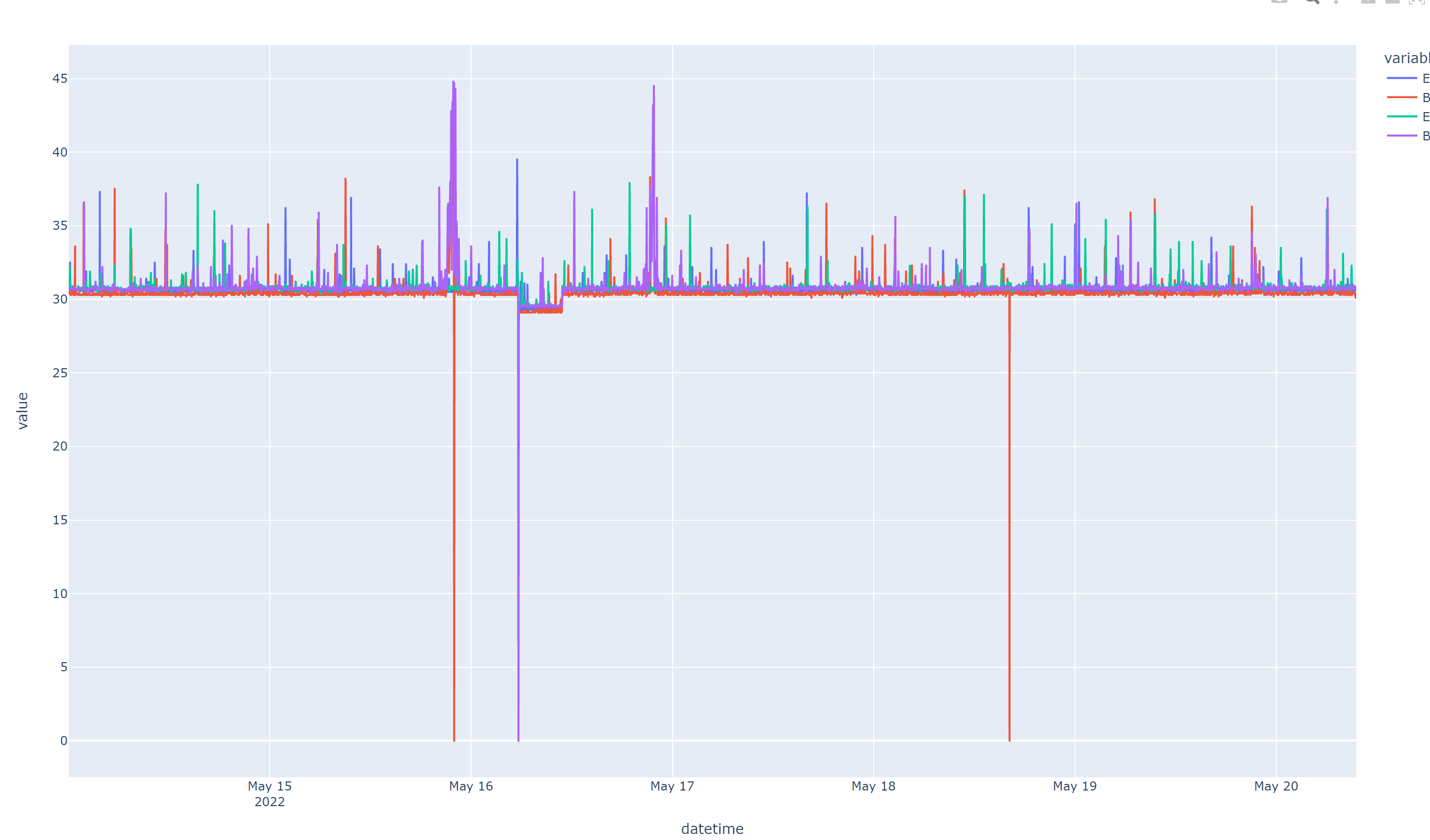
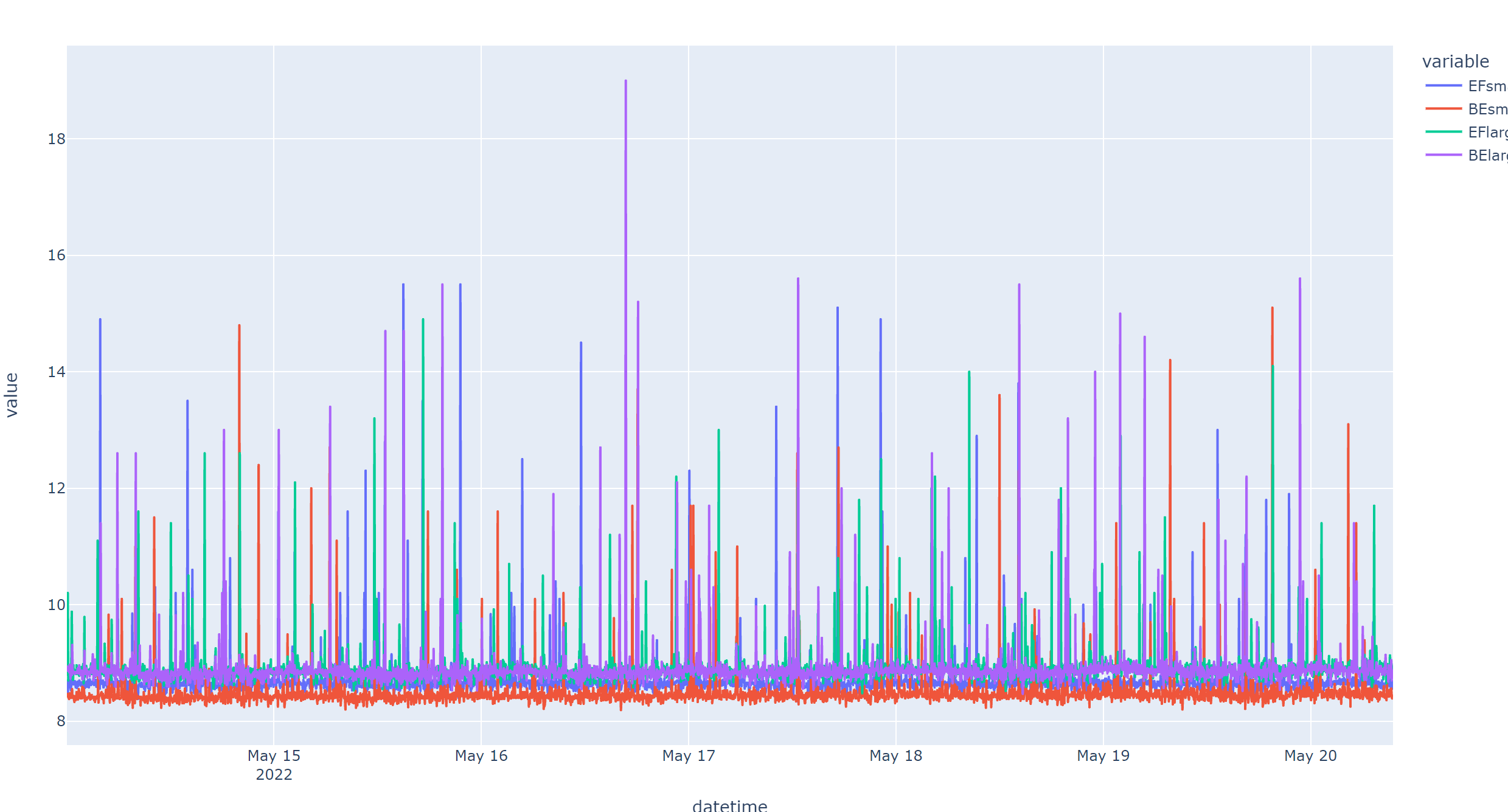


Figure .

The aim of the project is to correctly identify when a statistically significant change in the mean of the time series for each variable occurs. Alongside this, we want to ensure that when there is lag in the time series data, always we need to make sure that voice data is prioritised over data packets. The reasoning is simple: When on a mobile call, we can’t afford data to drop as the voice quality will sounds “computerised” and may not even get through at all. On the other hand, most mobile/internet devices have some form of local buffer, and therefore there is some tolerance if data is buffering/ delayed.

Changepoint Detection (CPD) is a well-researched area of time series analysis, for example, Truong et al investigate different offline change detection methods for multivariate time series (Charles Truong, 2020). In our particular use case, we can investigate offline changepoint detection methods as we don’t need real time monitoring. The different between offline and online change points is essentially as follows: Offline changepoint detects changes when all samples are received, whereas online changepoint attempts to detect changepoints in real time settings. If we can spot a real and significant change in the time series, as well as highlight to relevant areas in-between data being redownloaded, this is perfectly acceptable for this project. Alongside this, using offline detection allows us to visually see what the data looked like before a potential change point. This allows us to visually see the accuracy of any algorithm used.

After making the choice of using an offline detection algorithm, discussed in the previous paragraph, the next question is how do we approach detecting significant changes in a time-series? There are many approaches to address this question, simple CUSUM can be used to detect changes in data, however in this instance we attempt to reduce the number of false positives that are flagged to the user. CUSUM detection detects any change in mean, which can be highly inaccurate for the statistics of the data sat we have. In fig 1.1 we can clearly see a changepoint. Compared to fig 1.2, where the data is extremely noisy, and there is not a clear change in the mean of any of the pings observed.



We can see from the two plots shown above that we have very different types of data. Alongside this, the data supplied is unlabelled. This provided a challenge in itself, as to how to measure the accuracy of the model without manually analysing all plots to see if algorithm tested was detecting the correct changepoints.

For this project, attempting change point detection in the given data, a number of more advanced detection algorithms were used, and evaluated. Initial attempts included simple change point detections, which were investigated before settling on a much more accurate ensemble of changepoint detection algorithms. A python framework is also created, allowing the end user to rerun the different cost functions, and on different data. This allows customisable CPD in the future.

# Approach and methodology

There are many different change point detection algorithms available, via open-source python libraries. Some of these libraries offer a range of different detection methods for changepoint detection in time series data, while others offer similar algorithms wrapped in different libraries. Having unlabelled data meant that either the data could be hand labelled, or a method that didn’t require data to be labelled should be used. For this project, and given the fact that the data was only a small window of the timeseries data generated, I opted for an unlabelled approach, that would then be assessed graphically. This saved time initially in that labelling data was a manual process and was too time consuming, however, it meant that algorithms suitable for unsupervised learning had to be used. For example, one library initially looked at was a library called sktime (Alan Turing Institure) Many were tried and tested to assess their usefulness in detecting changepoints in the data highlighted above, until one was settled on and tested further. The three unsupervised changepoint detection algorithms tested are outlined below.

# Methods

## PELT

The Auto-detect number of change points (PELT) option uses the Pruned Exact Linear Time (Killick, 2012) algorithm to estimate the number and location of change points. PELT works by adding a penalty term for an inclusion of any additional changepoint. This is the means that the pay off between adding the new change point is weighed against the reduction in segment cost. For each point, if the cost is higher that the reduction in the segment cost, we will not include the segment as a change point Window

## Bottom up

This is described as a “fast signal segmentation” within the common libraries. The algorithm begins by creating the finest possible approximation of the time series, so that n/2 segments are used to approximate the length time series. Next, the cost of merging each pair of adjacent segments is calculated, and the algorithm begins to iteratively merge the lowest cost pair until a stopping criterion is met. (Keogh, 2001) Essentially the algorithm starts with many changepoints, and then iteratively deletes less significant ones. The complexity of order for the bottom-up algorithm is (n lon n)

## Window

The sliding window algorithm is also quick. It uses two windows that “slide” along the array of data. Then, the statistical properties of each window are calculated, and the discrepancies noted. If the discrepancies are significant, then we assume a boundary has been detected. The cost function here can be, for example, l2, which detected change in median and mean respectively

## Autoencoder

An autoencoder based approach to change point detection, using Time invariant representation (TIRE), was applied to our dataset. This approach detects change points including abrupt changes in the slope, mean, variance, autocorrelation function and frequency spectrum. (De Ryck, 2021). The TIRE approach returns a probabilistic value to each timestamp in the time series data, suggesting if a change point is at, or near, this point in time.

Of the methods highlighted above, the first three of these are well known CPD methods, with a plethora of literature online. For this project, the main python library used to aid in the investigation and results was The Ruptures library (Ruptures python library, 2022). This was the chosen as unsupervised machine learning library used to find the unknown number of changepoints within each multivariate time series. This was chosen after trials of other similar libraries, as it was much more customisable for the outcome of this project.

Libraries that used different approach, for example, Merlions change point library was initially approached as a reasonable approach (Sales Force) However, although we can see that there is potential for a significant change point detection in a generalised time series dataset (fig 3.1) This required a labelled dataset. As we have unlabelled and much nosier data in general, it was decided to not approach with this method.

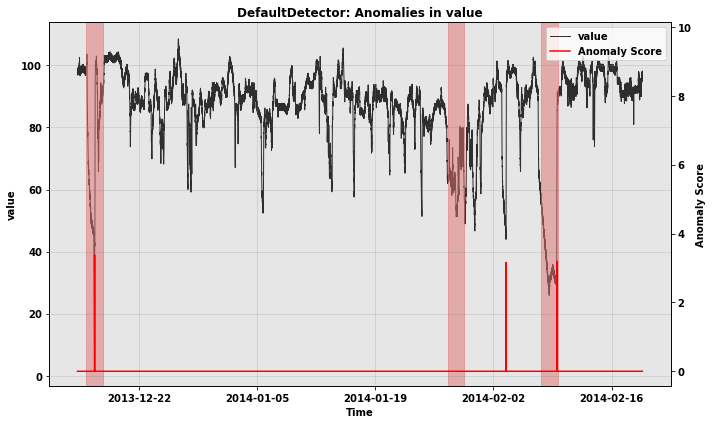


Figure .

Initially, approaching this project in a methodological way, wanted to test the three unsupervised methods above. To get a feel for is any of these models were a clear outlier in their detection ability for the data set, I went about setting trials of each of the three CPD approaches. Below is one of the first unsupervised methods using bottom-up algorithm, method 3.1 with a penalty value calculated as: (https://pro.arcgis.com, 2022)

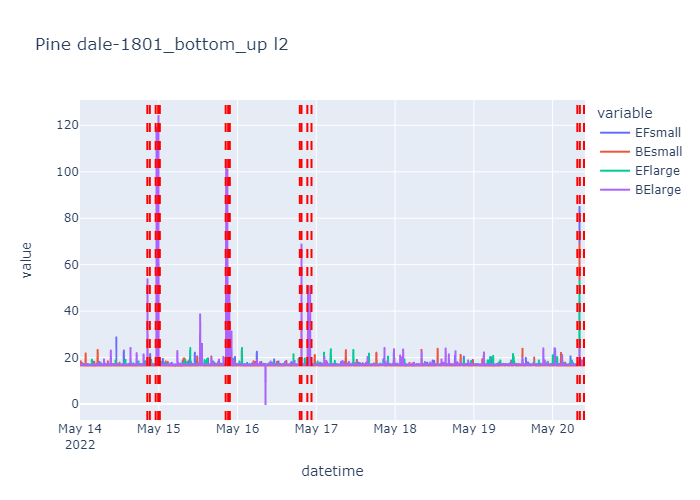


Figure .

You can see from fig 3.2 that the bottom-up algorithm, with the determined hyper parameters mentions, detects changes in the time series (albeit many more than possibly required). However, this particular CPD approach doesn’t seem to deal well with highly noisy, and non-substantial changes in mean. As fig 3.3 shows below, this is a noisy dataset and there is no real change in mean for any parameter. This would clearly show too many false positives.

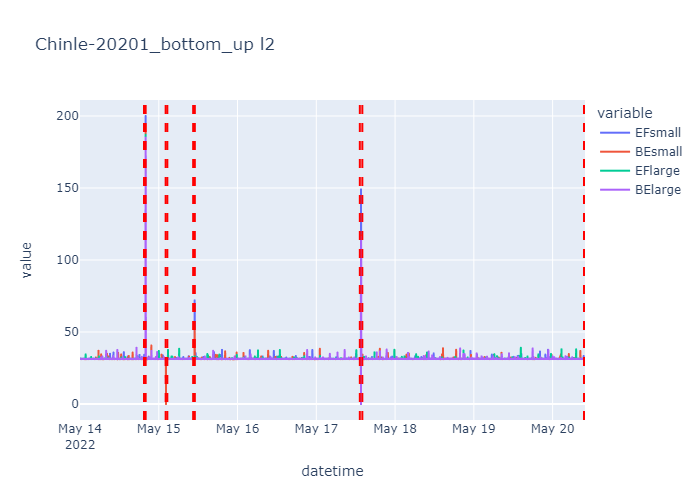


Figure .

Looking at the same CPD algorithm, but this time with a different cost function, we can see using a PELT CPD (method 3.1) algorithm but a kernel change point detection, fig 3.4 It can be see that we can prevent too much noise impacting the change detection algorithm.

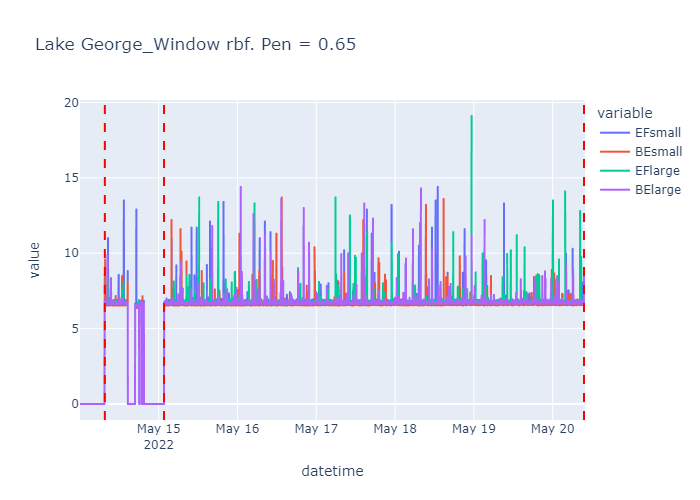


Figure .

However, comparing this to figure 3.5 where we used a bottom up with l2 cost function, we can see how many more changepoints we have detected.

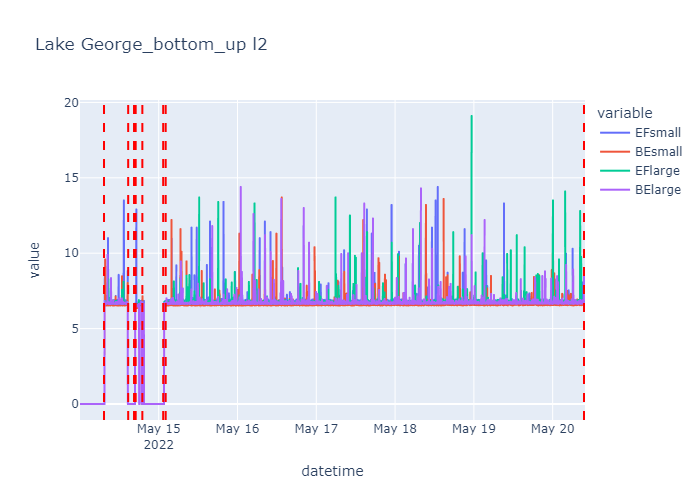


Figure .

Although the rbf cost function with the PELT model looks good, if we run this on the entire dataset, we come back with a total of only 2 graphs in all 363, where this algorithm detects a changepoint detections. This is clearly wrong. At this stage, we could go through each graph and manually label the changepoints, then run a supervised ML algorithm, however, I wanted to run with the unsupervised given that this data set is limited and we don’t have that much data to back test these algorithms on. It can be seen that a multitude of factors that can influence both the accuracy, time and noise robustness of the CPD models.

This then brings us to the approach favoured here in this report for the unsupervised models mentioned. The above highlights the issue in CPD method detection, as well as the cost function we can apply to each of the CPD algorithm. From an analytics perspective, starting with an initial reasonable guess is a good first step. However, it is clear that with the data we have, there doesn’t seem to be an applicable initial guess to run with. Learning hyperparameters would normally be approached using randomised grid search approach to finding the hyper parameters. I attempted to replicate this approach, but with an accuracy approach of the graphical representation. Initially, I looked at combining each of these models and looking at a weighting of CPD algorithms to test to see if some form of combination was applicable to improve the accuracy of the CPD whilst minimising false positives. We can see that for the initial plots, different algorithms are better under certain situations. However, this approach was superseded from further online reading. I attempted to combine each of the potential algorithms, initially through running each different algorithm using Pelt and Window methods mentioned in chapter 3. The issue was, as mentioned previously, time. It was too computationally expensive to run the multiple algorithms in quick enough time to make this worthwhile. The chosen method was to compare different cost functions (l1, l2, rbf) for each algorithm but also, henceforth described as Change point detection Ensemble (CPDE) (Katser, 2021). The novelty of this approach is that we can take the benefits of an ensemble method in choosing a cost function. This removes the user’s choice of cost function selection. As the cost function is always trying to be minimised, this approach then uses a scaling and aggregation decides how the “votes” of the aforementioned cost functions are chosen.

Although time taken to generate the initial results was longer than would be acceptable for the problem highlighted, once an accurate algorithm, aggregated cost function was obtained, running this on a dataset of 24 hours for each of the 363 masts was reasonably quick, and definitely within the timeframe needed between pulling datasets.

The investigation focused on 2 hyper parameters to tune for testing, one being the CPDE algorithm, one being the aggregation function, as this time, we have an aggregated cost function. I then ran through this algorithm with the test set (unobserved) to see if the algorithm picked up the correct changepoint location. The focus after testing all the above changepoint detection algorithms was then to find the quickest and most accurate one. Given the time constraints mentions above, we wanted the algorithm chosen to be reasonably quick, allowing sufficient time in between data pulldowns, to get a good picture of any significant changes within the time series data. Part of the methodology for achieving this was using both multiprocessing and multithreading within python. This allows us to maximise the speed for iterations through the data of cell sites (stack-overflow, 2022) This meant testing each CPDE scaling aggregation was doable within a reasonable timeframe. The biggest issue with all the above code is evaluating the results. We clearly have achieved an initial target of improving the change detection from a more basic level (for example, CUSUM) and we have back tested many of the hyperparameters explained above. We have to then look at accuracy. It is hard, without knowing the exact change points in the data (which we are not given in this instance) to methodically and statistically evaluate our results. I dealt with this by running an “evaluation” model, which set about plotting a plot of for each of the models and aggregation functions, for only a 24-hour lookback window

# Neural Net Approach

One last approach attempted to see if further improvements could be achieved was to attempt a Neural Network autoencoder. Autoencoders attempt to deconstruct the data, (here, we are referring to timeseries data) to a smaller feature vector, and then, using a decoder, reconstruct the data using a second Neural Network. An advantage of this approach is that it can be incredibly effective against noisy data. Making it a good approach to the highly noise data of the individual timeseries we currently have. Autoencoders are unsupervised, which again fits the specificity of the problem we have. This approach aims to locate significate changes in time series data. The auto-encoder basted mythology attempted was the Time Invariant Representation (TIRE) model (De Ryck, 2021). The use of the TIRE model allowed me to run the change detection with improvements: essentially from all the data, we are looking at observing a change in the time series, but also a significant and pronounced one. This can be difficult to remove false positives when the data is noisy. The output from the TIRE model, referenced above, with some changes in how the model decided that peaks were dissimilar enough, meant adding statistics on improving the differentiation score.

For example, we can look at the output of the model below:

The model produces two outputs, the first is a “similarity plot” this highlights the peaks the model detects, and how dissimilar they are to the other peaks/data within each variable for each site. The model then measures the width of each peak, but at a height ratio of 0.65. The reason this is chosen, is that a high peak, but very narrow, is most likely noisy data, that we don’t want to highlight as a change point. Measuring a peak slightly lower down should capture only that data that the peak is reasonably wide, thus equating to a time significant change in the mean.

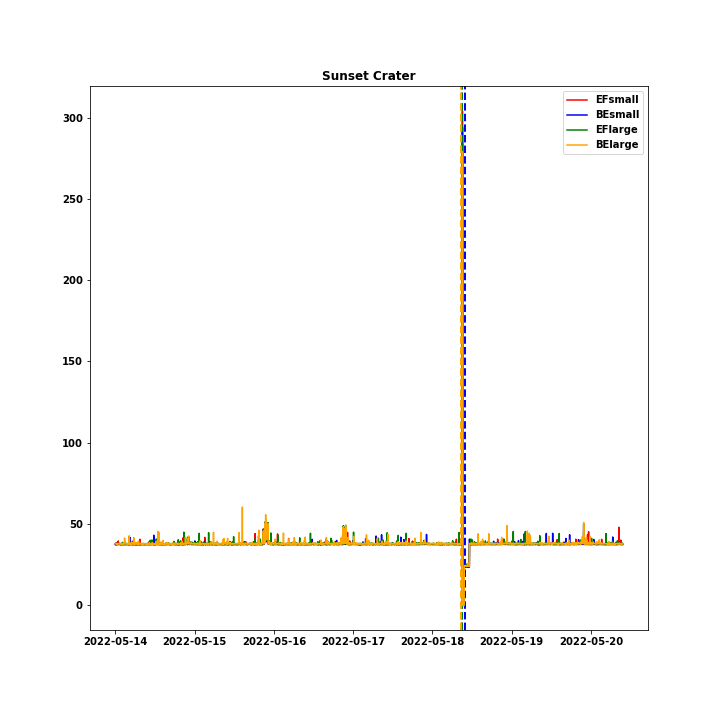


Figure .

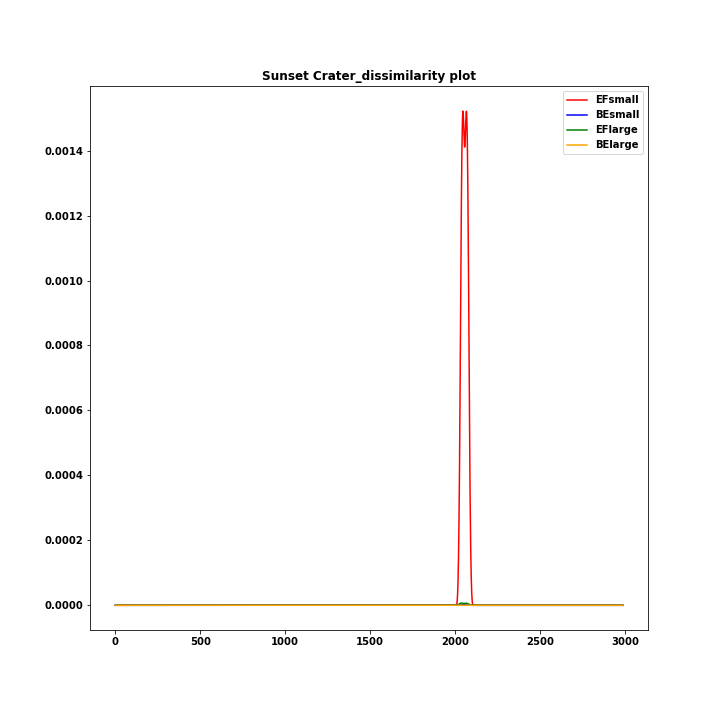
Figure 4.1 shows a reasonably noisy time series data. We can clearly see that for all variables there is a jump in the data, followed by a sharp drop. Upon then which we seem to see a significant change in the mean ping time. We can then view the model’s dissimilarity plots: 

Figure .2

Figure 4.2 shows a clear pick up form the model’s dissimilarity plots. In particular for BElarge. This means we can approach the changepoint detection in a slightly different way. Instead of trying to detect change points from extremely noisy data, by applying the change detection to the dissimilarity plots, we should have a much smoother function to be then able to apply statistically changes to. One route was to re-run the Change points we looked at earlier, however without making the algorithm too time consuming, we can leverage SciPy’s signal module. (https://docs.scipy.org/, 2022) This module allows us to detect peaks, and after applying some outlier analysis, we can obtain the index location of potential changepoints within the original data.

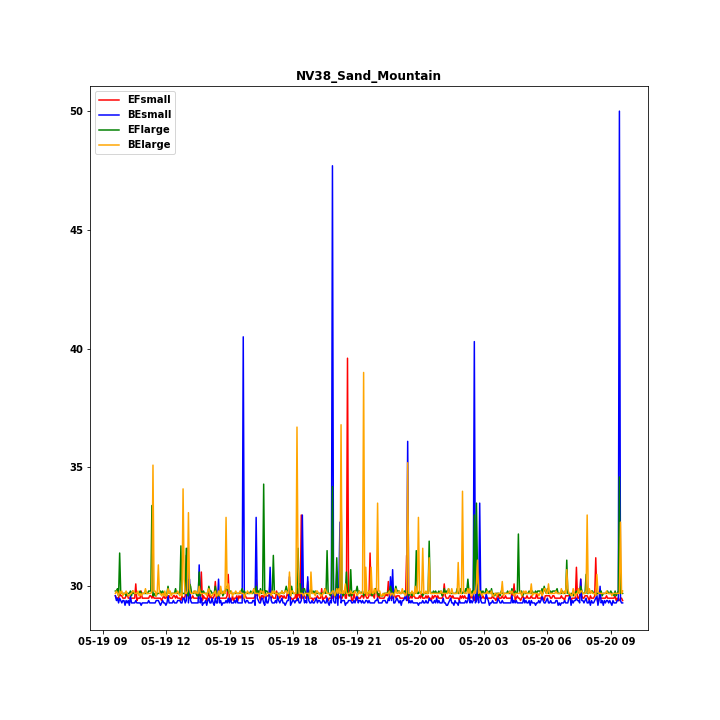
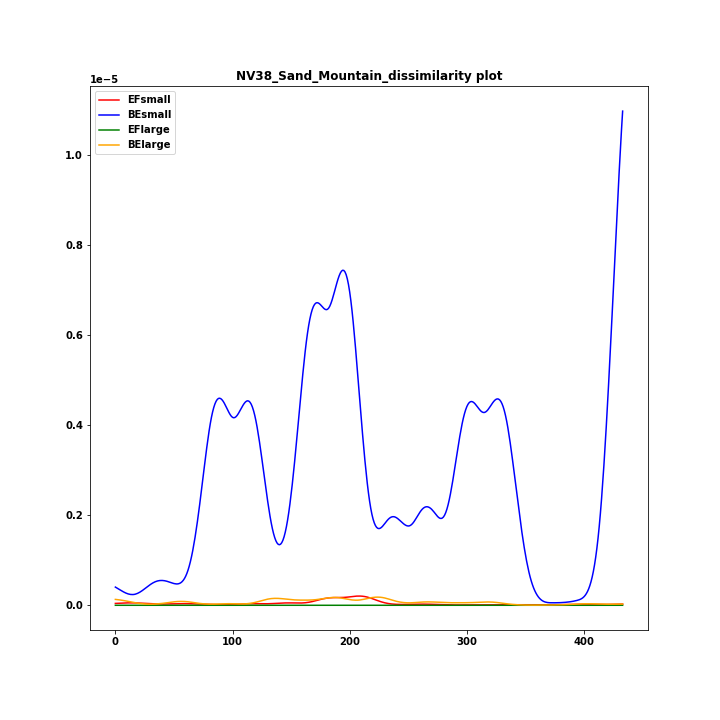


Figure 4.3

Figure 4.4

By looking at the dissimilarity plots, as opposed to the raw time series, we can see that identification of noisy data becomes much easier for the algorithm to observe. Fig 5.3 shows the extremely noisy ping data of one mast over 24 hours. The corresponging Dissimilarity plot in fig 5.4 shows that for the spikes in pingtime, they are extremely shot lived and their duration is realtively short.

The big advantage there is to the company this project is for, is that they now have a customisable Neural net, wrapped with statistical analysis of the dissimilar data, that allows this to be improved and enhanced. For example, one could take the model and use the parameters generate for each time series, and increase the number of epochs. Also, by saving down the weights of the Neural network, it may become clear over time that certain mobile sites have similar characteristics.

# Results and Discussion

The aim of this project was to apply and utilise the skills and analytic insight I have gathered over my time taking the MS analytics here at Georgia Tech. Applying it in a real-world situation, both in attempting to build on that knowledge, as well as aid the sponsor in improving their detection of errors within their Ping time series. Initially, the issue was approached with a simple application of detecting changes in time series with models that were relatively simple. However, that progressed into enhancing and improving libraries that were already available, with the python development framework. This let to a deep investigation of different, unsupervised, Changepoint detection algorithms. Each of these were investigated, and a final model chosen. The code passed to the sponsor should allow them to review this data, but also make changes to any of the implantation of the models themselves, to back test or run-on different subsets of data. The application of the models, as well as the outcome, will allow the sponsor to detect changes in the times-series more accurately, and within a reasonable time. This can be absorbed into their current detection framework seamlessly

The hardest part of the practicum was assessing the accuracy of the algorithms. This has been talked about throughout this practicum, and its not ever going to be perfect. By leveraging the knowledge gained from Neural networks, the final algorithm related to application of autoencoders on time series data, then looking at detection of changes in the dissimilarity graph, is a somewhat unique approach and one that can be built on. By having the time to focus on the different changepoint detection algorithms, it has been possible to improve the current detection process for ping data, and allow better and more accurate detection. Applying statistical and analytical knowhow to extremely noisy data, I was able to remove many of the false positives experienced in previous detection attempts. More work could be done on enhancing the Neural net, this seems to be the most promising: form increasing the number of epochs, tailoring the loss function and looing at different ways to detect peaks within the dissimilarity data, one would get even more accurate results.

# Conclusion and further work

As highlighted in the results, we found an improved approach to the detection of time series data from each cell tower, over and above a simple change in mean. This was done my harnessing the use of all the available data, as well as looking at the plethora of different ways we can detect changepoints within time series. One last approach was to try an Autoencoder model. This was implemented from this library (De Ryck, 2021), built off this paper (De Ryck, 2021)

# References

Alan Turing Institure, n.d. *sktime.* [Online]   
Available at: https://github.com/alan-turing-institute/sktime  
[Accessed 24 06 2022].

Charles Truong, L. O. N. V., 2020. *Selective review of offline change point detection methods.* [Online]   
Available at: https://arxiv.org/abs/1801.00718  
[Accessed 05 06 2022].

De Ryck, T. a. D. V. M. a. B. A., 2021. *https://github.com/deryckt/TIRE.* [Online]   
Available at: https://github.com/deryckt/TIRE  
[Accessed 1 06 2022].

De Ryck, T. D. V. M. a. B. A., 2021. Change point detection in time series data using autoencoders with a time-invariant representation.. *EEE Transactions on Signal Processing,* Volume 69, pp. 3515-3524.

https://docs.scipy.org/, 2022. *https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find\_peaks.html.* [Online]   
Available at: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find\_peaks.html  
[Accessed 5 7 2022].

https://pro.arcgis.com, 2022. [Online]   
Available at: https://pro.arcgis.com/en/pro-app/latest/tool-reference/space-time-pattern-mining/how-change-point-detection-works.htm  
[Accessed 01 07 2022].

Katser, I. K. V. L. V. &. M., 2021. *CPDE.* [Online]   
Available at: https://github.com/theovincent/CPDE  
[Accessed 1 7 2022].

Keogh, E. C. S. H. D. a. P. M. 2., 2001. , November. An online algorithm for segmenting time series.. *IEEE international conference on data mining,* pp. (pp. 289-296)..

Killick, R. F. P. a. E. I., 2012. Optimal detection of changepoints with a linear computational cost.. *Journal of the American Statistical Association,* pp. 1590-1598.

paperswithcode, n.d. [Online]   
Available at: https://paperswithcode.com/task/change-point-detection/codeless  
[Accessed 10 06 2022].

Ruptures python library, 2022. *ruptures python library.* [Online]   
Available at: https://centre-borelli.github.io/ruptures-docs/  
[Accessed 10 06 2022].

S. F., n.d. [Online]   
Available at: https://github.com/salesforce/Merlion  
[Accessed 01 05 2022].

stack-overflow, 2022. [Online]   
Available at: https://stackoverflow.com/questions/27455155/python-multiprocessing-combined-with-multithreading  
[Accessed 1 07 2022].