Technische Universität Berlin

Chair of Database Systems and Information Management

Master's Thesis

Empirical Comparison of Forecasting methods

Leonardo Araneda Freccero
Degree Program: ICT Innovation
Matriculation Number: 406022

Reviewers
Prof. Dr. Volker Markl

Advisor(s)
Behrouz Derakhshan
Bonaventura Del Monte

Submission Date
27.07.2021

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, 27.07.2021

. . . . . .

Leonardo,Araneda Freccero

Zusammenfassung

Tipps zum Schreiben dieses Abschnitts finden Sie unter [**?**]

Abstract

The abstract should be 1-2 paragraphs. It should include:

- a statement about the problem that was addressed in the thesis,
- a specification of the solution approach taken,
- a summary of the key findings.

For additional recommendations see [**?**].

Acknowledgments

For recommendations on writing your Acknowledgments see [**?**].

Contents

List of Figures

List of Tables

List of Abbreviations

NN Neural Network

DNN Deep Neural Network

RNN Recurrent Neural Network

List of Algorithms

1 Introduction

Time series forecasting, the ability to predict future trends from past data, is an important tool in science, for businesses and governments. However, it can be a double edged sword if the predictions are incorrect. Predicting the future is hard and predictions are only as good as the data and learning capabilities of the forecasting model used. Recently the Covid-19 pandemic had disastrous effects on society. Time series forecasting was used to predict the spread of the virus so that governments, hospitals and companies could plan accordingly to minimize its effects. However, many of the forecasts were overestimating the spread of the virus which lead to both organizational and health issues for hospitals, personnel and patients. With better forecasting solutions which would be capable of generating probabilistic forecasts this issue could have been avoided [**?**].

Improving forecasting accuracy is an active area of research and for that, new forecasting methods are continuously proposed [**?**, **?**, **?**, **?**, **?**]. As more forecasting methods are developed, these need to be compared in an accurate and reproducible way. The currently most popular method for comparing forecasting methods is through evaluating the algorithm on a couple of reference datasets [**?**]. As a part of this process, the datasets often undergo some preprocessing before a model is trained on them. Similarly, the hyperparameters of the algorithm are often tuned to fit the dataset in question. These steps are considered good practice as it allows the algorithm to perform optimally. However, if the method used to process the dataset is unclear in any way or if configuration details such as which hyperparameters used are missing, reproducing results becomes hard [**?**].

Some forecasting methods exhibit a non-deterministic behaviour where each subsequent run of the algorithm won't necessarily produce the same output. This is especially the case for deep learning algorithms as they are heavily relying on random processes such as dropout or random initialization of weights [**?**]. This makes it hard to reproduce results from these algorithms. Additionally, in the field of forecasting, multiple different metrics are employed in order to quantify the error of forecasts. This is due to some metrics being better suited for specific use cases. Often only a few of these metrics are used in papers when presenting new forecasting methods which makes reproducing results even harder.

In other disciplines of machine learning benchmarking suites are common tools for performing automatic reproducible comparisons between different algorithms. Some examples of these are MLBench and MLPerf. In time series forecasting

however, no such benchmarking suites exist instead results from competitions such as the M competitions are held as the reference which future papers compare to.

Reproducible results imply that an algorithm needs to produce the same output no matter when someone wishes to reproduce them. It should not matter if it is shortly after the algorithm was presented or a long time thereafter. This introduces difficulties as some algorithms are continuously being improved upon by their makers. If any of these improvements would change the predictive power of the algorithm this would make it impossible to reproduce any results. These types of performance changes can be handled by automated tests which ensure that the accuracy of the algorithm remains. However, defining such tests is hard for non-deterministic output and therefore accuracy regressions can pass through undetected [**?**]. Changes in predictive performance can also stem from third party updates of dependencies. These things are hard to control and potential problems are hard to foresee ahead of time.

1.1 Motivation

Previous comparisons of forecasting algorithms have found subpar performance of machine learning based approaches both in terms of accuracy and resource consumption when compared to classical methods such as Arima, ETS and Theta. However these comparisons are often made unfairly as the deep learning models which were tested were rather simple neural networks with few bells and whistles. More advanced deep learning methods such as DeepAR, DeepState, WaveNet etc. have been developed since. Thus, there is a clear benefit of thoroughly evaluating these modern algorithms in a fair and accurate way.

Accurate and fair comparisons can however only be made if the algorithms being tested all have an equal opportunity to perform well. Ensuring equal opportunity allowing forecasting methods to perform optimally is however hard and implementing a strategy for doing this is required when doing large scale comparisons. Additionally, the choice of error metric to use when comparing forecasting models is important as each error metrics has their own benefits and drawbacks. Using a flawed error metric can invalidate a seemingly fair comparison as some metrics are unreliable for certain applications or datasets. Implementing a strategy for when to use and not use certain error metrics is required in order to allow for fair comparisons.

Reproducibility of results is a core tenet of the scientific method. Despite this, being able to reproduce results from forecasting algorithms is not always straightforward [**?**]. Much of the difficulties regarding reproducibility stem from two core parts. Firstly, code and datasets are often not public and secondly the non-deterministic behaviour of certain forecasting models causes different runs of

the models to produce different results.

This thesis will focus on how fair, accurate and reproducible comparisons of modern forecasting methods can be made. Particular focus will be put on how comparisons can be made in a reproducible way for non-deterministic forecasting models. As part of this thesis, a system is implemented which automates the tuning and benchmarking of forecasting models. This system is then used to perform a large scale comparison of several modern forecasting algorithms over multiple datasets.

1.2 Research Question

This thesis has as an overarching goal to compare modern forecasting models. This question is complex in itself and can be separated into three sub-questions:

1. *Accuracy*: How to perform accurate comparisons between forecasting models?
2. *Fairness*: How can one fairly compare forecasting models?
3. *Reproducibility*: How to make comparisons reproducible?

1.3 Contribution

The main contribution of this thesis is Crayon, an open source library for benchmarking deep learning models in a fair, accurate and reproducible way. Many deep learning based forecasting algorithms are non-deterministic in nature which makes reproducing results hard. Crayon solves this by ensuring that the distribution of error metrics is reproducible. With distributions of error metrics, quantifying how good the accuracy of an algorithm is becomes a problem. To handle this, Crayon implements a custom scoring method, the "crayon score" for ranking forecasting models against eachother.

This thesis also investigates the efficiency of using various hypothesis tests such as the t-test and the Kolmogorov-Smirnov test to make non-deterministic output from forecasting models reproducible. Additionally, these tests are evaluated on forecasts generated by several modern forecasting solutions on several datasets. The practical benefits of applying hypothesis testing to verify distributions of forecasts is showcased on several modern forecasting algorithms trained on several real world datasets. Further, as a practical example this thesis shows how reproducibility by hypothesis testing can protect modern forecasting solutions from suffering accuracy regressions.

An additional contribution of this thesis is the analysis of 21 datasets with the purpose of identifying a representable subset to use when benchmarking forecasting methods.

As the final contribution a large empirical comparison where the predictive performance of multiple modern forecasting algorithms is compared over four popular datasets.

2 Scientific Background

A time series is a set of data points with a clear ordering in time. Some examples are the exchange rate between two currencies over time or the electricity consumption of households, see figures 1 and 2 [**?**].

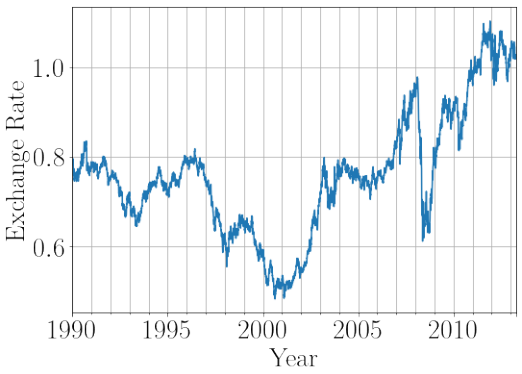

Figure 1: Exchange rate of two currencies from 1990 to 2013.

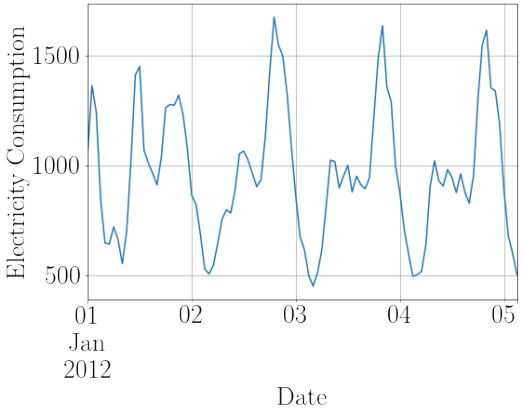

Figure 2: Electricity consumed by households.

Timeseries which increase or decrease in value over a longer period of time is said to have a upwards or downwards *trend*. If a timeseries has evenly distributed patterns such as the spikes in figure 2 the timeseries is said have a seasonal component with a frequency equal to the distance between the spikes. For the electricity timeseries this would be 24 as household electricity consumption follow the daily rythm of everyday life i.e. use more energy during the morning and evening with 24 hour intervals. If the timeseries exhibit spikes without a clear frequency, these are knowns as *cycles* [**?**]. The data which cannot be described by these features is known as the *remainder* [**?**]. Not all timeseries exhibit all of these patterns, for example, the timeseries in figure 2 does not display a trend.

2.1 Time series decomposition

Timeseries decomposition is the act of extracting features such as the seasonality and trend from timeseries data. This is often done in order to gain understanding

of the data [?]. While several methods of performing time series decomposition exists, one of the most powerful methods commonly used is STL decomposition [?]. After a timeseries has been split into its constituents one can quantify the strength of the trend and seasonality apparent in the data. This is done through compairing the variances of the seasonality S and trend T with that of the residual R. The strength of the trend F_t can thus be calculated as:

$$F_t = max(0, 1 - \frac{Var(R)}{Var(T + R)})$$

The strength of the seasonality F_s can be calculated as:

$$F_s = max(0, 1 - \frac{Var(R)}{Var(S + R)})$$

2.2 Time Series Forecasting

Time series forecasting is the art of predicting future trends of time series based on past observations. It is a key technology within many fields and is fundamental to the business decision process for many companies. Being able to interpret time-series in order to predict future trends makes it possible to plan for the future. For example power generation companies can adjust the amount of electricity which should be generated to match the expected demand and banks could make informed decisions about possible investment opportunities. There are many methods of generating such predictions, ranging from the use of simple heuristics to complex machine learning methods [?]. An overview of how to use deep learning to generate such forecasts is presented in section 2.4 and several state of the art forecasting models are presented in section 2.6.1.

Time series forecasting differs from common machine learning tasks such as image recognition or language processing in that predictions are not binary. More specifically, in time series forecasting a prediction is expected to be a close estimate of future values. Thus, quantifying the magnitude of the error from the ground truth is of importance in order to properly evaluate the accuracy of a forecasting model. This is commonly done by calculating an error metric, the choice of which varies depending on what is being forecasted and on preference of the forecasting practitioner. [?, ?, ?] In section 2.3.1 some such error metrics are presented.

Forecasts often comes in one of two forms; probabilistic forecasts or point forecasts. A point forecast is a forecast which consist of only one estimation of future values. This estimation could be either a single point in time or a series of points for several different timesteps. In figure 3 an example of a point forecast for several timesteps can be seen. Point forecasts has the disadvantage that they do not confer how accurate they are which may lead to prectitioners being overly confident

in the forecasts. This has historically caused several issues and more recently with covid-19 where over relying on point forecasts led to overly drastic measures for governments and hospitals [**?**]. Probabilistic forecasts such as the one in figure 4 includes this information by generating prediction intervals of the forecasts.

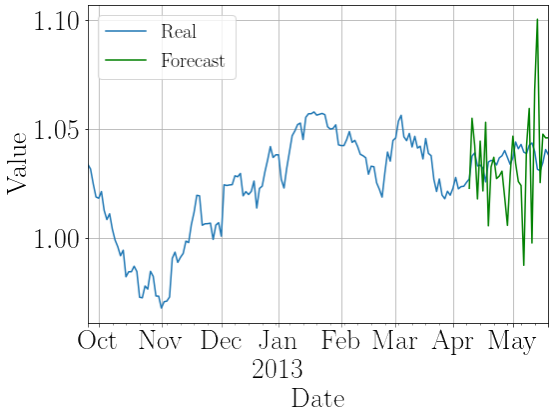

Figure 3: Point forecast

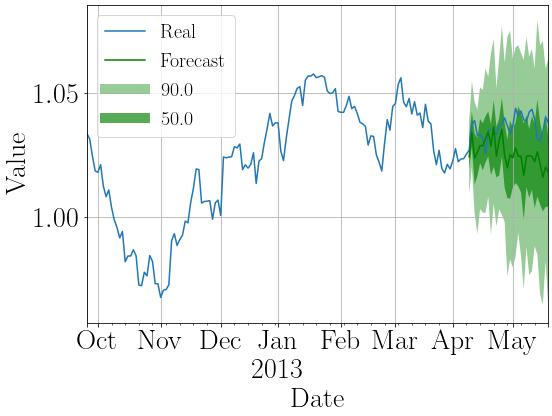

Figure 4: Probability forecast

Many methods exist for generating forecasts on timeseries data, two common groups of these are machine learning based methods and trivial methods. A trivial forecasting method would be, for example, to predict that the next value should be the same as the last recently observed value or the average of the last N observations. For certain timeseries, trivial methods perform competitively and as such these are often held as a baseline to which new forecasting methods are being compared too. Some more complex methods include those based on regression analysis such as linear regression or autoregressive models such as ARIMA [**?**]. Autoregressive models are a subset of regression models where the temporal ordering of the datapoints matter. I.e. autoregressive models use past observations to predict future values.

Lately, inspired by the successful use of neural networks in other domains, several neural network based approaches for time series forecasting have been introduced. Early implementations such as simple multi-layered perceptrons did not perform competitively with the more classical approaches such ARIMA. However modern deep learning models such as DeepAR [**?**] and MQCNN [**?**] have reported highly competitive accuracies.

Forecasting models for forecasting can also be split into further sub families such as whether they are local or global models. Local models are trained on individual time series while global models are trained on all time series in the training set. Essentially for a dataset with N timeseries, if using local models, one will have N individually trained models. If the algorithm would be a global model then only a

single model would be used for all N timeseries in the dataset. Generally speaking, local models performs well for timeseries with much historical data. However for short or new timeseries, local models often perform poorly due to the inability to train the model on sufficiently large amounts of data points. This is known as the cold start problem. Since global models train on the entire dataset across all timeseries, the length of any individual timeseries has less of an impact. I.e. generally global models suffer less from the cold start issue as the global model can use data from other timeseries as a basis for its predictions [**?**].

So far, we have seen the time series forecasting problem as generating a forecast from a single timeseries for example predicting future temperatures based on previously recorded temperatures. Predicting using singular timeseries such as this is known as univariate forecasting. Another family of forecasting solutions leverage related timeseries when making predictions. For example temperatures may depend on cloud coverage or the hours of sunlight. Algorithm which leverage information from related timseries are known as multivariate algorithms. The benefits of multivariate forecasting is that trends or patterns can be identified using these related timeseries in order to improve the accuracy for the target variable. For example, the temperature is higher whenever there is a large amou8nt of sun hours and no clouds.

2.3 Evaluating forecasting performance

If a model generates forecasts it is important to be able to quantify how good or bad its predictions are. This is generally done via a process called backtesting. When backtesting, first the algorithm is trained on a dataset, thereafter, the trained algorithm generates predictions on a test dataset. The generated predictions from the algorithm are then compared to the ground truth and a suitable error metric quantifying how wrong the prediction was from the expected value is calculated.

In other machine learning domains such as image recognition, the datasets used for training the models are disjoint from the test datasets. However, in the domain of time series forecasting this is not the case. Generally the train set contains the same time series as in the test set except for the last couple of data points for each time series. The number of removed data points is commonly the number of timesteps that a model should predict [**?**]. In figure 5 a train test split is presented of a time series with six data points. In this example, a forecasting algorithm should predict one timesteps into the future, thus, the last datapoint would be removed and the train set would contain five data points while the test series would contain six.

A popular method of increasing the amount of test data is to make the test dataset contain multiple copies of each time series where each one is successively

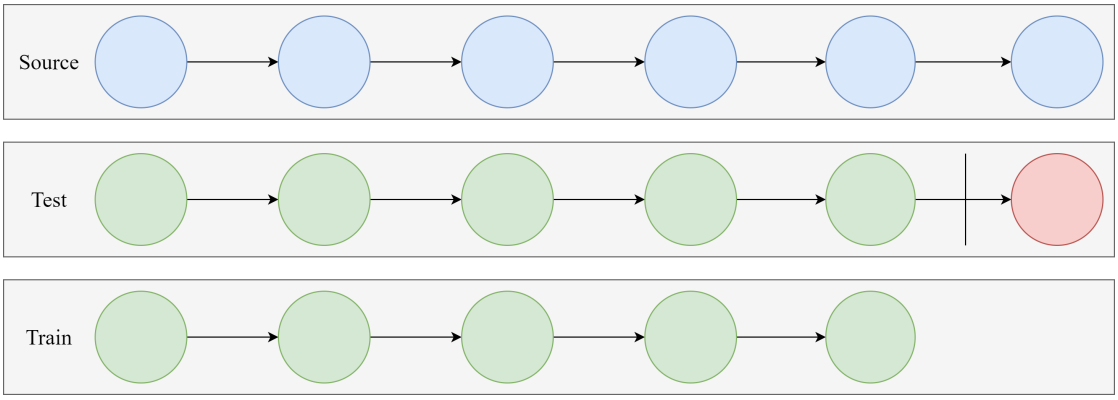

Figure 5: An example train test split of a time series with six datapoints. The red circle is the datapoint which prediction will be compared to.

shorter. The train dataset would contain the shortest of the truncated time series truncated one additional time. This process is called K-fold cross validation but is also known as forecasting on a rolling origin or time series cross validation. Performing a 3-fold cross validation on a dataset containing one time series of length six timesteps would result in a dataset containing three time series. An example of this is shown in figure 6. The advantage of the larger test dataset generated by this approach is that the performance of a model can be more accurately determined as it is excercised on more unseen data [**?**].

2.3.1 Error Metrics

When a forecasting model generates predictions on a timeseries, one needs to be able to quantify the error of the prediction from true observed values. In order to do this many different error metrics exists each with its own benefits and drawbacks. Often the choice of metric is dependent on data, and business application where the prediction will be used. In this section some of the error metrics which are commonly occurring in literature are presented. These metrics are calculated automatically when performing a backtest in GluonTS, see 2.6 for details about this process. Note that only a subset of the metrics available in GluonTS is presented here to form a basis for subsequent chapters.

Error metrics for forecasting can generally be split into three different categories: scale-dependent, scaled and percentage based [**?**]. Within each of these categories multiple different metrics exists. The remainder of this section describes some common error metrics within each category along with pros and cons and how they are calculated. For the rest of the thesis, a forecast is referred to as $\hat{Y}$ and the expected values are referred to as Y.

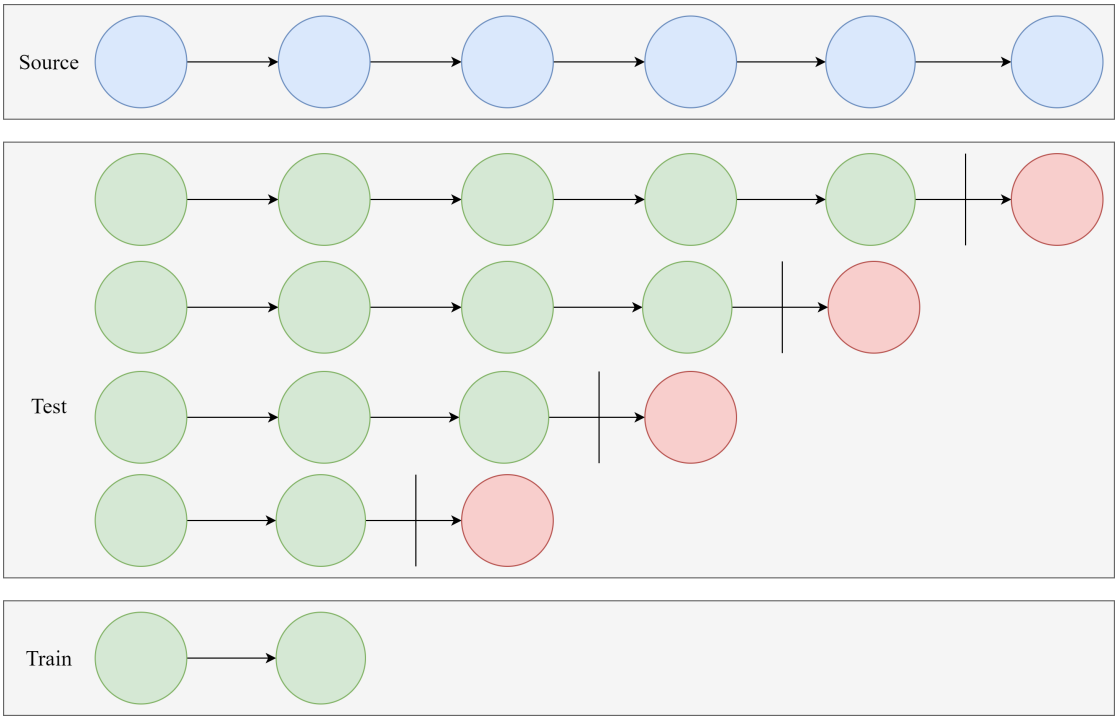

Figure 6: Example of a four fold cross validation dataset split. The red circles are the datapoints which predictions will be compaired to. Worth noting is that the train dataset here only has one timeseries of length two, i.e. one less than the shortest timeseries in the test dataset. This example assumes a prediction length for the dataset of one datapoint.

Absolute Error

The absolute error is calculated as the distance between the predicted value and the ground truth. An absolute error is positive and a value of 0 is optimal. It is an intuitively simple error, thus it can easily be understood and discussed when comparing algorithms.

However intuitive the absolute error is, there is a a key issue with this error and that is that it is scale dependent. I.e. the greater the values of the dataset being predicted on, the larger the absolute error will be. This makes it impossible to compare the accuracy of an algorithm across different datasets unless they have the same scale. [**?**] Furthermore, scale dependent metrics such as these may not be suitable even for comparing between timeseries within a single dataset. Imagine a dataset such as the electricity dataset (see table 3) where each timeseries is the electricity consumption from a household. One household may contain many people and appliances while another may be a factory full of equipment with high

power demands. These two timeseries would have two very different scales thus making it hard to compare the accuracy of the predictions.

$$AbsoluteError = \sum |Y - \hat{Y}|$$

Figure 7: Equation for calculating the absolute error

Root Mean Squared Error - RMSE

The root mean squared error is another scale dependent error such as the absolute error. RMSE is calculated by taking the square root of the mean of the squared error. The RMSE thus punishes larger errors more than smaller errors. The RMSE is more complicated to interpret than for example the Absolute Error due to the non-linear nature, despite this, RMSE is widely used in practise [**?**, **?**].

$$RMSE = \sqrt{mean((Y - \hat{Y})^2)}$$

Figure 8: Equation for calculating RMSE

Mean Absolute Percentage Error - MAPE

Percentage based errors such as the Mean Average Percentage Error normalizes the errors between 0 and 1 thus MAPE can be compared across datasets with different scales. While this is beneficial, MAPE suffer from other issues. For example MAPE becomes infinite or undefined if the ground truth is 0 for any parts of the prediction. Further issues exists for example that it penalizes negative errors more than positive errors. These issues has lead to variations of MAPE to be created such as Symmetric MAPE [**?**, **?**].

$$MAPE = mean(\frac{|Y - \hat{Y}|}{|Y|}))$$

Figure 9: Equation for calculating MAPE

Mean Average Scaled Error - MASE

MASE is a metric which takes the average error across all timeseries and scale is based on the error from a reference forecasting. An example a model which MASE uses for reference would be the Naive2Estimator detailed in section 2.6.1. Scaling the forecasting error in this way causes the MASE metric to be scale independent which makes it suitable when comparing across datasets. A MASE of 1 corresponds to that the algorithm under test is equally good as the naive model. A MASE below 1 means that the current model performs better than the naive model while a MASE above 1 means that the model performs worse than the naive baseline model [**?**, **?**].

In certain scenarios, for example if the reference model would result in a perfect prediction a division by zero would occur and MASE would tend towards infinity. This is a major drawback of the MASE metric which renders it unusable for certain datasets.

In equation 10 the denominator is the error of a seasonal naive model. Y_t means the value at time t and Y_{t-m} means the value of the timeseries at the previous period, i.e. if the expected seasonality is 24h Y_{t-m} would become the value 24 timesteps previously.

$$MASE = \frac{mean(|Y - \hat{Y}|)}{mean(|Y_t - Y_{t-m}]|)}$$

Figure 10: Equation for calculating MASE

2.4 Deep Learning forecasting methods

Neural Networks (NN) have in recent years been successfully applied in many different domains such as image recognition and natural language processing. However, within the field of forecasting NN based approaches has performed subpar compared to more traditional statistical methods [**?**, **?**, **?**, **?**]. This relative inefficiency between NN based approaches and classical methods and the success of NN in other domains has resulted in much research being made in improving tools and NN based models for forecasting [**?**].

A simple neural network, similar in architecture to the network described in section 2.6.1 is presented in figure 11.

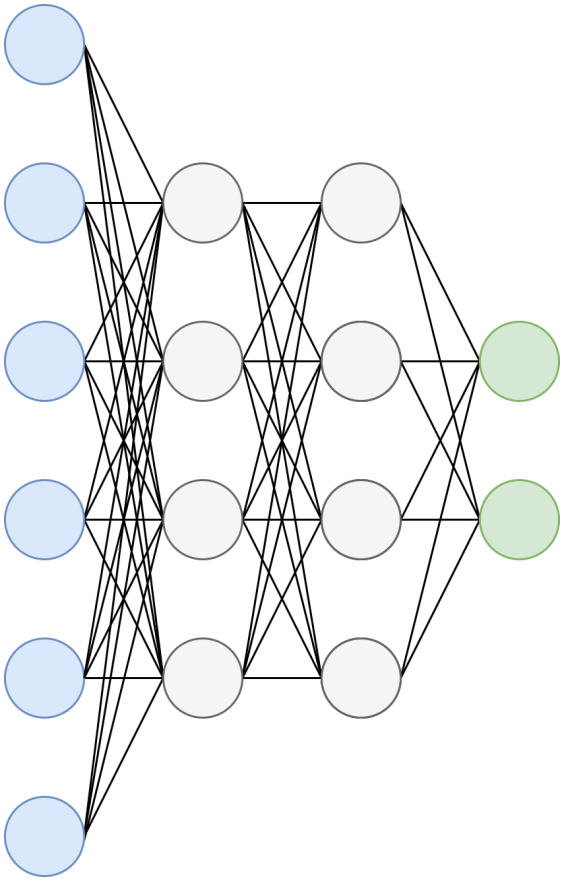

Figure 11: A simple neural network for forecasting based on the SimpleFeedForwardEstimator in section 2.6.1.

This very small network consists of four fully connected layers of different sizes with six input nodes, two fully connected hidden layers with four nodes each, and an output layer of two nodes. In this network, the input layer (marked in blue) has the same amount of nodes as the number of timepoints in the past which the network should make use of. This is will in the remainder of the thesis be called the *context length* of the network. The output nodes (in green) for this network produce a prediction of two timesteps. Thus this network has a *prediction length* of two timesteps. The context length and the desired prediction length are often exposed as hyperparameters for time series forecasting algorithms since they vary between datasets.

An issue with the simple network in figure 11 is that each timestep in the forecast is not influenced by the previous one. i.e. the prediction at timestep N does not depend on the prediction at timestep $N-1$. In real life scenarios, timeseries data often has a temporal dependency, for example, the temperature two days from now is affected by the temperature of tomorrow. This dependency is lost in simple neural network architectures such as this one. An alternative neural network architecture which can make use of temporal effects is known as a Recurrent Neural Network (RNN).

RNNs reuse the output of their nodes as inputs for the next iteration of the algorithm. This recurrent link causes previously seen values in the sequence to be represented in a hidden internal state. While this makes RNNs suited for use with sequential data such as time series, they become hard to train successfully as long term dependencies are lost [**?**]. One of the most successful techniques for use in RNNs to solve this is the Long Short-Term Memory (LSTM) node architecture. LSTM nodes enables RNNs to remember previously seen data for longer and forget or ignore parts of it [**?**].

Some issues with NN based approaches is that they require a large amount of data to be trained on properly. This is due to the fact that NNs are prone to overfit which makes them unable to generalize well to new data [**?**]. Despite this, NN has several advantages over classical approaches such as the capability to learn non-linear patterns from the data. Additionally, NN models are capable of cross learning between related time series without the need of manual feature extraction [**?**].

2.4.1 Sources of randomness

Neural networks heavily rely on random processess to function and thus randomnes is introduced at several points in a forecasting workflow. Some randomness can be introduced whenever an neural network is initialized as the weights and biases of a networks often are set randomly. Similarly, when training a neural network randomness can be introduced if optimization techniques such as dropout is used

as dropout randomly sets weights in the network to 0 at certain points in the training loop which causes the algorithm to be less prone of overfitting the data [?]. Even the method used for training neural networks, stochastic gradient descent introduces randomness.

Deep NNs are notoriously slow to train when compared to many other methods, thus lowering the time spent for training and inference is often done by parallelizing algorithms such that they run on several threads on a CPU or on hardware accelerators such as GPUs, FPGAs or ASICs. The execution speed of a forecasting model is thus highly dependent on what hardware the models are running on, however the hardware configuration also impacts the variance of the accuracy of ML models [?]. Parallelization of an algorithm often introduces a larger amount of data shuffling which impacts the rate of convergence of an algorithm. Parallelizing a forecasting model may make it unstable so that it cannot converge as will be shown in chapter 9. Due to this it is important that the hardware used when training and tuning forecasting models is presented in detail so that results can be properly reproduced [?].

A common method for reducing the randomness associated with ML models is to fix the seed used by the random number generator to a known number. While this removes randomness from for example random initializations or from backpropagation [?], other sources of randomness such as that introduced by the hardware or the OS remain. Furthermore, fixing the seed can be seen as an additional hyperparameter which needs to be tuned as it directly impacts the predictive performance of forecasting methods. While there are some major benefits of fixing the seed, there are also drawbacks. For example,luck becomes a factor in what accuracy can be expected for an algorithm. I.e. the accuracy of a model may seem highly competitive while if another seed would be used the accuracy of the model could change significantly [?]. In real life usecases such as when forecasting are used in companies or organizations, setting the seed may be beneficial or even necessary in order to optimize its performance for an individual task. In research however, setting the seed without reporting what it was can bias results and make them technically irreproducible [?, ?, ?].

One method of lowering the variance of machine learning methods is to use ensambles of forecasting methods as per the Bootstrap AGGregatING (Bagging) technique [?]. An ensamble is a set of multiple different ML models with slightly different configurations which are all trained in parallel.Bagging extends this such that a the input dataset is randomly sampled to create multiple independent datasets. Each model is then trained on one of these datasets and their forecasts are then combined via some voting or weighting scheme [?]. While bagging is often associated with decision trees they are also suitable for use with deep learning forecasting models such as the NBEATS Ensamble Estimator discussed

in section 2.6.1.

K-fold cross validation which was discussed in section 2.3 can also lower the variance of forecasting algorithms as it introduces multiple different train-test splits from a single dataset [?].

2.5 Benchmarking

Backtesting is a fundamental part of benchmarking, however benchmarking is also concerned with ensuring that different models can be fairly and reproducibly compared to each other [?]. In other domains of machine learning several benchmarking solutions exists, most notably MLPerf [?], MLBench [?], DLBS [?] and UCR [?]. Despite the success of benchmarks in other ML domains no established "go-to" benchmark exists in the domain of time series forecasting [?].

Attempts has been made to create such a benchmark, one such example is *Libra* [?] which uses custom datasets containing timeseries sampled from various well known datasets. The sampling methodology used in Libra was aimed to create datasets with a high amount of diversity between the timeseries. Due to this diversity, the datasets in Libra are not suitable for global forecasting models as they are highly unrelated. In a nutshell, Libra focuses on benchmarking the accuracy and time-to-result for univariate forecasts generated by local models.

Name	TS	Integration	DS	Focus	Info
Libra	X	R	X	A, S	Diverse datasets for univariate local models
UCR	X	CSV	X	A	Dataset repository with reference accuracies.
MLPerf		Docker, CLI		S	Benchmarking for hardware and ML frameworks
PMLB	X	Python, R	X	-	Dataset repository with 298 datasets for regression and classification.
MLBench		Kubernetes		S	For distributed ML frameworks
DLBS		Docker, Python		S	Benchmarking for hardware and ML frameworks

Table 1: Overview of six benchmarking suites for ML, the column marked TS depicts whether they support time series forecasting and the DS column is whether they offer custom datasets. A is shorthand for Accuracy and S for Speed.

Instead of using benchmarking suites when comparing forecasting models, new forecasting solutions are instead pitted against each other in forecasting competitions. The Makridakis Competitions organized by Makridakis Spyros et.al. are the most well known of these and has at the time of writing undergone five iterations; the M1 in 1983 [?], M2 in 1982 [?], M3 in 2000 [?], M4 in 2018 [?] and the M5 competition in 2020 [?]. These competitions have been highly impactful for the field of forecasting and has been the norm to which researchers compare too. However it was not until the M3 competition that NN based forecasters were included. These performed poorly compared to the non-NN algorithms which can partly be attributed to the limited size of the M3 dataset of 3003 timeseries [?]. This shortcoming of the M3 dataset lead to the creation of the M4 competition which had a similar but much larger dataset containing 100 000 timeseries [?]. Despite this, the findings of the M4 competition was inline with that of the M3 competition in that approaches purely using NN performed worse than even simple models. However, the M4 competition winner was a hybrid approach which combined RNNs with classical models. The latest competition, the M5, was focused on retail sales forecasting and found for the first time that ML based solutions, primarily Gradient Boosting Trees but also NN approaches such as DeepAR outperformed classical statistical methods [?].

In 2015 a thesis comparing forecasting solutions was published with the topic: *Benchmarking of Classical and Machine-Learning Algorithms (with special emphasis on Bagging and Boosting Approaches) for Time Series Forecasting* [?]. This thesis put the focus on performing a thorough dataset analysis and evaluating 14 algorithms on three real world datasets; Tourism, M3 and the NN5 [?]. Each algorithm was evaluated on multiple versions of each of these datasets with different preprocessing applied to them. In addition to evaluating the algorithms on the three real life datasets, a simulation study on artificial data was conducted. The purpose of this was to identify strengths and weaknesses of the algorithms for timeseries with simple characteristics such as only having trend or seasonality.

It is unclear whether time series cross validation was used as well as how many times each algorithm was executed. Further, as the title suggests there was little emphasis on deep learning methods and only a very simple neural network, similar to the one presented in figure 11 was used. This neural network performed worse than the naive approach for all datasets except for on NN5 [?].

2.5.1 Reproducibility

The ability to reproduce scientific results within the field of machine learning has been a hot topic in recent years with major conferences such as NeurIPS announcing dedicated reproducibility programs [?]. Certain fields of ML research such as within healthcare is reportedly in a *reproducibility crisis* [?, ?]. Surveys conducted

in 2018 reported that only 63% of 255 ML publications could be reproduced and only 4% of them could be reproduced without the original authors assistance [?]. Another survey in Nature sent out to more than 1500 scientists from various diciplines in 2016 reported that 50% researchers failed in reproducing even their own experiments [?]. Also Makridakis et.al., the organizers of the M-competitions, urged researchers to improve the reproducibility of their work. The following is a quote from their paper summarizing the M4 competition: *"We believe that there is an urgent need for the results of important ML studies claiming superior forecasting performance to be replicated/reproduced, and therefore call for such studies to make their data and forecasting algorithms publically available, which is not currently the case"* [?].

There are multiple methods of defining reproducibility. Pineau et.al. defined reproducibility as being able as *re-doing an experiment using the same data and same analytical tools* [?]. This definition aligns with that of Beam et.al. however, they point out that reproducibility is not concerned with the validity of the claims of a paper, only that practical results can be recreated [?]. McDermott et.al. further specifies the term reproducibility into three parts *technical*, *statistical* and *conceptual*. For technical reproducibility, the code and datasets used should be made publically available and in order to be statistically reproducible, the variance of the results should be published. To be conceptually reprodicble, it was important that multiple datasets should be used for the comparisons [?].

In addition to the requirement that code and datasets should be made available, Pineau et.al. suggested that standardized tools for supporting reproducibility should be developed and that any metrics used should be sufficiently specified. Furthermore it was argued that encapsulation tools such as Docker should be used as these would remove OS and dependency related issues [?]. This would also resolve issues pertaining to different versions of code being used [?]. The random seed which was discussed in section 2.4.1 is also an important value which should be reported as it can drastically affect the performance, especially for neural network based models [?, ?]. Increasing the use of statistical tests is another key aspect for improving the reproducibility [?]. In addition to this forecasting models should be executed multiple times with as much variation as possible between each run to further improve the accuracy and the statistical reproducibility of the results [?].

A summary of the requirements for achieving reproducibility is presented in table 2.

2.6 Gluon-TS

Gluon-TS is an open source library which includes several deep learning algorithms, datasets and tools for building and evaluating deep learning forecasting methods

Type of reproducibility	Requirements
Technical	Code and datasets should be publically available. If random seed is set it should be reported. Specific versions of code should be detailed and assets such as Docker containers provided.
Statistical	Variance of results should be published and statistical tests should be used.
Conceptual	Multiple complimentary datasets should be used for the comparison.

Table 2: Requirements for making ML workloads reproducible.

[?, ?, ?]. In GluonTS, the *Estimator* class corresponds to the implementation of a forecasting method. An *Estimator* can be trained on a dataset in order to generate a *Predictor* object. This *Predictor* can then ingest timeseries in order to produce forecasts for them.

In order to make it easy to evaluate the performance of an algorithm, GluonTS offers the *backtest_metrics* function. This function trains an *Estimator* on a train dataset, the generated *Predictor* is then used to generate predictions on a test dataset. Each of the timeseries in the test dataset will have the final N datapoints removed prior to passing them to the *Predictor*. The *Predictor* then takes the remaining time series data and produces a prediction of the same length N. Thereafter, the previously truncated datapoints are compared with the generated forecast in order to calculate error of the prediction from the ground truth [?]. There are multiple ways to calculate this error some of these are presented in detail in section 2.3.1.

GluonTS also contains Dockerfiles which makes it easy to create docker images with installations of GluonTS inside. These images are compatible with AWS SageMaker which allows them to be executed both in the cloud as well as locally. These images executes the *backtest_metrics* function of GluonTS when run and can thus be used to generate error metrics for any *Estimator* class in GluonTS [?].

2.6.1 Algorithms

GluonTS offers several forecasting models which can be used to generate predictions on timeseries data. Most of these algorithms are presented below. Only algorithms which were well documented in GluonTS or which I could explain by sifting through the source code are presented below.

CanonicalRNNEstimator

The CanonicalRNNEstimator is a bare bones RNN with a single layer of LSTM cells and is capable of producing probabilistic forecasts [**?**].

DeepAR

DeepAR is a RNN which produces probabilistic forecasts and was presented in *DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks* [**?**]. The model learns a global representation across all timeseries in a dataset and learns to identify seasonal behaviours in the data. DeepAR automatically adds certain meta information to the timeseries such as *day-of-the-week*, *hour-of-the-day*, *week-of-the-year* and *month-of-the-year*. Normally this type of meta information is manually added by the data scientist using the model, thus automating this procedure minimizes manual feature engineering. Another feature of DeepAR is the possibility to choose a likelihood distribution according to the data that it is to be trained on. Two different distributions are used by the original authors, the Gaussian Likelihood for real valued data and the negative binomial likelihood for postive count-data.

DeepFactorEstimator

The DeepFactorEstimator was presented in the paper *Deep Factors for Forecasting* in 2019 and consists of two main parts, a global and a local model [**?**]. The global model is a deep neural network (DNN) which is trained across all timeseries in order to capture complex non-linear patterns between the timeseries. The local model is a simpler model which is meant to capture local trends and patterns of individual timeseries. This hybrid architecture is thus able to leverage the DNN capability of learning complex patterns as well as having the computational efficiency of local models. Three different versions of the DeepFactorEstimator is presented in the original paper, the one implemented in GluonTS is the DF-RNN version. This version uses a second RNN to model the local timeseries. The DF-RNN presented in the paper performs better than Prophet and MQ-RNN on the Electricity, Taxi, Traffic and the Uber dataset both for short and long term forecasts. DeepAR performed worse than DF-RNN on average but was more accurate on the short term forecasts for the Uber dataset. The Uber dataset is not available as part of GluonTS however it is mentioned here for completeness, the other datasets are detailed in table 3.

DeepStateEstimator

The DeepStateEstimator combines linear state space models for individual timeseries with a jointly learned RNN which is taught how to set the parameters for the linear state space models. This has the benefit that the labour intensive tuning of multiple state space models is done automatically by the RNN without sacrificing performance. This approach was shown to perform better than DeepAR on the electricity dataset and on 4 out of 6 metrics on the traffic dataset. Furthermore, this approach outperformed ARIMA and ETS on all datasets [**?**].

GaussianProcessEstimator

The GaussianProcessEstimator is a local model where each timeseries in the dataset is modeled by a single gaussian process. The specific kernel used in the gaussian process can be specified as a parameter [**?**].

GPVAREstimator and DeepVAREstimator

The GPVAREstimator is a RNN model for handling multivariate timeseries which uses a gaussian copula technique for automatic data transformation and scaling. It utilizes a dimensionality reduction technique which minimizes the complexity of calculating a covariance matrix from $O(n^2)$ to $O(n)$. This allows much larger multivariate timeseries datasets to be used with it than what was previously possible. Additionally, the GPVAREstimator was compared against other multivariate forecasting algorithms, amongst them was a multivariate version of the DeepAR algorithm. This augmented version of DeepAR is implemented in GluonTS under the name DeepVAREstimator [**?**].

LSTNetEstimator

The LSTNetEstimator is a hybrid approach where long term patterns of the data is captured by a neural network architecture, these long term patterns are combined with a classical autoregressive model when generating predictions. The neural network architecture consists of four parts, a CNN, a RNN a novel RNN-skip layer and a fully connected layer. The RNN-skip layer makes up for the incapability of the LSTM cells in a RNN to remember very long term information by only updating the cells periodically [**?**].

NBEATSEstimator and NBEATSEnsembleEstimator

In the paper *N-BEATS: Neural basis expansion analysis for interpretable time series forecasting* [**?**] a univariate deep learning model for forecasting named N-

BEATS is presented. The authors of N-BEATS expressed that their goal with this algorithm was to disprove the notion that deep learning models had inferior performance compared to classical models. The authors reported a superior accuracy of N-BEATS model over all other models it was compared to.

The N-BEATS algorithm is an ensemble of 180 neural networks which were all trained to optimize for different metrics such as MASE, sMASE, MAPE and six different context lengths. Furthermore, bagging was used by running multiple runs of the algorithms with random initializations of the networks. A prediction of the N-BEATS model is the median value of the predictions of the algorithms in the ensemble.

Each model in the ensemble consists of multiple fully connected layers chained together to form blocks of networks. Each block is in turn chained together in order to form a stack. These stacks are also chained in order to form the final network.

GluonTS implements the N-BEATS model as the NBEATSEnsambleEstimator and the smaller algorithms in the ensemble as the NBEATSEstimator class. There are some differences between the implementation in the original paper and in GluonTS. Specifically, the training data is sampled differently in GluonTS than how it was in the source paper [**?**].

Naive2Predictor

the Naive2Predictor is a GluonTS implementation of the Naïve 2 forecasting method used as a reference in the m4 competition [**?**]. The Naive2Predictor predicts the future values to be that of the last datapoint in the timeseries adjusted by some seasonality. In GluonTS this seasonality can be deduced from the frequency of the data or by passing a custom seasonality via the *season_length* parameter [**?**].

NPTSPredictor

The NPTSPredictor is not a NN instead it predicts future values by sampling from previous data in the timeseries. The way that the samples are selected from the previous data can be modified via the hyperparameters. One can sample uniformly across all previous values in the timeseries or bias the sampling to more often sample from more recent datapoints depending on which kernel is used [**?**].

ProphetPredictor

Prophet is a nonlinear regression model developed at Facebook which frames the timeseries forecasting problem as a curve fitting exercise [**?**]. This is different from many other forecasting models for timeseries which leverage the temporal

aspect of timeseries in order to generate forecasts. Prophet was created with three goals in mind; it should be easy to use for people without much knowledge about timeseries methods, it should work for many different forecasting tasks which may have distinct features. Finally it should contain logic which makes it easy to identify how good the generated forecasts of Prophet are [**?**].

RForecastPredictor

The RForecastPredictor is a wrapper which allows a user of GluonTS to use the popular forecasting package *forecast* inside of GluonTS. The forecaster to use inside of the R package can be selected by passing the name of the method as a hyperparameter [**?**, **?**].

SeasonalNaivePredictor

The SeasonalNaivePredictor is a naive model which predicts the future value to be the same as the value of the previous season. This is a very simple model however an example explains its functionality best. If I want to use the SeasonalNaivePredictor to forecast the average temperature of June. The SeasonalNaivePredictor would return the average temperature for last year in June. In GluonTS, if not enough data exists (i.e. no data for last year in June) the mean of all the data in the timeseries is returned [**?**, **?**]

MQCNNEstimator, MQRNNEstimator

The GluonTS seq2seq package contains two forecasting algorithms, MQ-CNN and MQ-RNN. These are based on the MQ framework described in *A Multi-Horizon Quantile Recurrent Forecaster* [**?**]. The MQ framework is based on the sequence too sequence (Seq2Seq) architecture which consists of an encoder and a decoder network [**?**]. A Seq2Seq architecture encodes the training data into a hidden state which the decoder network then decompresses. Normally in Seq2Seq architectures, the RNN models tend to accumulate errors as the forecasts of an RNN for a timepoint t will be reused in order to generate a forecast for time $t+1$. By instead training the model to generate multiple point forecast for each timepoint in the horizon one wishes to forecast on, the errors tend to grow smaller. This is called *Direct Multi-Horizon Forecasting* and it is one of the changes introduced as part of the MQ framework. Further, the MQ framework allows for different encoders to used. Two of these are implemented in GluonTS, one with a CNN and one with a RNN. These are known as the the MQCNNEstimator and the MQRNNEstimator respectively [**?**].

SimpleFeedForwardEstimator

The SimpleFeedForwardEstimator in GluonTS is a traditional Multi Layer Perceptron (MLP) capable of generating probabilistic forecasts. The size of the network can be adjusted based on user parameters. Per default it has two densely connected layers containing 40 nodes in the input layer and 40 times the desired prediction length as the number of cells in the hidden layer.

The network has an additional layer which allows it to generate probability based predictions instead of only point predictions. This layer consists of a number of sublayers equal to the desired prediction length. Each of these layers consists of 40 nodes [**?**].

TransformerEstimator

The transformer is a Seq2Seq model which replaces the classical RNN encoder and decoders in a Seq2Seq model with more easily parallelizable NN components. A RNN requires data to be passed sequentially in order to learn dependencies between datapoints. This introduces a bottleneck and makes RNNs harder to train efficiently on modern hardware accelerators such as GPUs. The Transformer architecture presented in *Attention is All you Need* [**?**] alliviates this by leveraging parallelizable architectures such as feed forward networks as well as making heavy use of attention. Attention means that the architecture automatically can identify which parts of the input data is most relevant to the value we are trying to predict[**?**]. I.e. for the next value we are predicting, the most relevant previous values may be any or all of the previous n observations. In GluonTS the Transformer is implemented under the name TransformerEstimator [**?**].

2.6.2 Datasets

GluonTS offers 17 datasets ready to be used for training and evaluation, in table 3 an overview of these are presented. Of the available datasets 7 of them were first used in the M3, M4 and the M5 competitions [**?**, **?**, **?**]. The remaining datasets has been used in multiple research papers with various amounts of preprocessing applied to them [**?**, **?**, **?**, **?**, **?**, **?**].

Name	Freq	Description
Electricity	Hourly	Hourly electricity consumption of 370 clients sampled between 2011 - 2014. The original dataset on UCL was sampled each 15 minutes whilst the dataset in GluonTS had the data resampled into hourly series [?, ?].
Exchange Rate	B	Daily currency exchange rates of eight countries; Australia, Britain, Canada, China, Japan, New Zealand, Singapore and Switzerland between 1990 and 2016 [?].
Traffic	H	This dataset contains the hourly occupancy rate of 963 car lanes of the San Francisco Bay are freeways [?].
Solar Energy	10 Min	The solar power production records in the year of 2006, sampled every 10 minutes from 137 solar energy plants in Alabama [?].
Electricity NIPS	H	The Electricity dataset with additional processing [?].
Exchange Rate NIPS	B	The Exchange Rate dataset with additional processing [?].
Solar Energy NIPS	H	The Solar Energy dataset with additional processing [?].
Traffic NIPS	H	The Traffic dataset with additional processing [?].
Wiki Rolling NIPS	D	The Wiki dataset contains the amount of daily views for 2000 pages on Wikipedia [?].
Taxi	30 Min	Number of taxi rides taken on 1214 locations in New York city every 30 minutes in the month of January 2015. The test set is sampled on January 2016 [?].
M4 Hourly	H	Hourly timeseries used in the M4 competition randomly sampled from the ForeDeCk database [?].
M4 Daily	D	Daily timeseries used in the M4 competition randomly sampled from the ForeDeCk database [?].
M4 Weekly	W	Weekly timeseries used in the M4 competition randomly sampled from the ForeDeCk database [?].
M4 Monthly	M	Monthly timeseries used in the M4 competition randomly sampled from the ForeDeCk database [?].
M4 Quarterly	3M	Quarterly timeseries used in the M4 competition randomly sampled from the ForeDeCk database [?].
M4 Yearly	Y	Yearly timeseries used in the M4 competition randomly sampled from the ForeDeCk database [?].
M5 Dataset	D	Daily Walmart sales for 3049 products across 10 stores [?, ?].

Table 3: Datasets available in GluonTS.

2.7 Runtool

As part of an internship I had at Amazon Web Services (AWS) I created a Python toolkit for creating and executing large scale machine learning experiments. This tool was named *the runtool* and is open sourced as part of the GluonTS sister repository gluon-ts-tools [**?**]. Originally the runtool was meant to be a core part of this thesis, however the runtool diverged from the thesis and is now represented here as a third party package.

The runtool works in three parts, first assets such as algorithms and datasets are defined in a YAML config file. This file is then loaded by the runtool from within a python script. Thereafter experiments can be generated via mathematical operators such as + and *. Addition groups algorithms or datasets together into sets and multiplication generates an experiment of the cartesian product of the two sets being multiplied. Finally the runtool starts the generated experiments as training jobs in SageMaker. SageMaker is a cloud service enabling machine learning workloads such as model training and inference to be run on dedicated hardware.

Executing a few training jobs is not especially hard via a simple Python script as there are many powerful libraries available such as GluonTS. However scheduling and dispatching hundreds or thousands of different training jobs in parallel can quickly become complex. This is further complicated when multiple algorithms with different hyperparameter configurations should be evaluated on multiple different datasets. The runtool simplifies these things via so called $ operators within the config file. These operators enable, for example, inheritance between algorithms or datasets and dynamic updates of values in the config based on what the current experiment is.

In order to execute training jobs on SageMaker via the runtool there are certain requirements:

- The ML model needs to be in a SageMaker compliant docker container.
- The dataset has to be in an AWS S3 bucket.
- An IAM role granting the runtool access to the dataset and the docker image.

A key benefit of the runtool is its use of config files. Through these, it is possible to rerun any experiment with the exact same configuration as long as one has access to the docker container, the dataset and the config file. Due to that the dependencies are built into the docker image any experiment is fully rerunnable with minimum configuration.

The part of the Runtool which is responsible for starting training jobs is called the jobs dispatcher. While the builtin job dispatcher starts training jobs on SageMaker the runtool is built to make it easy to implement multiple backends. Thus

it is possible to extend the runtool such that training jobs can be executed on a local machine instead of on SageMaker.

In figure 13 a simple config file is displayed which defines two different algorithms and two datasets. This config is then used in the script presented in figure 12 to create four experiments where both algorithm are executed on both datasets.

```
import boto3
import runtool

# load config file
config = runtool.load_config("config.yml")

# create an experiment
my_experiment = (
    config.myalgo + config.anotherAlgo
    ) * (config.electricity + config.traffic)

# initialize runtool
tool = runtool.Client(
    role="arn:aws:iam::012345678901:role/my_role",
    bucket="my_bucket",
    session=boto3.Session(),
)

# dispatch the jobs
tool.run(my_experiment)  # blocking call
```

Figure 12: Python script using the config from figure 13 to create four experiments

```
electricity:
    meta:
        freq: 1H
        prediction_length: 24
    path:
        test: file:///path/to/dataset1/train.json
        train: file:///path/to/dataset1/test.json
traffic:
    meta:
        freq: 1H
        prediction_length: 24
    path:
        test: file:///path/to/dataset2/train.json
        train: file:///path/to/dataset2/test.json

base_algo:
    hyperparameters:
        epochs: 10
    instance: local
    metrics:
        MASE: 'MASE\): (\d+\.\d+)'
        abs_error: 'abs_error\): (\d+\.\d+)'
algo1:
    $from: base_algo
    image: image_with_algo1

algo2:
    $from: base_algo
    image: image_with_algo2
```

Figure 13: Config file describing two algorithms and two datasets. $from inherits the values from another node.

2.8 Hypothesis testing

Distributions occur in many areas within nature and science and being able to validate whether different samples come from the same underlying distribution is a long studied problem. This is known as hypothesis testing and several methods have been developed and one can group these tests into two major families, parametric and non-parametric tests [**?**]. Parametric tests are suitable whenever the type of the underlying distribution is known, otherwise non-parametric tests are more suitable.

The Student's T-test is a commonly used hypothesis test which compares the mean values of two distributions and only works if the samples are independent, normal and they have equal variance [**?**]. The requirement that the samples should have equal variance can make the T-test unsuitable in practice. However, an alternative to the Student's T-test is the Welch's T-test which allows for different variance between the two samples being tested. The Welch's test has however been shown to be unreliable if the two samples being tested have a considerable difference in size and variance [**?**].

A well known non-parametric test is the Diebold-Mariano (DM) two sample test which was designed to determine whether two samples of forecasts were statistically different [**?**]. Hassani et.al. showed that the DM test is outperformed by another non-parametric test, the Komogorov-Smirnov two sample test [**?**].

The Kolmogorov-Smirnov (KS) two sample test is a distance test based on the empirical cumulative distribution function (CDF) of the samples [**?**]. In essence, it tests if the distance between the two CDFs is too large, if they are to far apart, the test fails.

3 Approach

The purpose of this thesis is to perform an empirical comparison of forecasting methods. In order to do this, suitable methods, assets and systems need to be identified and evaluated. This chapter identifies the requirements these need to fulfill in order to be suitable for use in a benchmarking system. Further this chapter identifies assets and methods which need to be investigated further in a small experimental study to determine their suitability. In chapter 4 the results of this small experimental study is presented. The concepts discussed in this chapter are then combined into a benchmarking toolkit for forecasting methods which is presented in chapter 5. This system is then used for the empirical comparison in chapter 9.

3.1 Defining reproducibility

In section 2.5.1 the term reproducibility was split into three categories; *technical*, *statistical* and *conceptual*. A benchmarking toolkit should adhere to the requirements posed by these rules in order to be reproducible, thus following requirements need to be met:

In addition to these points, automating the benchmark would lead to a higher degree of reproducibility as the benchmarks would be standardized and thus less error prone. However, certain aspects of a benchmark, such as the error metric, are not suitable to be standardized since which error metric should be used depends on the use case.

3.2 Defining fair comparisons

The adjective Fair is defined as *acceptable and appropriate in a particular situation.* in the Oxford Dictionary. When benchmarking forecasting models, one method of achieving fair comparisons is to evaluate algorithms on multiple complementary datasets as this showcases the strengths and weaknesses of algorithms in different scenarios. Additionally, fair comparisons also require that the variance is shown as this provides more detail of an algorithms overall performance than a single value [**?**].

1. Encapsulation tools should be used for dependency and algorithm management.

2. Experiments should run multiple times in order to build up a distribution of errors such that the variance of the error is recorded.

3. Statistical tests should be applied to the results to enforce statistical reproducibility.

4. Multiple datasets should be used when benchmarking to cover various use-cases for conceptual reproducibility.

Figure 14: Requirements for a reproducible benchmark.

Simply executing an algorithm on multiple datasets is not enough as it is important to give each algorithm a fair opportunity to perform well on each dataset. This is normally achieved by *tuning* the hyperparameters of an algorithm for the dataset in question. A fair way of tuning algorithms is however non trivial as different strategies tend to be biased [**?**]. One method of achieving fairness could be to limit the time spent for hyperparameter tuning to some constant value. This would however be unfair towards slower methods such as DNNs or RNNs as the search space traversed would be smaller when compared to faster models such as simple NNs or classical models. Another approach is to limit the size of the search space by limiting the number of hyperparameter configurations to be tested. This would benefit models with few hyperparameters as each hyperparameter would be more thoroughly tuned than if more hyperparameters would be available. The inverse of this would also hold, one could limit the number of configurations for each hyperparameter which is tested but let the total number of hyperparameter configurations to be unbound. This would however be unfair towards models with few parameters.

Another approach would be to tune each algorithm such that it achieves its *optimal* performance for each dataset. This would be fair in the sense that it is independent of the complexity of the algorithm being tested. For ML competitions this is commonly the case as contestants tune their algorithms to perform optimally in order to win [**?**]. This is also the case when algorithms are used in practice as companies and organizations leveraging machine learning spend considerable effort in tuning these to fit their data [**?**].

To enable fair comparisons the implementations of the models being compared should be bug free and optimized. This is especially important for very complex models as the engineering effort of implementing such algorithms correctly is

considerable.

A summary of the requirements for fair comparisons which have been identified is presented in 15

1. Optimally tuned algorithms for each dataset
2. Variance should be reported
3. Models should be evaluated on multiple complementary datasets
4. Reference implementations of tested forecasting models should be used when possible

Figure 15: Requirements for a fair benchmark.

3.3 Defining accurate comparisons

In section 2.3.1 different metrics were presented for comparing forecasting accuracy. It was established that error metrics have different strengths and weaknesses which makes different metrics suitable for different domains or applications. Thus, in order to perform an accurate comparison of forecasting methods via a benchmarking system, multiple different metrics need to be made available.

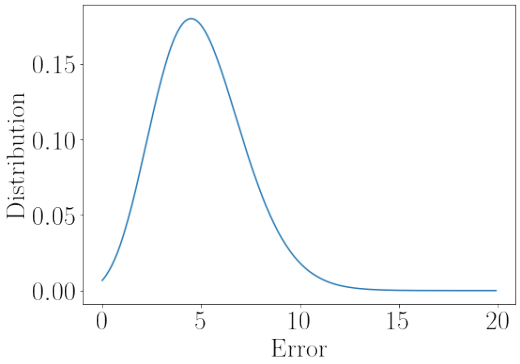

Figure 16: Poisson distribution

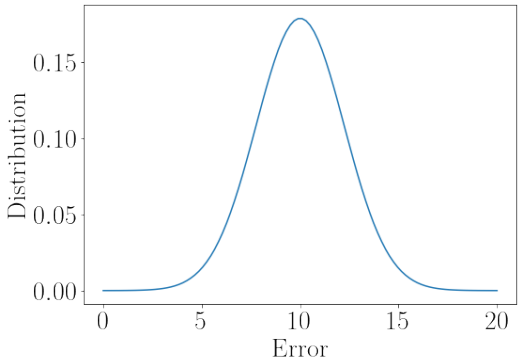

Figure 17: Gaussian distribution

Figure 18: Which error distribution is better?

Since, for reproducibility purposes, each model shall generate a distribution of error metrics one cannot any longer compare algorithms based on single metric

values. In figure 18 two examples of possible error distributions are presented. Since a smaller error is generally considered better, it makes sense that distributions which are concentrated around zero perform the best. For example in Figure 18 an algorithm generating an error distribution such as the poisson is more likely to produce errors close to 0 than an algorithm with an error distribution like the normal distribution in Figure 17.

A summary of what is required to perform accurate comparisons is presented in 19.

1. Suitable error metrics should be used for the domain and datasets.
2. The distribution of the error metric over multiple runs is used when comparing different algorithms.

Figure 19: Requirements for accurate comparisons

4 Designing a benchmarking system

While the requirements for a fair, accurate and reproducible comparison detailed in Chapter 3 is not an exhaustive list, they form a bare minimum of what should be expected from a benchmarking system for forecasting models. Some questions require to be investigated further however; Which encapsulation system should be used, what is a suitable hypothesis test for statistical reproducibility, how should distributions of errors be compared and how should a representable subset of datasets be chosen. At the end of this chapter an example architecture for a benchmarking system fulfilling these requirements is presented.

4.1 Design considerations

Due to the popularity and its native capability to encapsulate code and dependencies Docker containers are chosen as the encapsulation tool for this project. An advantage with docker containers is that popular tools such as Github, AWS, Azure and Dockerhub allows for storage and sharing of Docker images which makes it trivial for researchers to distribute their algorithms in a ready to use package.

Docker containers are however not sufficient for complete technical reproducibility as arguments such as hyperparameters or the number of CPU cores should be used can be passed to the containers when training them. If all arguments are not supplied when presenting these algorithms, technical reproducibility becomes hard to achieve. The Runtool described in Section 2.7 simplifies these things as the config file used when running the experiment can be used by a third party to rerun exactly the same experiment as long as the Docker image and dataset is available.

Reference implementations of complex algorithms are required for fair comparisons, GluonTS which was presented in Section 2.6 offers several algorithms ranging from simple naive solutions to highly complex DNN hybrid models. Furthermore, GluonTS offers ready to use Dockerfiles for both CPU and GPU execution of these algorithms. An additional benefit of leveraging GluonTS is that multiple datasets are available for training and testing algorithms on and that multiple error metrics are calculated as part of their backtesting system.

As per the discussion in Section 3.2 a fair and conceptually reproducible comparison should make use of multiple datasets with different complementary character-

Requirement	Accurate	Fair	Technically reproducible	Statistically reproducible	Conceptually reproducible
Multiple error metrics	X				
Compare distributions of error metrics	X	X	X	X	X
Optimally tuned models		X	X		
Multiple datasets used		X			X
Public datasets used			X		X
Reference implementations of ML models		X	X		
Use encapsulation tools		X	X		
Use statistical tests			X		
Optimally tuned models				X	

Figure 20: Requirements of a benchmarking system.

istics. GluonTS offers several datasets used in research papers and competitions, and an analysis of these needs to be performed to identify a representable subset. Such an analysis is performed in Section 4.5

In order to capture distributions of error metrics, the seed of the encapsulated algorithms should not be set from within the Docker image otherwise rule 2 of figure 14 would be violated as each run would be deterministic. Furthermore a suitable statistical test needs to be chosen to enforce rule 3. Three hypothesis tests for comparing distributions for reproducibility purposes were discussed in Section 2.8. However the practical performance of such tests when applied to real world distributions of error metrics need to be examined. Particularly, the amount of data needed for them to be accurate needs to be determined so that suitably large distributions of error metrics are collected when benchmarking. In Section 4.6 such an analysis is performed.

It is normal for benchmarks and competitions to generate tables where different forecasting models are compared using real valued numbers to describe their performance. Normally this number is an error metric [**?**, **?**, **?**, **?**, **?**, **?**]. Similarly, it would be suitable if a real number could be used to summarize the distribution of error metrics. The conversion from a distribution to a single number is unlikely to be lossless, however it is important that the conversion should accurately represent the distribution by benefiting errors close to 0 and punishing errors further away. Such a conversion should not be limited by the error metric used in the

distribution but be applicable to any error metric as per the requirements in Table 20.

Tuning of algorithms is a requirement for fair comparisons, thus the possibility of hyperparameter tuning should also be available as part of the benchmarking suite. Since many ML models exhibit non-deterministic behaviour, each hyperparameter configuration being tested should be executed multiple times and aggregated, otherwise rule 2 from 14 would be violated.

4.2 Proposed benchmarking system

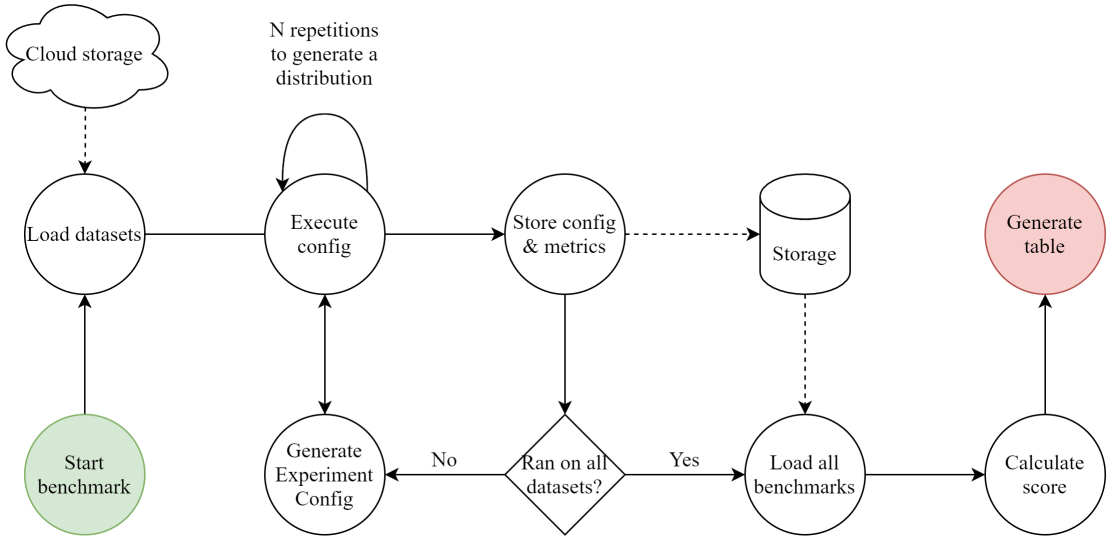

Figure 21: Proposed benchmarking system.

In figure 21 an overview of the proposed benchmarking system is displayed. Here the benchmark starts by a user providing the runtool configuration file containing the algorithm to use, the name of the image to use and other data such as which hyperparameters to pass to the algorithm. The system then downloads the datasets identified in 4.5 to the local machine for later use.

The benchmarking loop is as follows: the algorithm configuration file is first loaded into the runtool and an experiment is generated where the algorithm is executed on each of the datasets N times. The value of N is determined in Section 4.6. Each time the algorithm is executed, error metrics are reported by the GluonTS Backtesting functionality and these logs are captured by the Runtool and stored to a file or database for long term storage. This file will be referred to as the *Benchmark Result File* (BRF) for the remainder of the thesis. Along with

the distribution of error metrics, the config file used should be stored in the BRF to make it possible to rerun each benchmark.

The latest benchmarks error distribution is then aggregated through some scoring method which summarizes the distribution as a single value. This is then repeated for each previous benchmark which has been run. The resulting scores are then used to rank the algorithms against each other. A table is then presented to the user showing the relative rank of the latest benchmark compared to the previous benchmarks.

4.3 Reproducing benchmarks

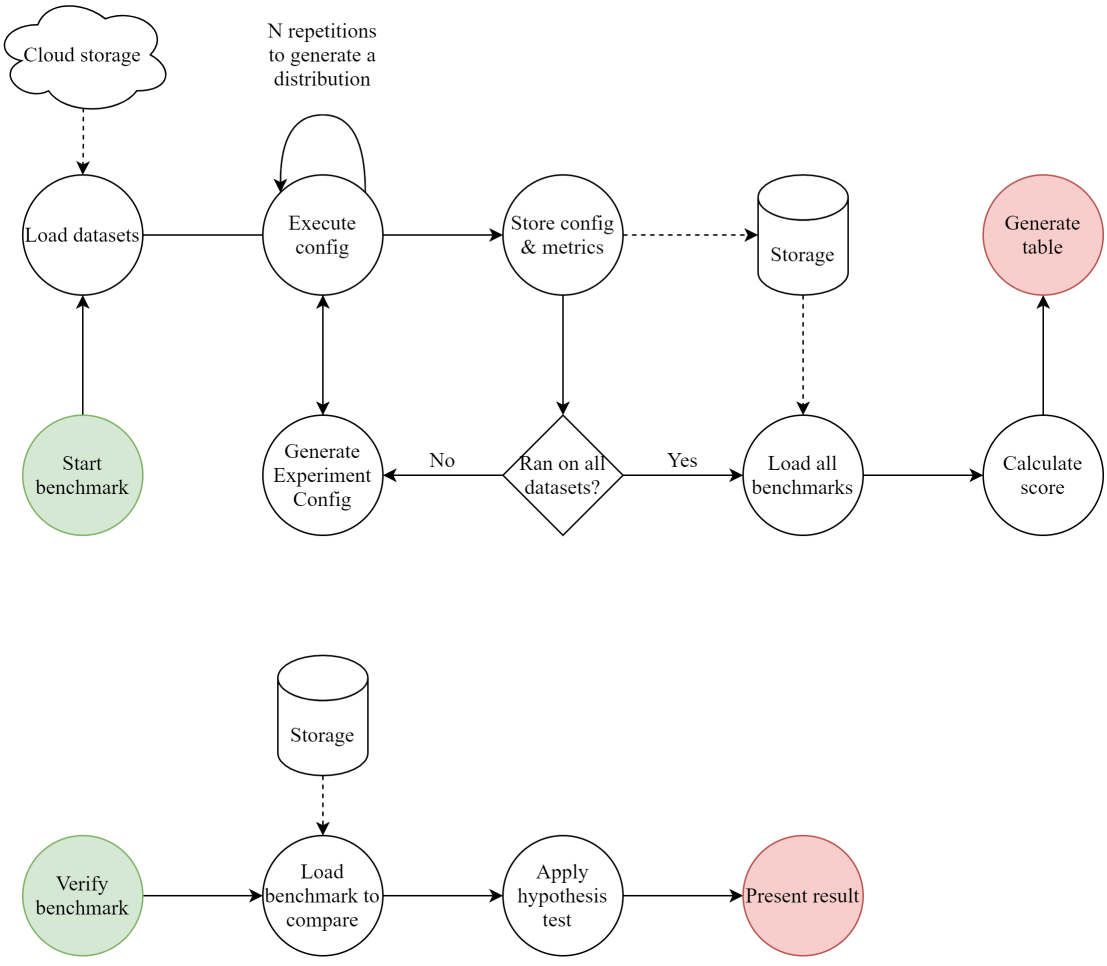

Figure 22: Proposed system for verifying benchmarks.

In order to both technically and statistically be able to reproduce a benchmark a

user would need access to the image used and the BRF created by the benchmarking system. Provided these, the verification system could rerun the benchmark using the config file stored in the BRF and the provided image. Rerunning a benchmark with the same setup should result in a similar distribution being generated by the algorithm. Statistically speaking, if the algorithm in the image would be the same as the reference algorithm, both error metric distributions would be sampled from the same underlying distribution. The new distribution can then be compared with that of the benchmark stored in the BRF through a suitable hypothesis test.

4.4 Hyperparameter tuning

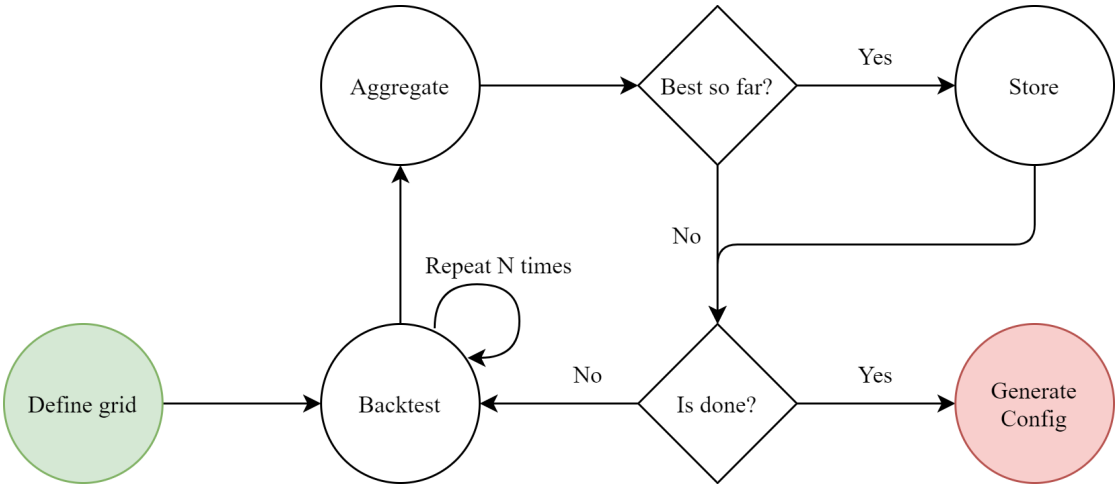

Figure 23: Proposed system for verifying benchmarks.

Hyperparameter tuning is a non-trivial task in itself and many advanced solutions exist such as Bayesian hyperparameter search, hyperband search or advanced meta learning approaches [**?**, **?**, **?**]. The hyperparameter tuning approach proposed here is not intended to supersede these. Instead it focuses on taking the non deterministic output of each run into account when tuning.

The proposed algorithm is in essence a simple grid search tuning algorithm with one major difference. Each configuration in the tuning loop is executed multiple times and then aggregated before being evaluated. Common ML frameworks which offer grid search such as SageMaker or SciKit-Learn (sklearn) surprisingly lack the capability to backtest each configuration multiple times [**?**, **?**]. While sklearn does offer a cross validation version of its grid search implementation, *GridSearchCV*, this splits the dataset into multiple parts for each iteration. Thus it does not serve

the same purpose as performing the backtest multiple times and aggregating the results.

Grid search was chosen as the tuning approach for this system as it is conceptually simple and it is a well known method. It does however have the downside that many unfavorable hyperparameter configurations are evaluated, something which e.g. bayesian search avoids [**?**].

An issue with the suggested tuning architecture is that time for tuning increases linearly with the number of repeated runs. Due to this, it may be suitable in real life scenarios to apply this approach as a second step after a set of promising hyperparameter configurations has been identified by another more time efficient approach.

4.5 Dataset Analysis

It is common that new forecasting methods are compared on several datasets as to showcase their predictive power in different scenarios. Different datasets exhibit different characteristics such as trend and seasonality. By identifying a representable set of datasets with complementary characteristics, one can evaluate how robust a forecasting algorithm is to different datasets. Choosing a representable subset of datasets is also needed as the time to train an algorithm increases with the amount of datasets that it should be trained on. For a empirical comparison such as this, it is unfeasible to run training and tuning jobs for all algorithms on all datasets available in GluonTS. In the remainder of this section an analysis of the datasets available in GluonTS is performed in order to identify a representable subset of them with diverse characteristics.

4.5.1 Methodology

For each dataset in GluonTS plots of the average time series along with one standard deviation from it is generated. This is done to get an overview of what the datasets looks like. Plotting timeseries for this purpose is common practice in time series forecasting, or as Hyndaman et.al so eloquently put in Forecasting: Principles and practice: *"The first thing to do in any data analysis task is to plot the data."*.[**?**]

Another common tool for visualising timeseries is to generate a histogram of the timeseries. However as the datasets available in GluonTS contains tens of thousands of timeseries, plotting each of these individually is unfeasible. Instead the average timeseries and one standard deviation of its values is plotted. Another common method for visualizing single timeseries is to generate a histogram of its values [**?**]. Instead of generating such histograms violin plots are used in as these

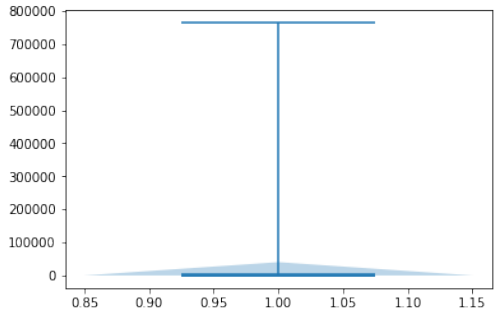

Figure 24: Unscaled violin plot of the electricity dataset

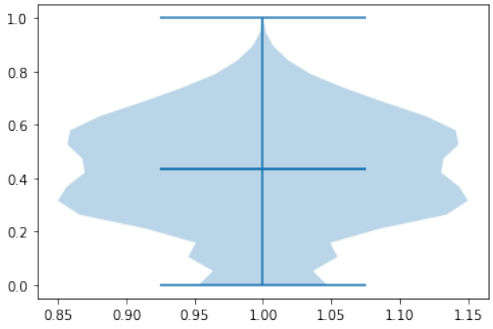

Figure 25: Scaled violin plot of the electricity dataset where all timeseries have been scaled by their maximum value.

capture both the distribution of values just as histograms do as well as the median value and the interquartile range. Since each timeseries is on its own scale, the aggregate violin plots are hard to read. By scaling each timeseries in the dataset by the maximum value of that timeseries before plotting it the violin plot becomes more easy to interpret.

In addition to the visual tools the following statistics is calculated for each of the datasets in GluonTS:

- Mean value
- Max value
- Min value
- Number of timeseries
- Number of datapoints
- Length of shortest timeseries
- Length of longest timeseries
- Strength of the trend (mean and standard deviation)
- Strength of the seasonality (mean and standard deviation)

Simple metrics such as the maximum, mean and minimum values for are useful to get a high level overview of the dataset. Further, simpler statistics such as these

can identify possible issues which can surface further down the line. For example, the MAPE metric is unstable for timeseries close to zero as a division with zero occurs [**?**].

The number of timeseries are important for global models as these then have more data which they can learn across which help to combat overfitting. The number of timeseries is however not important for local models as these are only impacted by the quality and length of each individual time series. Longer timeseries however, benefit forecasting models which can handle longer horizons [**?**]. For these reasons, the number of timeseries and the lengths of the shortest and longest timeseries are necessary metrics when comparing datasets. Optimally one would have both long timeseries and many series as this would benefit both global and local models.

In chapter 2 it was shown that the strength of seasonality and the strength of the trend of timeseries is a useful metric for differentiating timeseries. Thus, calculating the strengths of the timeseries in each dataset provides valuable information about the composition within them.

Since larger datasets make models less prone to overfit, the largest dataset is chosen in case a tie where two datasets are exhibiting the same characteristics. A summary of the criterias on which the datasets are evaluated is shown in Table 26.

1. One dataset with high trend and low seasonality
2. One dataset with low trend and high seasonality
3. One dataset with low trend and low seasonality
4. One dataset with high trend and high seasonality
5. Lower variance of strengths is prefered
6. Larger dataset chosen in case of a tie

Figure 26: Criteria for identifying a representable subset of datasets.

Limitations

The M4 dataset does not significantly differ to the M3 dataset since except for its size [**?**]. due to this, the M3 dataset is not considered in this comparison. Another set of datasets which are not going to be used are the NIPS datasets. This is due to them having had some unclear postprocessing applied to them. As the exact post

processing is not known, comparing any findings when training on these datasets with other papers becomes hard and introduces uncertainty.

4.5.2 result

In order to calculate the strengths of the trend, seasonality a STL decomposition is needed for each timeseries in the dataset. This decomposistion is done using the STL method of the Statsmodel package in Python [**?**]. The strengths of each timeseries is then calculated using the formulas in Chapter 2. In order to summarize the strengths, the average and the standard deviation was then calculated for each dataset.

The complete plots and statistics are available in the appendix as they were to numerous to be displayed here. However a summary of the extracted statistics is presented in Table 4.

Dataset	Trend	Seasonality	Trend Dev.	Seasonality Dev.
Exchange Rate	1.00	0.12	0.00	0.30
M4 Daily	0.98	0.05	0.05	0.10
M4 Yearly	0.93	0.09	0.13	0.16
M4 Quarterly	0.90	0.20	0.16	0.27
M4 Monthly	0.84	0.32	0.32	0.30
M4 Weekly	0.77	0.31	0.31	0.35
Electricity	0.65	0.84	0.17	0.19
M4 Hourly	0.62	0.88	0.37	0.16
Wiki Rolling	0.53	0.23	0.27	0.26
M5	0.38	0.28	0.32	0.33
Traffic	0.16	0.67	0.12	0.10
Solar Energy	0.09	0.84	0.03	0.02
Taxi	0.02	0.66	0.02	0.08

Figure 27: Strength of trend & seasonality for datasets in GluonTS.

There is only one dataset in Table 27 which exhibits low trend and low seasonality and that is the M5 dataset. However the M5 dataset does show the most variance of all the datasets which implies that it contains many timeseries with and without trend and seasonality. Despite this, since it is the largest dataset available and that there are no other datasets which exhibit low average trend & seasonality, the M5 dataset is deemed appropriate for use in the benchmarking system.

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Freq.
Elec.	2510.68	321	6755124	21044	21044	0.0	764000.0	1H
Elec.	2509.92	2247	47501580	21068	21212	0.0	764000.0	1H
Exch.	0.68	8	48568	6071	6071	0.01	2.11	1B
Exch.	0.68	40	246440	6101	6221	0.01	2.11	1B
Solar	40.35	137	960233	7009	7009	0.0	509.05	10min
Solar	40.25	959	6813695	7033	7177	0.0	509.05	10min
Traf.	0.06	862	12099032	14036	14036	0.0	0.72	H
Traf.	0.06	6034	85272488	14060	14204	0.0	0.72	H
Exch.*	0.68	8	48568	6071	6071	0.01	2.11	B
Exch.*	0.68	40	246440	6101	6221	0.01	2.11	B
Elec.*	607.95	370	2142282	1081	5833	0.0	168100.0	H
Elec.*	652.36	2590	10340239	1105	4000	0.0	168100.0	H
Solar*	40.35	137	960233	7009	7009	0.00	509.05	H
Solar*	40.25	959	6813695	7033	7177	0.00	509.05	H
Traf.*	0.05	963	3852963	4001	4001	0.00	1.00	H
Traf.*	0.05	6741	26964000	4000	4000	0.00	1.00	H
Wiki	3720.54	9535	7551720	792	792	0.00	7752515.00	D
Wiki	3663.55	47675	40619100	792	912	0.00	7752515.00	D
Taxi	8.79	1214	1806432	1488	1488	0.0	265.0	30min
Taxi	7.41	67984	54999056	149	1469	0.0	225.0	30min
M4 H	6827.69	414	353500	700	960	10.00	703008.00	H
M4 H	6859.56	414	373372	748	1008	10.00	703008.00	H
M4 D	4951.40	4227	9964658	93	9919	15.00	352000.00	D
M4 D	4960.15	4227	10023836	107	9933	15.00	352000.00	D
M4 W	3738.52	359	366912	80	2597	104.69	51410.00	W
M4 W	3755.97	359	371579	93	2610	104.69	51410.00	W
M4 M	4193.28	48000	10382411	42	2794	20.00	132731.31	M
M4 M	4207.51	48000	11246411	60	2812	20.00	177950.00	M
M4 Q	4141.00	24000	2214108	16	866	19.50	82210.70	3M
M4 Q	4287.13	24000	2406108	24	874	19.50	82210.70	3M
M4 Y	3630.52	23000	715065	13	300	22.10	115642.00	12M
M4 Y	4076.24	23000	852909	19	300	22.00	158430.00	12M
M5	1.12	30490	57473650	1885	1885	0.00	763.00	D
M5	1.13	30490	58327370	1913	1913	0.00	763.00	D

Table 4: Statistics of the datasets, top row for each dataset is the train split, the bottom row is the test split.

In order to choose a dataset which fits into the category of high seasonality and low trend there are three options; Solar Energy, Taxi and Traffic. Of these the Solar Energy dataset has the lowest variance for both the trend and the seasonality. This in addition to having the second lowest average trend of the three and $\tilde{2}3$ % higher average seasonality than the Taxi and the Traffic datasets makes it most suitable according to the criteria in Figure 26.

The datasets with the lowest seasonality and the highest trends are the M4 datasets except for the M4 Hourly as well as the Exchange rate dataset. The one with the lowest variance and the highest trend of these is the Exchange Rate dataset closely followed by the M4 Daily dataset. Since the M4 Daily has lower variance for both strength and seasonality, and since the size of the Exchange Rate dataset is much smaller than the M4 Daily dataset, with only 8 timeseries and 48k datapoints in comparison to the 4227 series and 9.9M datapoints the M4 Daily dataset is more suited for use in the benchmarking system. Furthermore, all other M4 datasets exhibit a higher variance than the M4 Daily which enforces the M4 Daily to a better choice for this comparison.

When it comes to finding a dataset which exhibit both high trend and a high seasonality, there are only two options, the Electricity dataset and the M4 Hourly dataset. They both have a high variance, however the variance of the Electricity dataset is slightly lower than that of M4 Hourly. In addition, the Electricity dataset has almost twice the amount of datapoints as M4 Hourly. Thus the Electricity dataset is chosen as it fits the criteria better.

To summarize the four datasets which most closely fit the criteria defined in 26 are:

	Low trend	**High Trend**
Low seasonality	M5	M4 Daily
High Seasonality	Solar Energy	Electricity

Table 5: Datasets with complimentary strengths of trend and seasonality

4.6 Comparing tests for validating forecasting performance

This section investigates the performance of three hypothesis tests for use on distributions of error metrics. The investigated tests are the T-test, Welch's T-Test, the Kolmogorov-Smirnov test.

GluonTS catches accuracy regressions through checking whether new forecasts are within 1.645 standard deviations, i.e. within the 95th percentile of past values

- Which test functions best for limited data
- Is a parametric test suitable or is a non-parametric test needed
- What is a reasonable sample size
- Which test has the lowest false reject ratio for related distributions
- Is a naive standard deviation based test sufficient

Figure 28: Questions for identifying a suitable hypothesis test.

[**?**]. This approach is based on the assumption that the distribution of error metrics is normally distributed. If this is True then, this simpler approach may be more suitable than a hypothesis test for reproducibility. Thus this approach will be evaluated along with the hypothesis tests. The questions this section aims to answer are presented in Figure 28.

4.6.1 Methodology

Several of the questions in Figure 28 require distributions of error metrics to be collected. For this purpose, the DeepAREstimator presented in Section 2.6.1 is suitable as it is both quick to train and a non-trivial forecaster. The dataset chosen is the Electricity dataset since it is a medium to small sized dataset, thus shortening the required training time.

The Runtool is then used to create 900 experiments where the DeepAREstimator is executed on the Electricity dataset. 300 of these runs has the hyperparameter *distr_output* set to the default value, 300 runs are executed with *distr_output* set to use the negative binomial distribution. The remaining 300 runs are executed using the Poisson distribution.

After the distributions are collected, histograms of the distributions are created to identify whether the distributions are visually different. This is done since if all three are visually similar it is unlikely that they can be used for evaluating the hypothesis test on. Further, a visual analysis can determine whether the naive method and or the T-test is suitable since it requires normally distributed data.

After the visual analysis, the ratio of false rejects are calculated, i.e. how often the test is unable to detect that two samples are sampled from the same distribution. To answer this questions, let N be the total distribution and X, Y be the two samples drawn from N. The size of X is then defined by

$$k \leq |X| \leq |N| - |Y|$$

where k is a constant value. Similarly, the size of Y varies between:

$$k \leq |Y| \leq |N| - k$$

For each sample of X and Y the tests are applied and false negatives are recorded.

In order to evaluate a suitable minimum sample size, heatmaps are generated where the tests are applied to the three distributions with varying sample sizes. Each test is performed with a *Pvalue* of 0.05. If the test accepted the samples to be from the same distribution the color green is used while a rejection, is colored red.

4.6.2 Results

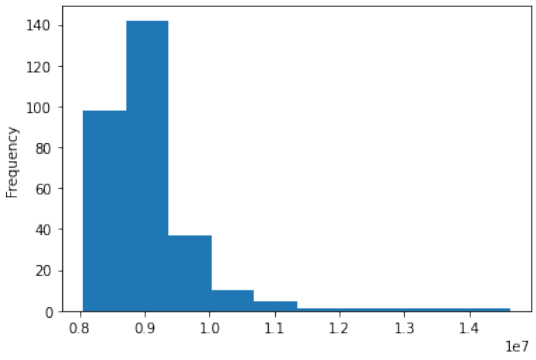

Figure 29: Student-T

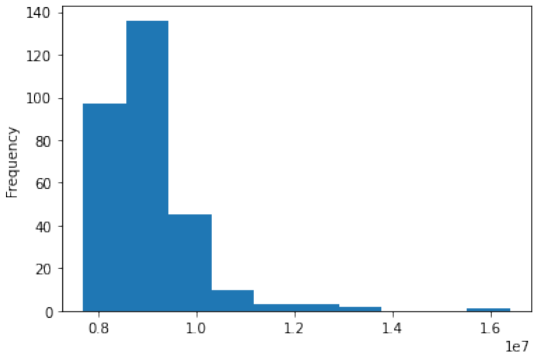

Figure 30: Neg-Binomial

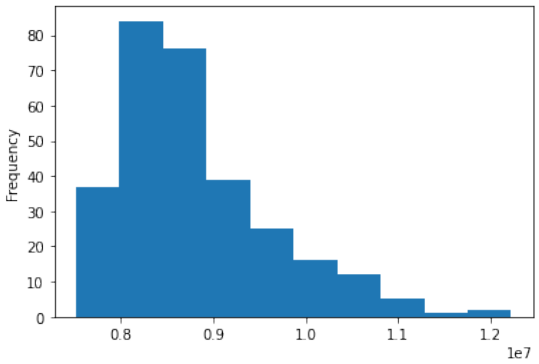

Figure 31: Poisson

Figure 32: Histograms of the absolute error for 300 runs of DeepAR on the Electricity dataset with three different values of the hyperparameter *distr_output*

The histograms shown in Figure 32 do not resemble a normal distribution. Instead all three histograms are more closely approximating a log normal distribution. Note that the histogram in Figure 29 and 30 are very similar while the one in Figure 32 differs. This indicates that different hyperparameter configurations for a single algorithm impacts the error distribution.

Name	%	**c**onfiguration
Naive	<1%	Negative Binomial
Naive	<1%	Student T
Naive	<1%	Poisson
T-Test	5%	Negative Binomial
T-Test	5%	Student T
T-Test	32%	Poisson
Welchs T-Test	11%	Negative Binomial
Welchs T-Test	8%	Student T
Welchs T-Test	31%	Poisson
Kolmogorv-Smirnov	3%	Negative Binomial
Kolmogorv-Smirnov	5%	Student T
Kolmogorv-Smirnov	21%	Poisson

Table 6: Number of times when the algorithm failed to recognize that the samples were from the same distribution

In Table 6 the amount of false rejections of the three hypothesis tests and the naive solution is presented. From this data it is clear that the Naive solution is the best performing with a 1% chance of failing to validate that the two samples are from the same distribution. The Kolmogorov Smirnov performs second best, however it is incorrect 21% of the time when evaluating on the distribution generated by the Poisson configuration. The two T-Tests are the worst performers, performing worse than or equal to the Kolmogorov-Smirnov test on all distributions.

Since the distributions of error metrics are not reminiscent of the normal distribution, the naive method, the T-test and the Welch's T-test which are parametric tests are unsuitable for use when verifying dataset distributions. The data in table 6 further enforces this since Kolmogorov-Smirnov is the second best performer for all distributions where only the naive method is better. In Figure 35 the naive method is evaluated on the Poisson and Negative Binomial distributions. It is clear from this heatmap that the naive method is prone to accept quite different distributions for all but small sample sizes. This is expected since the distributions investigated all have similar average values even though their distributions differ. In comparison, the Kolmogorov-Smirnov test was able to differentiate all

distributions given sufficient sample sizes, see figures 33, 34, 36.

The sample size required to differentiate between different distributions (true negatives) as well as the sample sizes required for true positives for the Kolmogorov Smirnov test is presented in Table 7. This table summarizes the heatmaps generated for the Kolmogorov-Smirnov test when applied to samples of varying sizes from the three distributions as described in 4.6.1. The heatmaps are available in Appendix 10.2

	Student T	Poisson	Negative Binomial
Student T	> 50		
Poisson	> 25	> 45	
Negative Binomial	> 160	> 60	< 130

Table 7: Required sample sizes for the Kolmogorov Smirnov test on three distributions of error metrics.

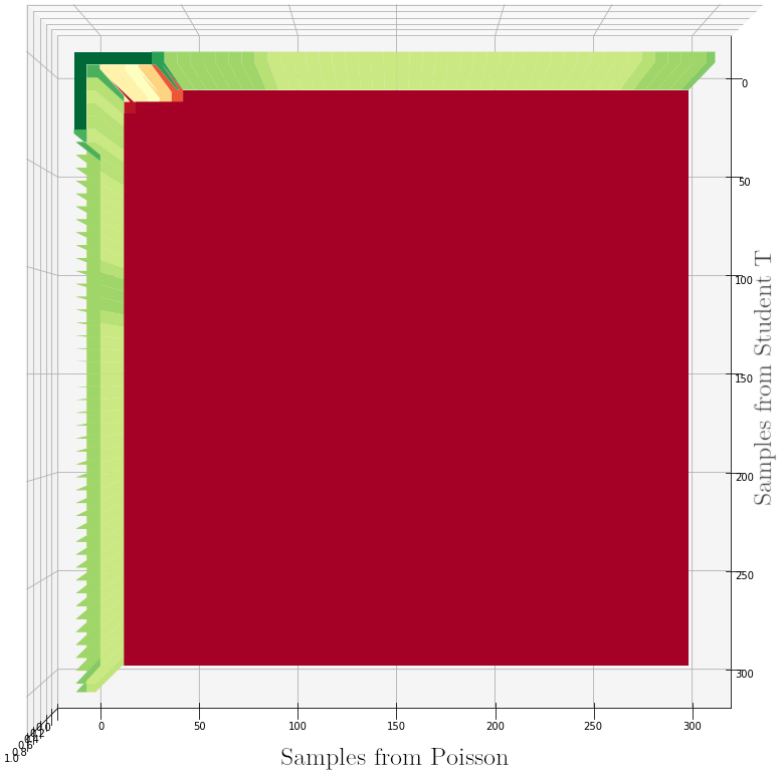

Figure 33: Heatmap of KS for Student-T and Poisson.

From Table 7 it is shown that the KS test becomes accurate for true positives at a sample size of at least 40 samples. However, it seems as if KS can become unstable for sample sizes above 130. To be able to use the KS statistic to distinguish between different distributions, sample sizes above 160 may be needed if the distributions are visually very similar, otherwise, sample sizes above 60 suffice. These values are aligned with the findings of the simulation study performed by Hassani et.al. **??**. There they showed the need for > 128 samples to achieve $> 99\%$ rejection rate for similarly looking distributions.

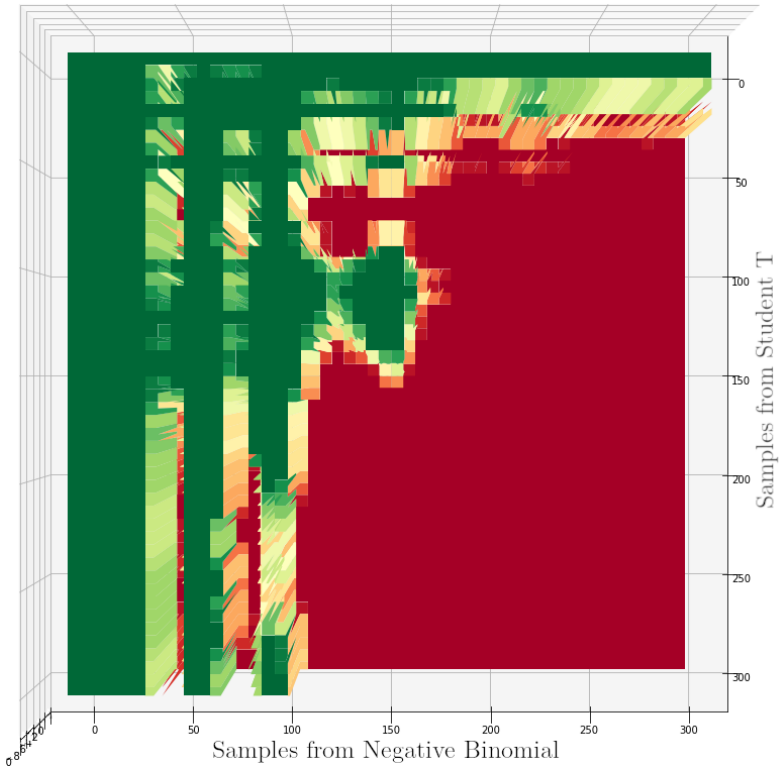

Figure 34: Heatmap of KS for Student-T and Negative Binomial.

To summarize, despite the limited data available for this comparison, the Naive test, the T-test and Welch's T-Test has been shown to be unsuitable for comparing distributions of time series forecasting errors due to possible non-normality of the data. When evaluating the Kolmogorov-Smirnov on samples from three different distributions it was shown that > 60 samples are required to verify distributions but more than 160 may be required to differentiate similar distributions.

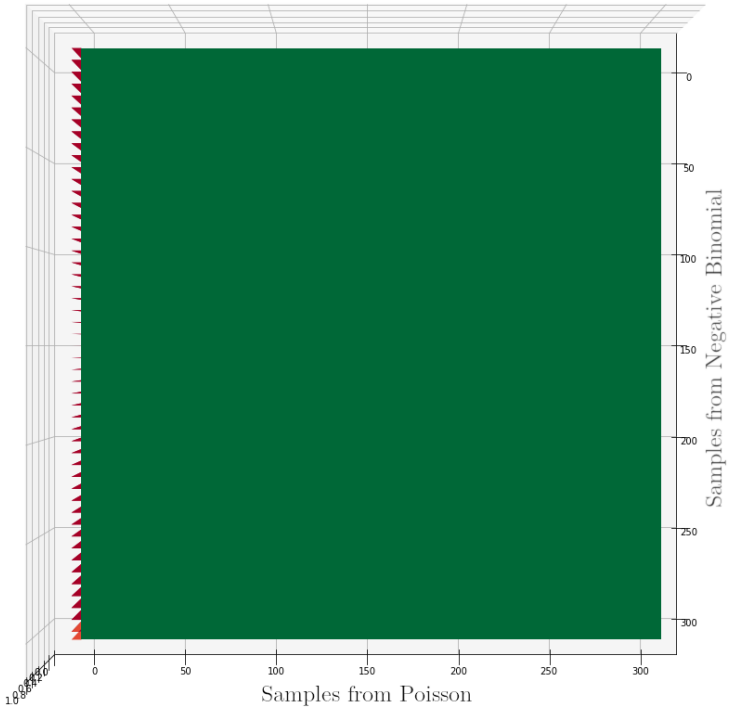

Figure 35: Heatmap of KS for Negative Binomial and Poisson.

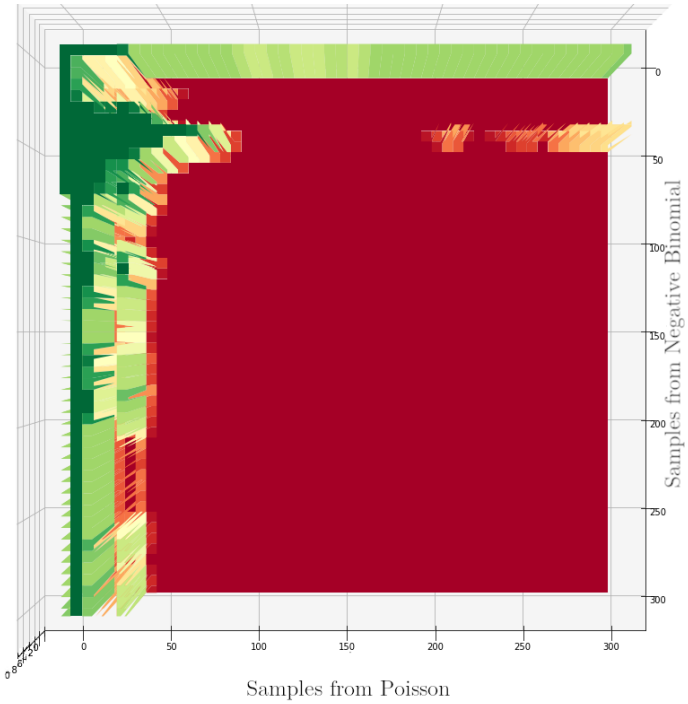

Figure 36: Heatmap of KS for Negative Binomial and Poisson.

5 Crayon

6 Empirical Comparison

7 Conclusion

... should include the following:

- problem restated and a brief summary of the methodology,
- student contributions (e.g., survey, open-source software, journal publication),
- a brief summary of the findings and results,
- limitations and generalizability of the findings and results.
- lessons learned,
- recommendations for future research.

8 Conclusion

... should include the following:

- problem restated and a brief summary of the methodology,
- student contributions (e.g., survey, open-source software, journal publication),
- a brief summary of the findings and results,
- limitations and generalizability of the findings and results.
- lessons learned,
- recommendations for future research.

9 Conclusion

... should include the following:

- problem restated and a brief summary of the methodology,
- student contributions (e.g., survey, open-source software, journal publication),
- a brief summary of the findings and results,
- limitations and generalizability of the findings and results.
- lessons learned,
- recommendations for future research.

10 Appendix

10.1 Example code

10.2 Heatmaps of hypothesis tests

This section contains heatmaps generated by applying hypthesis tests to samples of varying size sampled from a distribution of error metrics generated by the DeepAREstimator on the Electricity dataset. DeepAR was executed with three different values of the *distr_output* hyperparameter; NegativeBinomialOutput, StudentTOutput and PoissonOutput. The distributions generated by the different hyperparameter configurations are refered to as Negative Binomial, Student T and Poisson. See Section 4.6.1 for more information about how the distributions were generated.

In the heatmaps, green means the test approves the samples being from the same distribution otherwise it is red.

10.2.1 Kolmogorov Smirnov heatmaps of samples from a single distribution

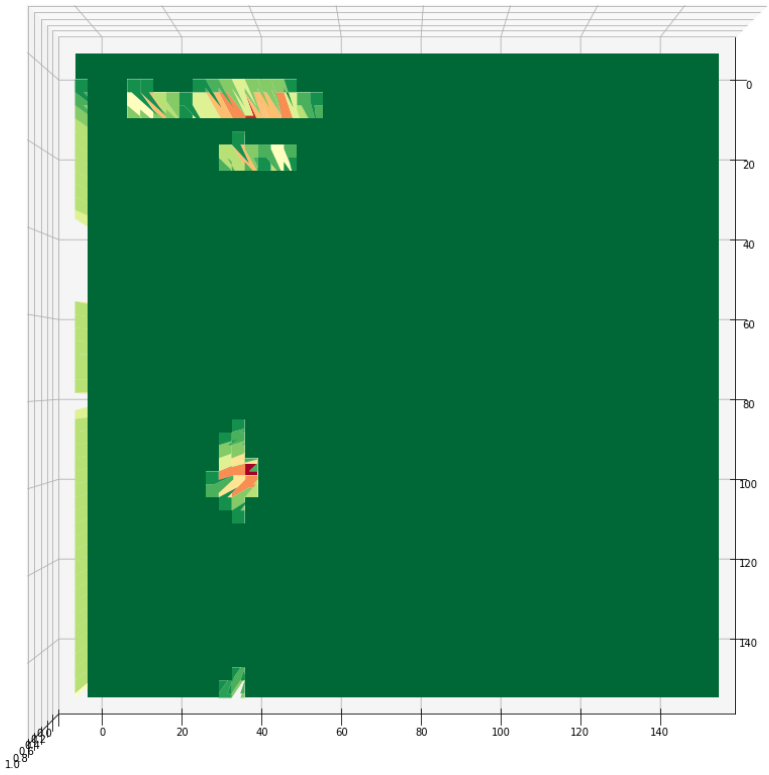

Figure 37: Negative Binomial

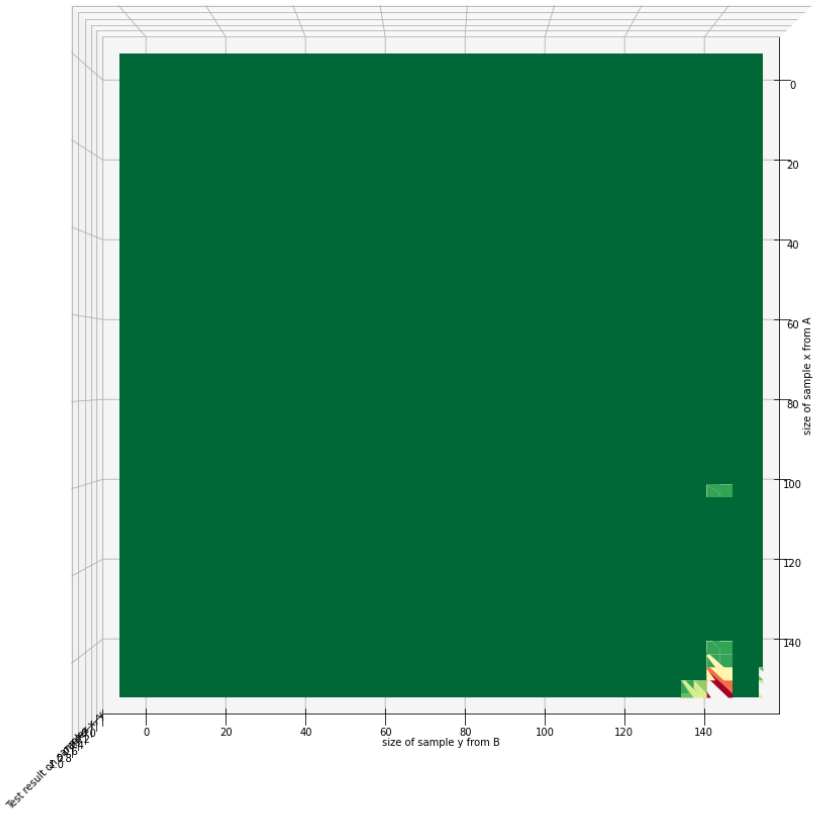

Figure 38: Poisson

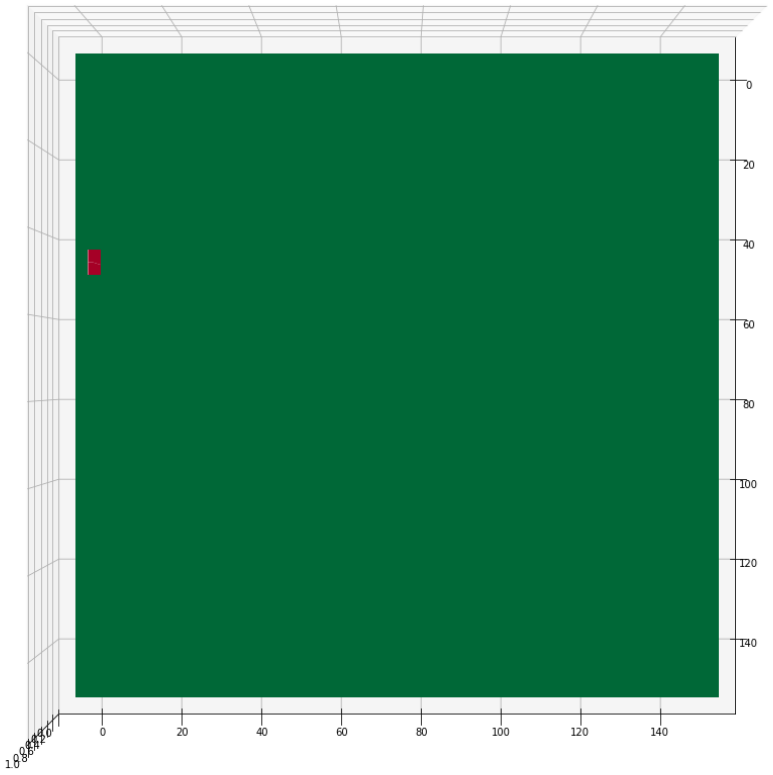

Figure 39: Student T

10.2.2 Kolmogorov Smirnov heatmaps of samples from two independent and shuffled distributions

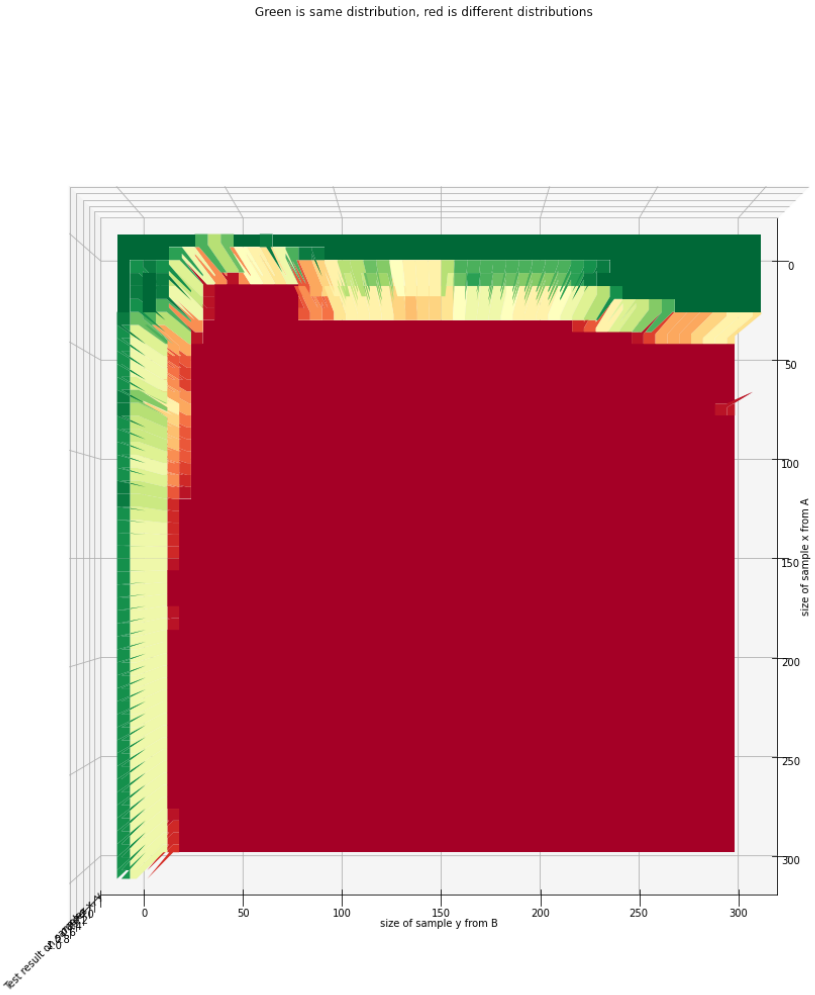

Figure 40: Negative Binomial on X axis and Student T on Y axis

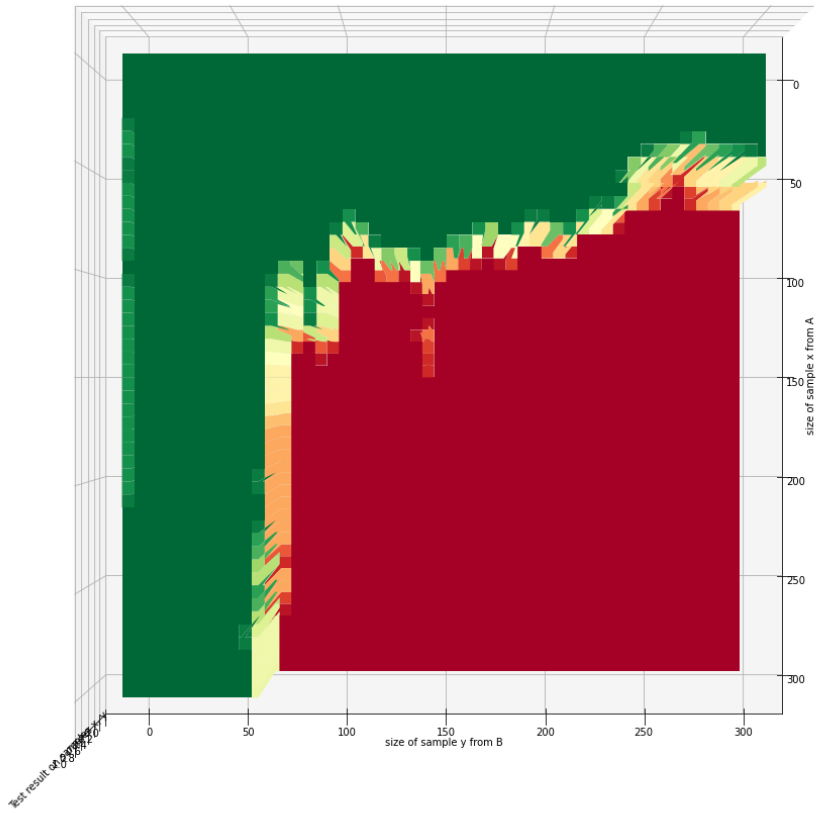

Figure 41: Poisson on X axis and Negative Binomial on Y axis

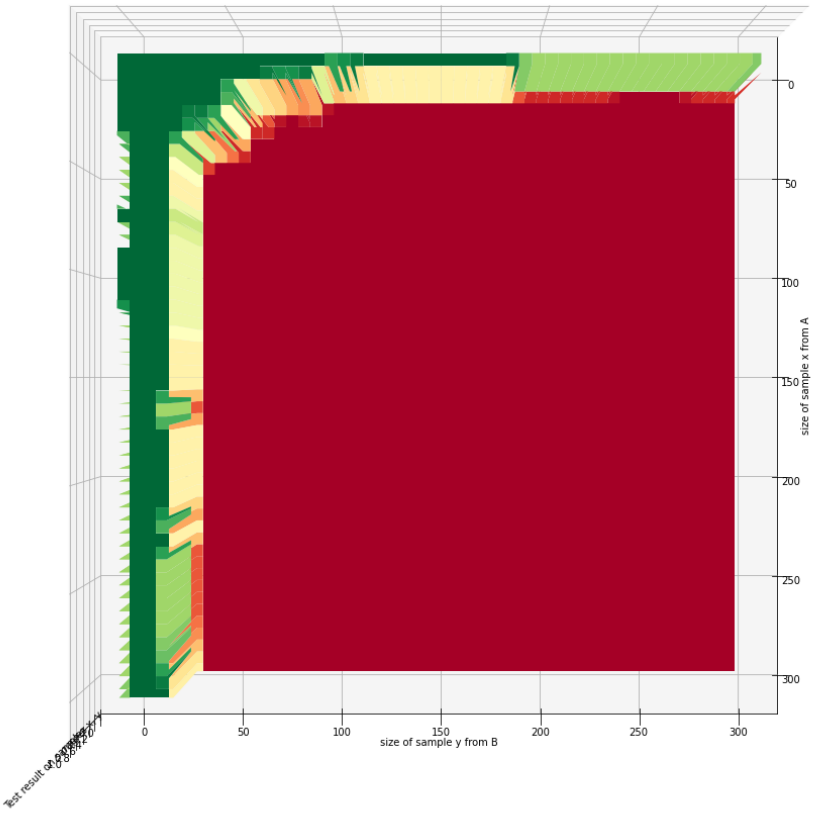

Figure 42: Poisson on X and Student T on Y axis

10.2.3 Heatmaps of the naive method applied to samples of varying length from a single distribution

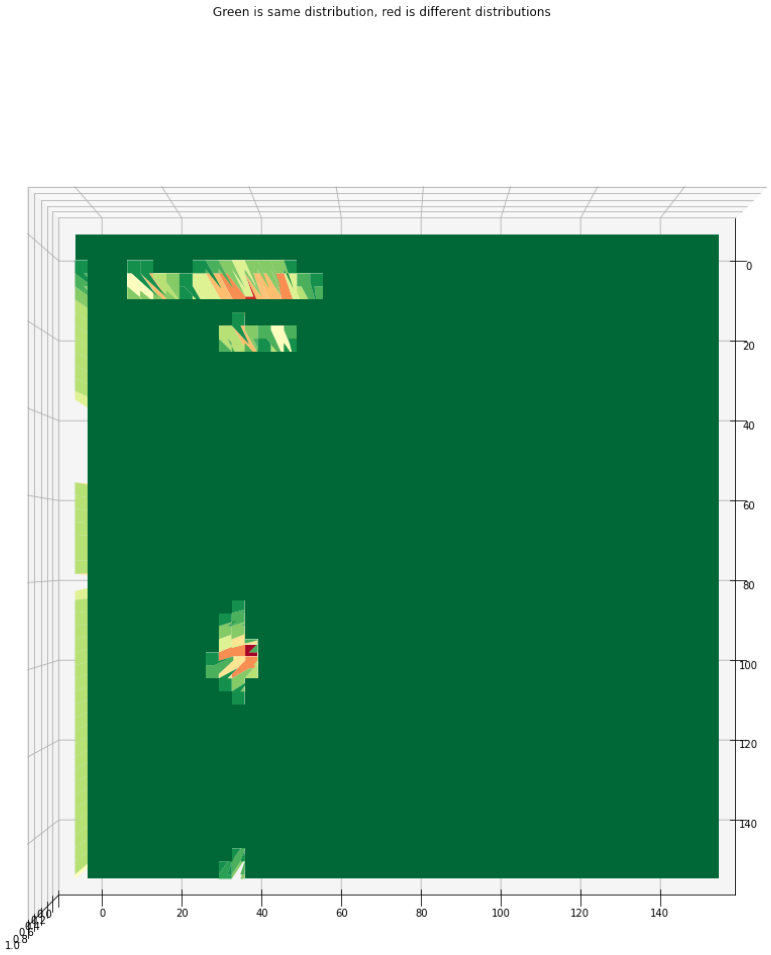

Figure 43: Negative Binomial

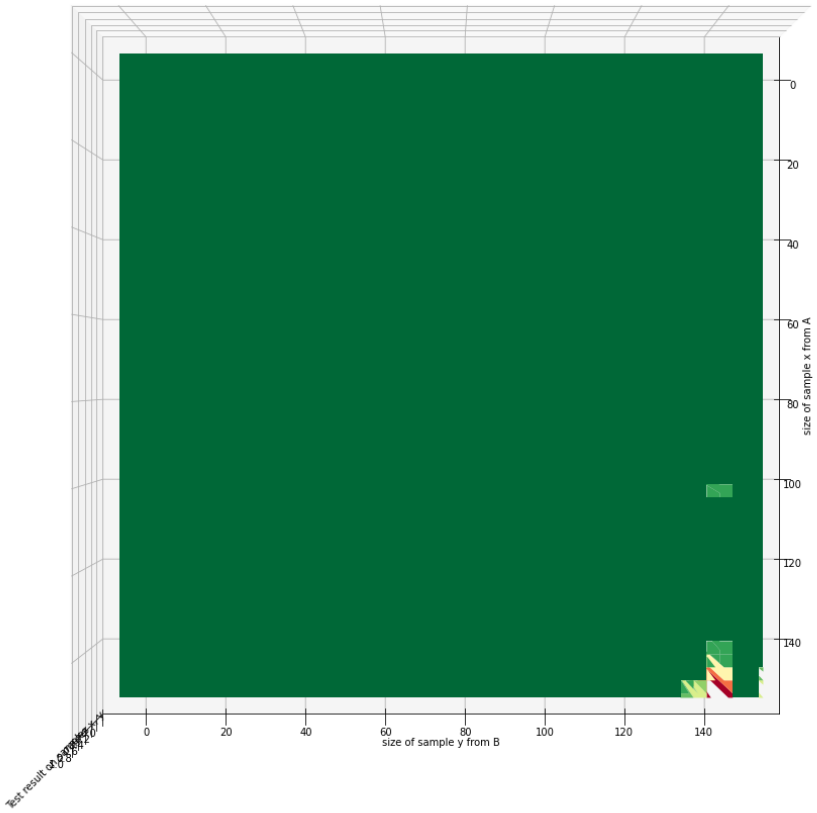

Figure 44: Poisson

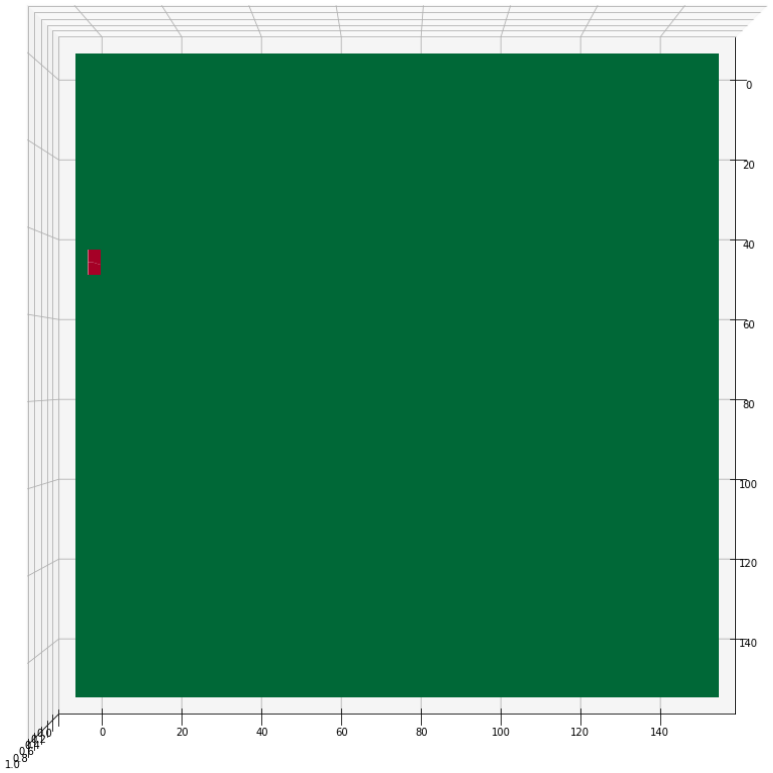

Figure 45: Student T

10.2.4 Heatmaps of the naive method applied to samples of varying length from two independent distributions

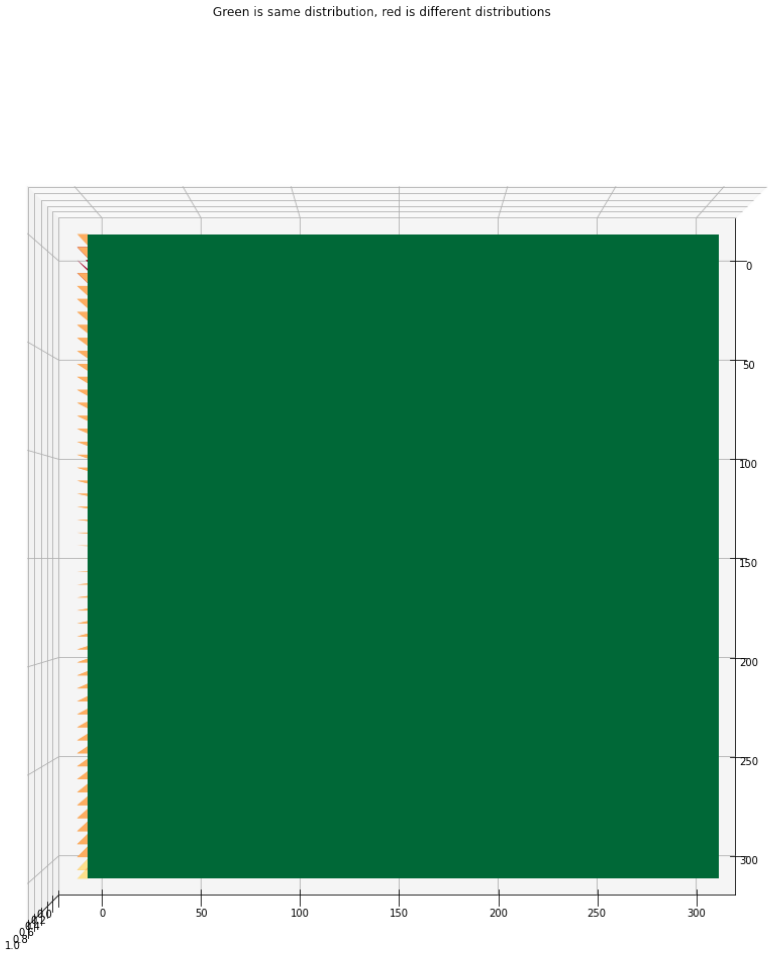

Figure 46: Negative Binomial on X axis and Student T on Y axis

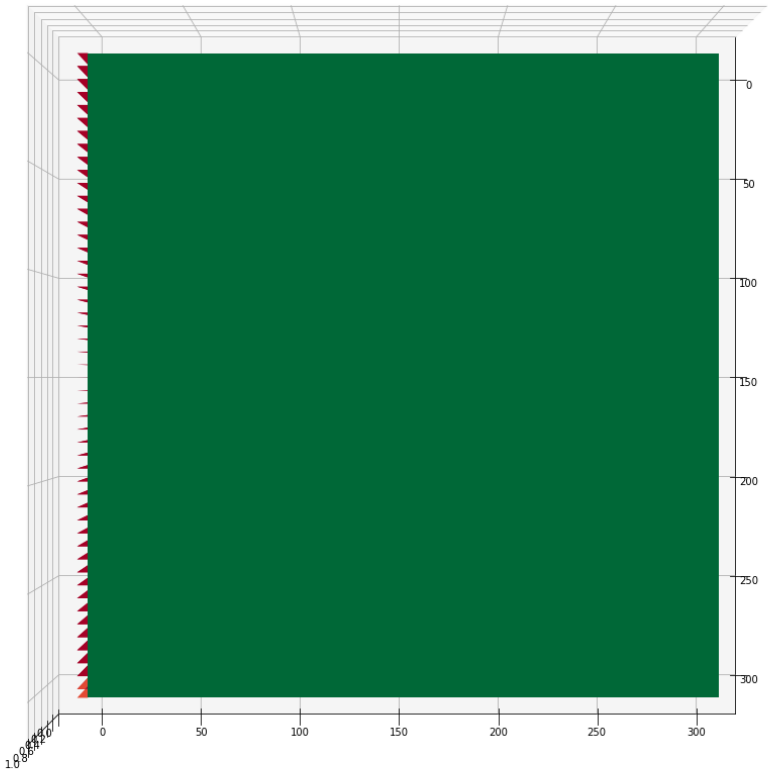

Figure 47: Poisson on X axis and Negative Binomial on Y axis

Figure 48: Poisson on X and Student T on Y axis

10.2.5 Heatmaps of the T-test applied to samples of varying length from a single distribution

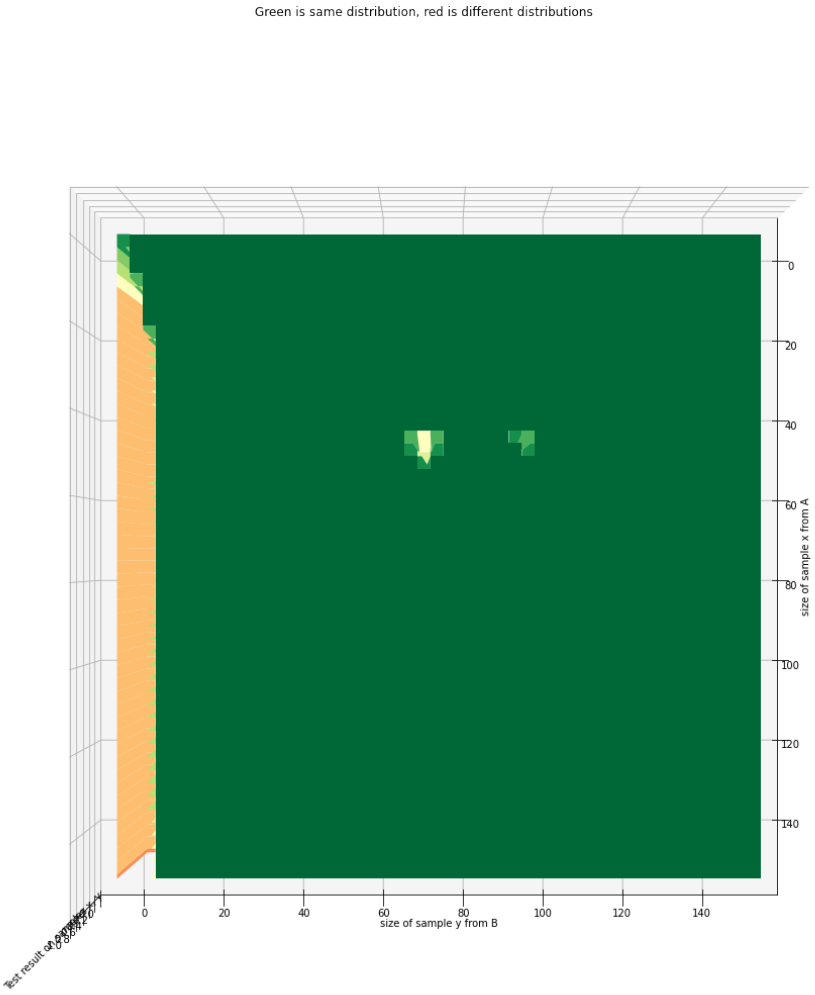

Figure 49: Negative Binomial

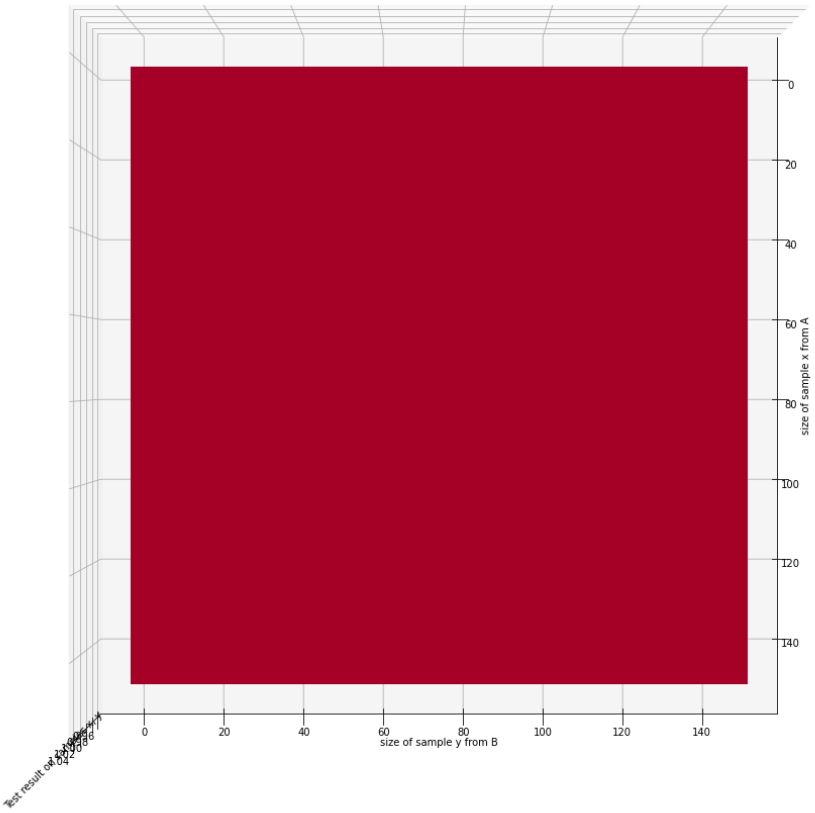

Figure 50: Poisson

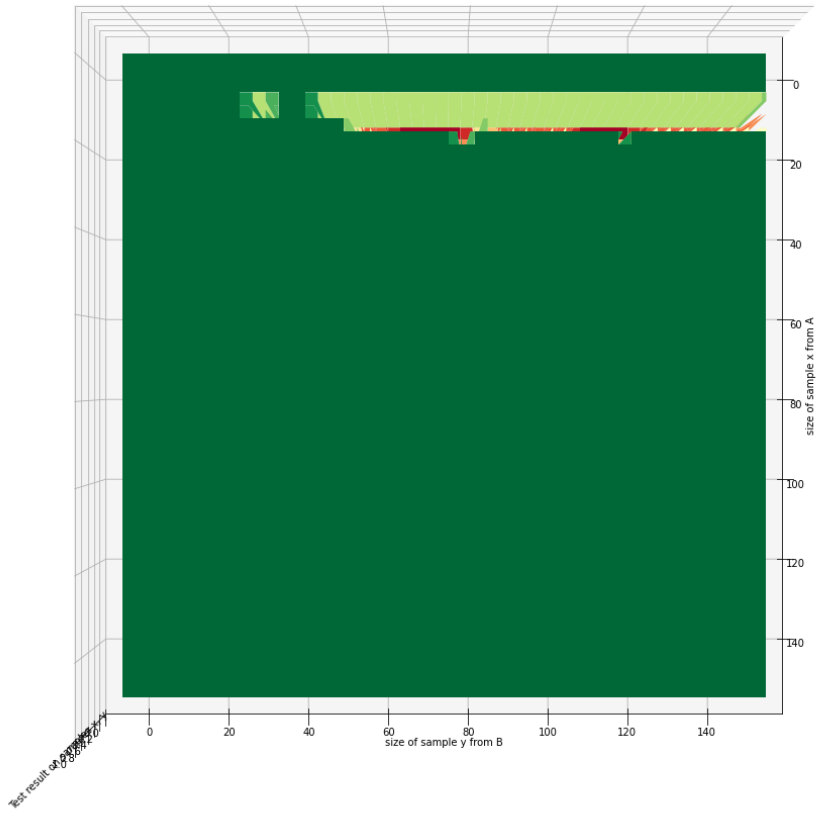

Figure 51: Student T

10.3 Dataset Plots and statistics

10.3.1 Electricity

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	2510.68	321	6755124	21044	21044	0.0	764000.0	1H
test	2509.92	2247	47501580	21068	21212	0.0	764000.0	1H

Table 8: Statistics of the Electricity dataset.

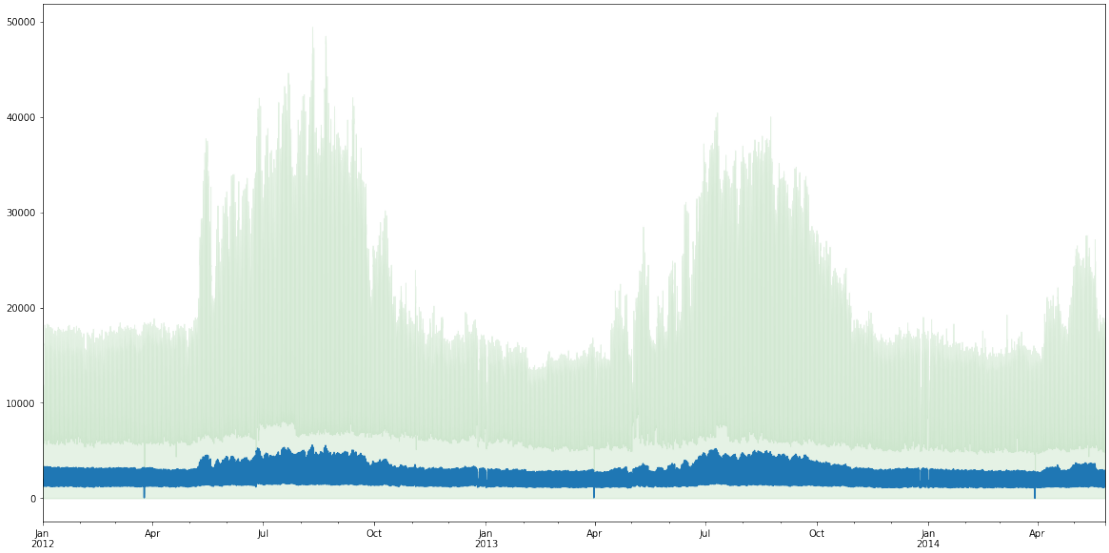

Figure 52: Plot over the average timeseries in the electricity dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

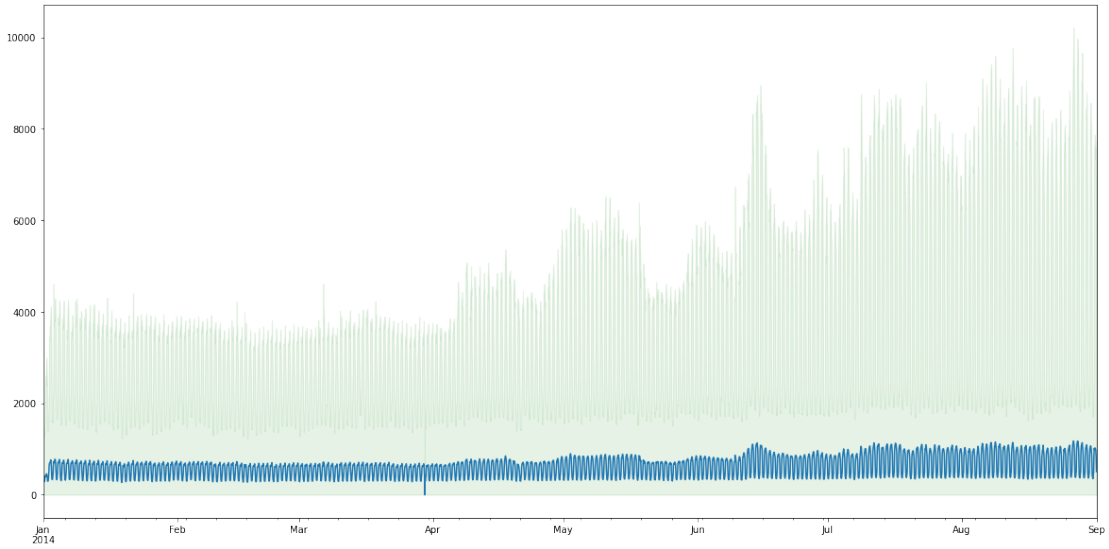

Figure 53: Zoomed version of 52

statistic	mean	deviation	max	min
trend	0.65	0.17	1.0	0.09
seasonality	0.84	0.19	1.0	0.0

Figure 54: Strength of trend and seasonality of the Electricity dataset

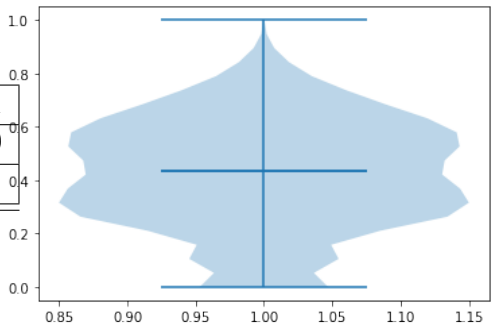

Figure 55: Scaled violin plot of the electricity dataset.

10.3.2 Exchange Rate

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	0.68	8	48568	6071	6071	0.01	2.11	1B
test	0.68	40	246440	6101	6221	0.01	2.11	1B

Table 9: Statistics of the Exchange Rate dataset

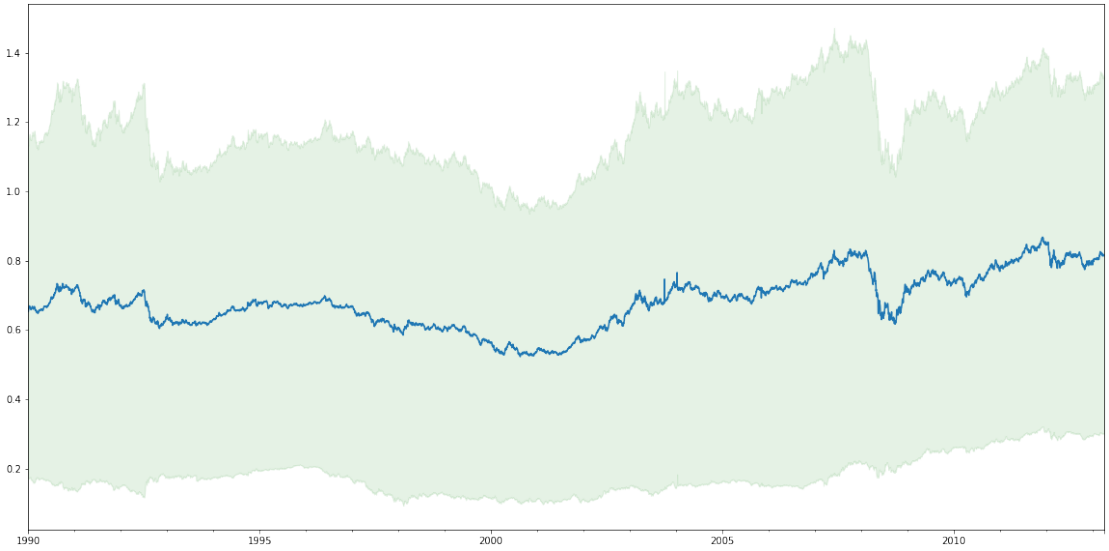

Figure 56: Plot over the average timeseries in the exchange rate dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	1.0	0.0	1.0	0.99
seasonality	0.12	0.30	0.90	0.0

Figure 57: Strength of trend and seasonality of the exchange rate dataset

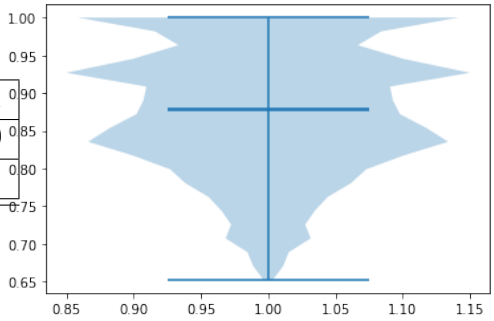

Figure 58: Scaled violin plot of the exchange rate dataset.

10.3.3 Solar Energy

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	40.35	137	960233	7009	7009	0.0	509.05	10min
test	40.25	959	6813695	7033	7177	0.0	509.05	10min

Table 10: Statistics of the Solar Energy dataset

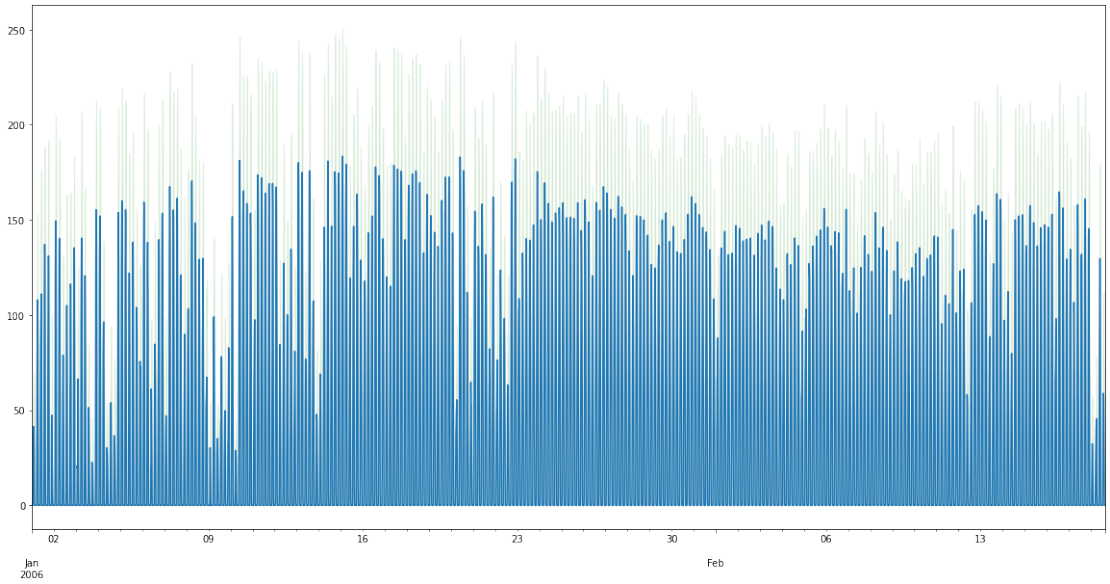

Figure 59: Plot over the average timeseries in the solar energy dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.09	0.03	0.15	0.0
seasonality	0.84	0.02	0.87	0.79

Figure 60: Strength of trend and seasonality of the solar-energy dataset

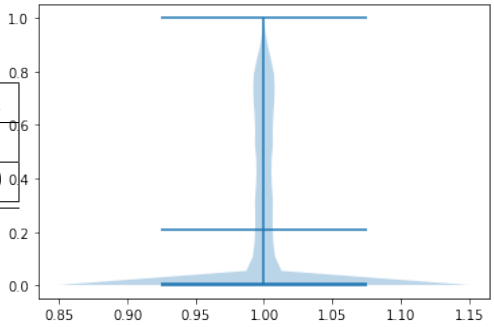

Figure 61: Scaled violin plot of the solar energy dataset.

10.3.4 Traffic

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	0.06	862	12099032	14036	14036	0.0	0.72	H
test	0.06	6034	85272488	14060	14204	0.0	0.72	H

Table 11: Statistics of the Traffic dataset

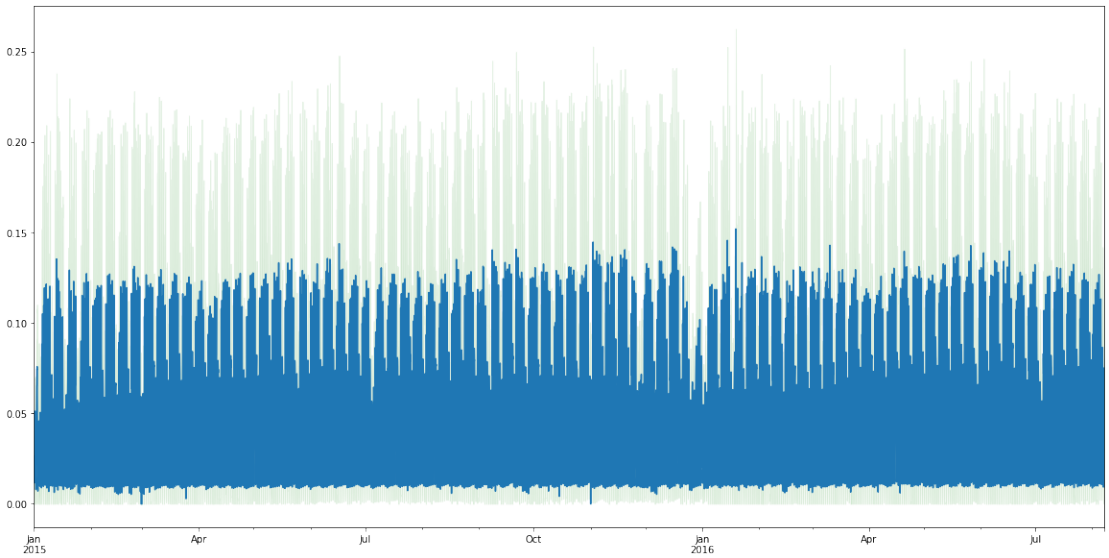

Figure 62: Plot over the average timeseries in the traffic dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.16	0.12	0.79	0.0
seasonality	0.67	0.10	0.93	0.12

Figure 63: Strength of trend and seasonality of the traffic dataset

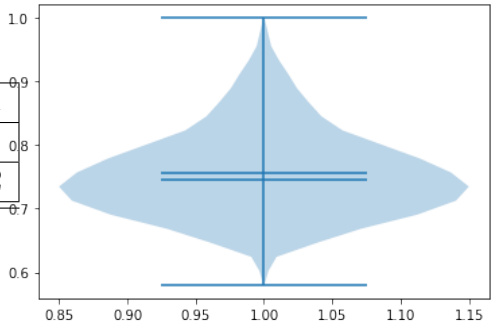

Figure 64: Scaled violin plot of the traffic dataset.

10.3.5 Exchange Rate NIPS

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	0.68	8	48568	6071	6071	0.01	2.11	B
test	0.68	40	246440	6101	6221	0.01	2.11	B

Table 12: Statistics of the Exchange Rate NIPS dataset

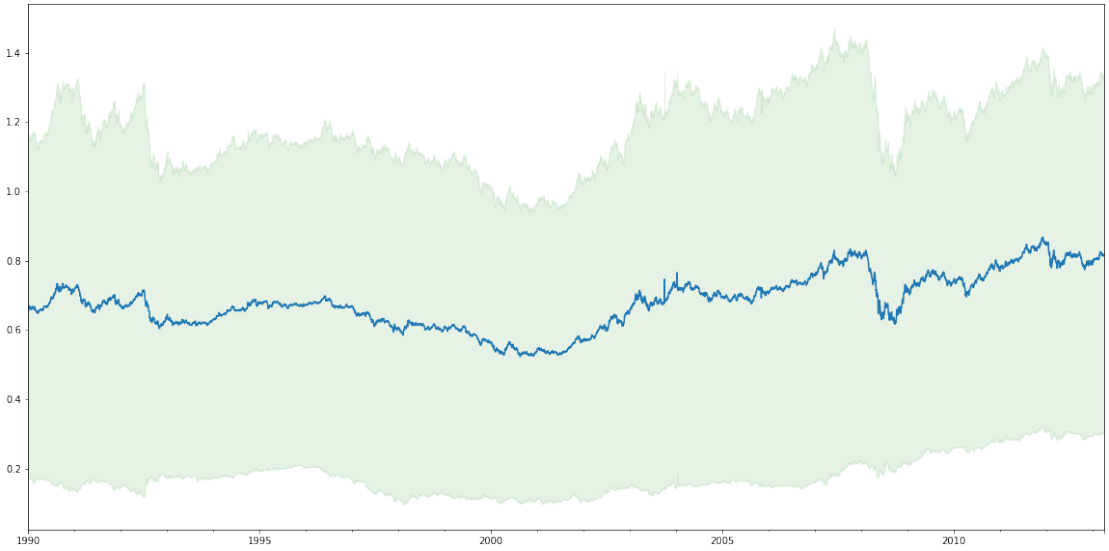

Figure 65: Plot over the average timeseries in the exchange rate nips dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	1.0	0.0	1.0	0.99
seasonality	0.12	0.30	0.90	0.0

Figure 66: Strength of trend and seasonality of the exchange rate nips dataset

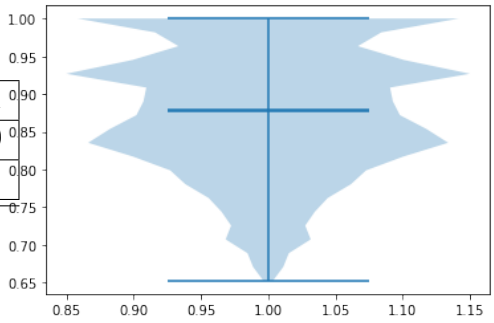

Figure 67: Scaled violin plot of the exchange rate nips dataset.

10.3.6 Electricity NIPS

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	607.95	370	2142282	1081	5833	0.0	168100.0	H
test	652.36	2590	10340239	1105	4000	0.0	168100.0	H

Table 13: Statistics of the Electricity NIPS dataset.

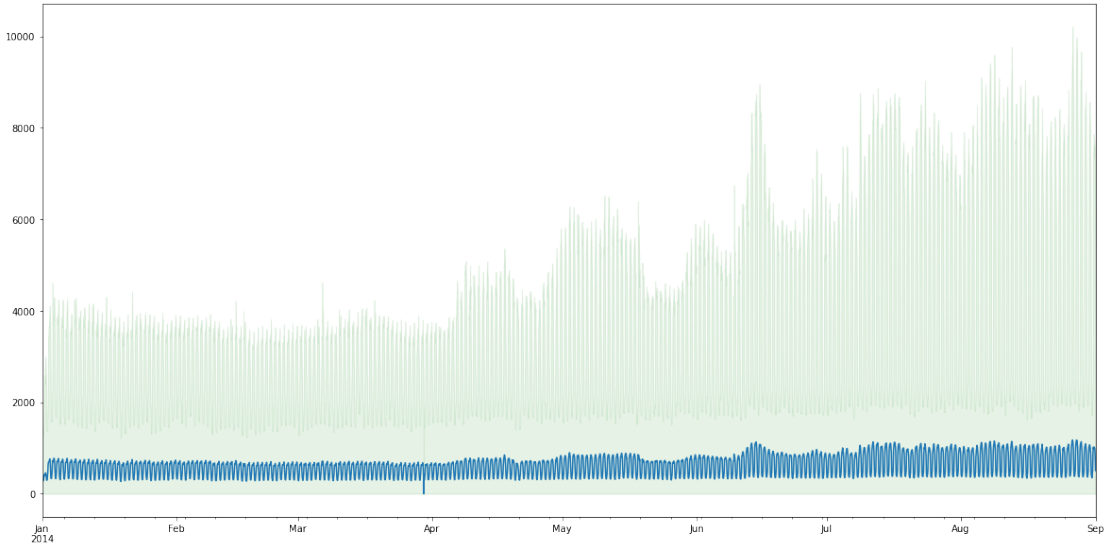

Figure 68: Plot over the average timeseries in the electricity nips dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.54	0.20	1.0	0.0
seasonality	0.86	0.16	1.0	0.0

Figure 69: Strength of trend and seasonality of the electricity nips dataset

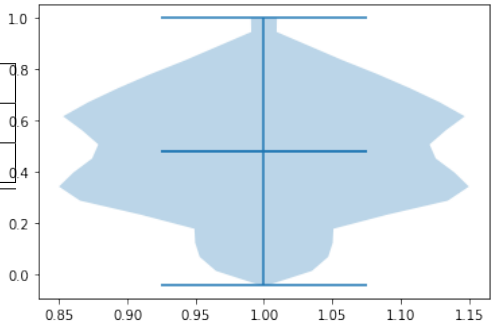

Figure 70: Scaled violin plot of the electricity nips dataset.

10.3.7 Solar Energy NIPS

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	40.35	137	960233	7009	7009	0.00	509.05	H
test	40.25	959	6813695	7033	7177	0.00	509.05	H

Table 14: Statistics of the Solar Energy NIPS dataset

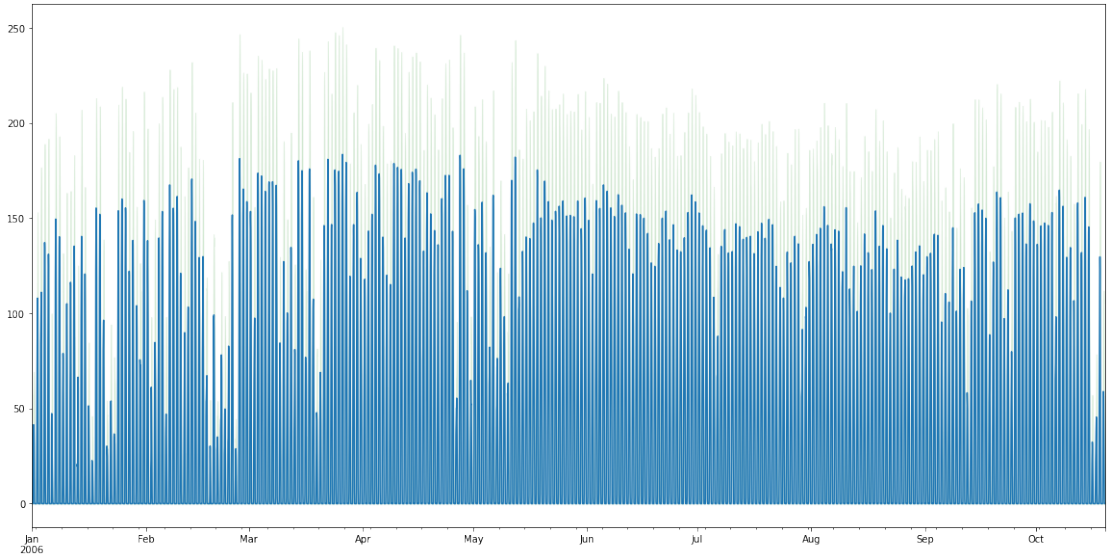

Figure 71: Plot over the average timeseries in the solar nips dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.17	0.02	0.24	0.11
seasonality	0.86	0.02	0.89	0.80

Figure 72: Strength of trend and seasonality of the solar nips dataset

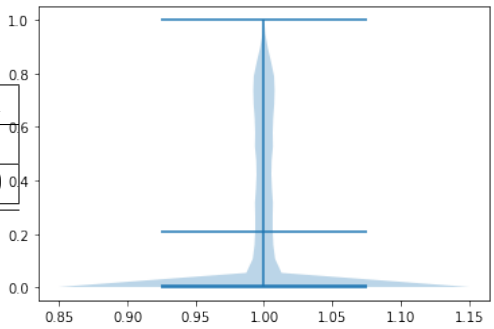

Figure 73: Scaled violin plot of the solar nips dataset.

10.3.8 Traffic NIPS

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	0.05	963	3852963	4001	4001	0.00	1.00	H
test	0.05	6741	26964000	4000	4000	0.00	1.00	H

Table 15: Statistics of the Traffic NIPS dataset

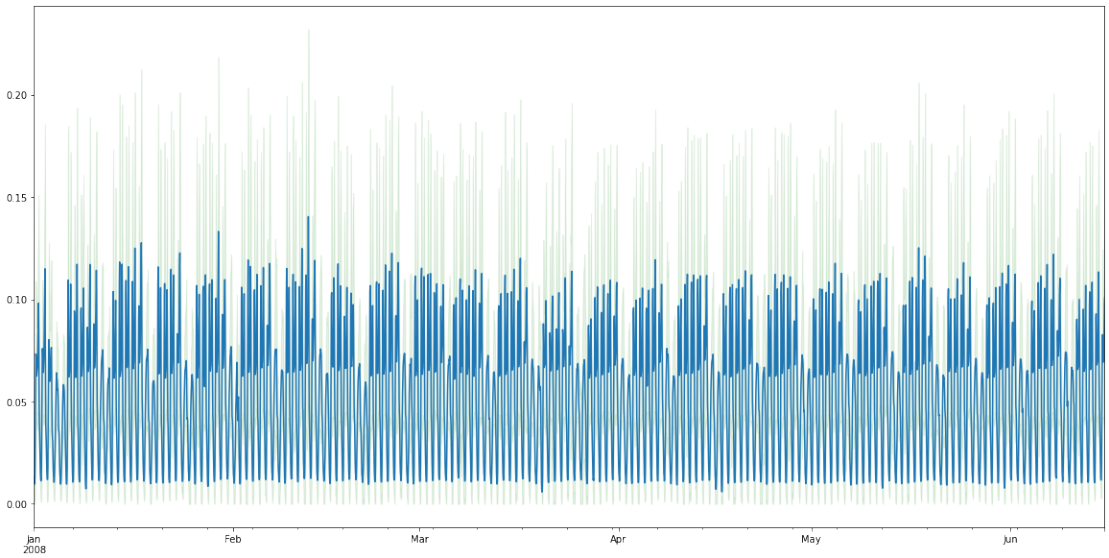

Figure 74: Plot over the average timeseries in the traffic nips dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.29	0.18	0.87	0.0
seasonality	0.76	0.12	0.94	0.0

Figure 75: Strength of trend and seasonality of the traffic nips dataset

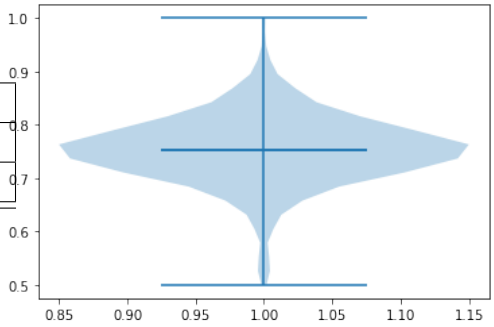

Figure 76: Scaled violin plot of the traffic nips dataset.

10.3.9 Wiki Rolling NIPS

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	3720.54	9535	7551720	792	792	0.00	7752515.00	D
test	3663.55	47675	40619100	792	912	0.00	7752515.00	D

Table 16: Statistics of the Wiki Rolling NIPS dataset

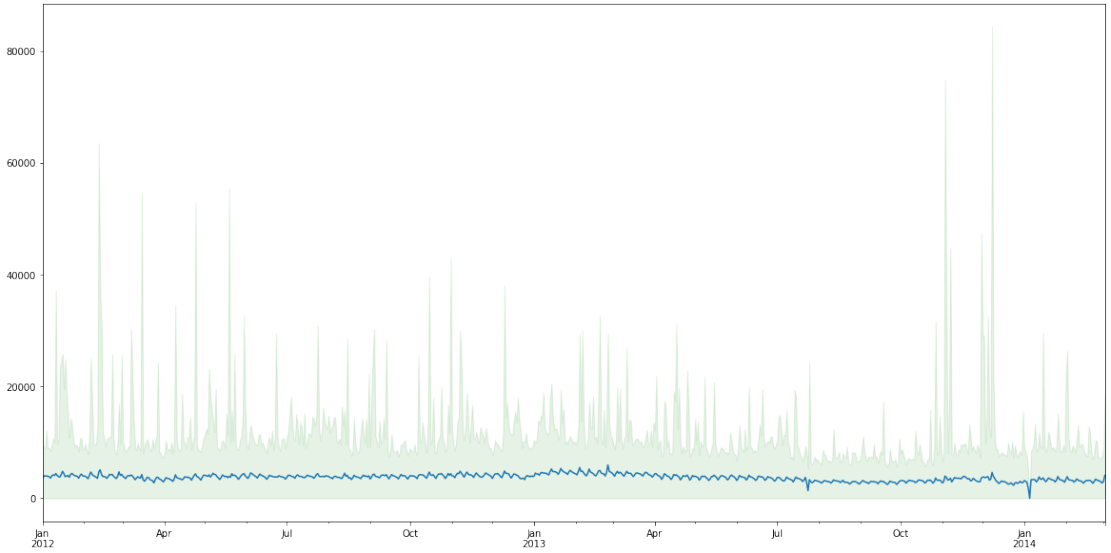

Figure 77: Plot over the average timeseries in the wiki-rolling nips dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.53	0.27	1.0	0.0
seasonality	0.23	0.26	1.0	0.0

Figure 78: Strength of trend and seasonality of the wiki-rolling nips dataset

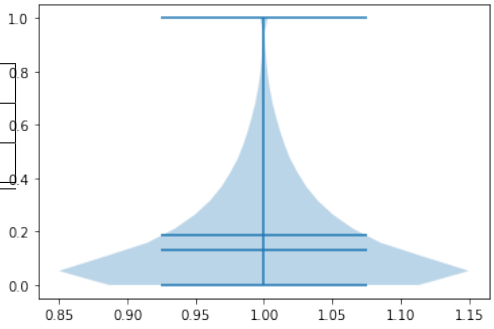

Figure 79: Scaled violin plot of the wiki-rolling nips dataset.

10.3.10 Taxi

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	8.79	1214	1806432	1488	1488	0.0	265.0	30min
test	7.41	67984	54999056	149	1469	0.0	225.0	30min

Table 17: Statistics of the Taxi NIPS dataset

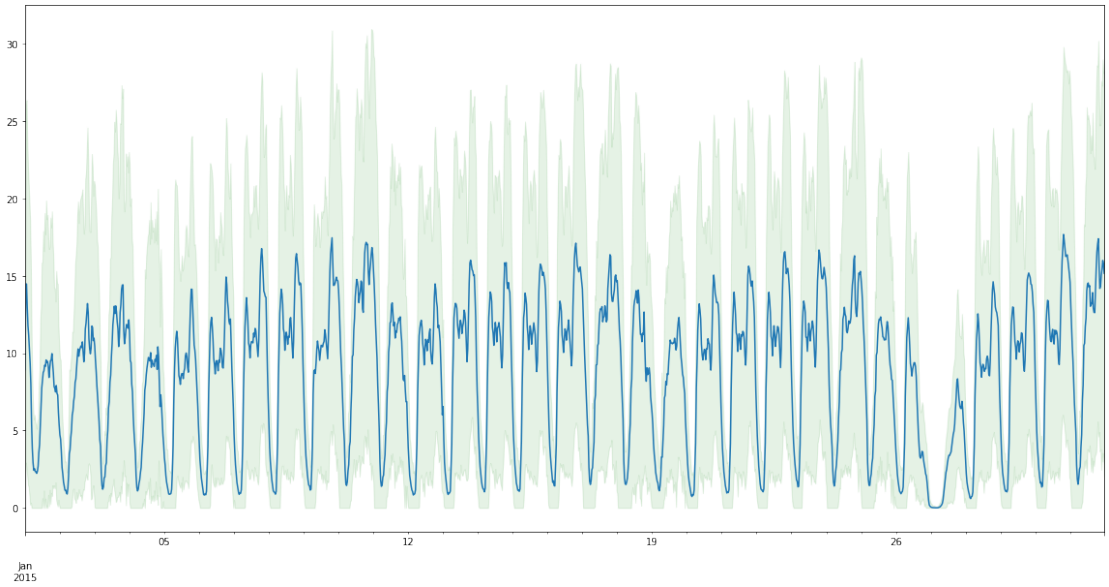

Figure 80: Plot over the average timeseries in the taxi dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.02	0.02	0.16	0.0
seasonality	0.66	0.08	0.92	0.41

Figure 81: Strength of trend and seasonality of the taxi dataset

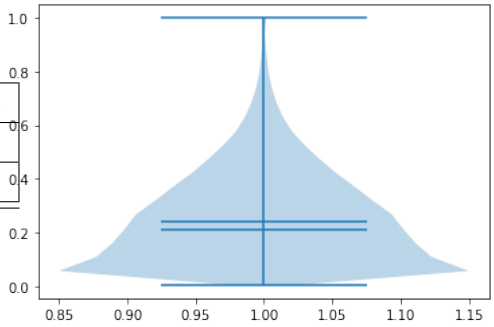

Figure 82: Scaled violin plot of the taxi dataset.

10.3.11 M3 Monthly

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max
train	4928.47	1428	141858	48	126	80.00	86730.00
test	4971.28	1428	167562	66	144	-1200.00	86730.00

Table 18: Statistics of the M3 Monthly dataset.

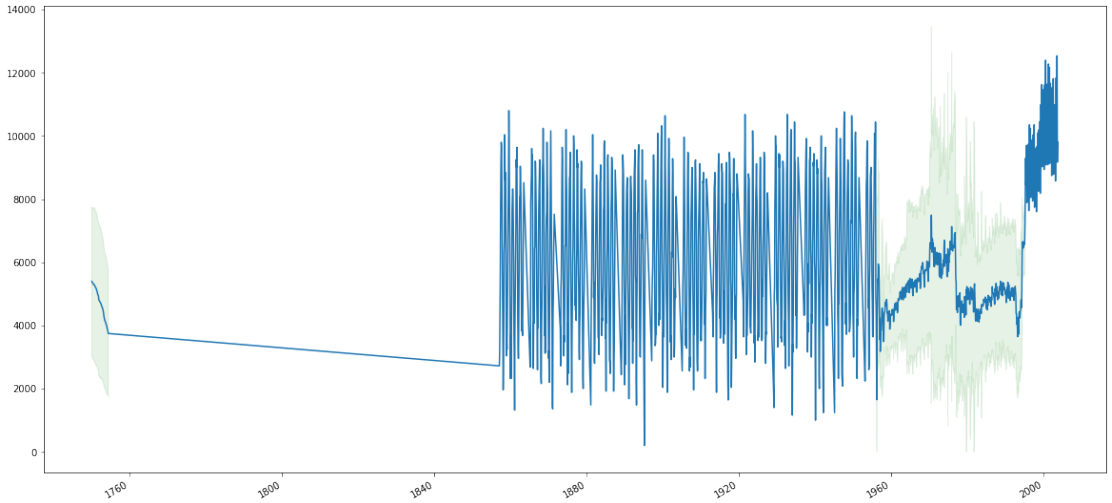

Figure 83: Plot over the average timeseries in the m3 monthly dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.68	0.31	1.0	0.0
seasonality	0.35	0.29	1.0	0.0

Figure 84: Strength of trend and seasonality of the m3 monthly dataset

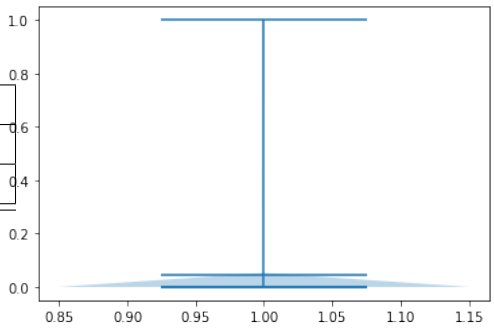

Figure 85: Scaled violin plot of the m3 monthly dataset.

10.3.12 M3 Quarterly

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max
train	4819.27	756	30956	16	64	126.00	20245.00
test	4983.53	756	37004	24	72	121.00	20375.00

Table 19: Statistics of the M3 Quarterly dataset.

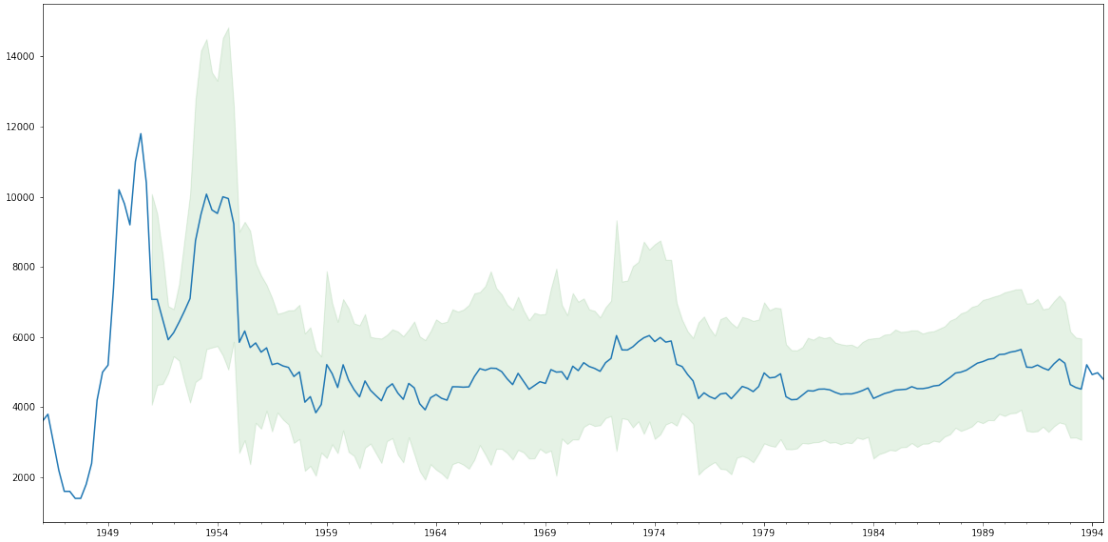

Figure 86: Plot over the average timeseries in the m3 quarterly dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.88	0.19	1.0	0.0
seasonality	0.33	0.35	1.0	0.0

Figure 87: Strength of trend and seasonality of the m3 quarterly dataset

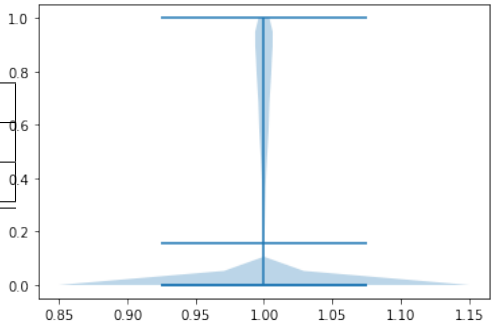

Figure 88: Scaled violin plot of the m3 quarterly dataset.

10.3.13 M3 Yearly

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max
train	4417.05	645	14449	14	41	30.00	39666.22
test	4815.77	645	18319	20	47	30.00	45525.66

Table 20: Statistics of the M3 Yearly dataset

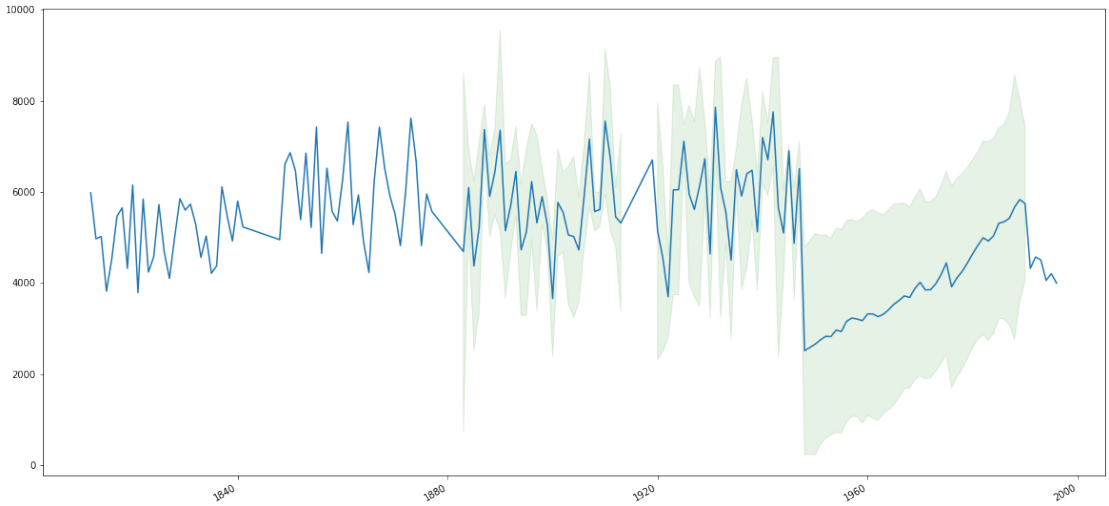

Figure 89: Plot over the average timeseries in the m3 yearly dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.89	0.17	1.0	0.0
seasonality	0.09	0.17	0.96	0.0

Figure 90: Strength of trend and seasonality of the m3 yearly dataset

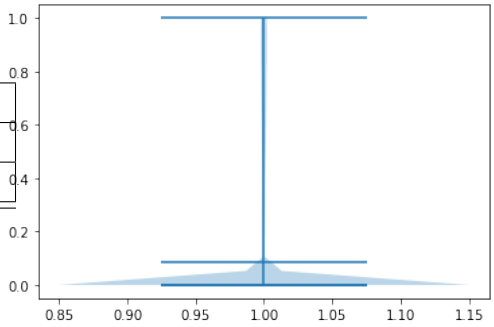

Figure 91: Scaled violin plot of the m3 yearly dataset.

10.3.14 M3 Other

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max
train	6152.18	174	11933	63	96	28.00	59472.00
test	5999.87	174	13325	71	104	28.00	59472.00

Table 21: Statistics of the M3 Other dataset

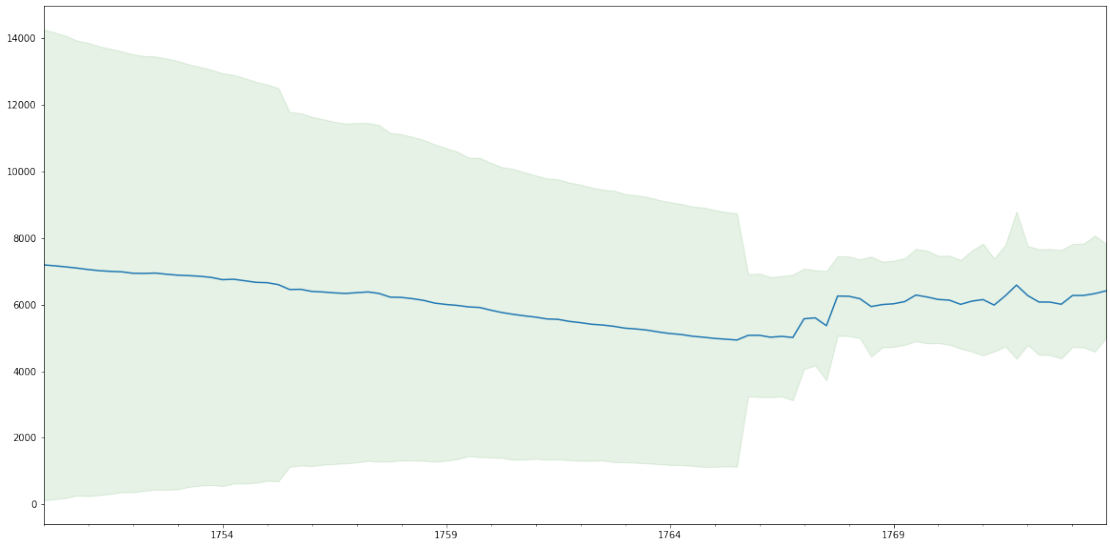

Figure 92: Plot over the average timeseries in the m3 other dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.94	0.17	1.0	0.35
seasonality	0.07	0.08	0.44	0.0

Figure 93: Strength of trend and seasonality of the m3 other dataset

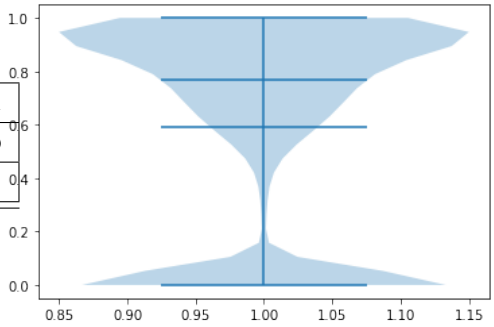

Figure 94: Scaled violin plot of the m3 other dataset.

10.3.15 M4 Hourly

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	6827.69	414	353500	700	960	10.00	703008.00	H
test	6859.56	414	373372	748	1008	10.00	703008.00	H

Table 22: Statistics of the M4 Hourly dataset

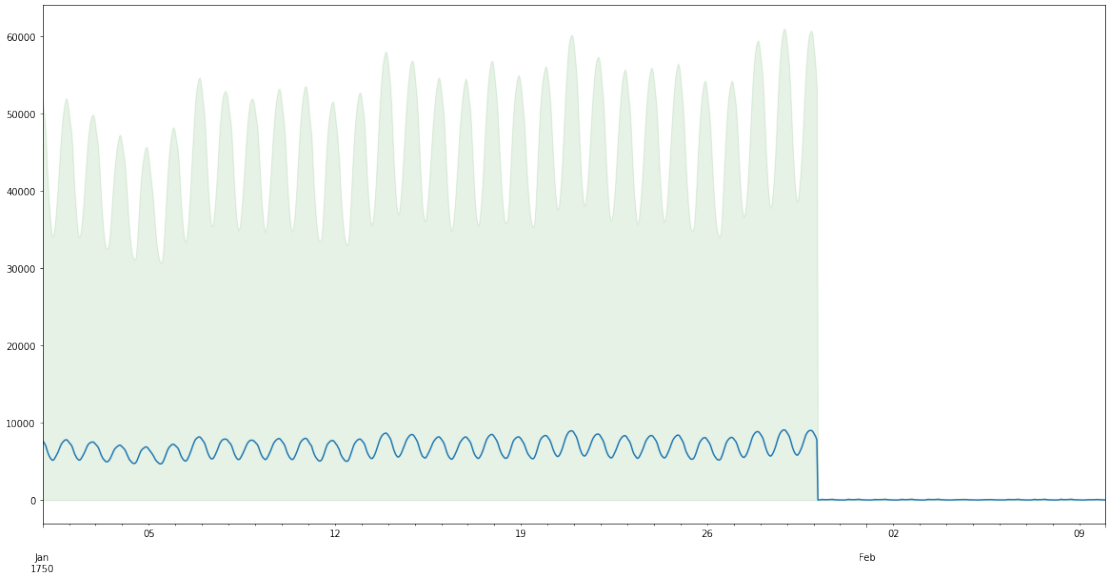

Figure 95: Plot over the average timeseries in the m4 hourly dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.62	0.37	1.0	0.0
seasonality	0.88	0.16	1.0	0.0

Figure 96: Strength of trend and seasonality of the m4 hourly dataset

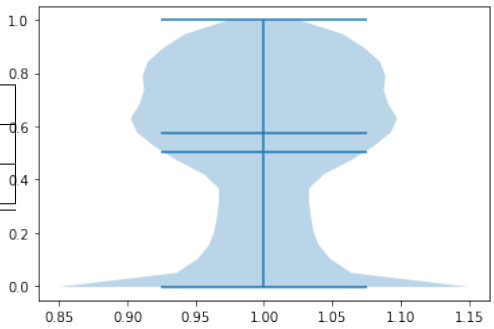

Figure 97: Scaled violin plot of the m4 hourly dataset.

10.3.16 M4 Daily

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	4951.40	4227	9964658	93	9919	15.00	352000.00	D
test	4960.15	4227	10023836	107	9933	15.00	352000.00	D

Table 23: Statistics of the M4 Daily dataset

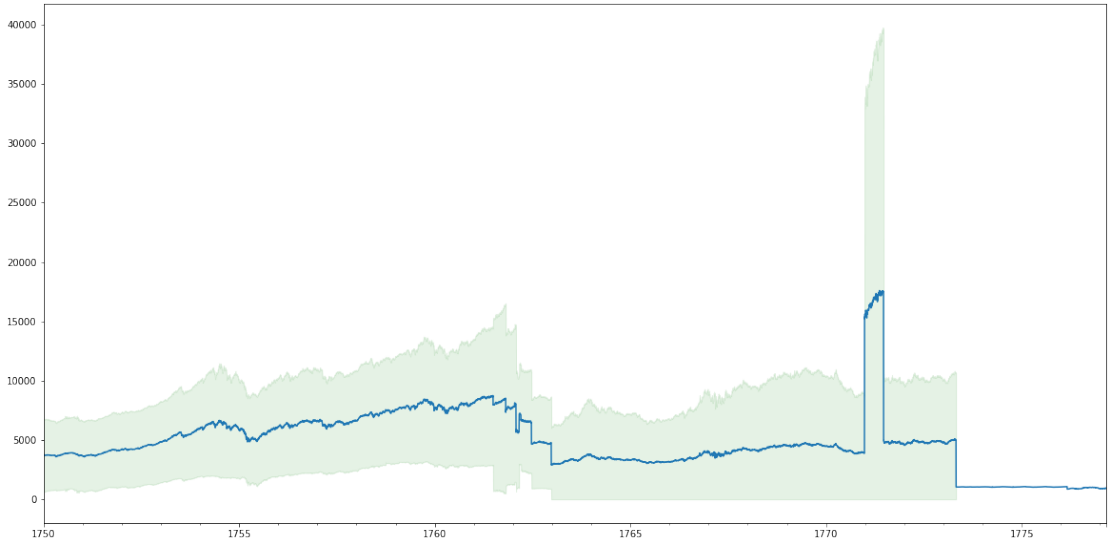

Figure 98: Plot over the average timeseries in the m4 daily dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.98	0.05	1.0	0.0
seasonality	0.05	0.10	1.0	0.0

Figure 99: Strength of trend and seasonality of the m4 daily dataset

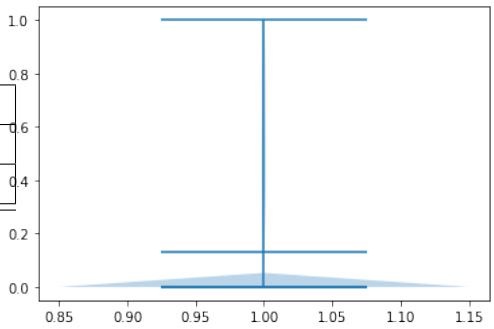

Figure 100: Scaled violin plot of the m4 daily dataset.

10.3.17 M4 Weekly

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	3738.52	359	366912	80	2597	104.69	51410.00	W
test	3755.97	359	371579	93	2610	104.69	51410.00	W

Table 24: Statistics of the M4 Weekly dataset

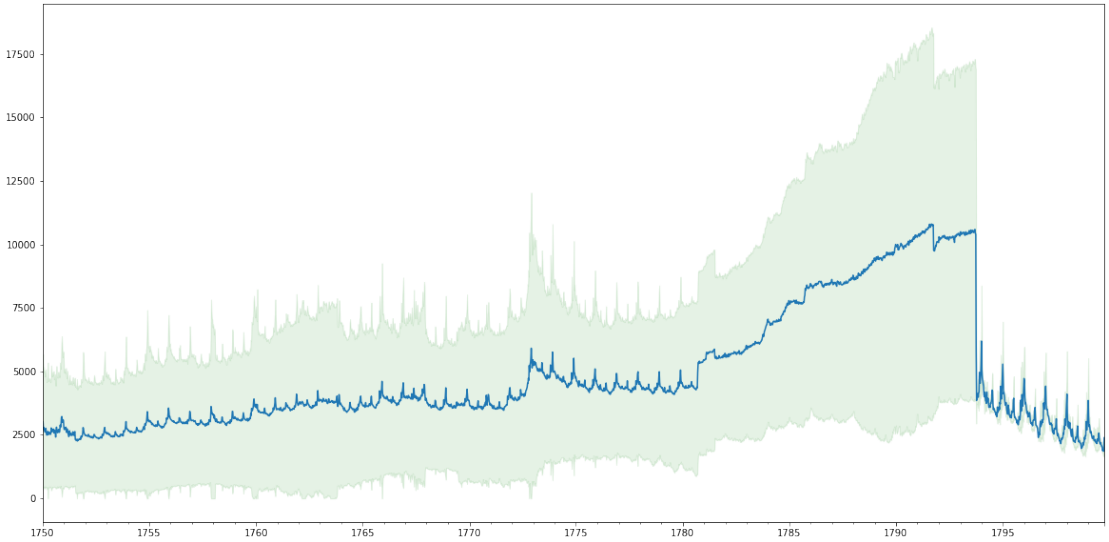

Figure 101: Plot over the average timeseries in the m4 weekly dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.77	0.31	1.0	0.0
seasonality	0.34	0.35	1.0	0.0

Figure 102: Strength of trend and seasonality of the m4 weekly dataset

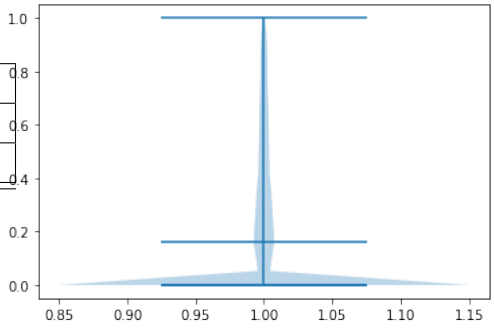

Figure 103: Scaled violin plot of the m4 weekly dataset.

10.3.18 M4 Monthly

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	4193.28	48000	10382411	42	2794	20.00	132731.31	M
test	4207.51	48000	11246411	60	2812	20.00	177950.00	M

Table 25: Statistics of the M4 Monthly dataset

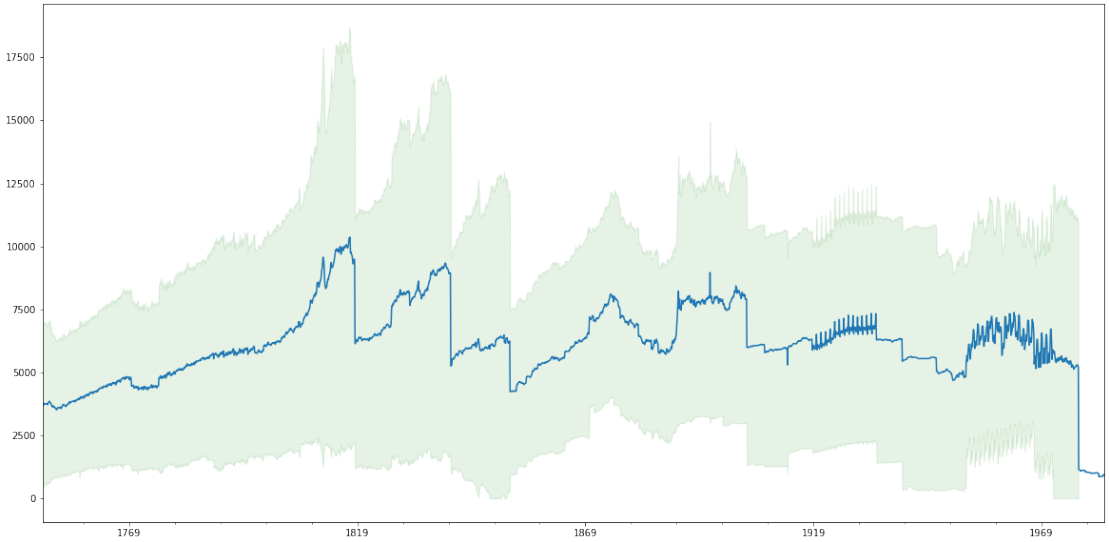

Figure 104: Plot over the average timeseries in the m4 monthly dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.84	0.23	1.0	0.0
seasonality	0.32	0.30	1.0	0.0

Figure 105: Strength of trend and seasonality of the m4 monthly dataset

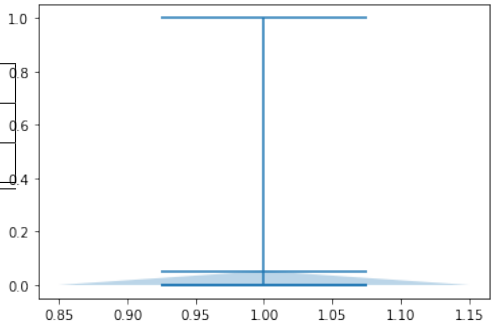

Figure 106: Scaled violin plot of the m4 monthly dataset.

10.3.19 M4 Quarterly

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	4141.00	24000	2214108	16	866	19.50	82210.70	3M
test	4287.13	24000	2406108	24	874	19.50	82210.70	3M

Table 26: Statistics of the M4 Quarterly dataset

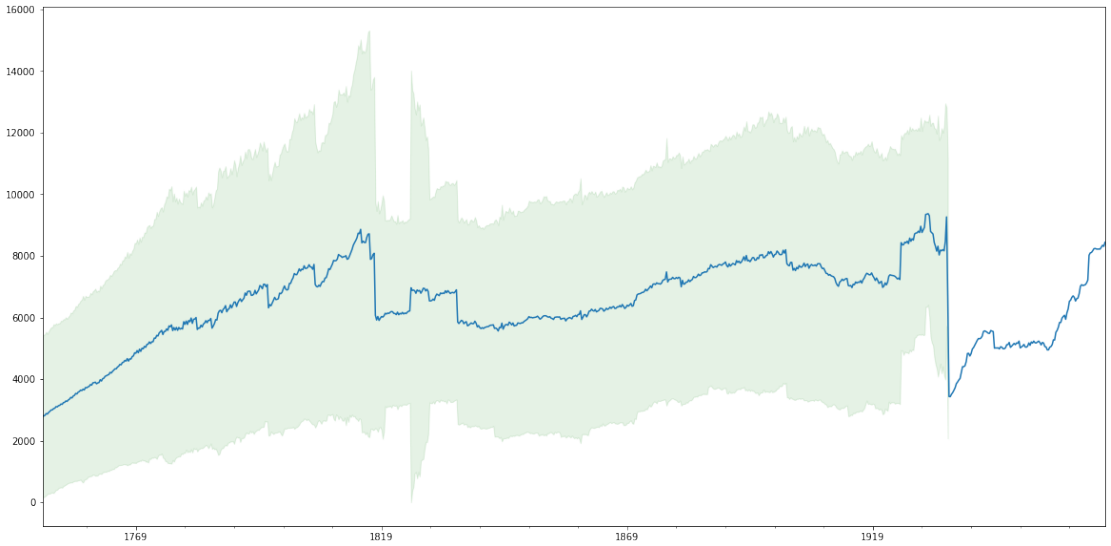

Figure 107: Plot over the average timeseries in the m4 quarterly dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.90	0.16	1.0	0.0
seasonality	0.20	0.27	1.0	0.0

Figure 108: Strength of trend and seasonality of the m4 quarterly dataset

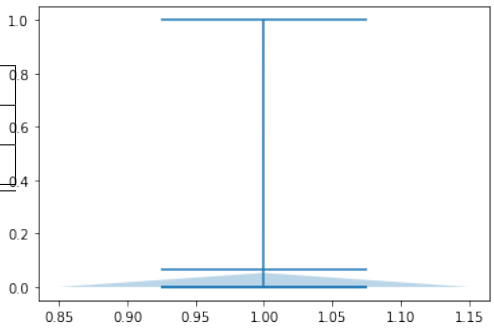

Figure 109: Scaled violin plot of the m4 quarterly dataset.

10.3.20 M4 Yearly

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	3630.52	23000	715065	13	300	22.10	115642.00	12M
test	4076.24	23000	852909	19	300	22.00	158430.00	12M

Table 27: Statistics of the M4 Yearly dataset

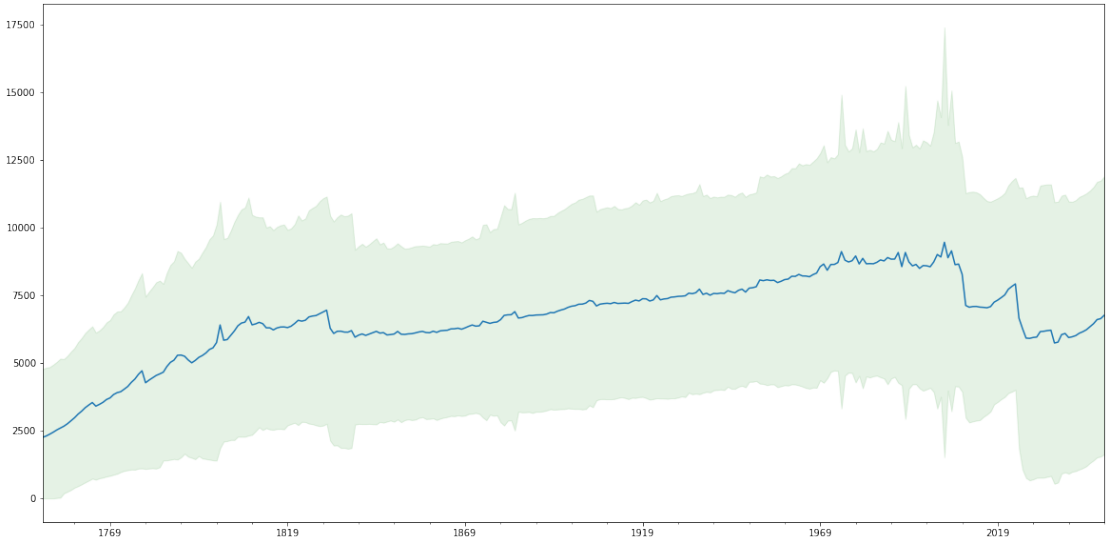

Figure 110: Plot over the average timeseries in the m4 yearly dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.93	0.13	1.0	0.0
seasonality	0.09	0.16	0.98	0.0

Figure 111: Strength of trend and seasonality of the m4 yearly dataset

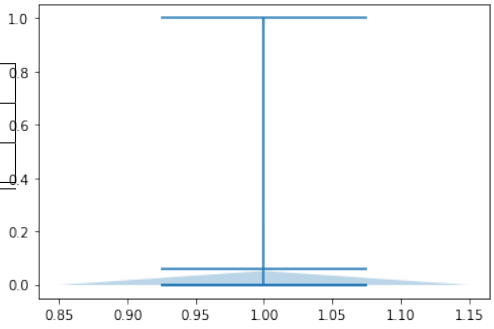

Figure 112: Scaled violin plot of the m4 yearly dataset.

10.3.21 M5

Dataset	Mean	Series	Items	Shortest	Longest	Min	Max	Frequency
train	1.12	30490	57473650	1885	1885	0.00	763.00	D
test	1.13	30490	58327370	1913	1913	0.00	763.00	D

Table 28: Statistics of the M5 dataset

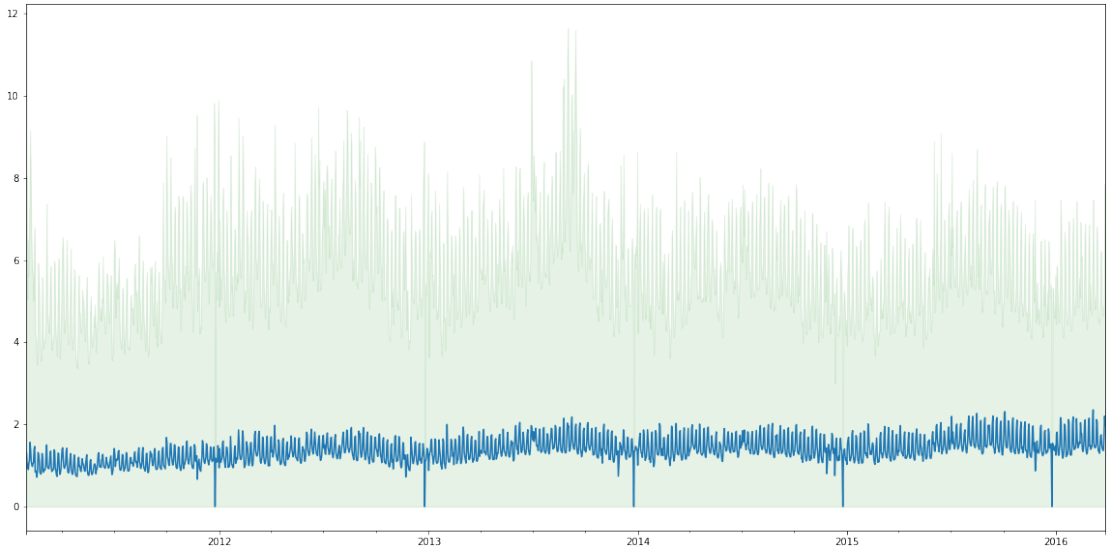

Figure 113: Plot over the average timeseries in the m5 dataset with one standard deviation shown in green. Only positive values are shown as the dataset is non-negative.

statistic	mean	deviation	max	min
trend	0.38	0.32	1.0	0.0
seasonality	0.28	0.33	1.0	0.0

Figure 114: Strength of trend and seasonality of the m5 dataset

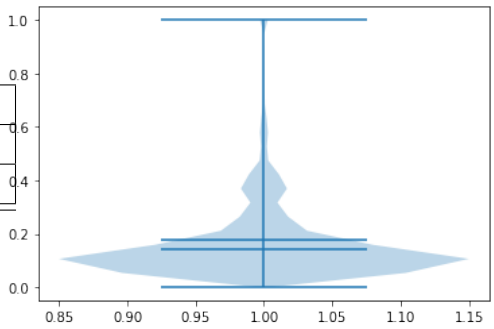

Figure 115: Scaled violin plot of the m5 dataset.