Statistical and Deep Learning Approaches to Anomaly Detection in Financial Time Series

Ethan Li[*], Rohan R. Kulkarni[*]
Daniel Fernandez[†], Yossi Cohen [†], Ali Hirsa [*]

May 2023

Contents

[*]Columbia University Department of Industrial Engineering and Operations Research
[†]Wellington Management

1 Abstract

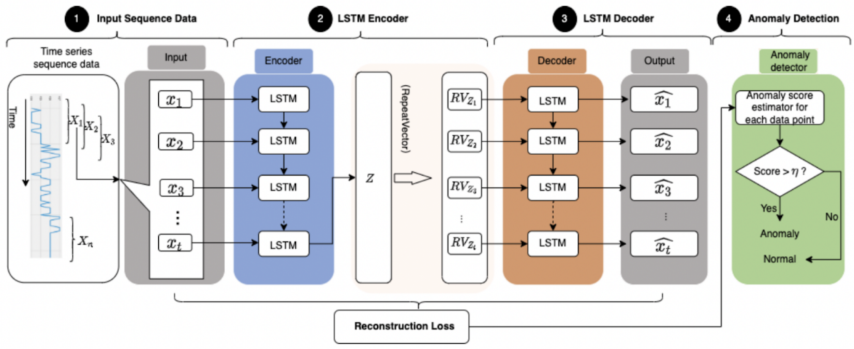
Detecting anomalies in time series data is crucial for various applications, ranging from finance to healthcare. In this project, we investigate the performance of three machine learning methods for detecting anomalies in credit card transaction time series data provided by Wellington Management: Long Short-Term Memory (LSTM) networks, Bayesian Hierarchical Models, and Wavelet Analysis. Our objective is to compare the effectiveness of these approaches and determine the best method for accurate anomaly detection. We preprocess and transform the collected data, then apply each method, evaluating their performance in terms of accuracy, precision, recall, and F1 score. Our findings reveal that each method has its strengths and weaknesses, with varying degrees of success in detecting anomalies. By comparing the results, we provide valuable insights for practitioners and researchers in selecting the most suitable technique for their specific use cases, while also suggesting future research directions to further improve anomaly detection in time series data.

2 Introduction

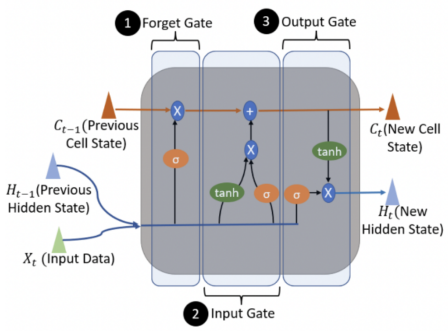
2.1 Background

This work is a continuation of the anomaly detection in time series from a previous group. In the context of anomaly detection, the past research group utilized a Git repository called `deep-learning-wellington-od`. We build off of this repository and add our own code for the models that we implemented. The complete code repository filetree and description is given in Appendix A. They planned to select a baseline model from the outlier detection library `pyod` to address univariate cases [12]. Once the baseline model was selected, they intended to develop both univariate detection models and multivariate prediction models simultaneously. [1] We continue their efforts in this work by branching off the same repository.

To implement the anomaly detection, they employed an LSTM Autoencoder (Long Short-Term Memory) model. The essence of this model is to train on the time series using the LSTM recurrent neural network and an autoencoder structure [11]:



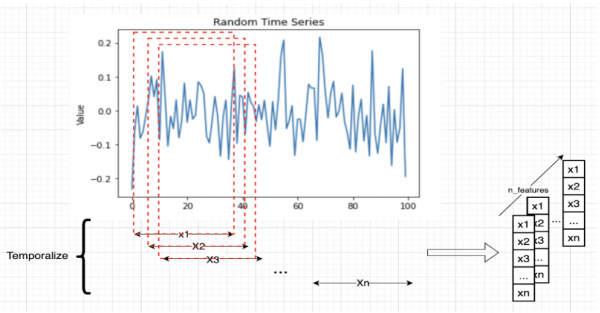
The LSTM (Long Short-Term Memory) is a type of recurrent neural network architecture designed to process and model sequential, and in this case temporal, data. A basic unit of the LSTM is shown below [11]. It incorporates memory cells that can store information over long periods, allowing it to capture and learn complex dependencies in the data. The main advantage of using LSTMs for time series prediction is that they can learn long-term dependencies in data. This allows them to make more accurate predictions than other models, such as ARIMA models. The main disadvantage of using LSTMs for time series prediction is that they can be computationally expensive to train. This is because LSTMs have a large number of parameters that need to be learned [7, 6, 9].



The LSTM model is utilized to make predictions on the time series and forecast. This works with an autoencoder architecture, which first compresses the data into a low-dimensional latent representation which is then reconstructed. The reconstruction, a prediction of the time series behavior, is then compared with the actual time series to quantify a reconstruction error. Anomalies are denoted as regions in the time series with exceptionally high reconstruction error [7, 6, 8].

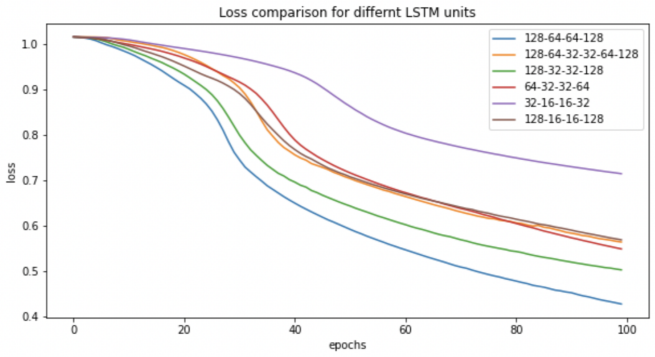
Note that the data must be preprocessed into lookback window for training in the LSTM. This quantifies the amount that the LSTM RNN can rely on observations from the past during training. This is done through a process caused temporalization, which was implemented by the previous group as shown

below [7, 9]. The standard lookback window of size 5 was chosen for this project [1].



The past group implemented this model on sample data that they generated, specifically normal Brownian motion, to model random behavioral fluctuations over time. The model was trained on this normal data. They also created abnormal data by extrapolating from the normal data with regime shifts at two specific indices. The encoder and decoder structure of the model consisted of two layers, and the LSTM blocks in the decoder were flipped. The results showed that the abnormal data had outlier shifts at two indices ([2000, 3000] and [6000, 7000]) [1]. The prediction curve generated by their model successfully reconstructed the normal curve and detected these two outlier incidents.

In their research, the past group analyzed the loss function across various sizes of LSTM layers [1]:



They discovered that the most optimal LSTM model consisted of four layers with units=[128,64,64,128] respectively. The group observed that if the unit number in the other layers was too small, the LSTM model struggled to adequately capture the input data, leading to an inability to achieve the desired loss within the specified number of epochs. Conversely, when the layer sizes were too large, the loss function encountered the issue of vanishing gradient descent, resulting in a slower convergence to the desired loss compared to the model with the

optimal layer configuration [1]. Due to their findings, we choose the optimal LSTM structure as the consistent architecture behind all of our LSTM models.

2.2 Objectives

To understand the objectives of this work, it is important to reconcile with the limitations of the last group's work. We pinpoint some key areas for improvement in the last group's work and expand upon them in this study. Our primary goal is to enhance the resilience of the previous LSTM autoencoder anomaly detection approach as well as explore alternative approaches to time series anomaly detection. The earlier method involved training the model on the complete dataset and reconstructing predictions using the latent low-dimensional representation of the time series to detect anomalies. However, a significant drawback of this method is the absence of explicit response labeling, which means that we might unintentionally train on anomalous regions within the dataset. Consequently, the reconstruction loss and subsequent anomaly predictions may be unreliable [1]. To address this issue, we experiment with different, more rigorous methods of using the LSTM model to predict anomalies in the data. We also propose alternative methods that utilize Bayesian models and wavelets. Our study involves implementing these methods on primarily the credit card transaction financial time series data by company provided by Wellington Management, though we also explore the quarterly sales by company dataset.

3 Methodology

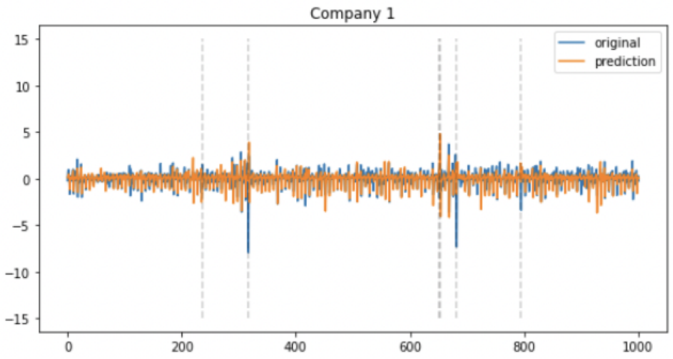
3.1 Long Short-Term Memory (LSTM) Models

First, we consider deep learning time series prediction/forecasting models using the Long Short-Term Memory (LSTM) architecture. This section is a continuation of the last group's work in LSTMs. In the previous paper, the authors build the LSTM model in Tensorflow and create a suite of methods to train, validate, and test the model on simulated time series data [7, 1, 6]. In this paper, we extend the previous work by using a applicable financial dataset provided by Wellington Management, the credit card transactions time series, and a more sophisticated LSTM model. We will also evaluate the performance of the model using a more rigorous methodology by making sure that we do not train on the entire dataset, which can lead to confirmation bias in the validation phase. The two primary approaches of LSTM anomaly detection we use are "brute force" and "STL-assisted".

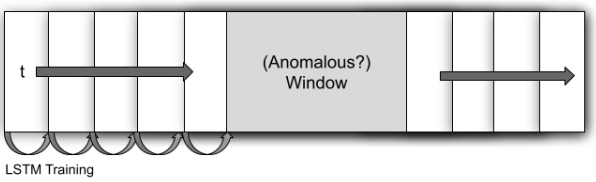
3.1.1 Brute Force LSTM Approach

The "brute force" approach for implementing LSTM time series forecasting was borne from the observation that the prior group's work trained on the entire time series, used the LSTM model to reconstruct the entire time series and

arbitrarily denoted the top k-highest reconstruction errors as anomalous, as shown below with $k = 6$ [1]:



Training and testing on the same dataset in-sample can lead to confirmation bias and gives rise to doubt in the efficacy of our results for outliers. We make this approach more rigorous by utilizing a rolling windows brute force method, making sure to only train on a fixed number of windows in the time series up until a denoted test window. The methodology is shown in the diagram below:

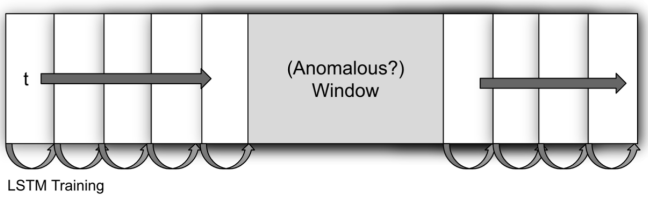


To implement this approach, the time series data is first divided into fixed-size moving windows. These windows act as input sequences for the LSTM network. Each window contains a sequence of consecutive data points, capturing local dependencies within the time series. During the training phase, the LSTM network is presented with these moving windows sequentially. The network processes each window, and its hidden state is updated at each time step. The output of the LSTM at the final time step is compared to the original data points in the window, typically using a mean squared error (MSE) loss function. The objective of training is to minimize this reconstruction error, encouraging the LSTM to learn the underlying patterns of the time series. The moving windows propagate through the LSTM network by feeding them as sequential inputs. At each time step, the LSTM considers the current input and the previous hidden state to generate the output and update the hidden state. This recursive nature of LSTMs allows them to capture long-term dependencies in the time series. Once the LSTM model is trained, it can be used for anomaly detection. During the testing phase, the time series is divided into the same fixed-size moving windows. These windows are then fed into the trained LSTM,

and the model generates predictions for the denoted test window that follows the training windows. The difference between the original data points and the LSTM's predictions can be measured using the reconstruction error, such as MSE. Anomalies can be identified by comparing the reconstruction error to a predefined threshold [7, 6, 8]. Data points or windows with reconstruction errors above the threshold are considered anomalies, indicating deviations from the learned normal behavior.

In this process, we choose a window size and train the LSTM model on rolling windows on the credit transaction time series. This allows us to train the model on more concentrated time frames and capture more granular behavior. We do NOT train on a window of the time series that we "guess" to be anomalous, i.e. the test window. This test window moves across the entire time series over iterations. We experiment with different window sizes to examine different granularities with which we can investigate the data and pinpoint anomalous regions. We look at periods of 5,10,15,20, and 25 days. The lookback window we choose is the one recommended by the past group, that of 5 days [1]. We utilize a "rolling step" of 5 days, which indicates the number of days to propagate the rolling window forward.

One mistake that we made in our development process was the subtlety of training only before the test window, not after. A visual of this incorrect approach to model training is shown below:



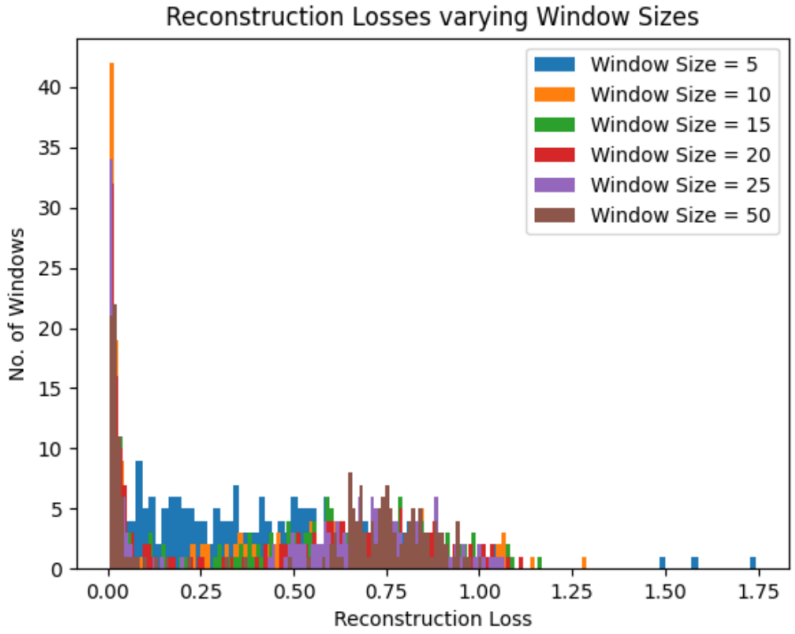
This data leakage comes from the fact that in time series, we cannot train on the future to predict the past. Especially in our windows approach this was initially not obvious, but it is essential to notice because this can lead to large confirmation biases during learning and forecasting.

Our revised approach only trained on windows leading up on the test window and tested on the test window, ignoring time series after the denoted test window. Then, test window was rolled forward by the rolling step amount. Since we start at the beginning of the time series and progress chronologically, the model improves reconstruction for later points in training. The initial implication was that we can avoid this by using an "random restarts" methodology as used in gradient descent to estimate find global optima, however classical random restarts in time series will still lead to the aforementioned data leakage. Finding a more effective way to partition the training data into windows while avoiding data leakage and training on potential anomalous regions can be an area of further development, as this will increase our confidence in regions we denote as anomalous. The methods that we show in this research lead us to be able to pinpoint anomalous regions with granularity and no confirmation bias,

but we must have the trade off that the initial detected outliers are likely not anomalous and can be attributed to a model that has not been trained on the time series enough. Hence, the further along the time series the prediction, the more confident we are in anomalous designations.

3.1.2 Brute Force LSTM Results and Analysis

Utilizing the brute force methodology on company 1 of Wellington's credit card transaction time series, we can pinpoint anomalous regions and plot the discrepancy between the predicted and actual values. The error between these is known as the reconstruction loss or error. Due to the architecture of the encoder/decoder, we know that all decoding will likely have some reconstruction loss, but we want to designate windows with abnormally high loss as anomalous. Hence, histogramming the reconstruction loss is helpful, as shown below.



We can see from this that the distribution of reconstruction losses varies with the window size chosen, or the granularity with which we examine anomalies in the time series. For a fixed window size, we can z-score the reconstruction losses and selectively plot the actual and reconstruction windows for losses above two standard deviations of the mean. Shown below are select anomalous windows that happen to overlap over a certain time period in the time series:

Clearly, there are some areas of the time series that we can more confidently claim as anomalous as the reconstruction shows anomalous behaviors even when examined with different granularities or window sizes. In this case, we identify an anomalous region in late July-early August 2016. The exact locations and whether the prediction is under or over predicting the true time series can be investigated with a more granular plot given a particular window size. The other results for the most prominent identified anomalous windows have similar analysis of overlapping are shown in Appendix B.

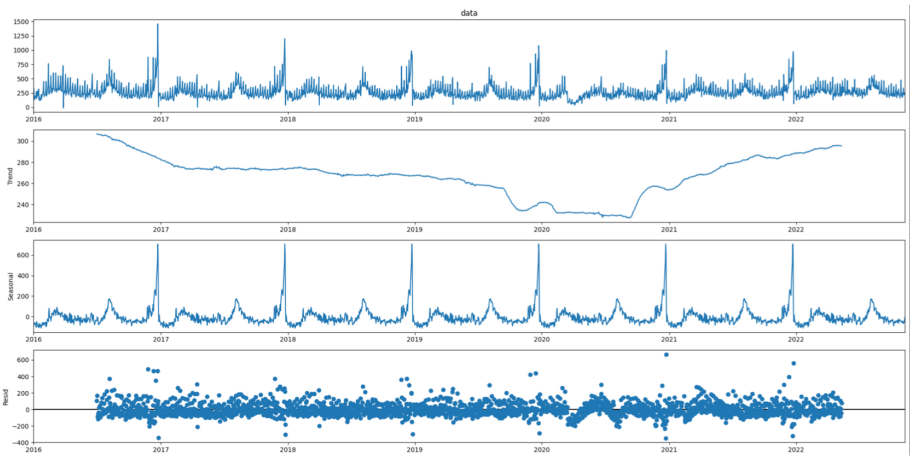
3.1.3 STL-assisted LSTM Approach

A drawback we found with the LSTM brute force approach was precisely in its name - the methodology required rolling the test window through the time series

with brute force from start to end. This is slow and impractical for large time series. Instead of iterating through the entire time series, we sought to find intelligent ways to denote areas of the time series that are most likely anomalous, and then verify that with the reconstruction error given by the test window on that region using the LSTM Autoencoder model.

We propose utilizing the STL decomposition (Seasonal and Trend decomposition using Loess) of the time series to give us this intelligent means. STL decomposition is a technique used to break down a time series into its underlying components: trend, seasonal, and residual. The trend component captures the long-term, slowly changing pattern of the series, while the seasonal component identifies the repeating patterns or cycles. We can input the seasonality that we think best models the temporal data's behavior as our modeling assumption. The residual or remainder component represents the unexplained variation in the time series that remains after removing the trend and seasonal components. It contains the random and unpredictable fluctuations that are not captured by the trend and seasonal patterns. It is precisely this residual component that is of interest.

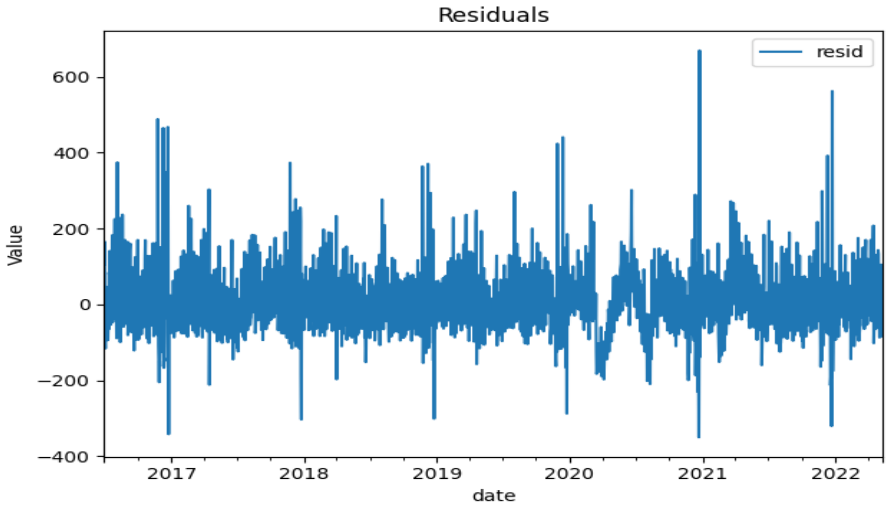
We utilized the last group's work on Exploratory Data Analysis of the credit card transactions dataset to be able to generate the STL decompositions [1]. An example STL for the first company with yearly seasonality is shown below:



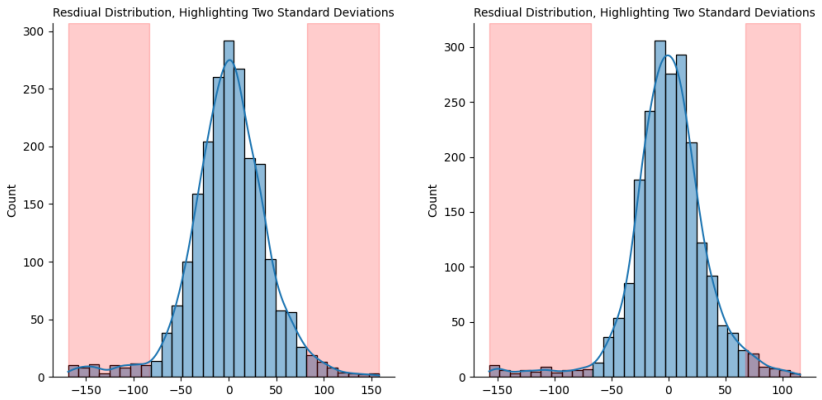
Our STL-assisted methodology computes the STL decomposition of the particular time series and z-scores the residual values. In other words, the residual values are computed across the time series, histogrammed, and the values beyond of two standard deviations of the mean residual value. Residual values within two standard deviations are considered variation due to inherent noise in our modelling assumption. The others denote possible anomalous regions. We then feed this potential anomalous regions as test windows in our LSTM Autoencoder model, training up to and testing only on the proposed test window. We then observe and compare the reconstruction error. This method eliminates the needs for rolling windows through the entire time series as in the brute force

method as the test windows are intelligently gathered from data points with higher residual value z-scores.

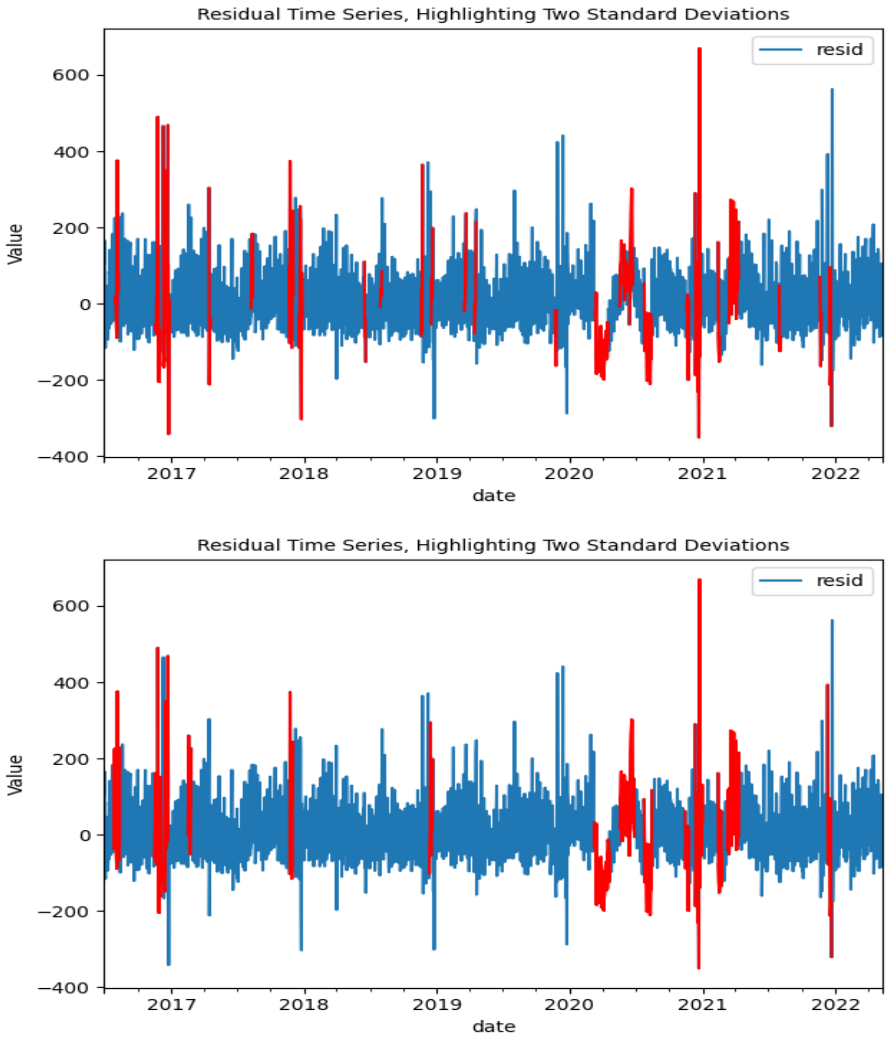
We will now show one full STL-assisted analysis for anomaly detection for a window size of 5 and 10. First, we find the residual for the time series for company 1:



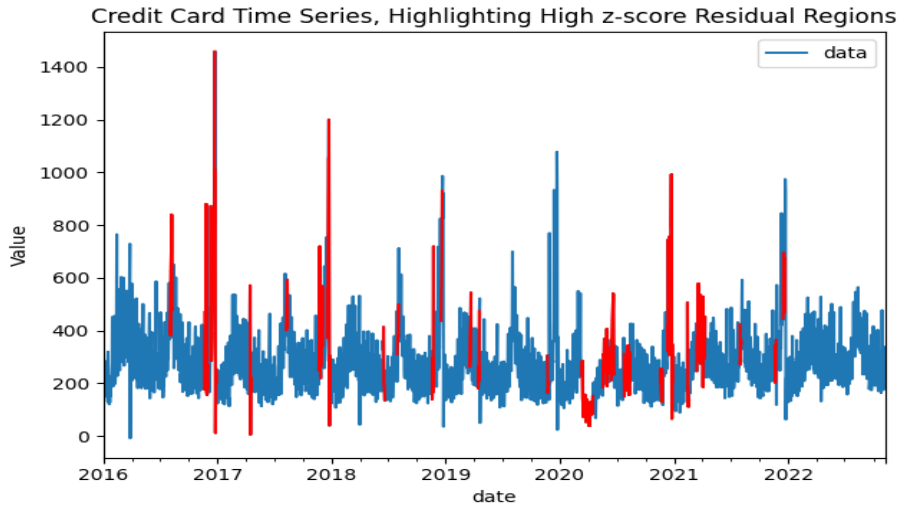
Next, we can histogram these residuals and designate those with high z-scores as likely to be anomalous regions. However, depending on our window size the distribution of the residuals will be different. See the histogram with a window size of 5 on the left and that of 10 on the right:

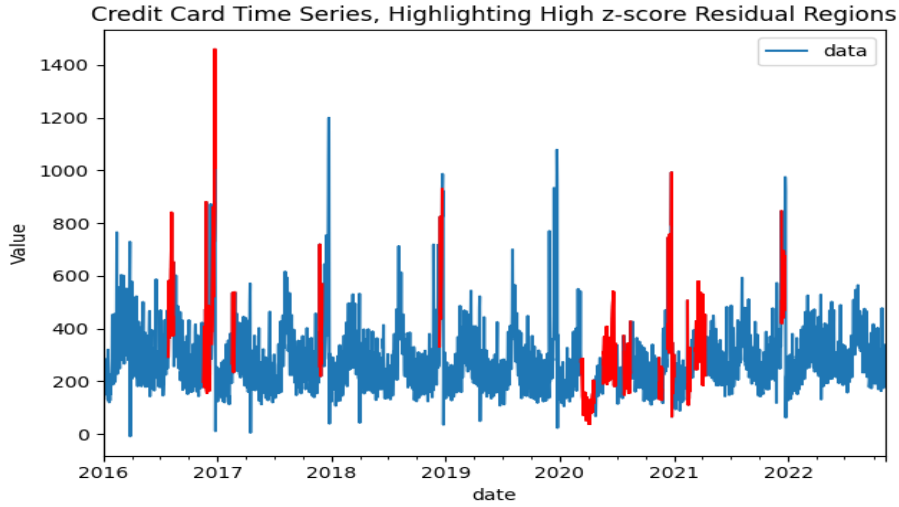


These distributions are not identical and therefore the regions of the residual time series that have high z-scores (likely anomalous regions) are not necessarily identical. However, when we plot the high z-score regions of the residual for each we can see that there is a lot of overlap:



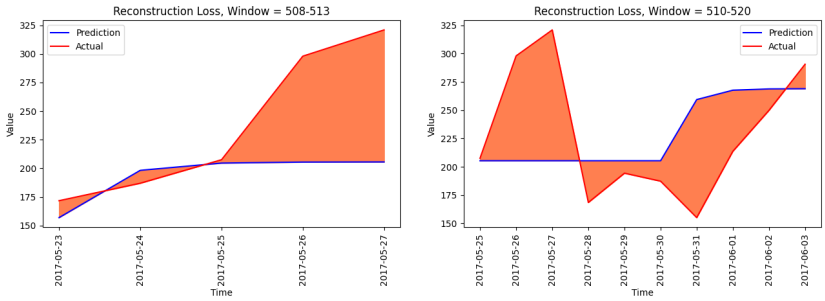
Because the indexing of the time series is the same, we can find the corresponding likely anomalous regions in the original time series. These will be the "test windows" that we can utilize our LSTM rolling windows model to train up to and test on:





There indeed is a lot of overlap on the likely anomalous windows of varying window sizes, though they are not necessarily identical. This analysis of the residual time series, histogramming and z-scoring the values for potential anomalies in the original time series was done for various window sizes (5, 10, 15, 20, 25). The results are shown in Appendix C.

When we train the LSTM model and test on the windows, we can plot the reconstruction and the actual time series to measure the reconstruction loss. Select results for window sizes of 5 and 10 are shown in Appendix D. Two examples identified anomalous windows from the window size of 5 and 10 are shown below:



Indeed, these two do overlap on the same time span, in late May of 2017. Therefore, due to the information provided by the STL decomposition's residual, we were able to pinpoint an anomalous region using the LSTM model while avoiding brute forcing through the entire time series. The overlapping of results with different window sizes further increases our confidence in the assertion that this region is anomalous.

3.2 Bayesian Hierarchical Models

Bayesian anomaly detection is a probabilistic approach used to identify outliers or anomalies in data. Rooted in Bayesian inference, it leverages the Bayesian framework to model uncertainty, incorporate prior knowledge, and update beliefs based on new data. Bayesian anomaly detection works by first modeling the regular patterns and behavior in the data, and then determining the degree to which a given observation deviates from this expected behavior. Such techniques have been shown to be superior to standard algorithms in both classification and anomaly detection performance in the presence of uncertainties. ([3]).

Our approach to anomaly detection involves Bayesian hierarchical modeling. Bayesian hierarchical models, also known as multilevel models, offer a flexible framework to combine information at multiple levels to estimate features of interest, while accounting for the full range of uncertainty present in these data sourced. [2] These models are hierarchical in the sense that they involve parameters that vary at more than one level. In financial time series data, the underlying processes might exhibit different behavior at different time scales, necessitating a multilevel modeling approach.

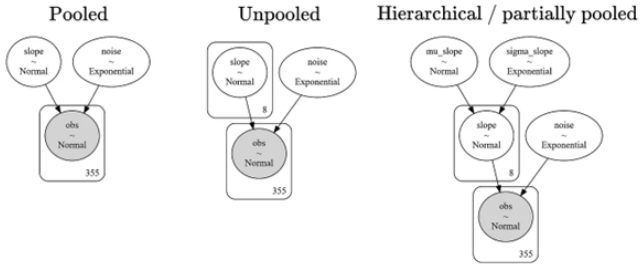
3.2.1 Model Formulation

A *pooled model* assumes that all groups or individuals in the dataset are drawn from the same underlying distribution, and that any differences between groups are due to random variation.

An *unpooled model* assumes that each group or individual in the dataset has its own unique underlying distribution, and that these distributions are independent of each other.

A *partially pooled model* is a compromise between the pooled and unpooled models. It assumes that the groups are not completely independent, but rather share some underlying structure or information. In other words, a partially pooled model assumes that there are systematic differences between the groups, but some of these differences are shared across groups.

In this study, we first test an unpooled model, then proceed with a pooled and partially pooled model. Our test data is the Columbia credit data set, which includes a series of company IDs and transaction values over time. We measure the Mean Squared Error of each model to determine which model can best capture the seasonality and trends.



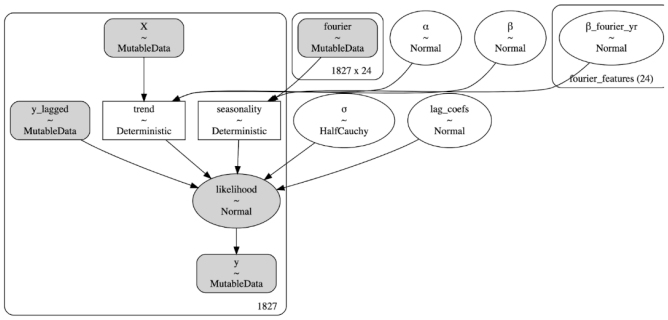
Since there is only one feature dimension (time) and one target dimension (data), we need to construct some predictive features. We will add a seasonality term, a trend term, and an autoregressive term. We also calculate a set of fourier features to decompose the seasonality.

3.2.2 I. Unpooled Model

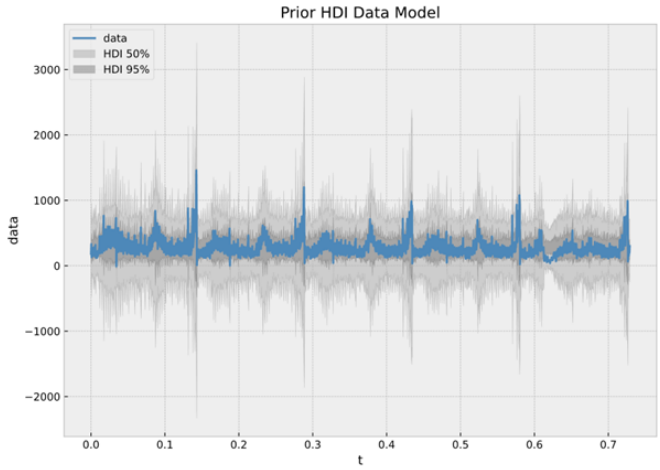
For our unpooled model, we choose specific single companies to model, one at at time. We specify the prior distribution to be a function of additive seasonality, trend and an autoregressive term. We feel that this is an appropriate prior as company transactions tend to have strong seasonality, as well as dependency on previous transaction volume. We add seasonality terms as a standard fourier decomposition with a yearly period. Our specification is as follows:

$$\begin{aligned}
\text{data} &\sim N(\text{seasonality} + \text{trend} + \text{autoreg}, \sigma) \\
\text{seasonality} &\sim \sum_k (\beta \sin\left(\frac{2\pi kt}{m}\right) + \beta \cos\left(\frac{2\pi kt}{m}\right)) \\
\text{trend} &\sim \beta t + a \\
\text{autoreg} &\sim c \cdot \text{data}_{t-1}
\end{aligned}$$

We place normal priors on all the coefficients (except for sigma, for which we use half-cauchy). For the target, we use a Normal with the training sample mean and standard deviation as the prior. The final model specification is as below:



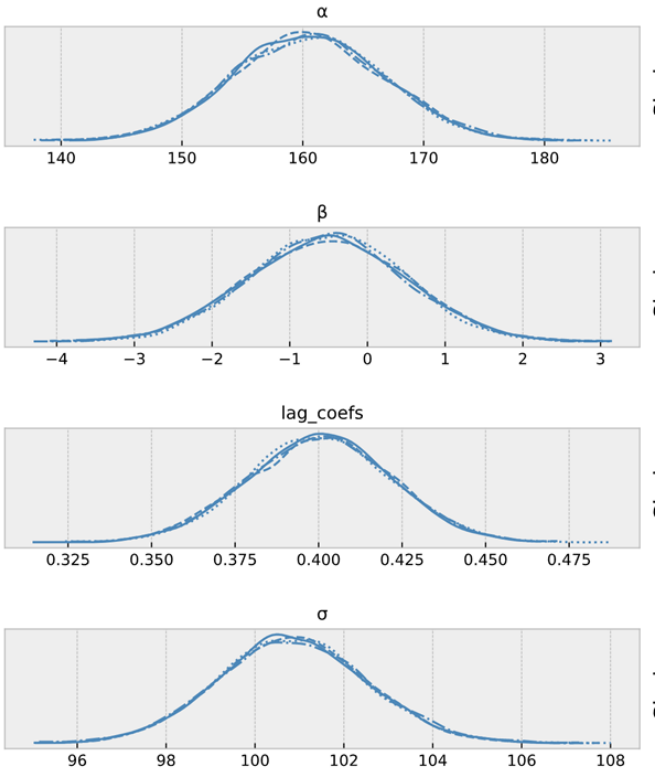
We then run a prior predictive check to validate that the values chosen are not too constraining.



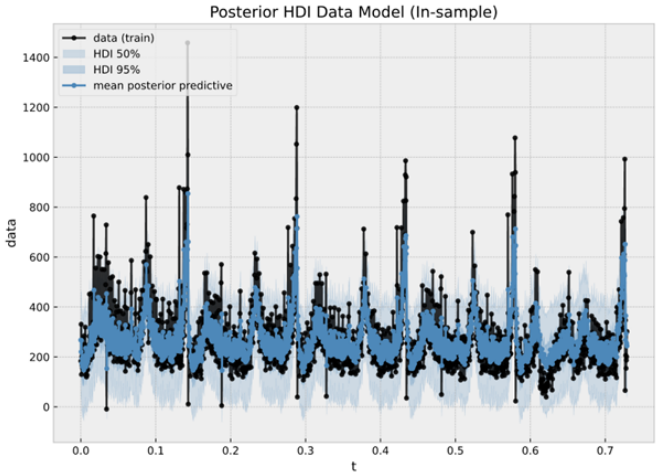
In Bayesian statistics, the No-U-Turn Sampler (NUTS) is a variant of the Hamiltonian Monte Carlo (HMC) method used for sampling from a probability distribution. NUTS addresses a few key problems that HMC has, particularly the difficulty in choosing appropriate values for the parameters that govern the step size and number of steps.

NUTS automatically determines the number of steps to take. It does this by simulating a potential trajectory both forwards and backwards in time until it senses that it's about to start doubling back on itself - hence the name, No-U-Turn Sampler. By automatically tuning this parameter, NUTS can make Hamiltonian Monte Carlo a lot more efficient and easy to use in practice.

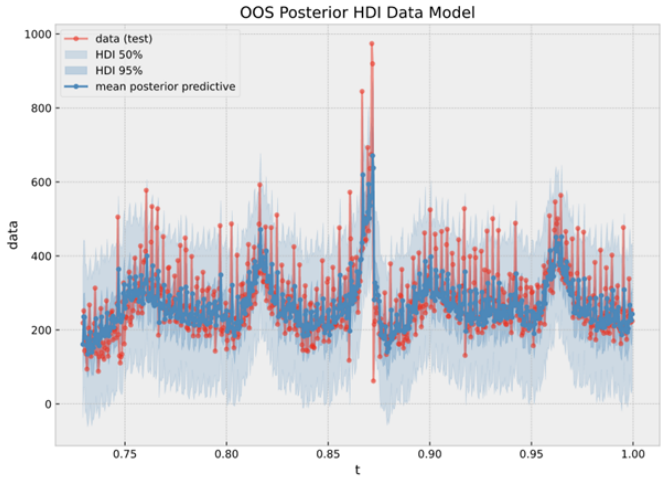
Using the NUTS (No U-TURN Sampler) we generate the in-sample posterior distribution as well as the 50% HDI (high density interval) and 95% HDI levels. From visual inspection the distribution is able to capture seasonality and the autoregressive features of the data.



The distributions inform us of the location and scale of the parameters. Alpha has a mean around 160 shows the mean transaction value without seasonality. Lag coefficient value of around 0.4 show the level of autocorrelation to lag 1 data. Sigma of 100 informs us that the standard deviation is large relative to the mean data.



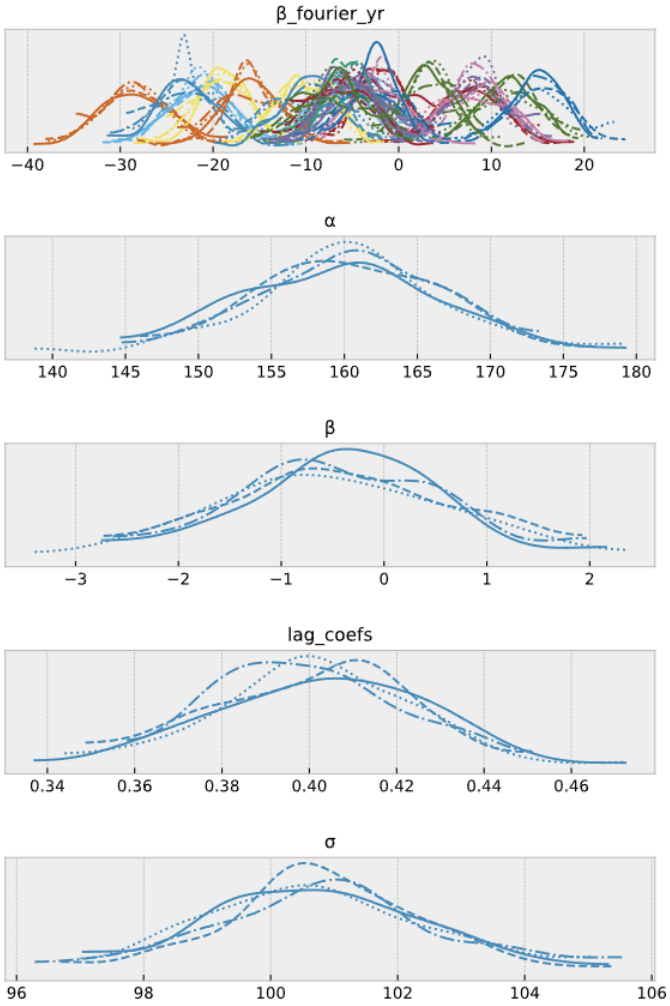
Now we generate the predictions on out of sample data. The model is clearly able to capture the seasonality on out of sample data. Comparing the test results and mean posterior draws, we get a MSE of 7184.3. We then use the model for anomaly detection by simply marking any data points outside the 50% HDI density as anomalous.



3.2.3 II. Pooled/Partially Pooled Model

A partially pooled / hierarchical model first generates a posterior from some overall population, then uses those as prior inputs to the unpooled model. Our goal is to generate priors for the mean, trend and lag coefficient terms by first running a posterior sample for the test data set on the entire set of company data. Then, we use the fit mean coefficients as the prior inputs for the singular

company model. This process is a hierarchical partially pooled model. The result, as seen below, pulls the drawn sample coefficients towards



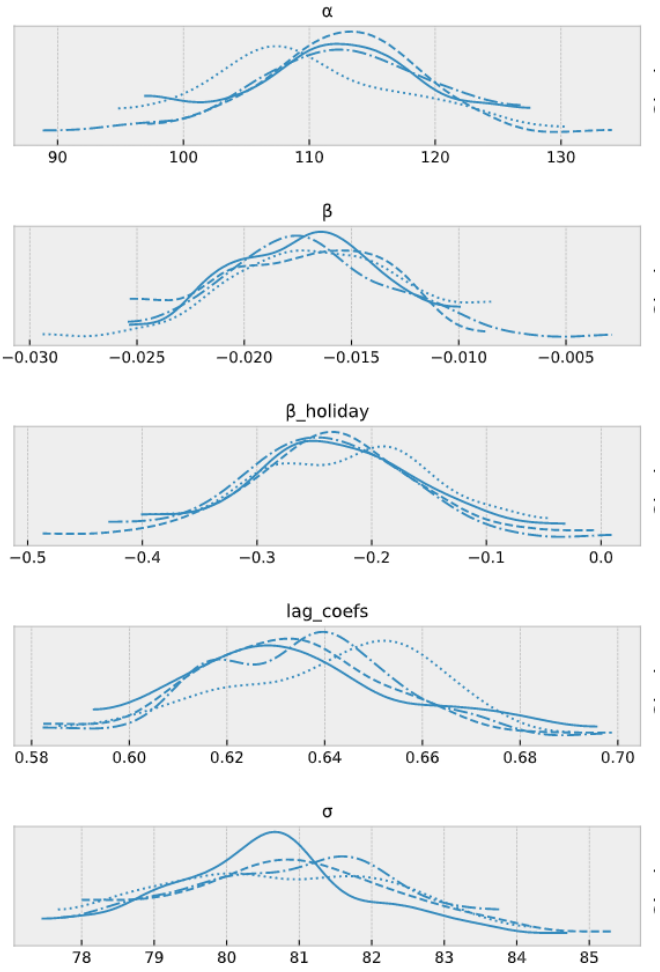
3.2.4 III. Adjusted Seasonality Model

Next, we fine-tune our Bayesian model for higher accuracy by adjusting some seasonality and autoregression terms and adding holidays. We will return to using a non-pooled model, as the hierarchical model takes an excessive amount of time to run without a large benefit to model accuracy.

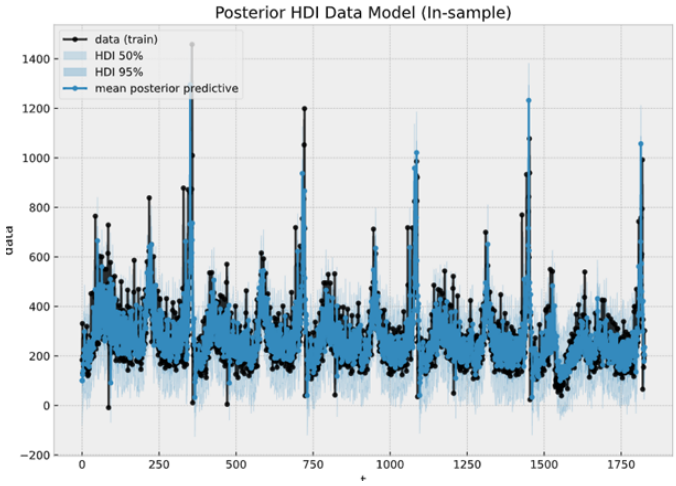
For our new model, instead of modeling with an AR lag 1 process, we use a lag of 7 to capture the weekly effect. We keep the yearly seasonality term but remove the Fourier features for the weekly, as this is captured with the AR term. We also use a new multiplicative seasonality model, as opposed to additive

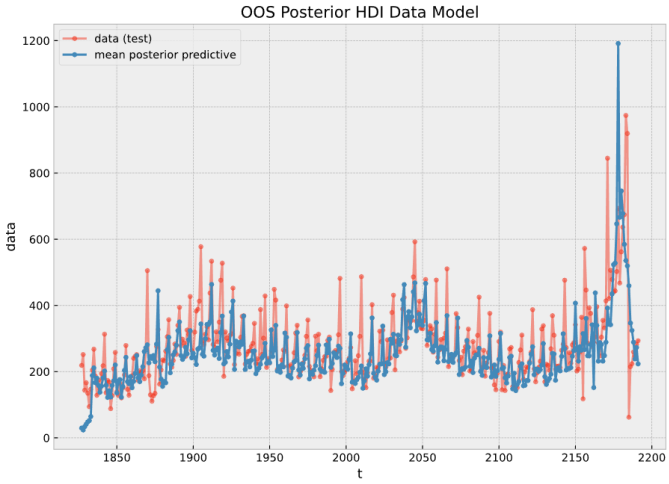
seasonality, with a new multiplier term for holidays. Holidays are marked as 1 for a national holiday; 0 otherwise.

$$\mu = (\text{trend} + \text{ar}) \cdot (1 + \text{seasonality} + x_{\text{hol}} \cdot \beta_{\text{holiday}})$$



The posterior samples of the coefficients show a significant negative value for the holiday coefficient (-0.3) and AR lag coefficient (0.64). This suggests a large negative effect for holidays (no shopping on holidays) and high correlation between weekdays.





Our adjusted model has an MSE of 5986.553.

3.2.5 IV. Model Results

Model	MSE
Model 1 : Unpooled	7184.3
Model 2 : Partially Pooled	7177.92
Model 3 : Seasonality Adjusted, Unpooled	5986.5

Table 1: MSE Comparison of 3 Models

Although our target objective is anomaly detection, we use MSE to determine the overall accuracy of the model. A higher MSE corresponds to a better ability for the model to capture seasonality and trends effects; thus a better ability to detect what 'true' anomalous regions / data points are. Our results show that the adjusted unpooled model performs best as the multiplicative seasonality was captured better. The partially pooled model performed slightly but not significantly better than the unpooled model.

3.3 Wavelet Analysis

Our final evaluation method involves wavelet decomposition. Traditional spectral analysis techniques, such as the Fourier Transform, provide insight into the frequency components of a signal but often struggle to capture the dynamics of non-stationary time series data with complex seasonalities. The wavelet transform, on the other hand, can adapt to these challenges by utilizing a series of time-localized basis functions called wavelets. The continuous wavelet transform uses wavelet functions that allow both frequency and location (or scale) analysis providing time and frequency information simultaneously, hence giving a more detailed analysis for non-stationary signals. [4]. These wavelets are generated by scaling and translating a mother wavelet function, resulting in a flexible and adaptable framework for time-frequency analysis.

3.3.1 Wavelet Transform

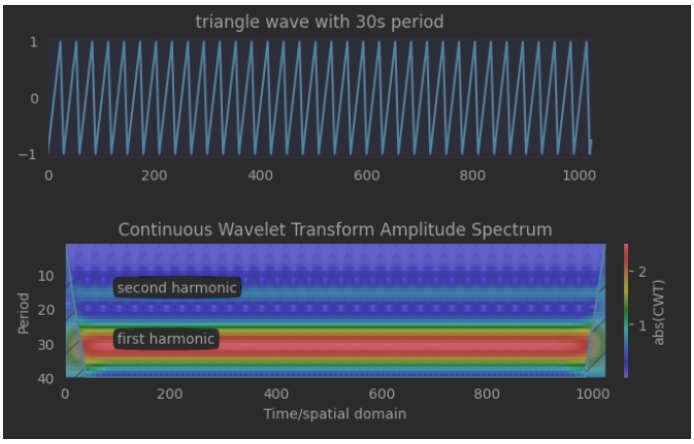
A continuous wave transform of a signal f(t) takes the form of the following:

$$CWT_{\psi}(a,b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} f(t)\psi^* \left(\frac{t-b}{a}\right) dt$$

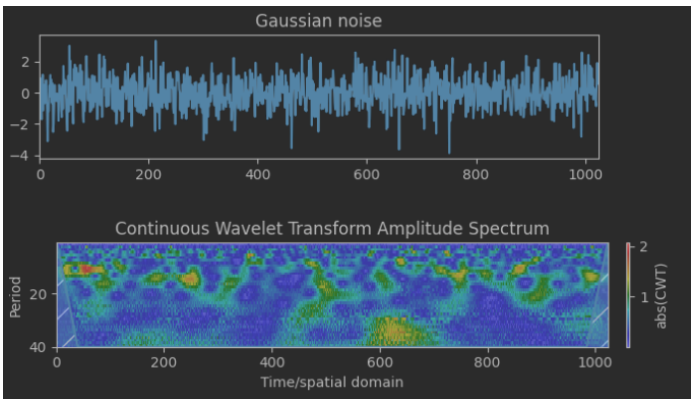
Where a and b represent the scale and translation parameters in the Continuous Wavelet Transform and the * denotes the complex conjugate of the wavelet function ψ.

Using the python library PyWavelets and scaleogram [5]., we can generate spectrograms that depict the periodicities to which the wavelet transform is sensitive to. We use the complex morelet wavelet as the default wavelet function. Different wavelets (Meyer, Gaussian, etc.) will have compromises between scale and time resolution.

These diagrams depict a 1-D signal via a 3 dimensional visualization (across time - x axis, period - y axis and intensity - color). For example, we can generate a repeating triangle wave and decompose the wave using CWT, and depict it with the spectrogram. There is an obvious first harmonic at the 30 second period level where periodicity is greatest (red area). It is also continuous across time, which is why we see the red area stay constant (horizontal) throughout the graph.

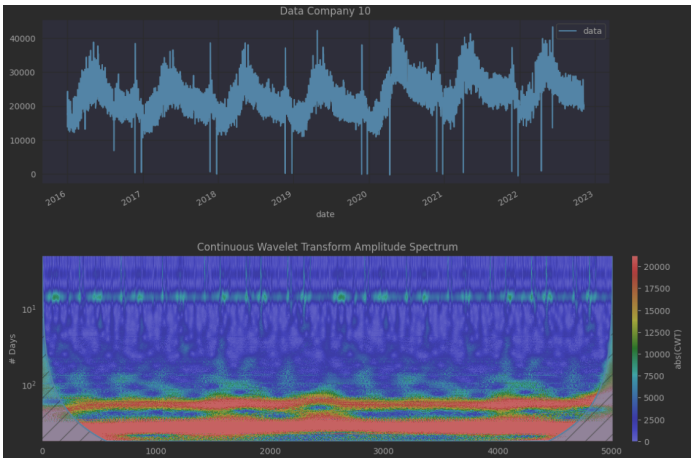


If we generate pure Gaussian noise, the wavelet transform is unable to reveal any seasonalities with the spectrogram:



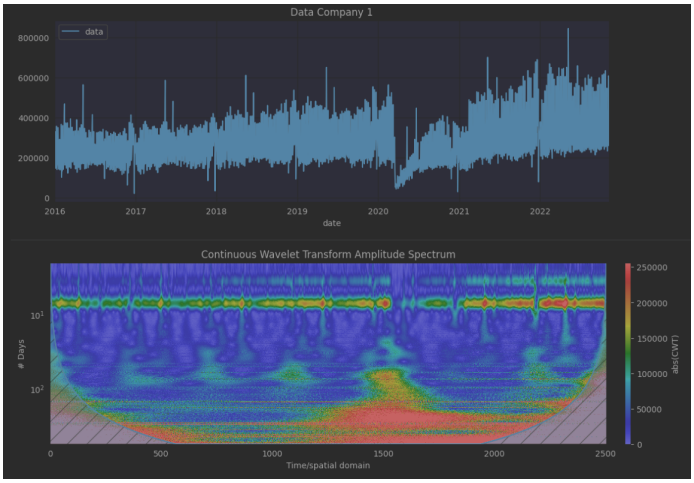
3.3.2 Wavelet Results and Analysis

We now perform the continuous wave transform on the Columbia transaction dataset. Generating a spectrogram for company 10, we see a strong weekly seasonality as well as strong harmonic at the yearly level- this pattern is very consistent. We also see end of year effects that last about a week- this is highlighted by the stronger light blue color on the weekly scale- this is also obvious from looking at the data itself. From the spectrogram itself, there is no obvious anomalous region in this data set.



For company 1, the spectrogram computed shows a strong harmonic at the 7 day level. This confirms our intuition in the previous studies that there is a strong weekly frequency of data. There is also a strong yearly seasonality (bottom red region) which dissipates in 2018 but resume in further years.

We also see what looks to be repeating features on a yearly basis (blue cone region at the end of each year). This seems to be an end of year holiday effect which spans 1 quarter.



We can also see an anomalous region during 2020 pandemic when transactions fell to 0 and the weekly periodicity dissipates. Towards the end of 2022 we see the weekly periodicity increase in amplitude (weekly stripe tending more red). With this visualization it is straightforward to detect the anomalies as a break in the continuous 7-day harmonic.

When conventional methods, such as trend or Fourier analysis, might fail to detect anomalies, wavelet decomposition shines by deconstructing the data

at varying scales. This multi-resolution analysis allows for the identification of abrupt changes, cyclical patterns, or transient phenomena that may be obscured in regular analysis due to noise or the complex nature of the signal.

4 Conclusion

In conclusion, our research provided a comprehensive comparison of various powerful time series anomaly detection methods. Each technique has demonstrated unique strengths and weaknesses depending upon the characteristics of the dataset, the need for interpretability, and the computational resources available. In our study we describe deep learning approaches for anomaly detection with brute force and STL-assisted LSTM anomaly detection as well as statistical approaches for anomaly detection with Bayesian hierarchical models and wavelet analysis of the time series. The implementation on Wellington Management's credit card transaction data set lead to common findings of anomalous regions.

4.1 Comments

Some comments can be made on the research done. If done differently, the authors would like to have explored more ways to incorporate deep learning methods in anomaly detection beyond brute force. The STL-assisted methodology was just a first step in that direction and still requires some implementation changes in order to have more confident and narrower designations of anomalous regions in the time series, as well as increasing the speed of training. The most enjoyable part of this project was comparing the results for the variety of different methodologies explored and the excitement of learning that they converge on identifications of anomalous regions despite having different architectures. The most tedious was likely making sure that the graphs for the analyses were plotted, saved, and compared correctly.

4.2 Future Research

Some future work to consider is finding a more effective way to train the LSTM model in rolling windows across the time series while further tackling the challenge of not having explicit response labelling in the data. This will decrease the likelihood of confirmation bias in training and lead to more confidence in our anomalous designations. The STL approach needs to be further refined as per the comments aforementioned. In addition, the Bayesian and wavelet analyses can be modified to incorporate additional terms that are more tailored to the time series at hand in order to better fit a particular time series and give even more specific and confident anomaly detection.

5 References

[1] Lin, R., Liu, T., & Xue, G. (2022). *LSTM-AutoEncoder for Multivariate Anomaly Detection* (IEOR4742 Project Report). Columbia IEOR.

[2] C. Edson Utazi, Warren C. Jochem, Marta Gacic-Dobo, *Bayesian hierarchical modelling approaches for combining information from multiple data sources to produce annual estimates of national immunization coverage*. arXiv preprint arXiv:2211.14919, https://arxiv.org/abs/2211.14919 2022.

[3] Roberts, Ethan and Bassett, Bruce A. and Lochner, Michelle, *Bayesian Anomaly Detection and Classification*, arXiv preprint arXiv:1902.08627, https://arxiv.org/abs/1902.08627, 2019.

[4] Aguiar, Luís Francisco and Soares, Maria Joana, *The continuous wavelet transform : a primer*, https://www.semanticscholar.org/paper/The-continuous-wavelet-transform-%3A-a-primer-Aguiar-Soares/3157685e993c71e3c5a41f039d8f0432718877c1, 2011.

[5] Scaleogram, https://pypi.org/project/scaleogram/

[6] Chollet, F. (n.d.). *The keras blog*. The Keras Blog ATOM. https://blog.keras.io/building-autoencoders-in-keras.html

[7] Grogan, M. (2020, November 7). *One-Step predictions with LSTM: Forecasting hotel revenues*. Medium. https://towardsdatascience.com/one-step-predictions-with-lstm-forecasting-hotel-revenues-c9ef0d3ef2df

[8] Intro to autoencoders. *Tensorflow Core*. (n.d.). https://www.tensorflow.org/tutorials/generative/autoencoder#third_example_anomaly_detection

[9] Jayawardhana, S. (2020, July 30). *Sequence Models & Recurrent Neural Networks (RNNs)*. Medium. https://towardsdatascience.com/sequence-models-and-recurrent-neural-networks-rnns-62cadeb4f1e1

[10] Sharma, S. (2019, March 5). *Epoch vs Batch Size Vs iterations*. Medium. https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

[11] Wei, Y., Jang-Jaccard, J., Xu, W., Sabrina, F., Camtepe, S., & Boulic, M. (2022, April 14). *LSTM-Autoencoder based anomaly detection for indoor air quality time series data*. arXiv.org. https://arxiv.org/abs/2204.06701v1

[12] Wilkinson, P. (2021, November 5). *Univariate outlier detection in Python*. Medium. https://towardsdatascience.com/univariate-outlier-detection-in-python-40b621295bc5

6 Appendix

6.1 Appendix A - Code Repository File Structure

- **Description**
  - **Project**: Wellington Management - Multivariate Time-Series Anomaly Detection
  - **Contributors**: Rohan R. Kulkarni, Ethan Li, Ruoyu Lin, Tianhao (Raymond) Liu, Guangrui Xue
  - **Supervisors**: Daniel Fernandez, Yossi Cohen (Wellington Management), Ali Hirsa (Columbia IEOR)

    This is a project repository for 2022-2023 Columbia Engineering AI Applications in Finance Wellington Management (group 1).

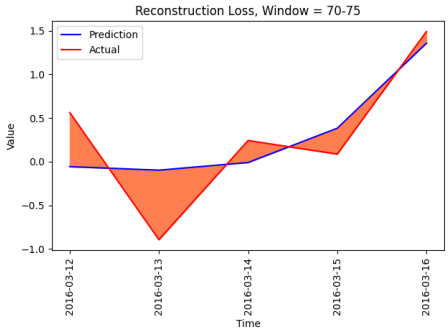
- **Files**
  - **Data** (`./data`)
    - `data4columbia_credit.csv`: credit card transaction data of 50 companies, provided by Wellington Management
    - `data4columbia_sales.csv`: quarterly sales data of 50 companies, provided by Wellington Management
    - `featured_credit.csv`: credit data set after processing and feature engineering. The actual data set fed to the model.
  - **EDA** (`./eda`)
    - `class_singlecompanyeda.py`: contains EDA class SingleCompanyEDA for the EDA pipeline of a single company.
    - `class_multicompanyeda.py`: contains EDA class MultiCompaniesEDA for the EDA pipeline of multiple companies.
    - `wellington_data_processing.py`: the data processing pipeline for the credit card data set.
    - `EDA_demo.ipynb`: the EDA pipeline aggregated.
    - `EDA_credit.ipynb`: EDA conducted on the credit dataset.
    - `EDA_sales.ipynb`: EDA conducted on the sales dataset.
    - `bayesian.ipynb`: EDA for the Bayesian hierarchical model.
  - **Model** (`./model`)
    - `__init__.py`: for modularization purpose only.
    - `lstm_autoencoder.py`: contains two classes:
      - `DataGeneration`: a legacy class used to call simulation toolkit. Deprecated in later stage of studies, but still salvageable for future simulations.

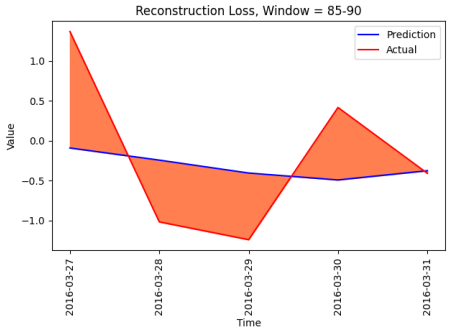
        - `LSTM_Model_Base`: the main class to define an LSTM-AutoEncoder model. The class is designed such that an LSTM-AE architecture can be passed to initialize the model, and the class will automatically check whether the passed specification is a valid (symmetric) architecture.
    * `lstm_windows.py`: contains two classes:
        - `LSTMWindows`: the main class for LSTM windows analysis that defines all the training and testing methods for running brute force or STL-assisted LSTM anomaly detection in rolling windows.
        - `LSTMWindowPlot`: the plotting class that contains helper methods to easily plot results of deep learning methods, original time series and reconstructions of the LSTM autoencoder, and more.
    * `model_exec.py`: the main model execution file that contains utility class `OutlierMetric` that enables the classification of outliers using different methods, and other helper functions that enable the execution of LSTM-AE models. See docstrings for documentation.
- `run_windows.py`: runs the LSTM rolling windows brute force method for anomaly detection.
- `run_stl_windows.py`: runs the STL-assisted LSTM rolling windows method for anomaly detection.
- `plot_windows.py`: plots and/or saves results and analysis from an LSTM windows train/test session.
- **Bayesian and Wavelet models** (`./bayes_wavelet`)
    * `readme.md`: readme for using these models.
    * `requirements.txt`: Python environment requirements for using these models.
    * **data**: data for using these models in local reference.
    * `bayesian.py`: defines the architecture of the Bayesian Hierarchical model.
    * `bayesian_adjusted.py`: implementation of the Bayesian model.
    * `bayesian_partial_pool.py`: defines the architecture of the Bayesian partially-pooled model.
    * `bayesian_unpooled.py`: defines the architecture of the Bayesian unpooled model.
    * `wavelet.ipynb`: implementation of the wavelet model and analysis.
- **LSTM Windows Results** (`./lstm_windows_res`)
    * Contains text files of reconstructions, anomalous designations, and other data from LSTM windows train/test sessions.

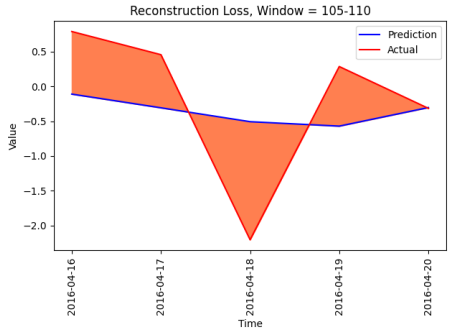
    - **plots**: generated plots from LSTM windows brute force and STL-assisted analysis.
    - **anom_plots**: generated plots of anomalous regions and reconstructions from LSTM windows brute force and STL-assisted analysis.
  - **Simulation Toolkit** (`./sim_util`)
    - `__init__.py`: for modularization purpose only.
    - `class_basesimulation.py`: contains base simulation class `BaseSimulation`, parent class of `Multisimulation` in `class_multisimulation`.
    - `class_multisimulation.py`: contains inherited simulation class `MultiSimulation`, child class of `BaseSimulation` in `class_basesimulation`. For future users: both these two classes can be modified to produce more sophisticated simulation tools.
    - `class_simulationhelper.py`: contains helper functions used to implement previous classes. It also contains other helper methods for plotting.
    - `data_generation.py`: contains a single function that showcases how to automatically generate simulated datasets using only random seeds.
    - `gen_datasets.py`: a caller file for functionalities in `data_generation.py`.
  - **LSTM Model Artifacts** (`./lstm_model`)
    - Contains TensorFlow/Keras model saves, sessions, and artifacts for the deep learning models used.
  - `demo_2022.ipynb`: the 2022 demo notebook, kept for completeness.
  - **Deprecated** (`./deprecated`)
    - Contains deprecated but perhaps still useful code.

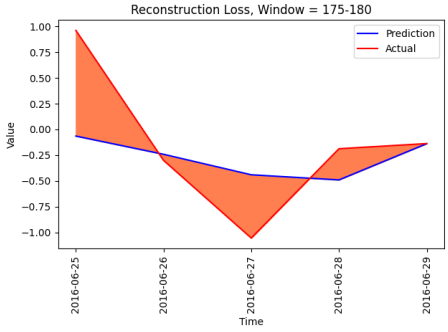
6.2 Appendix B - Brute Force LSTM Anomaly Results

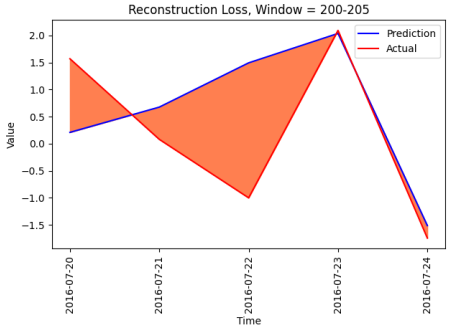
Window Size = 5:

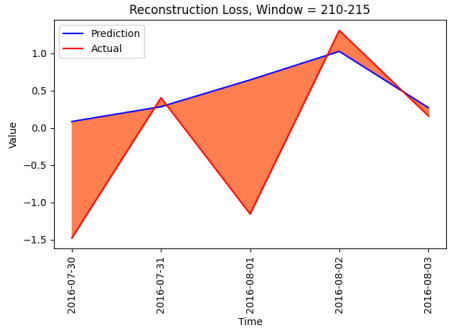


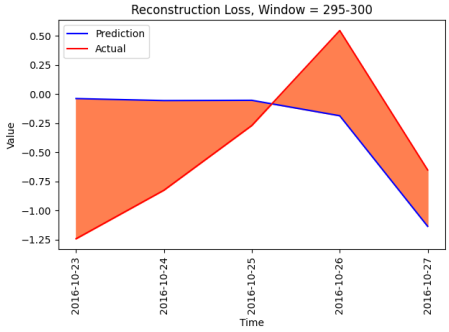


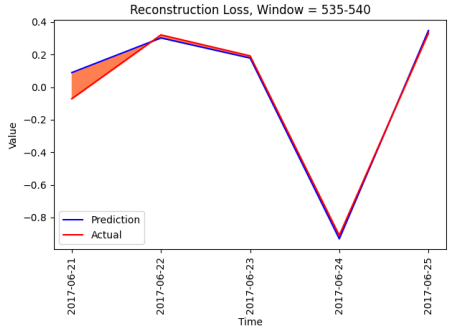




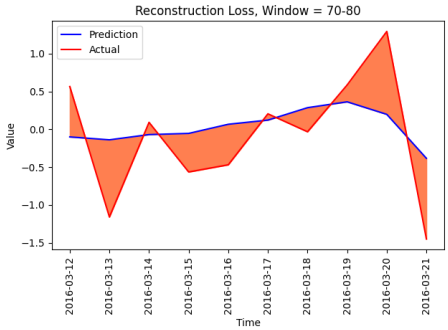


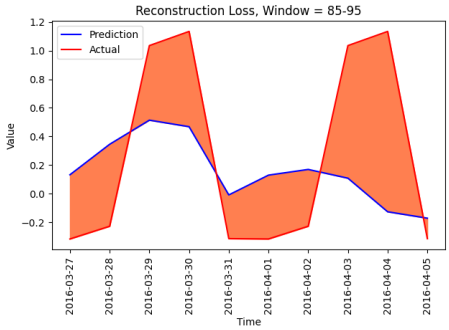


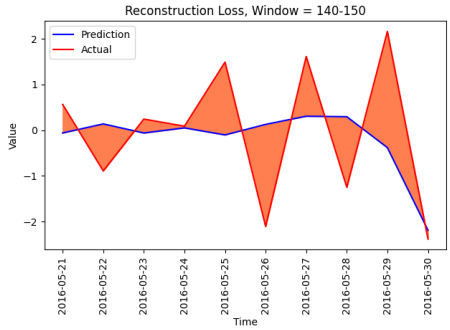


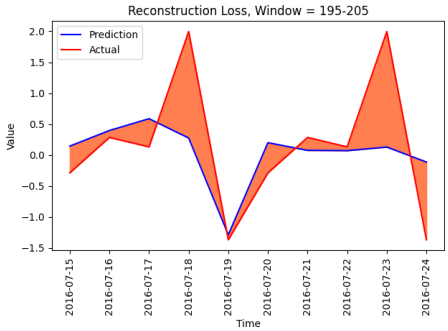


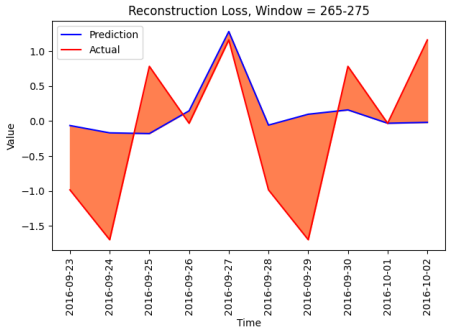
Window Size = 10:

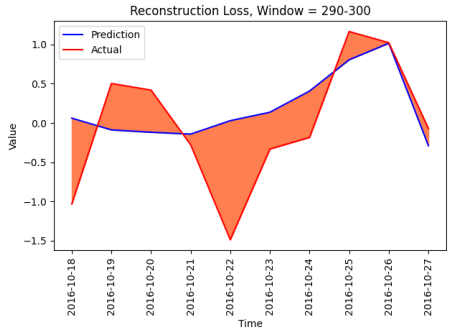

Reconstruction Loss, Window = 70-80
Prediction
Actual
Value
Time

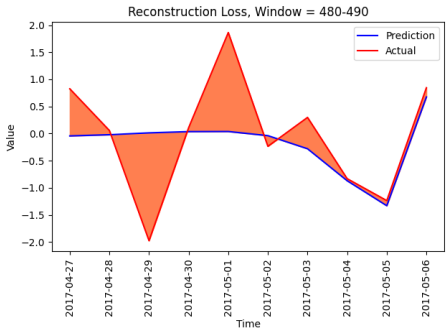

Reconstruction Loss, Window = 85-95
Prediction
Actual
Value
Time

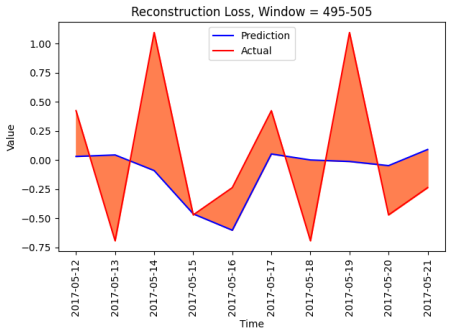

Reconstruction Loss, Window = 140-150
Prediction
Actual
Value
Time

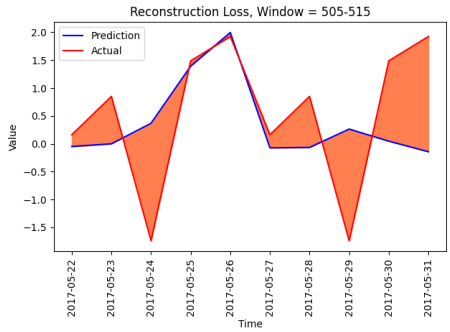

Reconstruction Loss, Window = 195-205
Prediction
Actual
Value
Time

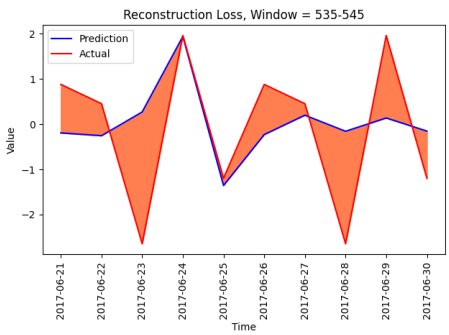

Reconstruction Loss, Window = 265-275
Prediction
Actual
Value
Time


Reconstruction Loss, Window = 290-300
Prediction
Actual
Value
Time

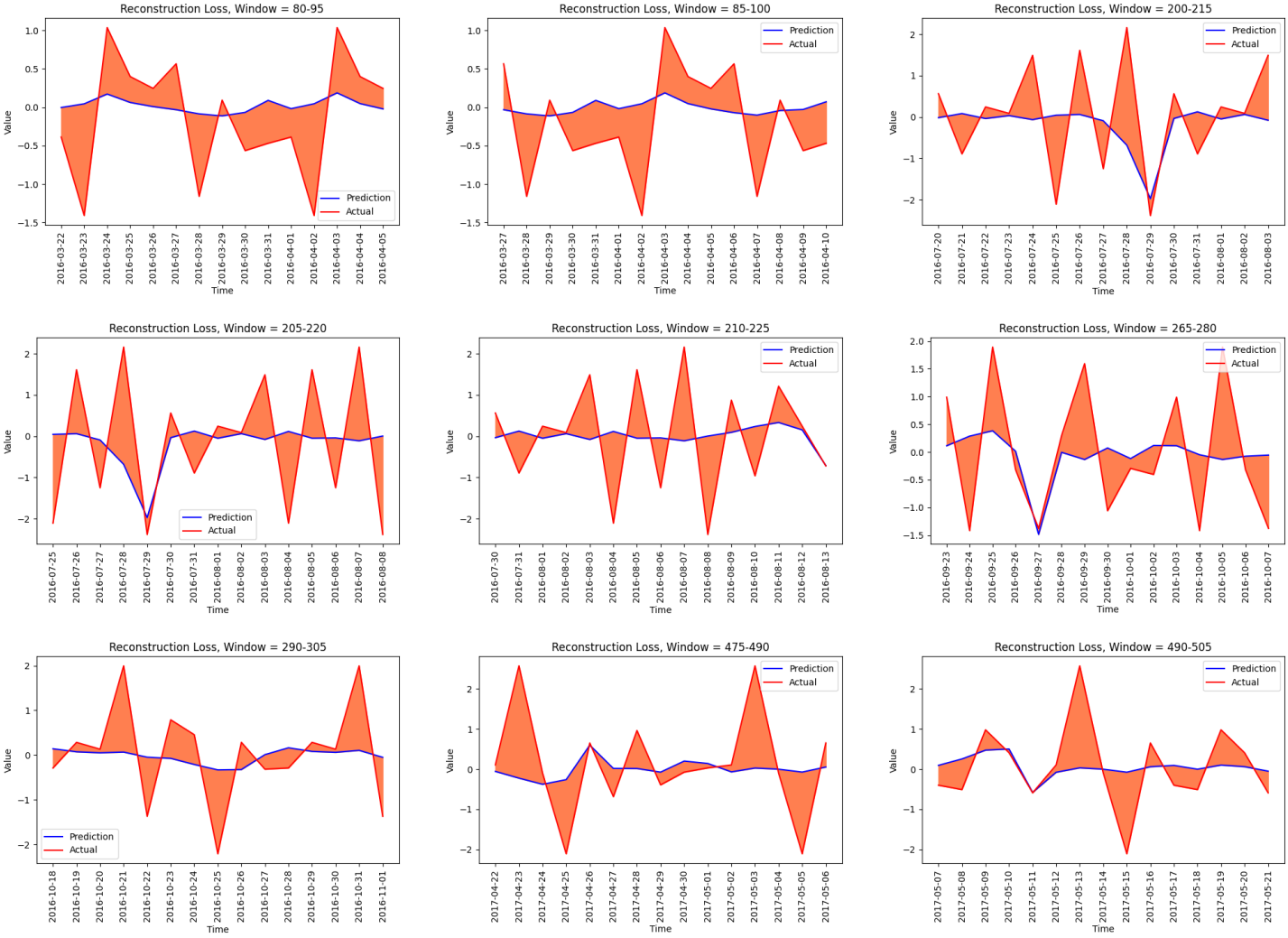

Reconstruction Loss, Window = 480-490
Prediction
Actual
Value
Time

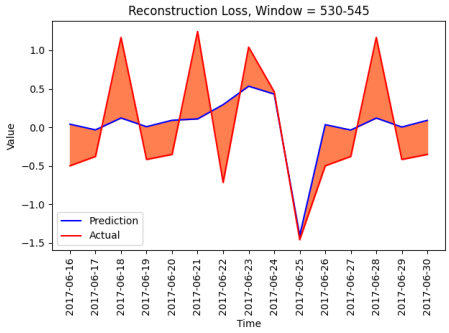

Reconstruction Loss, Window = 495-505
Prediction
Actual
Value
Time


Reconstruction Loss, Window = 505-515
Prediction
Actual
Value
Time

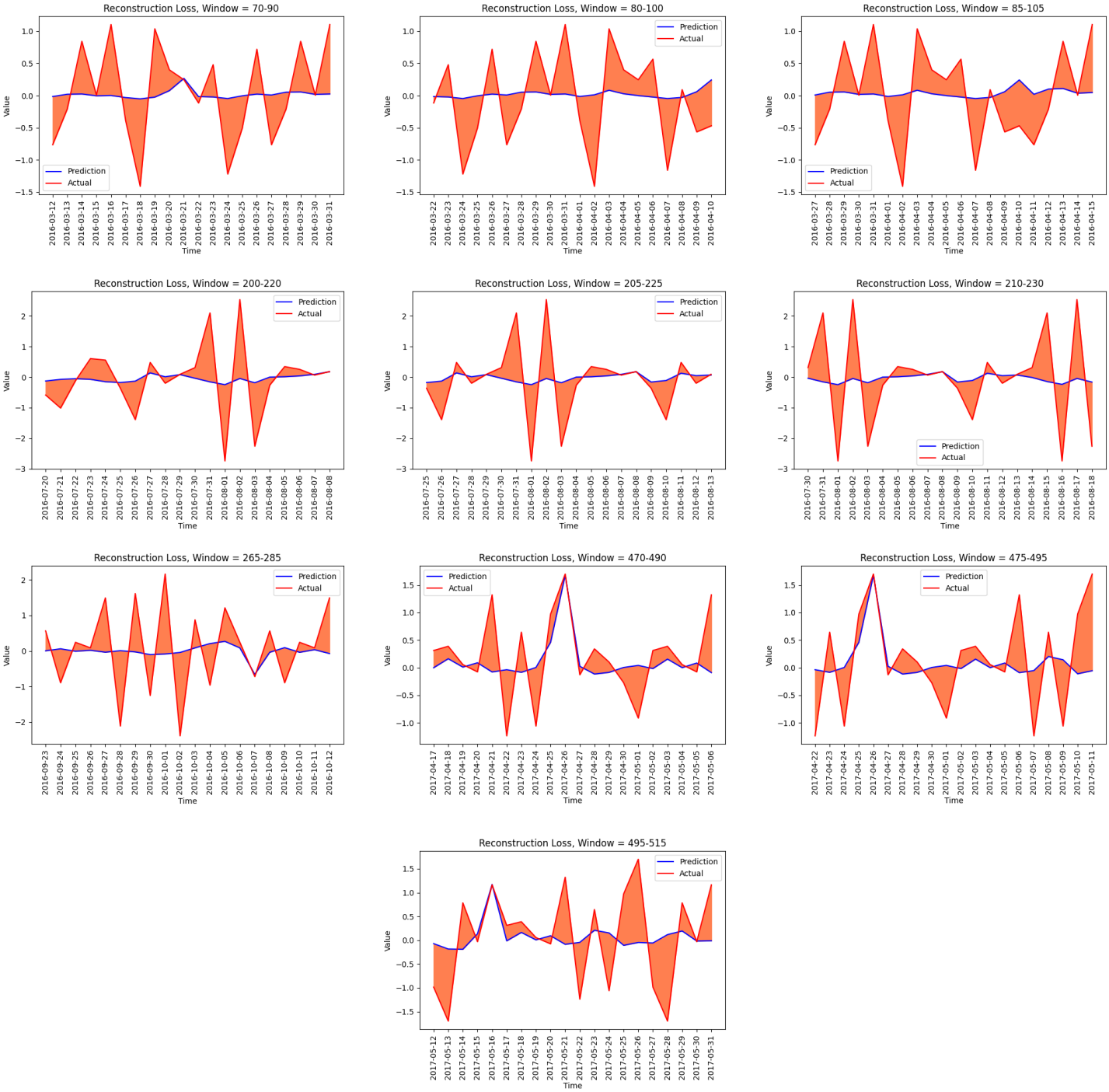

Reconstruction Loss, Window = 535-545
Prediction
Actual
Value
Time

Window Size = 15:

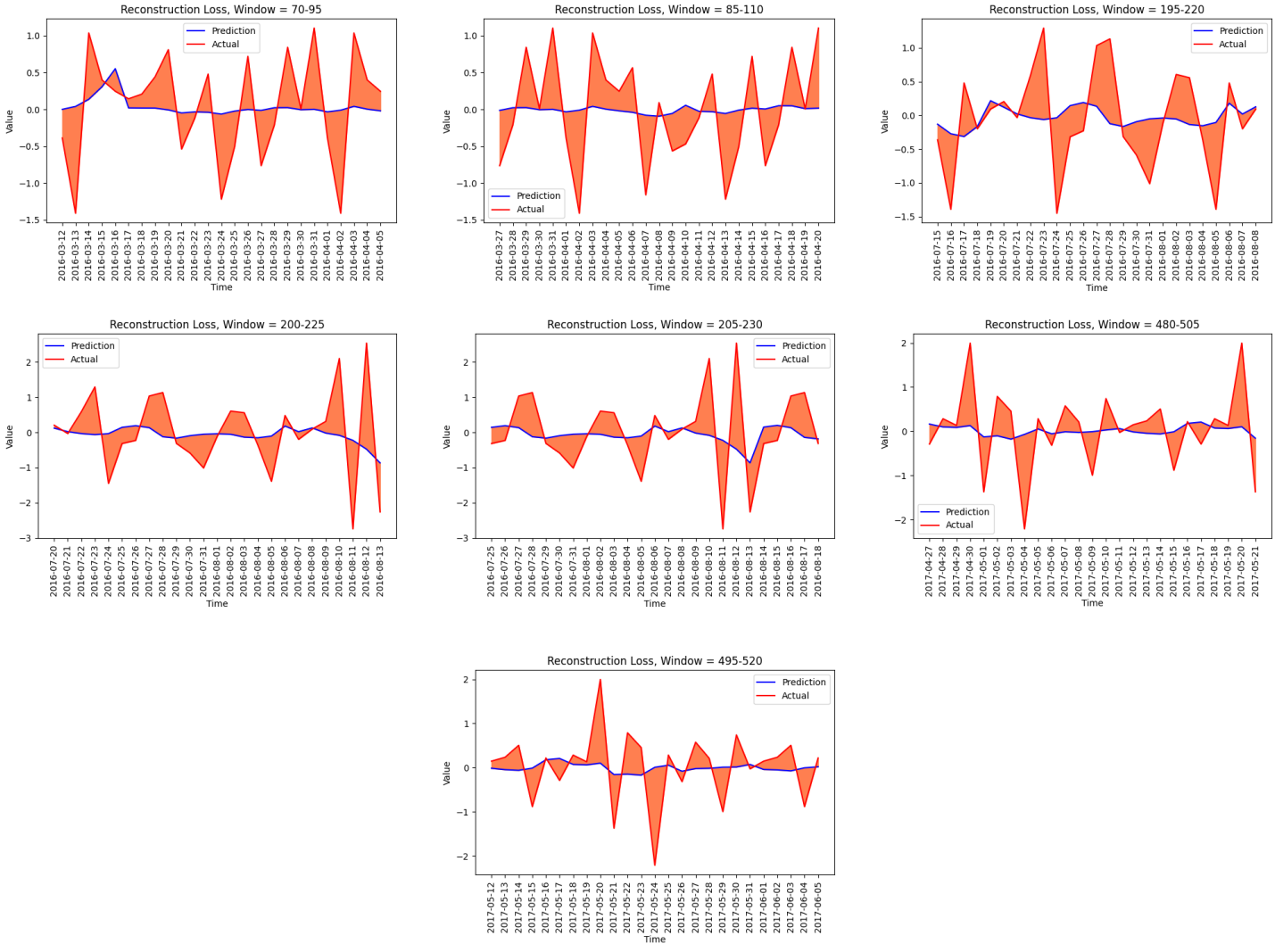




Window Size = 20:

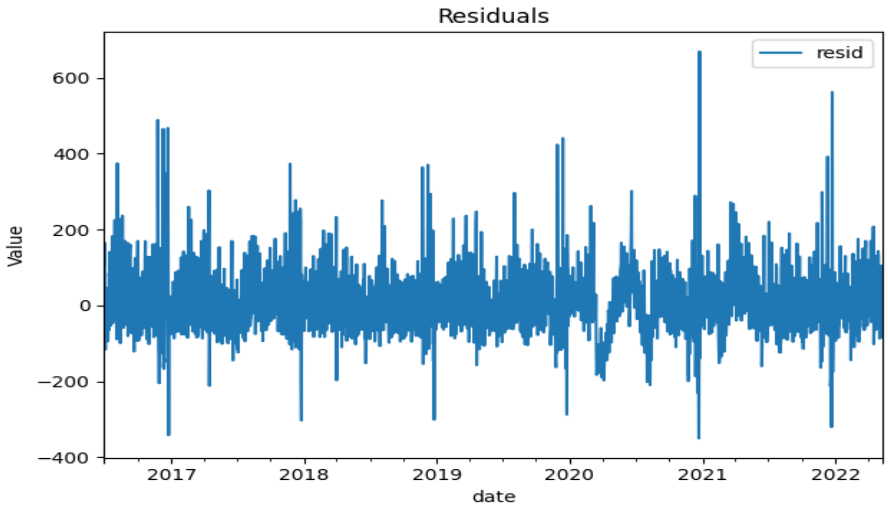


Window Size = 25:

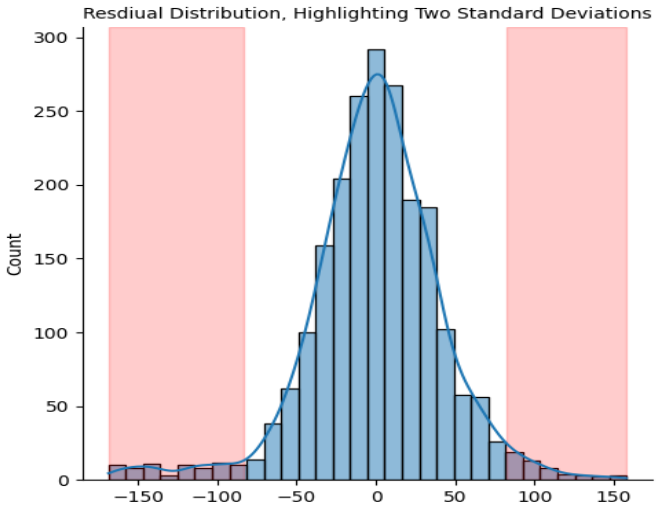


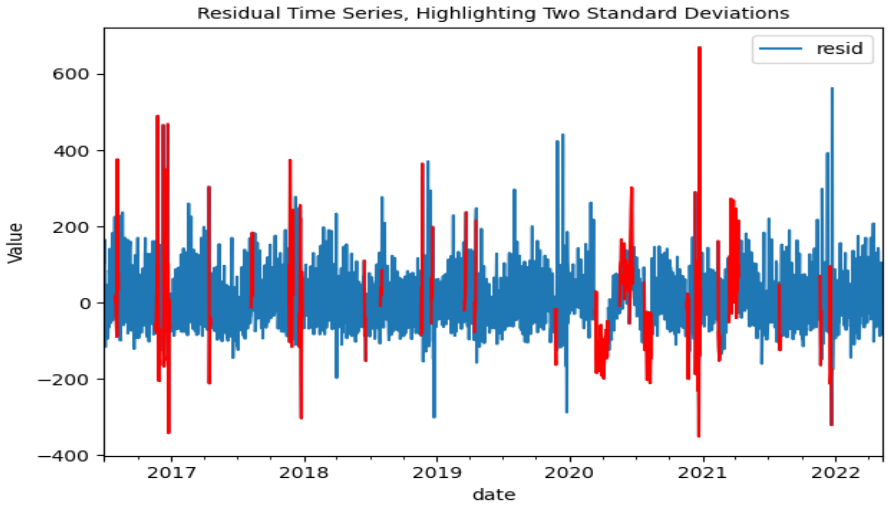
6.3 Appendix C - STL Residual Results

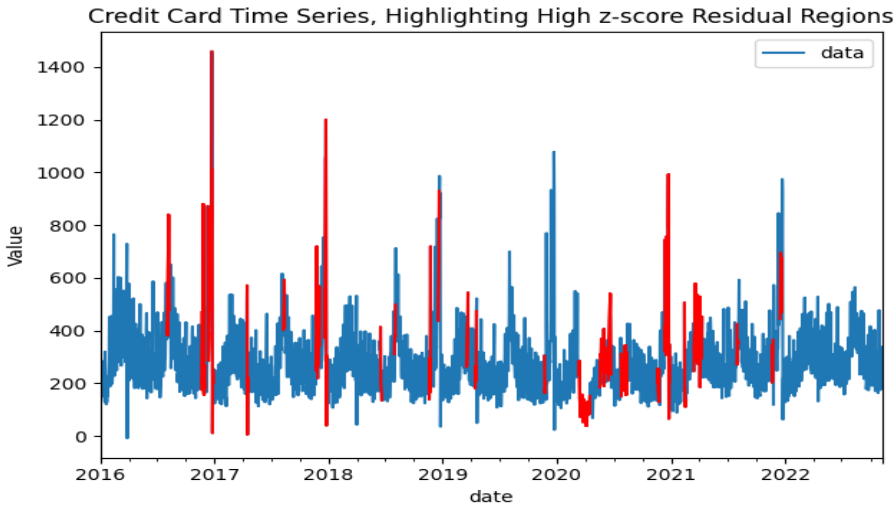
Original Residual:



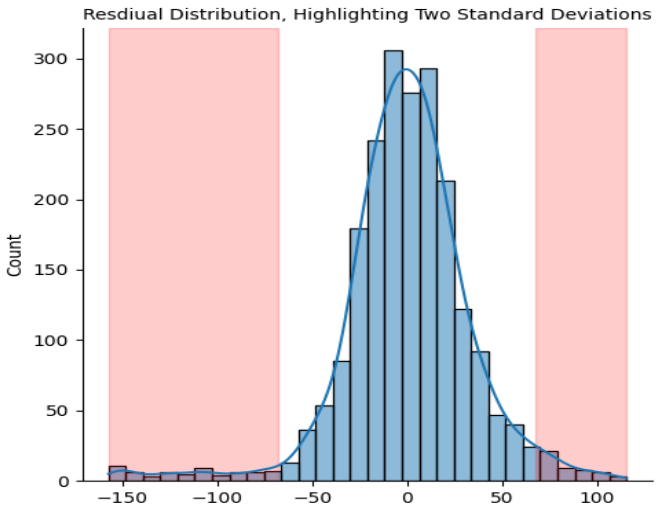
Window Size = 5:

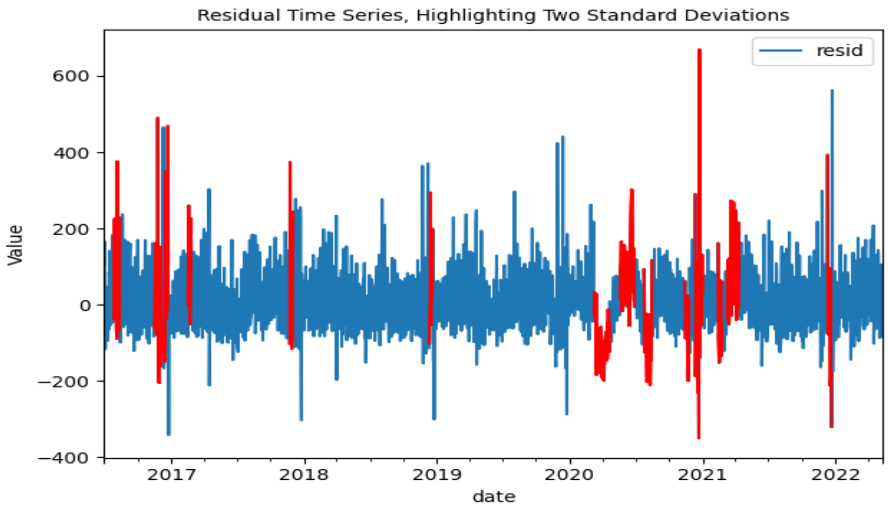


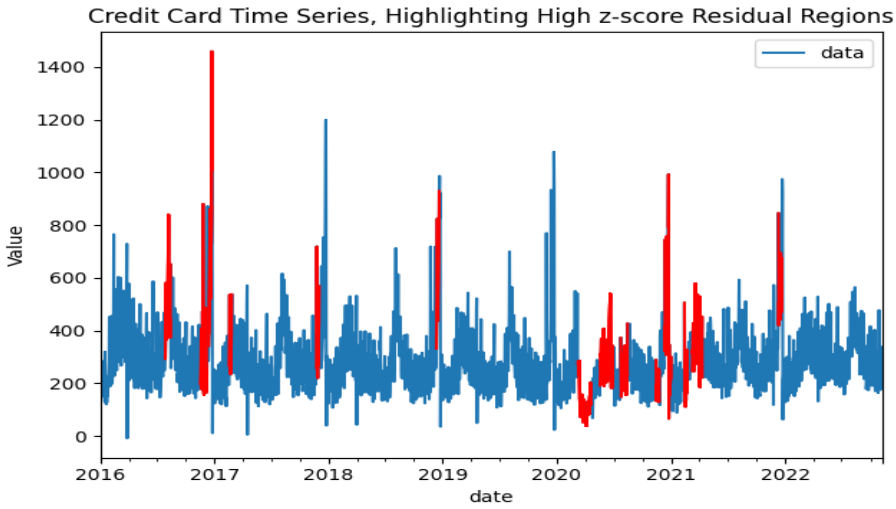




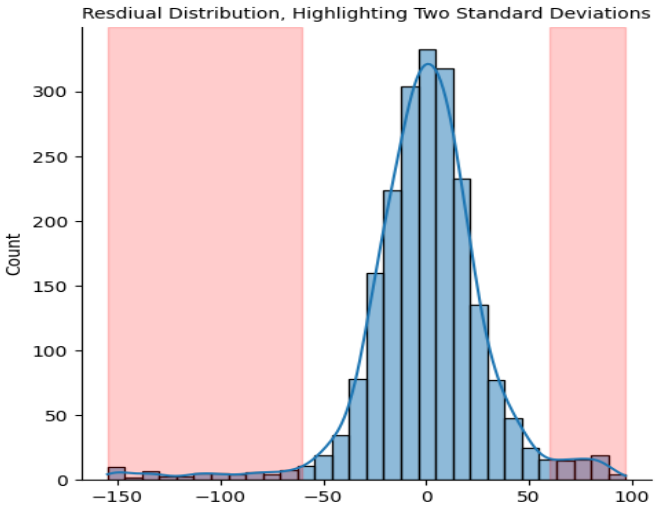
Window Size = 10:

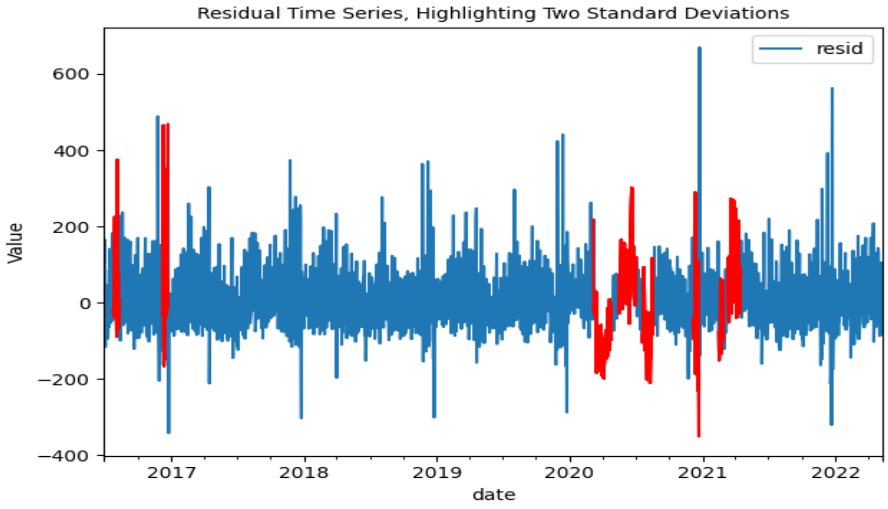

Resdiual Distribution, Highlighting Two Standard Deviations
Count
300
250
200
150
100
50
0
−150
−100
−50
0
50
100

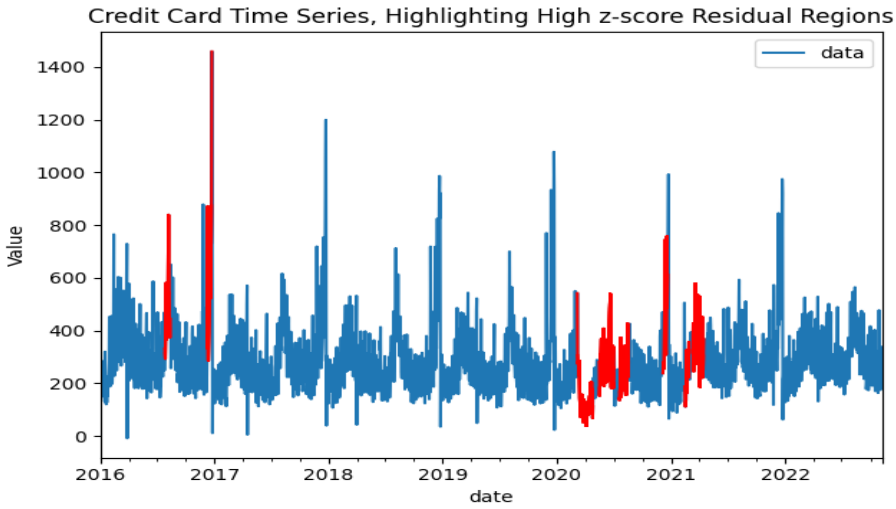

Residual Time Series, Highlighting Two Standard Deviations
resid
Value
600
400
200
0
−200
−400
2017
2018
2019
2020
2021
2022
date



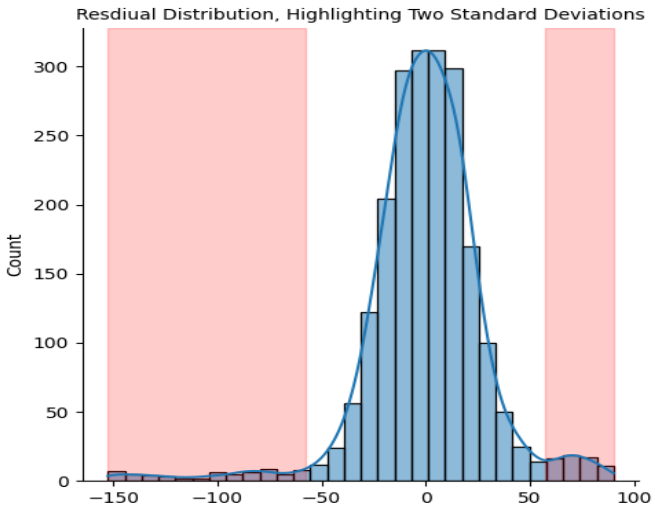
Window Size = 15:

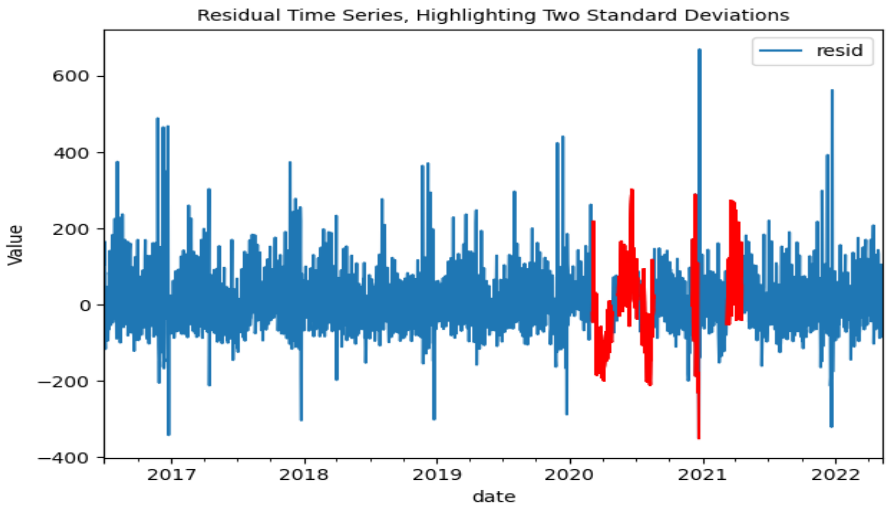


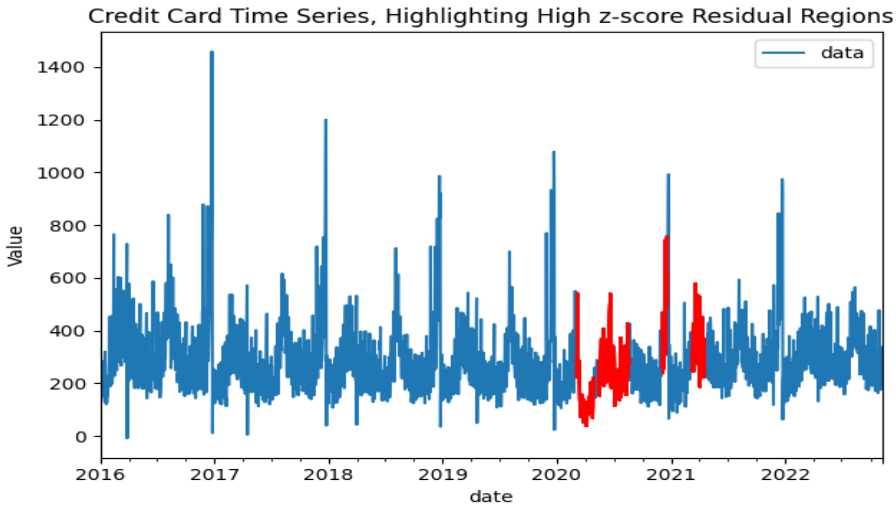




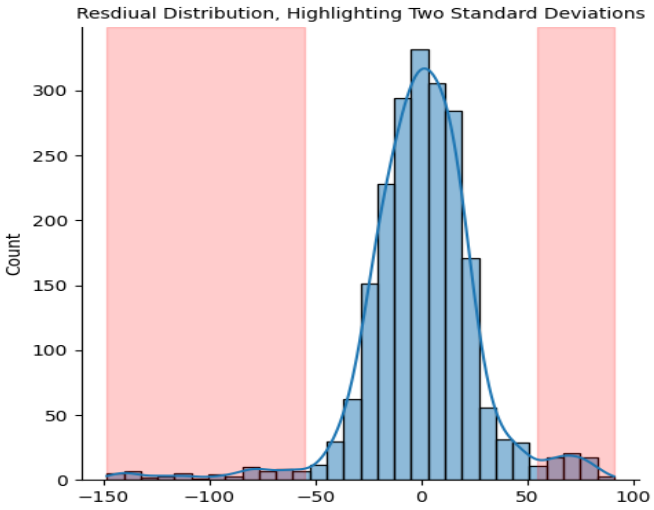
Window Size = 20:

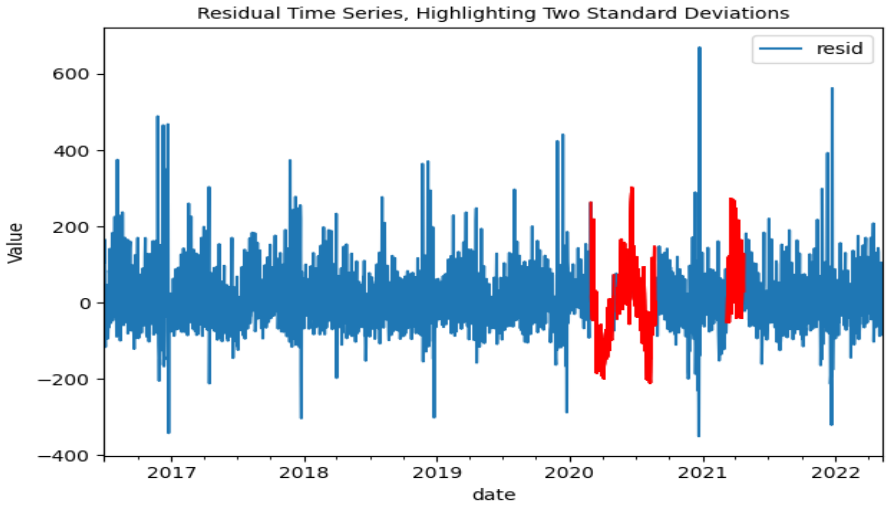

Resdiual Distribution, Highlighting Two Standard Deviations
Count
300
250
200
150
100
50
0
−150
−100
−50
0
50
100

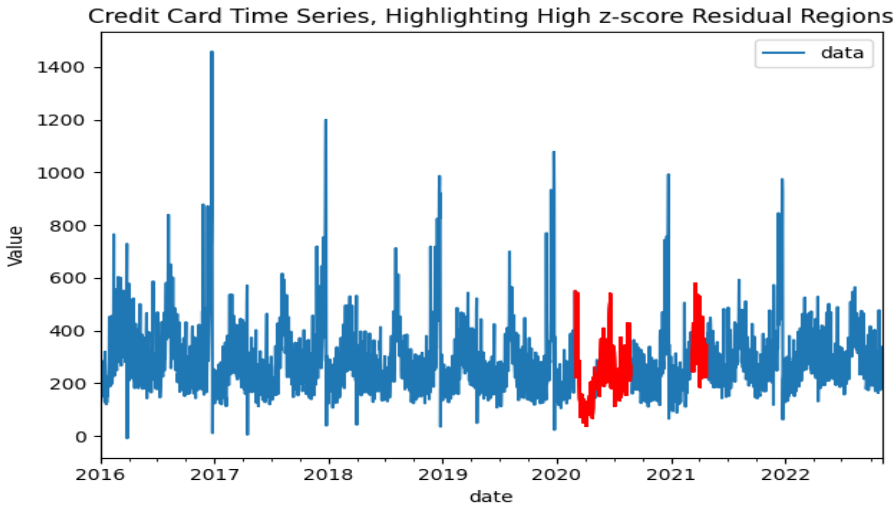

Residual Time Series, Highlighting Two Standard Deviations
resid
Value
600
400
200
0
−200
−400
2017
2018
2019
2020
2021
2022
date



Window Size = 25:

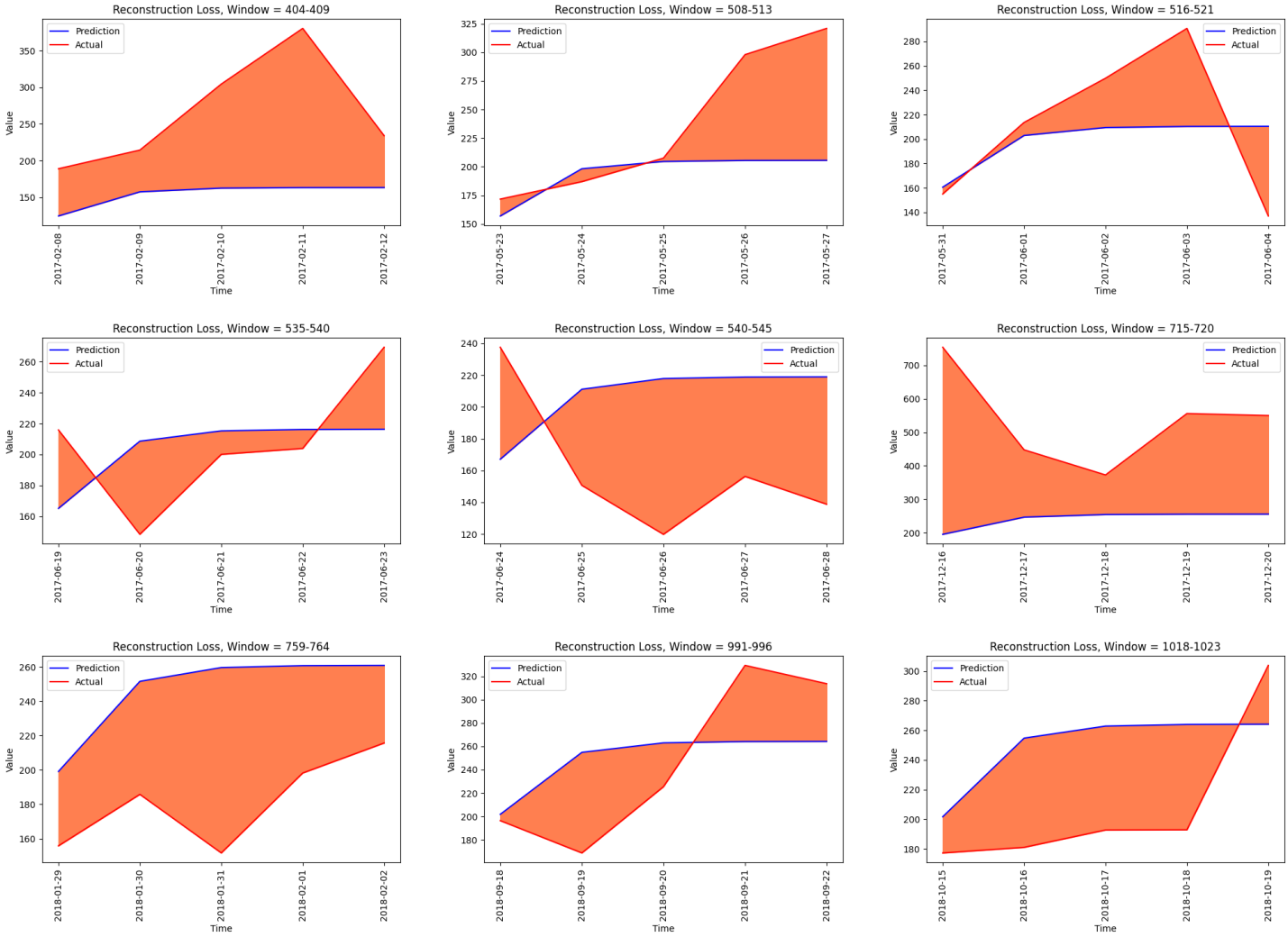




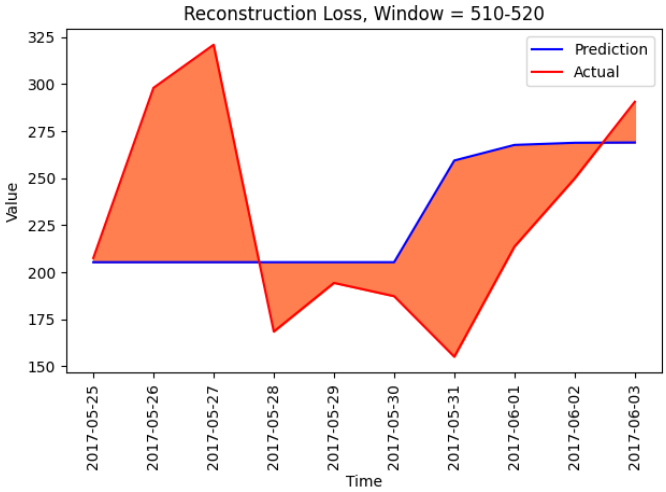


6.4 Appendix D - STL-assisted LSTM Anomaly Results

Window Size = 5:



Window Size = 10:


Reconstruction Loss, Window = 510-520
Prediction
Actual
Value
325
300
275
250
225
200
175
150
2017-05-25
2017-05-26
2017-05-27
2017-05-28
2017-05-29
2017-05-30
2017-05-31
2017-06-01
2017-06-02
2017-06-03
Time