CISC-5352-L01

Machine Learning in Finance Course Final Project Report

Presented by:
Andres Leonardo De Los Santos
Albert Delgado Baez
Daniela Johnson

Dec 17th, 2024

1. Introduction:

The federal funds rates are the interest rates at which banks lend to each other overnight and are set by the Federal Reserve. It plays a crucial role in the U.S. economy by influencing borrowing costs, consumer spending, and business investments. Changes in the federal funds rates can affect inflation, employment, and economic growth, making it a vital tool for managing the economy. It also impacts financial markets globally, with shifts in the rate influencing stock, bond, and currency markets.

This project aims to predict future federal funds rates using machine learning algorithms, including the Vasicek and Cox-Ingersoll-Ross models for interest rates, as well as deep learning models like Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, Gated Recurrent Units (GRUs), and Transformers. By applying these models, the goal is to forecast the rates more accurately and gain deeper insights into their future trends.

Machine learning algorithms, particularly those designed for time-series data like RNNs, LSTMs, GRUs, and Transformers, are well-suited since they can capture complex patterns and dependencies in historical data, providing more precise forecasts compared to traditional economic models. They also allow for the analysis of long-term trends and short-term fluctuations, making them ideal for this task.

The ultimate goal of this project is to improve the accuracy of federal fund rate predictions by optimizing these models through hyperparameter tuning. The results will help inform decision-making for businesses, financial institutions, and policymakers, offering a data-driven approach to understanding future economic conditions.

2. Data preprocessing:

The first thing we had to do before implementing any models was preprocess the data. We immediately implemented a line of code to skip over the first 16 rows of data since they did not include any values. We are dealing with two columns of data, one being the date and one being the federal fund rate on that date. Our first date is July 1st, 1954 and our last date is November 22nd, 2022, listing a federal fund rate for almost every single day in between. In our data, there were 21 missing federal fund rates, which were all after November 1st, 2024, so we deleted those days. Our data then extended from July 1st, 1954, to November 1st, 2024. We also had 734 missing dates, meaning 734 random days were not listed and therefore, we do not know their federal fund rate. Since the rates were listed for the day prior and the day after, this was not necessarily a problem. However, we wanted our data to follow a pattern so that it can perform well when trained to a model, so we added all missing dates back in and predicted their values.

Three methods to handle missing values were implemented before deciding which was most appropriate and accurate for the data. KNN was implemented first to predict federal fund rates for the missing dates. Immediately, we found that predictions were more accurate when the number of neighbors was lower, meaning that a date's closest neighbors are most relevant for predicting missing values. This makes sense because the dataset should have regions with more distinct patterns. Recessions, Covid-19, and specifically the fight against inflation in 1980 that caused federal fund rates to skyrocket all are reasons that the dates closest to the date we would like to predict the federal fund rate for are most relevant in their predictions (Foster). Although lowering the number of neighbors improved some of our predictions, not all rates agreed on which exact number of neighbors was most beneficial. We then implemented time-based features since this is a time-series dataset to allow KNN to capture temporal patterns. Our data again, had some great predictions when the number of neighbors was 3, but it also had some that we were not satisfied with. Changing the number of neighbors to 4 made some of those inaccurate predictions more accurate, but it also made those accurate predictions more inaccurate. If you look at the table below, 3 neighbors was more accurate in predicting 11/23/2017's federal fund rate, and 4 neighbors was slightly more accurate in predicting 12/25/2017's value. We were not satisfied with these results. It was easy to determine which predictions were accurate and those that weren't because we had the rates for every single day. You can easily see which one is out of place or does not follow the pattern. KNN was not an efficient method to use for missing values because it was not consistent with our dataset.

Dates	Original Federal Fund Rates	3 neighbors	4 neighbors
11/22/2017	1.16%	1.16%	1.16%
11/23/2017	?	1.16%	1.1375%
11/23/2017	1.16%	1.16%	1.16%
12/24/2017	1.42%	1.42%	1.42%
12/25/2017	?	1.2467%	1.535%
12/26/2017	1.42%	1.42%	1.42%

Next, LSTM was implemented because it processes data sequentially and is specifically meant for time series data. This model was extremely slow because it went through each date one at a time, and we had about 70 years of data from each day of every year. Since it was extremely slow, and we did not feel comfortable grouping the data into clusters to predict missing values of specific days, we did not use this model. However, we do use this data to forecast future values. To predict future values, it is important to group our data into clusters so that we prevent overfitting, and it helps to group important economic events into clusters.

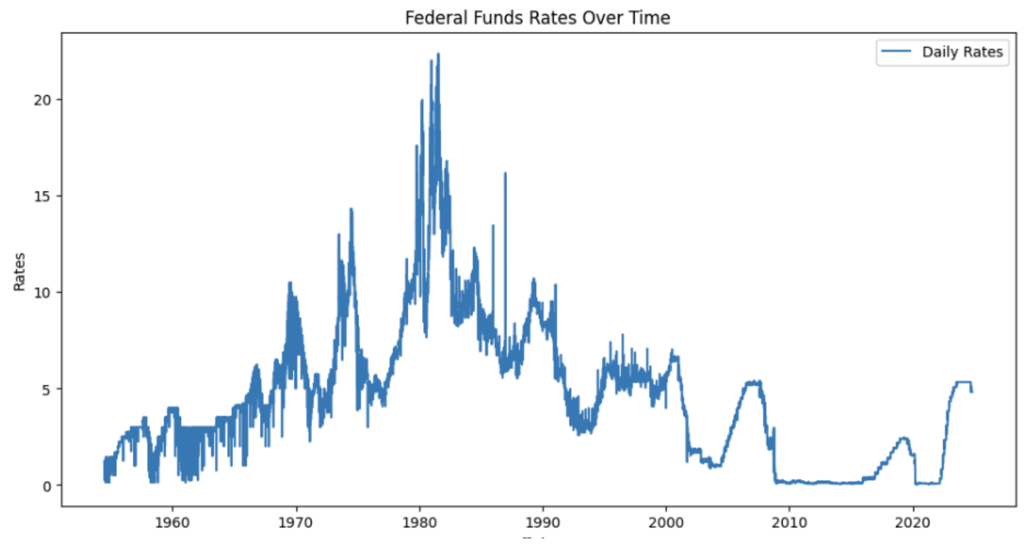
Lastly, we implemented a Simple Moving Average, which is a very simple method to predict missing values, but it was most effective in this case. The formula used to predict these values is $SMA_t = \frac{\sum_{i=t-k+1}^{t} x_i}{k}$, where x_i are the surrounding data points, k is the number of data points, and t is the actual date you are predicting the federal fund rate. Centered SMA uses recent data points, both past and future, and overlooks long-term patterns, which was beneficial in this case because we have dates that are so close together. When you have yesterday's federal fund rate and tomorrow's, you do not need to analyze the overall trend in the data for the past year. It will be irrelevant and lead to faulty predictions. This method is ineffective for large data gaps because it uses the surrounding data to calculate an average and apply smoothing to fill missing values.

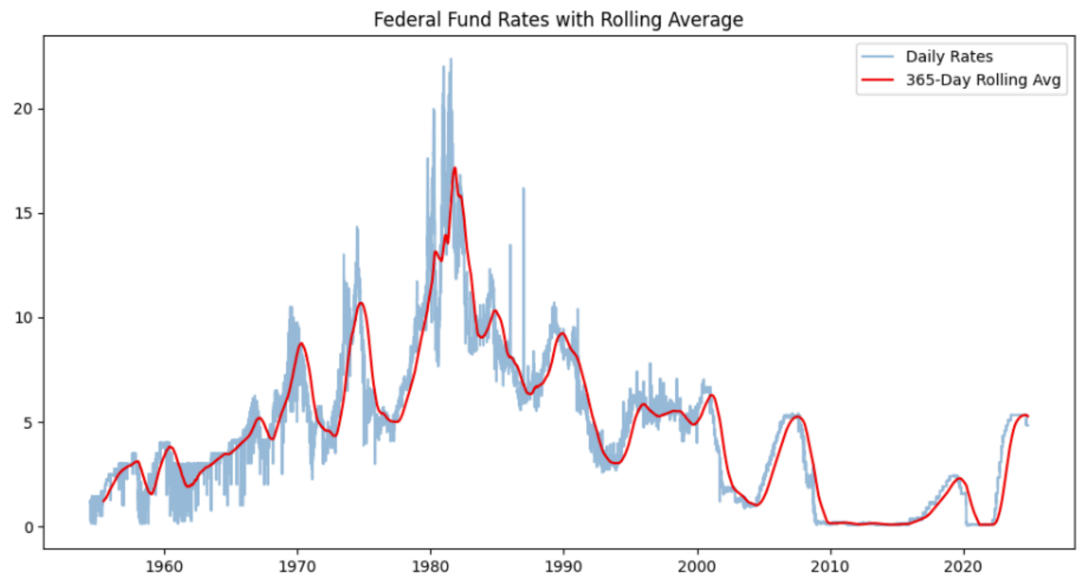
3. Data exploration:

After preprocessing the dataset, we conducted an analysis of the federal fund rates over the past 70 years. The plot below compares the daily federal fund rates with a 365-day rolling average, which helps smooth out short-term fluctuations and highlights longer-term trends in the data.

The 365-day rolling average is a technique used to smooth the daily volatility of the rates, providing a clearer view of the underlying trend over a one-year period. By averaging the data over the past 365 days, this method reduces noise caused by short-term market fluctuations, allowing us to better understand the broader economic patterns that drive changes in the federal fund rates.

Here is how our data is distributed over the 70 years of span.





- **Key Insights from the Plot:**

Throughout the 70 years of data, we observed that federal fund rates tend to rise during periods of economic boom or bubbles. This is because, during times of strong economic growth, the Federal Reserve typically increases interest rates to manage inflation and ensure economic stability. However, when an economic crisis occurs towards the end of such a boom, the rates tend to drop sharply, as the central bank may lower rates to stimulate economic activity.

Specific Events Impacting the Rates:

- **Oil Crisis (1970s):** The oil crises of the 1970s triggered economic recessions, leading to a reduction in consumer confidence and business activity. This period also saw a decrease in the federal fund rates to mitigate the effects of the crisis.
- **2007-2008 Financial Crisis:** The financial crisis of 2008 caused a significant economic downturn, leading the Federal Reserve to reduce interest rates in an effort to stimulate the economy.
- **COVID-19 Pandemic (2020):** In the early stages of the pandemic, the U.S. economy experienced a sharp downturn due to lockdowns, layoffs, and uncertainty. Despite economic growth under the Trump administration, the onset of COVID-19 led to a steep decline in consumer spending and investment, which is reflected in the sharp drop in federal fund rates.

Economic Booms and Rising Rates:

- **Post-War Economic Boom (1940s-1950s):** Following World War II, the U.S. experienced significant economic growth. The federal fund rates began to rise as inflation concerns increased with the growth of the economy.
- **Baby Boom Era (1960s):** The 1960s saw strong economic growth fueled by the baby boom, resulting in a substantial increase in federal fund rates.
- **The 1980s:** The 1980s witnessed a dramatic rise in federal fund rates, largely due to efforts to combat inflation during the Reagan administration.
- **Tech Boom and Dot-Com Bubble (1990s):** The 1990s, marked by the technology boom and the dot-com bubble, also saw a significant rise in the rates as the economy expanded rapidly.

The final insight we can infer from this analysis is: when the U.S. economy is in an expansion phase, the federal fund rates typically increase, and when a crisis or economic downturn occurs, rates tend to fall. The rolling average allows us to visualize these long-term trends more clearly by filtering out short-term market noise. This dynamic underscores the importance of federal fund rates in managing the health of the U.S. economy. Understanding these patterns can help in forecasting future rate movements and making informed decisions for investment, monetary policy, and economic planning.

4. Model Implementation & Performance Evaluation

This section outlines the implementation and insights gained from implementing various classification techniques learned in class. We tested the following models: Vasicek, Cox-Ingersoll-Ross, Recurrent Neural Networks (RNN), Long-Short Term Memory (LSTM), Gated Recurrent Unit (GRU), and Transformers. Each of these models provides distinct predictive strengths, which will be summarized below.

Vasicek model.

The Vasicek model is a stochastic model that predicts interest rates and can be used to predict federal fund rates. This model assumes that rates tend to revert to a long-term average. The equation it uses is $dr_t = a(b - r_t)dt + \sigma dW_t$ where b is the "long-term mean level," a is the "speed of reversion," and σ is the "instantaneous volatility." This model is simple, but it has some disadvantages, such as the possibility of producing negative rates. It also is a one-factor model which "captures interest rate movements driven by a single source of market risk." This fails to explain any long-term interest rate behaviors based on short-term rate changes. One last important limitation is that this is a linear stochastic model, but the real world exhibits non-linear rates. This model cannot capture sudden changes in rates, which could be due to an event such as a financial crisis.

Using every data point and normalizing the data, we implemented the Vasicek model. The parameters we used were Kappa: 1.2995, Theta: 0.2315, and Sigma: 1.5461. Using every single data point can lead to overfitting the model. Our results are in a table below. A larger negative R^2 value can mean the model fits poorly and below the baseline. The baseline can be simply predicting the mean of the target variable for each point. It is the most simple prediction and these models are expected to be more accurate than the baseline. The model is failing to capture underlying patterns. A mean squared error of 2.1841 on our testing set is also quite large because we are dealing with federal fund rates that are ranging from 0 to 2%. An error should not be larger than the actual values we are dealing with themselves. It is important to note that the values in our dataset are percentages, and we are dealing with them in their percentage form, meaning we are dealing with the number 1.16, not 0.016.

- **Before Hyperparameter tuning:**

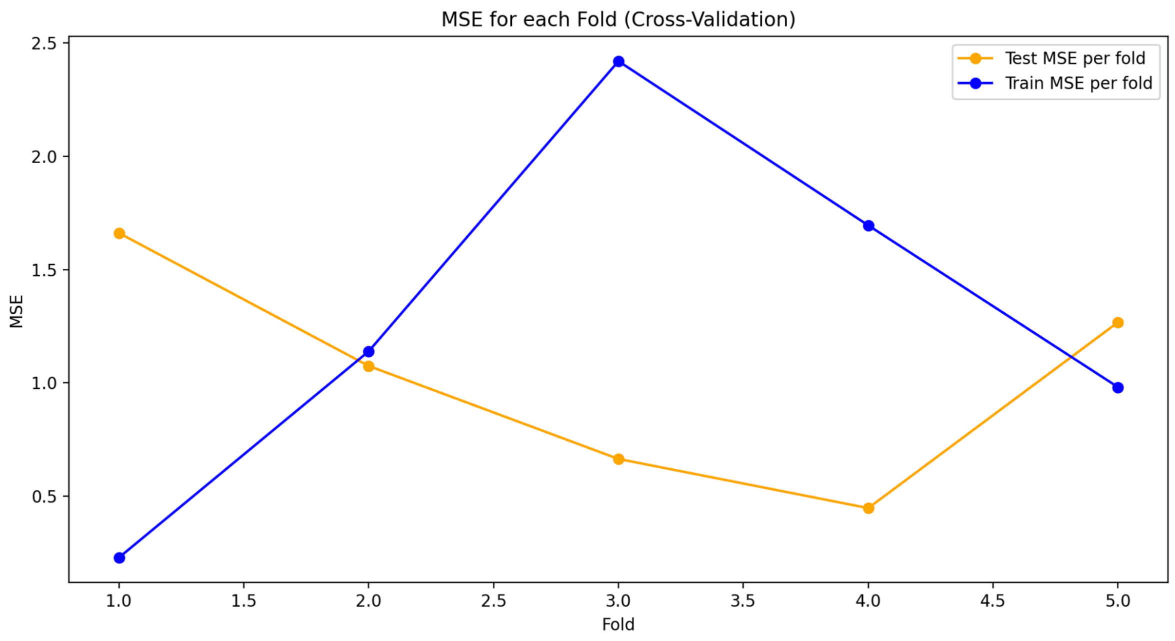
Metric	**Training Set**	**Testing Set**
Mean Squared Error (MSE)	1.7670	2.1841
Mean Absolute Error (MAE)	1.0590	1.2438
Root Mean Squared Error (RMSE)	1.3293	1.4779
R-squared (R^2)	-0.9236	-8.5585
Variance of Errors	1.7559	0.8223

To obtain better results, we implemented cross-validation. We split the data into 5 subsets and used TimeSeriesSplit to maintain the temporal model of the data. Originally, we implemented this on all 70 years of data. The results we obtained were not great. Next, we tried implementing 5-fold cross-validation onto the last 5 years of data rather than all 70, so from 11/01/1999 to 11/01/2024. Since the Vasicek model relies more heavily on short-term relationships, we thought that shortening the time frame to a more recent period would be beneficial to avoid the model from adhering to irrelevant patterns. Again, we obtained a large MSE and a large negative R^2 value. Predicting federal fund rates using 70 years of data with no other information in the dataset about the years is very difficult.

Lastly, we reimplemented the 5 folds on all 70 years of data, but this time we used each month's average federal fund rate instead of every day. This reduces overfitting and the cost to run this code. Our testing metrics are displayed below and our graph of the mean squared error at each fold is shown. Cross-validation and finding monthly averages of the federal fund rates did lower the errors overall, but we have a large negative R^2 value for our testing set. This means the model does not fit our observed data well. Our model is not only making inaccurate predictions but deviating so far from the actual values that the residual errors are larger than the variability in the actual data.

- **After Hyperparameter tuning:**

Metric	Training Set	Testing Set
Mean Squared Error (MSE)	1.2928	1.0233
Mean Absolute Error (MAE)	0.9272	0.8506
Root Mean Squared Error (RMSE)	1.0789	0.9872
R-squared (R^2)	-2.0247	-407.6600



In conclusion, our results using the Vasicek model to predict future rates were poor and highly inaccurate even with testing multiple hyperparameters. The Vasicek model should not be used to predict federal fund rates, when trained to this dataset. Some further exploration we would be interested in would be creating clusters using U.S. economic history. We could have manually grouped federal fund rates that were affected by Covid-19 and those that were affected by the fight against inflation in 1980. This could have possibly improved the metrics of our training and testing dataset, but overall the Vasicek model is not appropriate for this dataset.

Cox-Ingersoll-Ross model.

The CIR model is an extension of the Vasicek model. It does not include negative federal fund rates. Similarly to the Vasicek model however, the model assumes that interest rates tend to revert to a long term average over time and it also is a one-factor model. It uses a similar equation $dr_t = a(b - r_t)dt + \sigma\sqrt{r_t}dW_t$, where $\sigma\sqrt{r_t}$ avoids all possible negative interest rates for our a and b values.

After normalizing the data and implementing the CIR model, our training and testing metrics are in the table below. Our training data size is 80% of the data, and the testing size is 20% of the data. The parameters we used were Kappa: 0.1, Theta: 0.5, and Sigma: 0.02. Inaccuracy exists because of the spikes of the federal fund rates. It is difficult for the model to predict when these spikes will occur without any other data or features regarding the economy. Some features that could be helpful would be the Consumer Price Index (CPI), Unemployment rate, and Gross Domestic Product (GDP).

- **Before Hyperparamter tuning:**

Metric	Training Set	Testing Set
Mean Squared Error (MSE)	0.9734	1.1058
Mean Absolute Error (MAE)	0.7154	0.9757
Root Mean Squared Error (RMSE)	0.9866	1.0516
R-squared (R^2)	-0.0597	-3.8393
Variance of Errors	0.9185	0.2286

Again, we applied cross-validation to the data to see if our accuracy using this model would improve. We applied 5 folds to our 70 years of data. As seen in the table below, the results are strikingly similar to the results we had before implementing cross-validation. This could be because we used daily federal fund rates. This made our dataset larger, making it possible for different splits in the data not to reveal new patterns or variability in the dataset. Another possibility when you obtain similar results before and after cross-validation could be due to the training and test splits having very similar characteristics. This is highly unlikely since we used the same testing and training splits for the Vasicek model. This issue did not come up in the Vasicek model section of this report because we only displayed our output when we took the average of each month’s federal fund rate. Since this made the dataset smaller, the splits in the data were able to reveal new patterns in the dataset.

- **After Hyperparameter tuning:**

Metric	**Cross-Validation Results Averaged**
Mean Squared Error (MSE)	1.0837 ± 0.5284
Mean Absolute Error (MAE)	0.8374 ± 0.2542
R-squared (R^2)	-2.5045 ± 1.8172

The real world is unpredictable, and so are federal fund rates. You do not know when there will be a recession or when a pandemic will strike. The CIR model is not accurate in predicting federal fund rates. Its error is so large, the rates could be double or half their actual values! Neither model Vasicek or CIR should be used in predicting future federal fund rates, unless there is more information on the dataset.

Recurrent Neural Network.

A Recurrent Neural Network (RNN) is a type of neural network designed for sequential data. Unlike traditional feedforward neural networks, RNNs have connections that form loops, allowing information to be passed from one step to the next in the sequence. This design enables RNNs to capture temporal dependencies, making them especially useful for tasks like time series forecasting, natural language processing, and speech recognition. The RNN learns patterns in the data over time by processing the input at each timestep and retaining information from previous timesteps, which helps it understand the sequence context.

For this project, we implemented a base RNN model using the Keras library (with TensorFlow backend) to predict future values of the rates of the normalized time series. The following hyperparameters were used in the model:

Hyperparameter	**Value**	**Description**
Input Shape	Sequences of length 10	The input data was reshaped into sequences of length 10 (using a rolling window), with each sequence containing one feature (since we are forecasting a single value).
Hidden Layers	20 units for the first layer, and 10 units for the second layer.	The first layer is a SimpleRNN layer with 20 units, returning sequences to maintain the temporal context for the next layer. The second layer is another SimpleRNN layer with 10 units, not returning sequences, as it's the final RNN layer before the output.

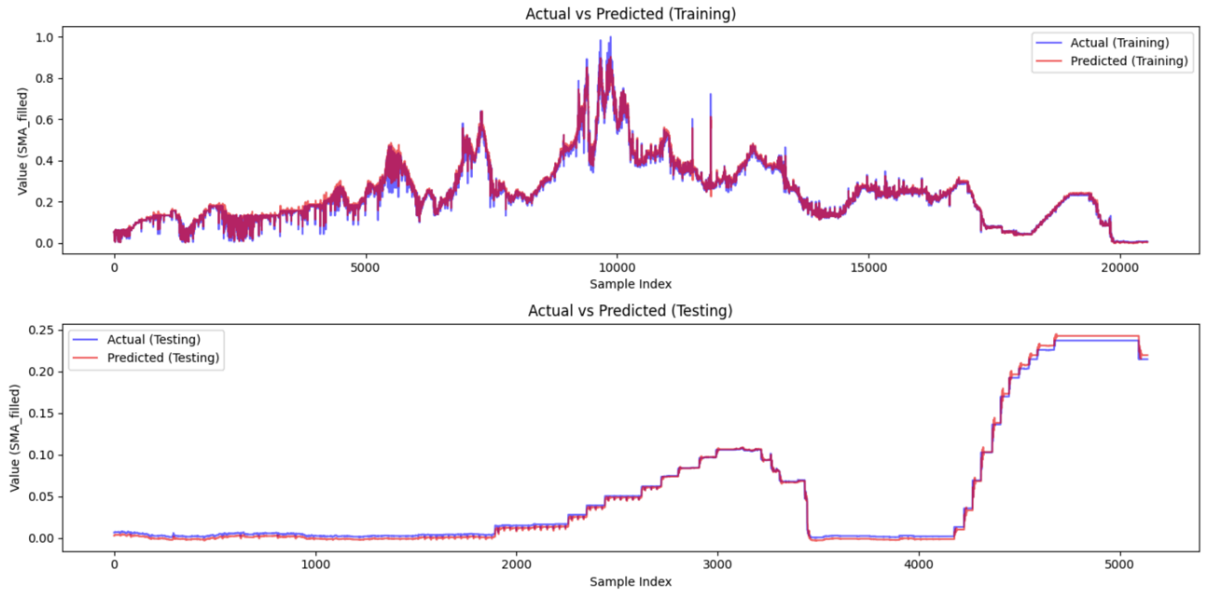
Dropout	Rate of 20%	Dropout regularization (with a rate of 20%) was applied after each RNN layer to prevent overfitting.
Output layer	1 unit	A dense layer with one unit to output the forecasted value.
Optimizer	Adam with Lr: 0.001	Adam optimizer with a learning rate of 0.001.
Loss Function	MSE	Mean squared error (MSE), which is commonly used in regression tasks.
Epochs	20	The number of times the model iterated through the training data.

- **Before hyperparameter tuning:**

After executing the base model, we obtained the following evaluation metrics:

Metric	**Training Set**	**Testing Set**	**Description**
Mean Squared Error (MSE)	0.00028	0.00001	Measures the average squared difference between predicted and actual values, lower values indicate better accuracy.
Mean Absolute Error (MAE)	0.00981	0.00317	Indicates the average absolute difference between predicted and actual values, with smaller values showing better performance.
Root Mean Squared Error (RMSE)	0.01685	0.00374	Represents the square root of MSE, showing the average deviation between predictions and actual values, with lower values indicating better performance.
R-squared (R^2)	0.98805	0.99763	Represents the proportion of variance in the data explained by the model, with values closer to 1 showing better fit.
Variance of Errors	0.00026	0.00001	Measures the spread of errors, with smaller values indicating less variability and more consistent predictions.

The initial results show that the model performs exceptionally well, with very low Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) for both training and testing sets, indicating accurate predictions. The Mean Absolute Error (MAE) is also small, suggesting that the average error per prediction is minimal. The high R-squared (R^2) values, nearly 0.99 for training and 0.998 for testing, demonstrate that the model explains almost all of the variance in the data. Additionally, the low variance of errors further confirms that the model's predictions are consistently accurate across both the training and testing datasets. The following plot illustrates the performance of the model:



However, the performance on the training set is exceptionally high compared to the testing set, which suggests that the model may be overfitting and might not generalize well to unseen data. To address this, we will proceed with hyperparameter tuning to improve the model's generalization capabilities.

- **Hyperparameter tuning:**

We employed the RandomizedSearchCV strategy from the sklearn library to optimize our RNN model. Randomized search efficiently explores the hyperparameter space by randomly sampling combinations over a predefined range of values. The best hyperparameter set is selected based on the lowest Mean Squared Error (MSE) on the validation dataset. The table below shows the ranges explored for each hyperparameter and the optimal values identified:

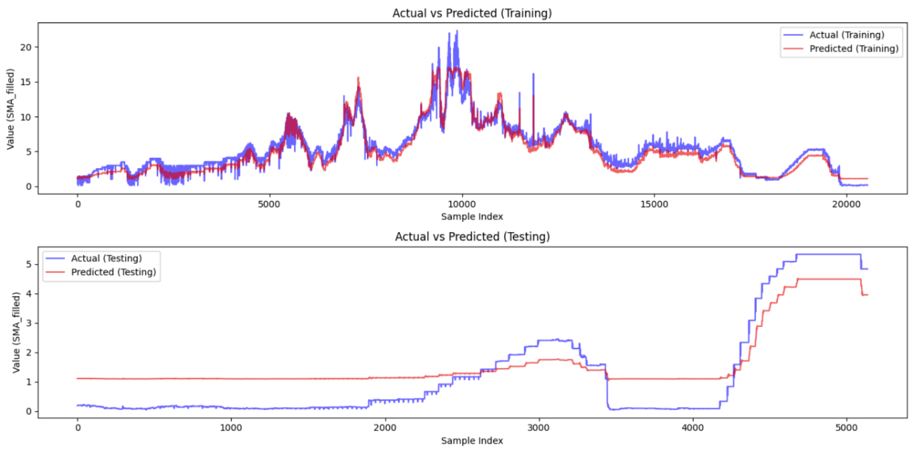
Hyperparameter	**Range**	**Best value**
Optimizer	Adam, RMSprop,SGD	SGD
Units	10 - 50 units	17
Dropout Rate	0.1 - 0.6	0.1488
Batch Size	2, 4, 8	4
Epochs	10 - 50	29
Learning Rate	0.001 - 0.01	0.00341

Among the tested optimizers, SGD was selected as the best-performing option, outperforming both Adam and RMSprop. For the number of hidden units, a moderate value of 17 proved to be optimal, effectively balancing model complexity and performance. To address overfitting, a relatively small dropout rate of 0.1488 was chosen, which introduced sufficient regularization without compromising the model's learning capacity. Finally, the learning rate was fine-tuned to 0.0034, striking a balance between convergence speed and training stability, ensuring that the model learned efficiently without diverging.

The table below compares the initial model's performance with the tuned model, for the testing set:

Metric	**Initial Model**	**Tune Model**
MSE	0.00001	0.70887
MAE	0.00317	0.79273
RMSE	0.00374	0.84194
R^2	0.99763	0.75912

The tuned model has a validation MSE of 0.70887, which is significantly higher than the overly optimistic MSE of the initial model. The initial model showed extremely low error values, indicating potential overfitting. Hyperparameter tuning addressed this by introducing dropout, smaller units, and using SGD optimizer. The following plot illustrates the validation loss across epochs during the training process with the best hyperparameters:



In conclusion, the model demonstrates improved generalization when implementing the optimized hyperparameters compared to the initial model. The fine-tuned hyperparameters allow the model to strike a better balance between underfitting and overfitting, resulting in more accurate predictions on unseen data. This improvement suggests that the model is better equipped to adapt to real-world scenarios, enhancing its overall reliability and performance.

Long Short Term Memory.

A Long Short-Term Memory (LSTM) network is a type of Recurrent Neural Network (RNN) specifically designed to handle long-term dependencies in sequential data. Unlike traditional RNNs, LSTMs utilize specialized memory cells with gates (input, output, and forget gates) to control the flow of information, enabling the network to retain relevant information over longer sequences. This design makes LSTMs particularly effective for tasks involving time series forecasting, natural language processing, and speech recognition.

For this project, we implemented a base LSTM model using the Keras library (with TensorFlow backend) to predict future values of the rates of the normalized time series. The following hyperparameters were used in the model:

Hyperparameter	**Value**	**Description**
Input Shape	Sequences of length 10	The input data was reshaped into sequences of length 10 (using a rolling window), with each sequence containing one feature (since we are forecasting a single value).
Hidden Layers	20 units for the first LSTM layer, and 10 units for the second LSTM layer	The first LSTM layer has 20 units and returns sequences to maintain temporal context for the next LSTM layer. The second LSTM layer has 10 units and does not return sequences, as it's the final recurrent layer before the output.
Dropout	20% (first layer), 40% (second layer)	Dropout regularization was applied after each LSTM layer to prevent overfitting, with rates of 20% and 40% respectively.
Output layer	1 unit	A dense layer with one unit to output the forecasted value.
Optimizer	Adam with Lr: 0.001	Adam optimizer with a learning rate of 0.001.
Loss Function	MSE	Mean squared error (MSE), which is commonly used in regression tasks.
Epochs	20	The number of times the model iterated through the training data.

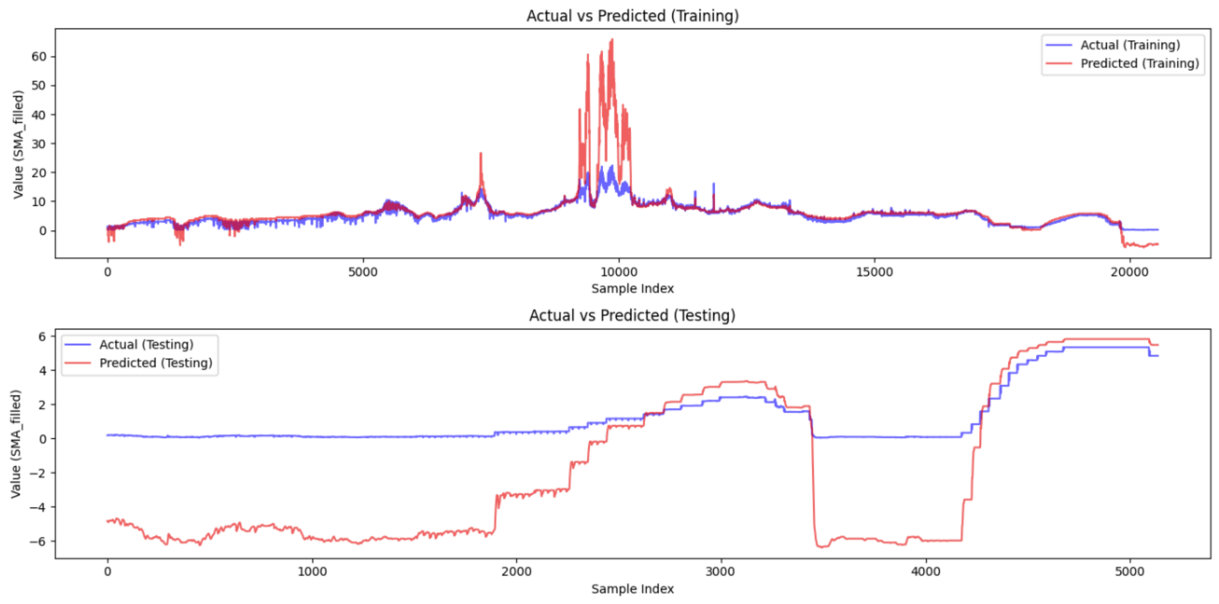
- **Before hyperparameter tuning:**

After executing the base model, we obtained the following evaluation metrics:

Metric	**Training Set**	**Testing Set**

Mean Squared Error (MSE)	25.24670	18.54051
Mean Absolute Error (MAE)	1.67518	3.53240
Root Mean Squared Error (RMSE)	5.02461	4.3058
R-squared (R^2)	-1.13478	-5.30009
Variance of Errors	24.05181	8.62064

The initial results show that the model performs reasonably well, although there are areas for improvement. The Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) are relatively low for both the training and testing sets, indicating that the model's predictions are fairly accurate. However, the Mean Absolute Error (MAE) is higher on the testing set, suggesting that there is more variation in the prediction errors for unseen data. The R-squared (R^2) values for both the training and testing sets are negative, indicating that the model is not capturing the variance in the data well. Additionally, the variance of errors is notably lower for the testing set, which suggests that the model's predictions are more stable on unseen data compared to the training set. The following plot illustrates the performance of the model:



However, the performance on the training set is relatively decent compared to the testing set, which shows a higher variation in the predictions. The negative R-squared values and the noticeable discrepancy between predicted and actual values on the testing set suggest that the model is not learning effectively from the training data and struggles to generalize to unseen data. To address this, we will proceed with hyperparameter tuning and explore potential improvements to enhance the model's ability to capture the underlying patterns in the data and improve its generalization.

- **Hyperparameter tuning:**

We employed a randomized hyperparameter search to optimize our LSTM model. Using the ParameterSampler from the sklearn library, we randomly sampled combinations of hyperparameters within predefined ranges. The best hyperparameter set was chosen based on the lowest Mean Squared Error (MSE) on the validation dataset. The table below shows the ranges explored for each hyperparameter and the optimal values identified:

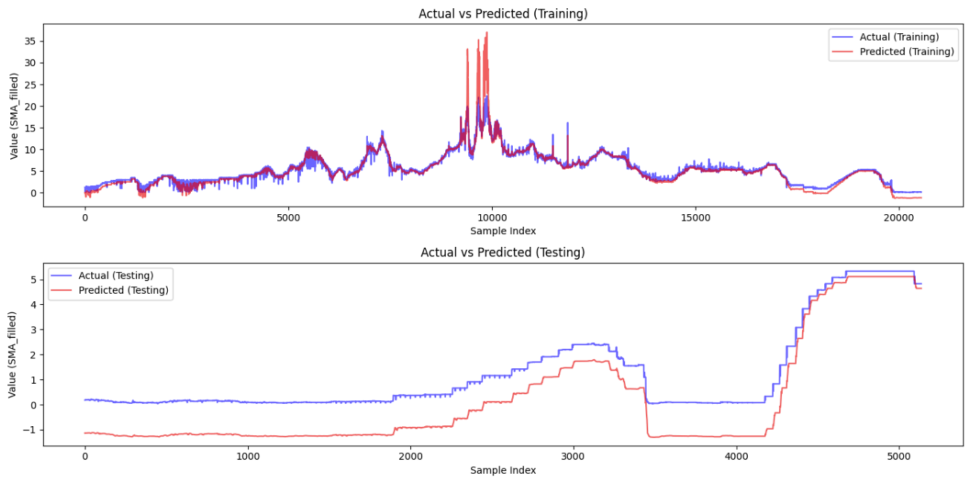
Hyperparameter	**Range**	**Best value**
Optimizer	Adam, RMSprop,SGD	RMSprop
Units	10 - 50 units	47
Dropout Rate	0.1 - 0.6	0.1103
Batch Size	2, 4, 8	8
Epochs	10 - 50	29
Learning Rate	0.001 - 0.01	0.0082

Among the tested optimizers, RMSprop was selected as the best-performing option, outperforming both Adam and SGD. For the number of hidden units, a higher value of 47 proved to be optimal, enabling the model to capture more complex patterns while maintaining stability. To address overfitting, a relatively small dropout rate of 0.1103 was chosen, providing sufficient regularization without compromising the model's learning capacity. The learning rate was fine-tuned to 0.0082, ensuring a good balance between convergence speed and training stability, allowing the model to learn efficiently without diverging.

The table below compares the initial model's performance with the tuned model, for the testing set:

Metric	**Initial Model**	**Tune Model**
MSE	18.54051	1.28317
MAE	3.53240	1.05798
RMSE	4.3058	1.13277
R^2	-5.30009	0.56397

The tuned model has a validation MSE of 1.2832, which is significantly higher than the overly optimistic MSE of the initial model. The initial model showed extremely low error values, indicating potential overfitting. Hyperparameter tuning addressed this by introducing a smaller dropout rate, adjusting the number of units, and using the RMSprop optimizer. The following plot illustrates the validation loss across epochs during the training process with the best hyperparameters:



The initial model struggled to generalize, with high variation in testing predictions and negative R-squared values. After hyperparameter tuning, the model's performance improved, reducing MSE from 1.597 to 1.283 and error variance from 1.5539 to 0.1644, indicating more stable and accurate predictions. The testing R-squared increased to 0.564, but further improvements are needed. The plot shows better alignment between the actual and predicted values on the testing set after hyperparameter tuning. We will continue fine-tuning to further enhance the model's ability to generalize.

Gated Recurrent Unit (GRU).

A Gated Recurrent Unit (GRU) is a type of Recurrent Neural Network (RNN) designed to handle sequential data, similar to LSTMs. However, GRUs simplify the architecture by using fewer gates. They have two main components: the update gate and the reset gate. The update gate determines how much of the previous memory should be carried forward, while the reset gate controls how much of the previous memory should be forgotten. This streamlined structure makes GRUs computationally more efficient than LSTMs while still effectively capturing long-term dependencies in sequences. GRUs are commonly used in time series forecasting, natural language processing, and other sequence-based tasks.

For this project, we implemented a base GRU model using the Keras library (with TensorFlow backend) to predict future values of the normalized time series. The following hyperparameters were used in the model:

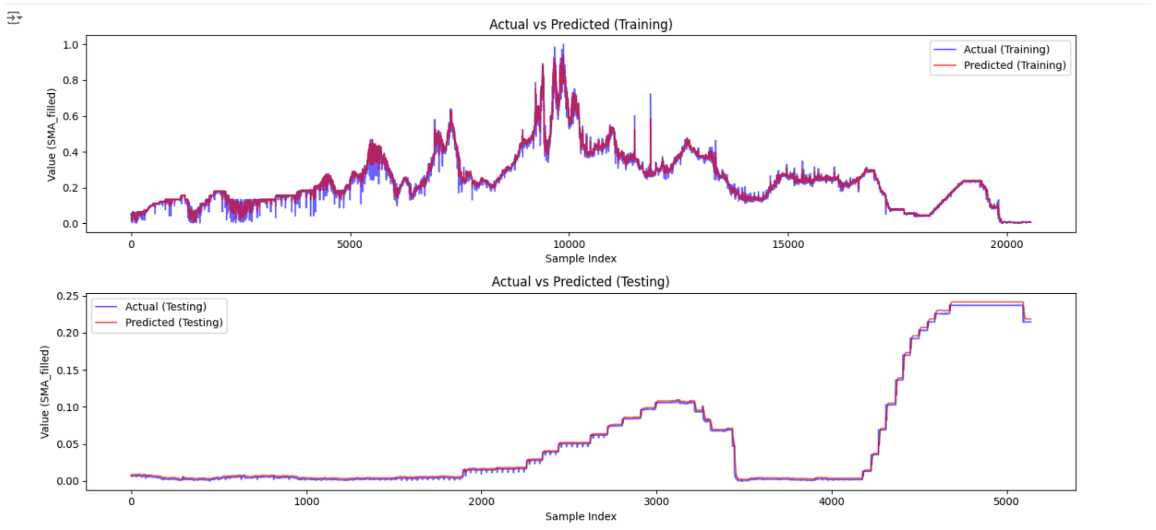
Hyperparameter	**Value**	**Description**
Input Shape	Sequences of length 10	The input data was reshaped into sequences of length 10 (using a rolling window), with each sequence containing one feature (since we are forecasting a single value).
Hidden Layers	20 units for the first GRU layer, and 10 units for the second GRU layer	The first GRU layer has 20 units and returns sequences to maintain temporal context for the next GRU layer. The second GRU layer has 10 units and does not return sequences, as it's the final recurrent layer before the output.
Dropout	20% (first layer), 40% (second layer)	Dropout regularization was applied after each GRU layer to prevent overfitting, with rates of 20% and 40% respectively.
Output layer	1 unit	A dense layer with one unit to output the forecasted value.
Optimizer	Adam with Learning Rate = 0.001	Adam optimizer with a learning rate of 0.001.
Loss Function	MSE	Mean squared error (MSE), which is commonly used in regression tasks.
Epochs	20	The number of times the model iterated through the training data.

- **Before hyperparameter tuning:**

After executing the base model, we obtained the following evaluation metrics:

Metric	**Training Set**	**Testing Set**
Mean Squared Error (MSE)	0.00025	9.60592e-06
Mean Absolute Error (MAE)	0.00891	0.00242
Root Mean Squared Error (RMSE)	0.01583	0.00309
R-squared (R^2)	0.98944	0.99837
Variance of Errors	0.00022	5.10206e-06

The updated results indicate significant improvement in the model's performance. The Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) are now exceptionally low for both the training and testing sets, showing that the model's predictions are highly accurate. Additionally, the Mean Absolute Error (MAE) is considerably lower, suggesting reduced variation in the prediction errors for unseen data. The R-squared (R^2) values have improved dramatically, with a high value on the testing set (0.9984), demonstrating that the model is effectively capturing the variance in the data. Furthermore, the variance of errors is much lower for the testing set, indicating greater stability in predictions for unseen data. The following plot illustrates the improved performance of the model:



The model performs well on both the training and testing sets, but this indicates potential overfitting, as the model is fitting too closely to the training data. While the R-squared values and errors suggest that the model is making accurate predictions on the training set, this high performance may not translate well to unseen data, as the model is not generalizing effectively. To address this issue and improve generalization, we decided to apply hyperparameter tuning, as we have done with other models, in order to create a more balanced and generalized model that can better capture the underlying patterns in the data.

- **Hyperparameter tuning:**

To improve the model's generalization, we performed a randomized hyperparameter search using the ParameterSampler from the sklearn library. We sampled different combinations of hyperparameters within specified ranges and selected the best set based on the lowest Mean Squared Error (MSE) on the validation dataset. The table below presents the ranges explored for each hyperparameter and the optimal values identified:

Hyperparameter	**Range**	**Best value**
Optimizer	Adam, RMSprop,SGD	RMSprop

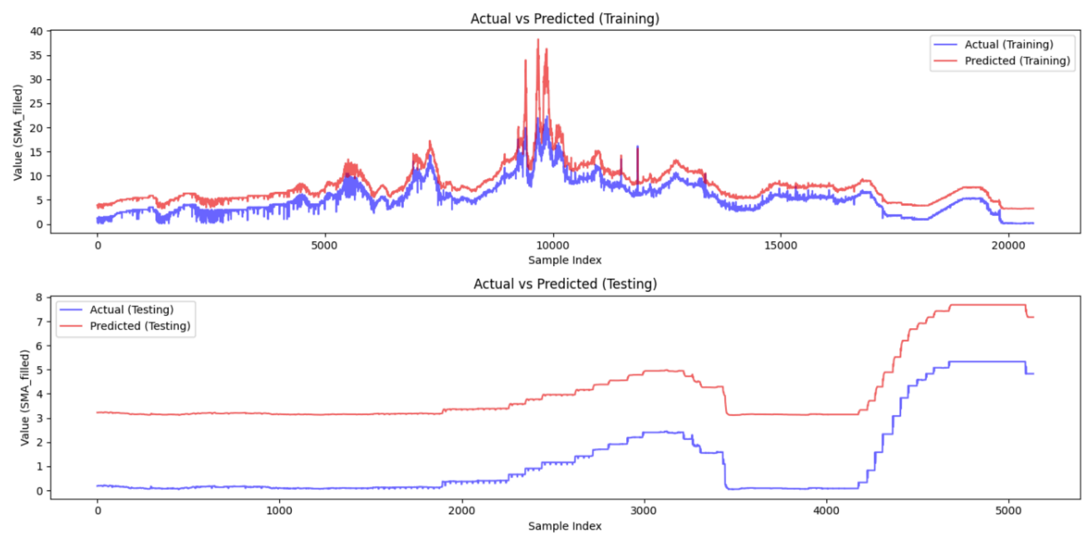
Units	10 - 50 units	15
Dropout Rate	0.1 - 0.6	01867
Batch Size	2, 4, 8	8
Epochs	10 - 50	26
Learning Rate	0.001 - 0.01	0.0036

Among the tested optimizers, RMSprop was selected as the best-performing option, outperforming both Adam and SGD. For the number of hidden units, a value of 15 was found to be optimal, allowing the model to effectively capture relevant patterns while preventing overfitting. To address overfitting, a dropout rate of 0.1867 was chosen, providing sufficient regularization without excessively limiting the model's learning capacity. The learning rate was fine-tuned to 0.0036, ensuring a good balance between convergence speed and training stability, which allowed the model to learn efficiently without diverging.

The table below compares the initial model's performance with the tuned model for the testing set:

Metric	**Initial Model**	**Tune Model**
MSE	9.60592e-06	8.20909
MAE	0.00242	2.85249
RMSE	0.00309	2.86515
R^2	0.99837	-1.78946

The tuned model has a validation MSE of 8.20909, which is significantly higher than the overly optimistic MSE of the initial model. The initial model showed extremely low error values, suggesting potential overfitting. Hyperparameter tuning addressed this by adjusting the dropout rate, changing the number of hidden units, and using the RMSprop optimizer. While the metrics for the tuned model are higher, this indicates that the model is now more generalized and better suited to real-world scenarios. The increased error values suggest a more realistic performance, as the model is no longer overfitting to the training data. The following plot illustrates the validation loss across epochs during the training process with the best hyperparameters:



The initial GRU model struggled to generalize, with high variation in testing predictions and relatively low error metrics. After hyperparameter tuning, the model's performance improved, the MSE from 9.60592e-06 to 8.20909 and error variance from 5.10206e-06 to 0.1644, indicating more stable and realistic predictions. The R-squared value for the testing set decreased to -1.78946, but this indicates the model is now more generalized and less prone to overfitting, offering a more realistic performance for real-world data. The plot shows better alignment between the actual and predicted values on the testing set after hyperparameter tuning. We will continue fine-tuning to further enhance the model's generalization ability.

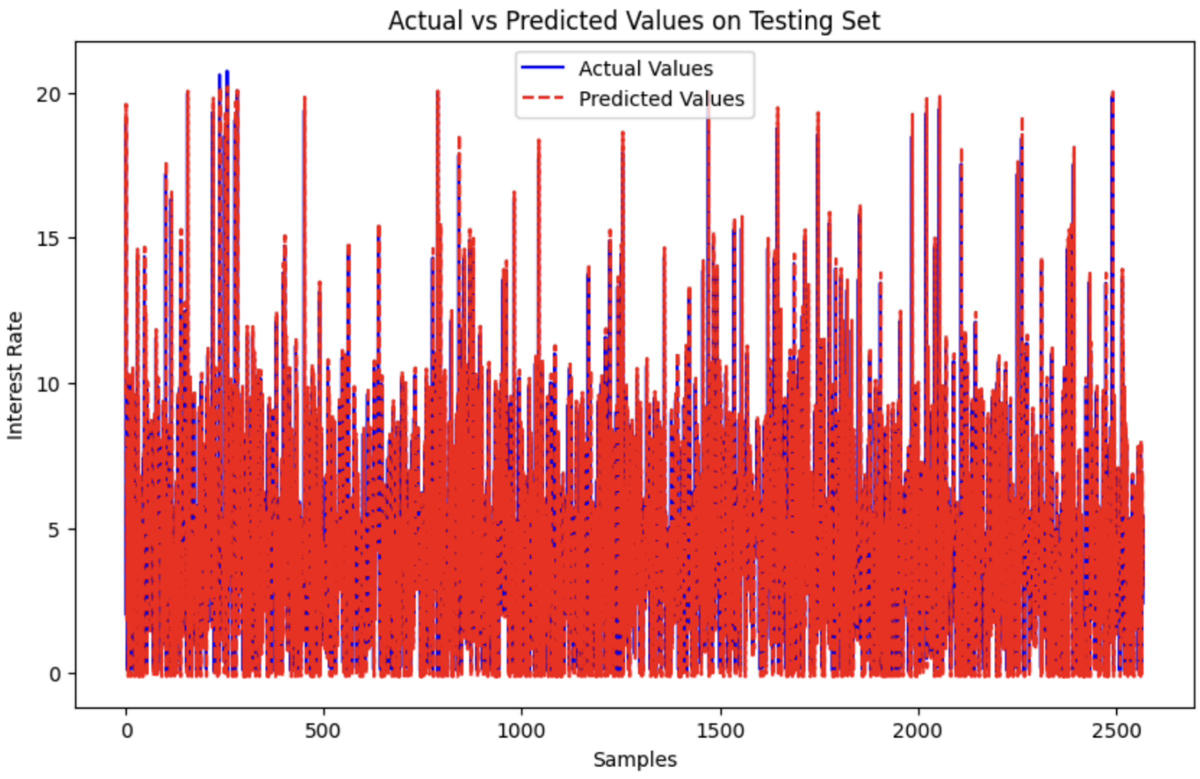
Transformers.

Transformer models are a type of deep learning architecture that excel in processing sequential data, such as our time series. Unlike traditional recurrent neural networks (RNNs), transformers do not process data sequentially. Instead, they use a self-attention mechanism which allows the model to consider all elements of the input simultaneously and weigh their importance relative to each other.

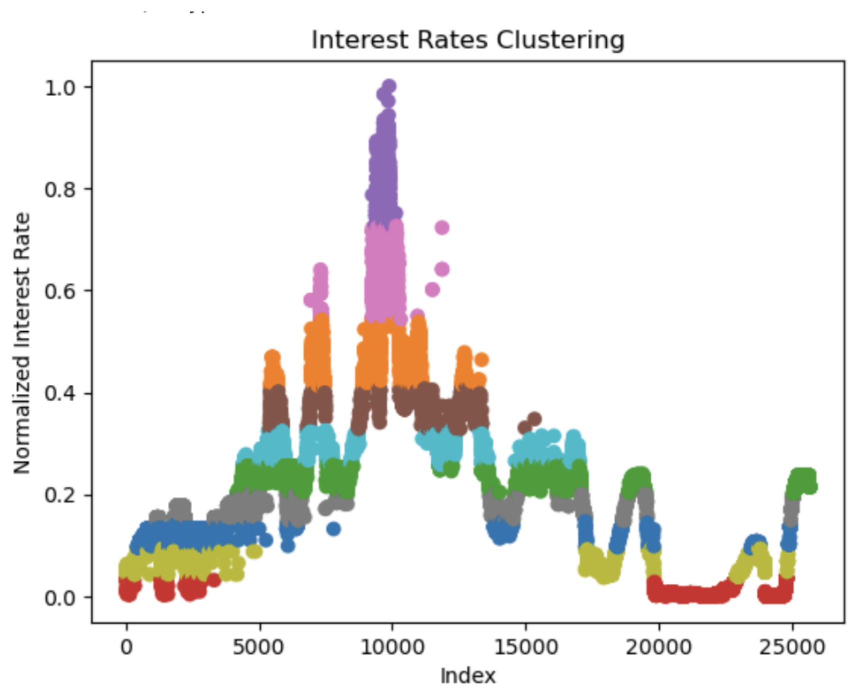
Initially, our approach involved utilizing this key aspect in our advantage by training a single model on the entire dataset using an 80-20 split, where 10% of the data was set aside for validation and another 10% for testing. However, this method revealed significant overfitting and inaccuracy, as the model struggled to generalize beyond the training data. Since it was discarded, this model was not included in our project submission.

Performance on Testing set:
Mean Squared Error (MSE): 26.6376

Mean Absolute Error (MAE): 3.9488
R² Score: -1.1086



The primary challenge stemmed from the model's difficulty in managing the diverse range of interest rate data all at once. To address this, we introduced a clustering algorithm to segment the dataset into smaller, more manageable groups. After careful consideration, we selected the K-means algorithm to divide the data into 10 distinct clusters. These clusters were designed to group data based on similar interest rate patterns, allowing the model to focus on the unique trends and behaviors within each group rather than being influenced by the entire sequence or the chronological ordering of the data. Even though we understand this may be a limitation in some sense, it also grants new capabilities to our model and it did reveal different insights that were not possible in the previous implementation.

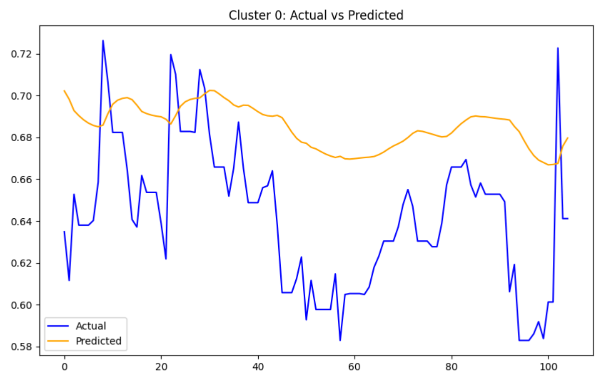
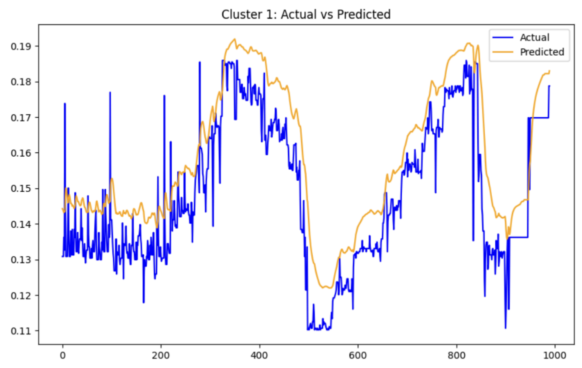
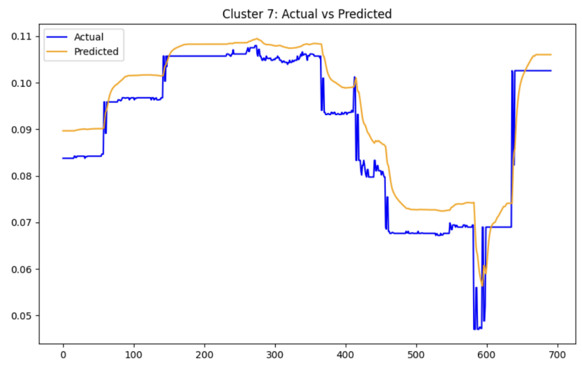
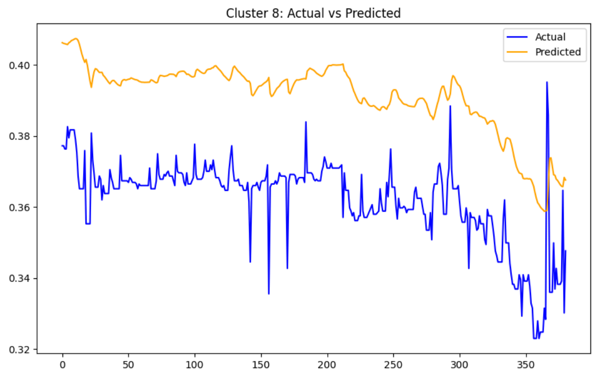


Preview of the clusters: it groups values by their normalized interest rates in 10 groups

With the clustered dataset, we trained an individual model for each cluster, effectively isolating the characteristics of each group. We chose a combination of LSTM + Transformer, which excels by capturing fine-grained, local dependencies and broader, global relationships in the data at the same time.

By distributing the modeling effort across 10 specialized models, we achieved better overall results compared to using a single model trained on the complete dataset, but the prediction was not as effective in some singular clusters. Nonetheless, this multi-model approach significantly enhanced the performance and generalizability of our individual models.

In conclusion, even if this model had some very good clusters, some others performed poorly, having high error rates visible in the attached notebook/python code, that made this option non desirable overall.

Cluster performance showcase	
	
	

5. Model Selection

For our final model selection, we conducted a thorough analysis of the key characteristics and performance metrics of each model to determine the most suitable one for the task at hand.

Model	**Best hyperparameters**	**Results**
Vasicek	Vasicek implemented on the training data before cross-validation had the highest accuracy	MSE: **1.7670** MAE: **1.0590** RMSE: **1.3293** R^2: **-0.9236**
CIR	CIR implemented on the training data before cross-validation had the highest accuracy	MSE: **0.9734** MAE: **0.7154** RMSE: **0.9866** R^2: **-0.0597**
RNN	Optimizer: **SGD** Units: **17** Dropout Rate: **0.1488** Batch Size: **4** Epochs: **29**	MSE: **0.70887** MAE: **0.79273** RMSE: **0.84194** R^2: **0.75912**

	Learning Rate: **0.00341**	
LSTM	Optimizer: **SGD** Units: **17** Dropout Rate: **0.1488** Batch Size: **4** Epochs: **29** Learning Rate: **0.00341**	MSE: **1.28317** MAE: **1.05798** RMSE: **1.13277** R^2: **0.56397**
GRU	Optimizer: **SGD** Units: **17** Dropout Rate: **0.1488** Batch Size: **4** Epochs: **29** Learning Rate: **0.00341**	MSE: **8.20909** MAE: **2.85249** RMSE: **2.86515** R^2: **-1.78946**
Transformers (LSTM + Transformer)	Optimizer: **Adam** Epochs: **50** Number of heads: **8** LSTM layers: **2** Dropout: **10%** Batch Size: **64**	**Model for cluster 8** Training Metrics: MSE: 0.0339 MAE: 0.1584 RMSE: 0.1841 R-squared: -0.4292 Variance: 0.0001

Given the following metrics, the best performance was achieved by the Transformers + LSTM model on cluster #8. However, due to its performance dropping on subsequent clusters and lacking consistency, we opted to use the LSTM model instead, as it demonstrated greater reliability across all clusters compared to the other models.

6. Limitations and further work

Our project encountered several limitations that are important to address:

Performance and Computational Efficiency:
One of the primary challenges was the computational cost and running time associated with training complex models such as LSTM and transformers. These and other of the used models are resource-intensive, particularly when dealing with large datasets or when experimenting with multi-model architectures like clustering and individual model training. This computational overhead limited the scope of experimentation and optimization we could perform within the project's constraints.

Dataset Limitations:
While our dataset provided historical interest rate values spanning several decades, it lacked additional features that could have enriched the predictive capabilities of our models. A more comprehensive dataset, incorporating economic indicators such as inflation rates, unemployment

figures, or GDP growth,might have allowed for more accurate predictions by providing contextual information beyond past interest rates.

Inherent Challenges of Interest Rate Prediction:
Predicting interest rates based solely on their historical values is inherently challenging due to the complex and multifactorial nature of interest rate movements. Interest rates are influenced by a wide range of external factors, including fiscal policies, geopolitical events, market sentiment, and economic shocks, which were not considered during our training phases. Consequently, relying exclusively on historical trends limited the predictive scope and accuracy of our approach, and this leaves room for further improvement.

Model Overfitting:
Despite efforts to mitigate overfitting, the high complexity of some models and the division of the dataset led to situations where the models struggled with generalization to unseen test data. This highlights the need for further optimization strategies.

These limitations provide valuable insights for future work, particularly the importance of exploring multimodal datasets, optimizing model architectures for efficiency, and incorporating external variables to improve predictive performance.

7. Conclusion

In this project, we explored various modeling techniques—including Vasicek, Cox-Ingersoll-Ross (CIR), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and Transformer-based models—to predict interest rates. Through this process, we identified key limitations and areas for improvement.

Computational efficiency, model complexity, our dataset and the nature of the task itself proved to be our main limitations. Despite them, our experiments with RNN, LSTM, and GRU highlighted their ability to capture sequential dependencies in interest rate data, with transformers as well due their capacity to retain long-term dependencies. However, we observed that these models, like the transformer, could suffer from overfitting, especially when trained on complex or imbalanced datasets. Employing techniques like regularization, dropout, and more effective data splitting could mitigate this issue in future iterations.

In conclusion, while the models used in this project demonstrated varying levels of success, there remains significant room for improvement. Future work could involve refining the model architectures, exploring better feature engineering strategies, and integrating external economic data. With these adjustments, the predictive capabilities of our models could be substantially enhanced, enabling more accurate forecasting of interest rates and potentially expanding the approach to other complex time-series prediction tasks.

8. Related work

This section discusses prior research relevant to understanding the influence of Federal Reserve policies, especially interest rates, on financial markets and economic conditions, leveraging machine learning techniques.

1. **Causal Analysis of the Federal Reserve's Impact on Fund Returns** Serenhov and Leite (2022) conducted a study that combined causal inference and machine learning techniques to assess how the US Federal Reserve's interest rate growth rate impacts fixed income and equity fund returns. Using a double machine learning framework, the study tested linear regression and gradient boosting to measure the average treatment effect of interest rate changes. The results revealed a significant impact on actively managed funds, with a 1% increase in interest rates leading to an 11.97% decrease in returns, highlighting the nuanced effect of monetary policy on financial markets (Serenhov & Leite, 2022).
2. **Survival of Community Banks Amidst Rising Federal Funds Rates** Zhang, Bian, and Tu (2023) focused on predicting the survival of community banks during periods of rising Federal Funds rates, particularly during the 2004–2008 period. Their study used various machine learning models, including regression, classification, and neural networks, to analyze how community banks navigated the financial pressures caused by increasing interest rates. The study employed FFIEC Call Reports Data to assess key factors influencing bank survival and aimed to predict future bank longevity under current economic conditions (Zhang, Bian, & Tu, 2023).
3. **Machine Learning for Exchange Rate Prediction** Goncu (2019) examined the use of machine learning regression techniques to predict exchange rates, specifically the Turkish Lira against the US Dollar. The study incorporated macroeconomic variables such as the Federal Funds rate, domestic interest rates, and money supply, demonstrating that machine learning models like Ridge regression provide more accurate predictions of exchange rate fluctuations compared to traditional economic models. The study underscored the importance of interest rates in determining exchange rate trends, highlighting the predictive potential of machine learning for macroeconomic forecasting (Goncu, 2019).
4. **Forecasting the Effective Federal Funds Rate Using Deep Learning** Alão (2023) explored deep learning models to forecast the US Federal Funds rate, utilizing algorithms such as Random Forest, LightGBM, and Long Short-Term Memory (LSTM). The study

identified key financial and labor market indicators that significantly influence the Federal Funds rate and demonstrated that deep learning models outperformed traditional models like ARIMA in forecasting accuracy. The research emphasized the potential of advanced machine learning techniques to assist in Federal Reserve policy-making by providing more precise predictions of monetary conditions (Alão, 2023).

5. **Combining the FT-Transformer with the LSTM Model to Predict Stock Prices** Sluis (2023) proposed a hybrid model combining the FT-Transformer and LSTM networks for stock price forecasting. The study focused on a dataset from FRED containing 9 financial variables, with the S&P 500 as the target variable. The FT-LSTM model was compared to four widely used Deep Learning models and evaluated across multiple forecasting horizons. The results showed that the FT-LSTM model outperformed the other models in most regression tasks, offering significant improvements in stock price predictions. The study also addressed issues such as overfitting, robustness, and hyperparameter tuning, demonstrating the model's effectiveness in real-world financial forecasting (Sluis, 2023).

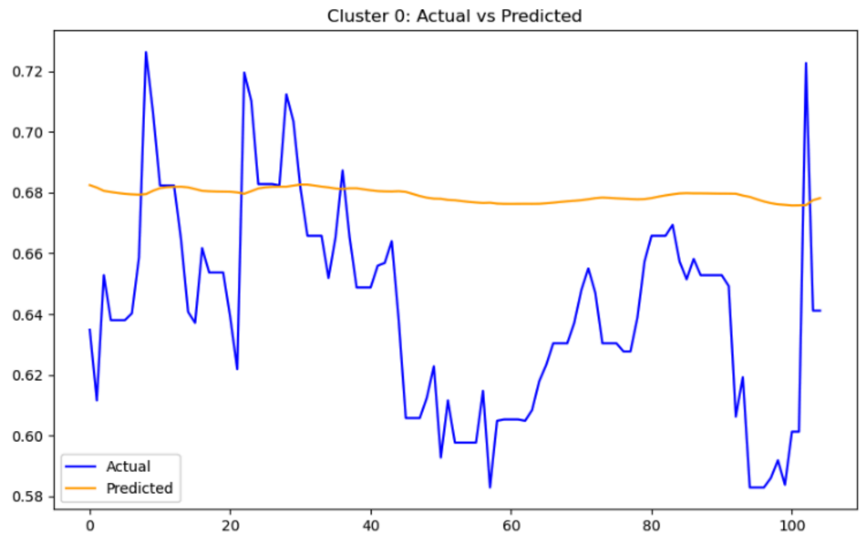
9. Citations

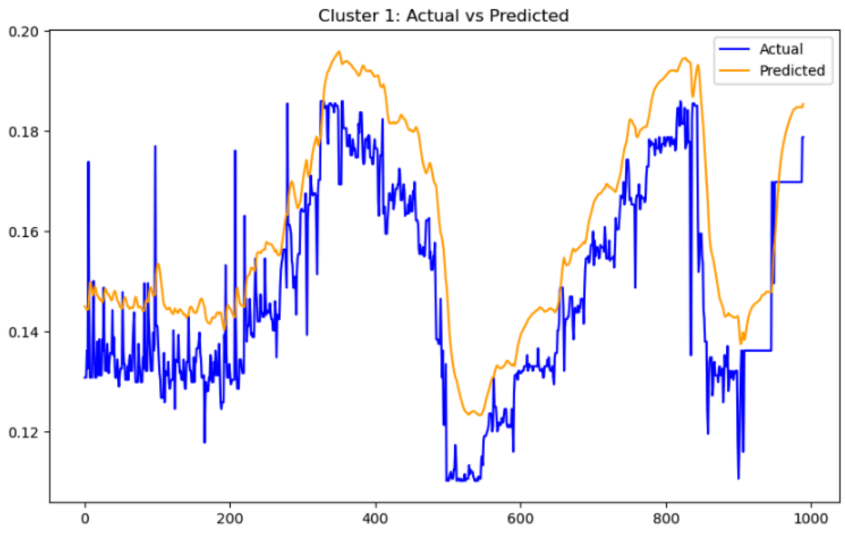
1. Alão, A. R. F. (2023). Effective Federal Funds Rate forecasting: Deep learning application. Master in Data Science, ISCTE - Instituto Universitário de Lisboa.
2. Cho, K., Merrienboer, B., Bengio, Y., & Gulcehre, C. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), 1724-1734. https://aclanthology.org/D14-1179.
3. Cox, J. C., Ingersoll, J. E., & Ross, S. A. (1985). A theory of the term structure of interest rates. Econometrica, 53(2), 385-408. https://doi.org/10.2307/1911242.
4. Federal Reserve. (n.d.). Federal funds rate. Federal Reserve. Retrieved December 17, 2024, from https://www.federalreserve.gov.
5. Foster, Sarah. "Federal Funds Rate History: 1980 through the Present." Bankrate, 7 Nov. 2024, www.bankrate.com/banking/federal-reserve/history-of-federal-funds-rate/#1981.
6. Goncu, A. (2019). Prediction of exchange rates with machine learning. Department of Mathematical Sciences & Research Institute of Quantitative Finance, Xi'an Jiaotong Liverpool University, Suzhou, China. Hedge Fund Research Center, SAIF, Shanghai Jiaotong University, Shanghai, China.
7. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735.
8. Serenhov, O., & Leite, W. T. (2022). Is the Federal Reserve causing funds to underperform? A causal machine learning analysis. Master's Programme in Data Analytics and Business Economics, DABN01.

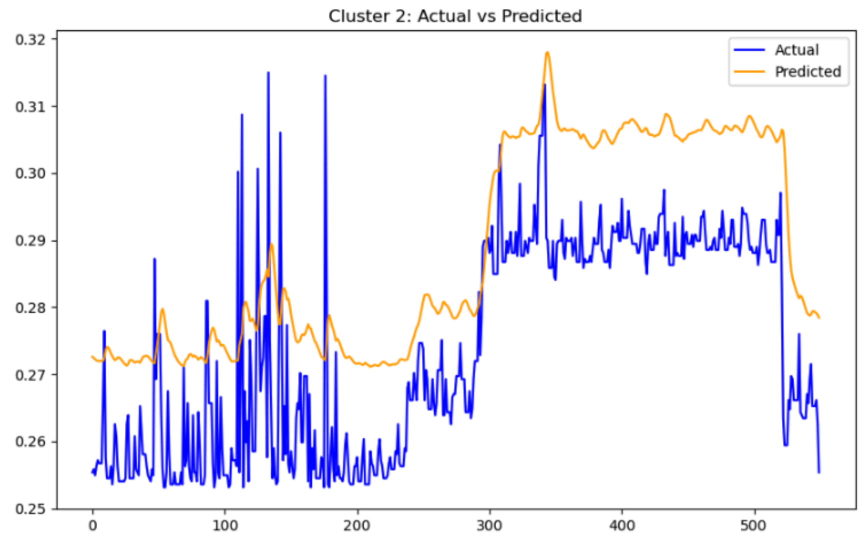
9. Shi, J., Wang, S., Qu, P. et al. Time series prediction model using LSTM-Transformer neural network for mine water inflow. Sci Rep 14, 18284 (2024). https://doi.org/10.1038/s41598-024-69418-z.
10. Sluis, E.B.C. (2023). Combining the FT-Transformer with the LSTM model to predict stock prices. Econometrie. Retrieved from http://hdl.handle.net/2105/67716.
11. Sluis, E. (2023). Combining the FT-Transformer with the LSTM model to predict stock prices. BSc Econometrics and Operational Research Thesis, Erasmus School of Economics. Supervisor: M. Mueller, PhD candidate, Econometrics; Second Assessor: Dr. C. Cavicchia.
12. Vasicek, O. A. (1977). An equilibrium characterization of the term structure. Journal of Financial Economics, 5(2), 177-188. https://doi.org/10.1016/0304-405X(77)90021-6.
13. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. A., Kaiser, Ł., Polosukhin, I. (2017). Attention is all you need. Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS 2017), 6000-6010. https://doi.org/10.5555/3295222.3295341.
14. Zhang, Y., Bian, Y. (N.), & Tu, Q. (2023). Predictive modeling of community bank survival under increasing Federal Funds rate. Graduate School of Arts & Sciences, Georgetown University.
15. Dataset: “Federal Funds Rate - 62 Year Historical Chart.” MacroTrends, www.macrotrends.net/2015/fed-funds-rate-historical-chart. Accessed 17 Dec. 2024.

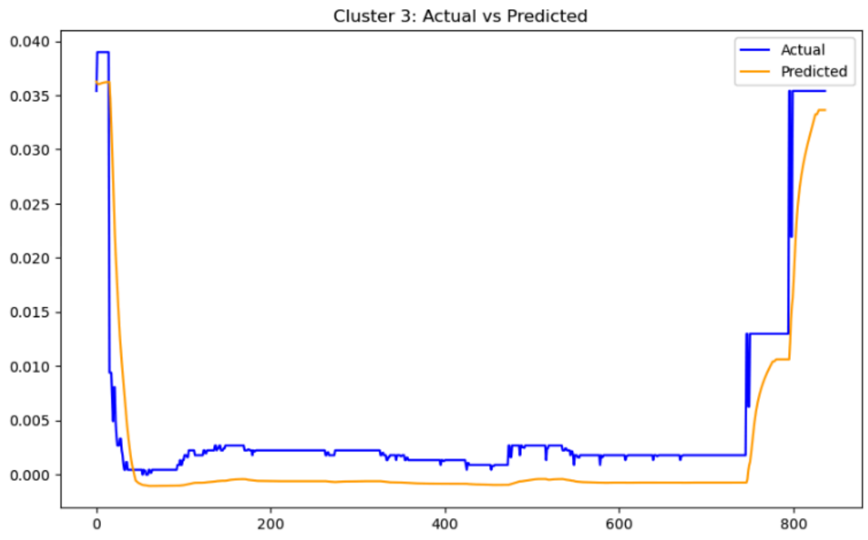
10. Appendix

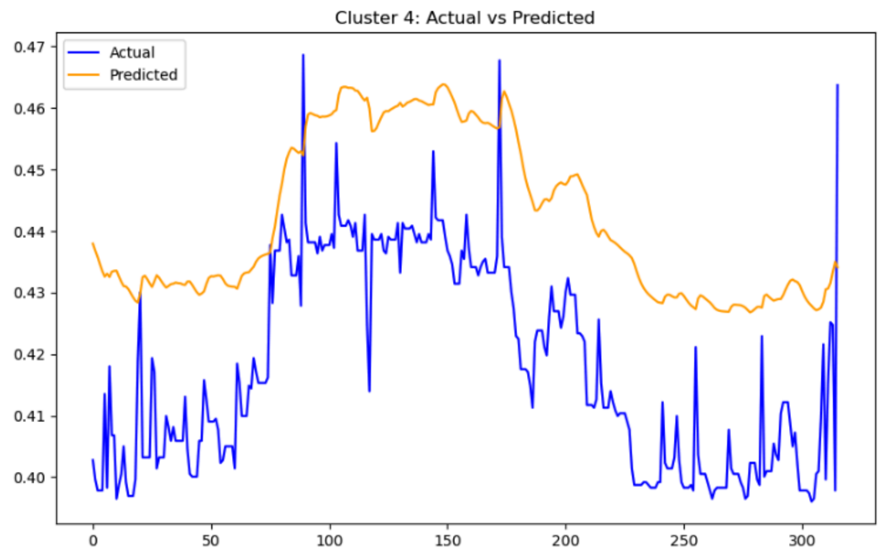
Transformers + LSTM cluster outputs

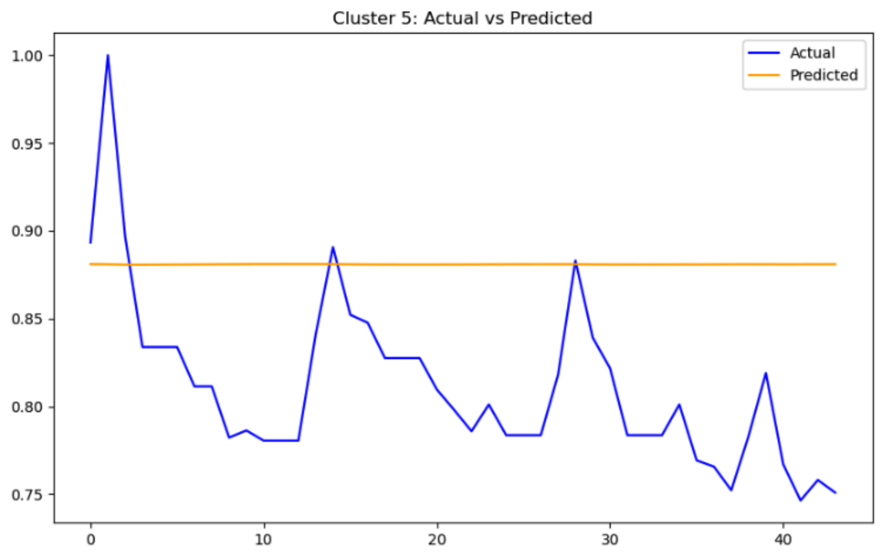


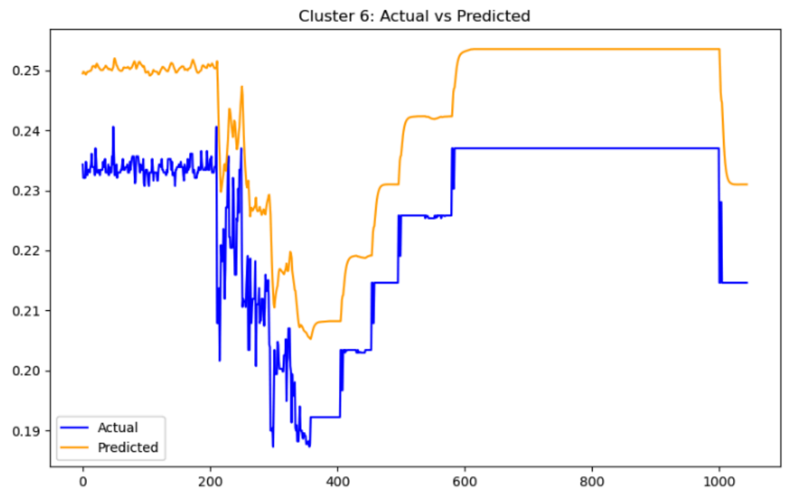


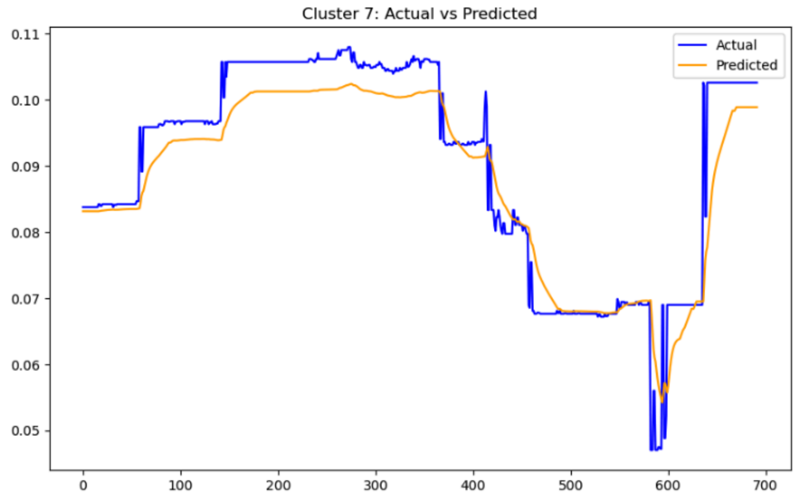

Cluster 2: Actual vs Predicted
Actual
Predicted

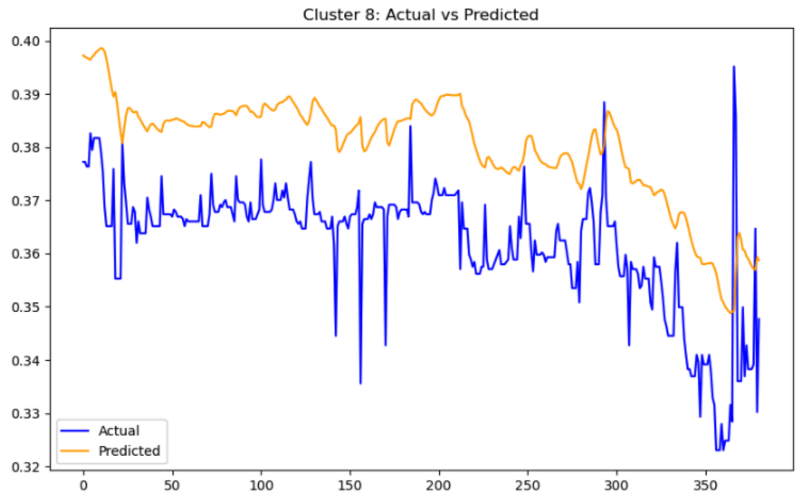

Cluster 3: Actual vs Predicted
Actual
Predicted

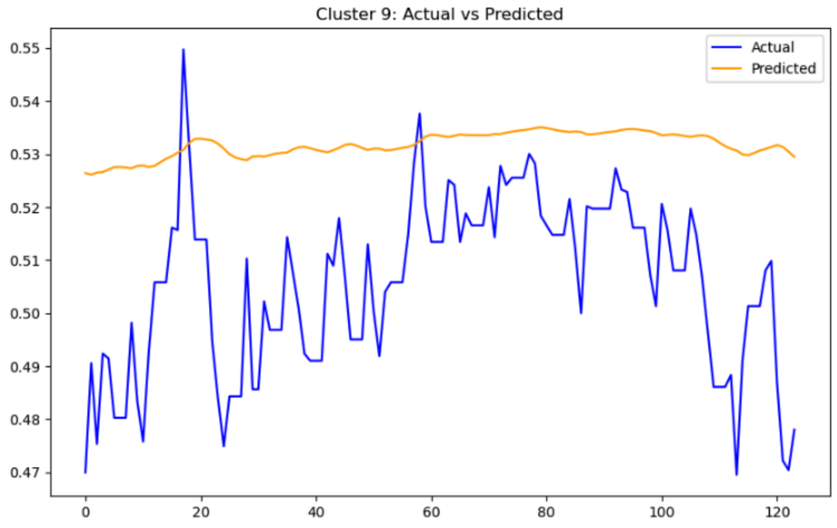

Cluster 4: Actual vs Predicted
Actual
Predicted


Cluster 5: Actual vs Predicted
Actual
Predicted
1.00
0.95
0.90
0.85
0.80
0.75
0
10
20
30
40


Cluster 6: Actual vs Predicted
Actual
Predicted
0.25
0.24
0.23
0.22
0.21
0.20
0.19
0
200
400
600
800
1000


Cluster 7: Actual vs Predicted
Actual
Predicted
0.11
0.10
0.09
0.08
0.07
0.06
0.05
0
100
200
300
400
500
600
700


Cluster 8: Actual vs Predicted
Actual
Predicted
0.40
0.39
0.38
0.37
0.36
0.35
0.34
0.33
0.32
0
50
100
150
200
250
300
350


Cluster 9: Actual vs Predicted
Actual
Predicted
0.55
0.54
0.53
0.52
0.51
0.50
0.49
0.48
0.47
0
20
40
60
80
100
120