PREDICTION OF FEDERAL FUNDS RATE USING FEDERAL RESERVE SPEECHES

Fariya Bano (fb198) - 219035744

University of Leicester

Abstract

This project report presents a comprehensive analysis of the Federal Reserve speeches, somplemented by the economic data of Unites States of America, to predict the Effective Federal Funds Rate (EFR) using advanced technology, including Natural Language Processing (NLP), Deep Learning, and Machine Learning Techniques. The EFR is a critical indicator of the United States' monetary policy, making its accurate prediction essential for various stakeholders like financial market participants, policymakers, economists, foreign exchange traders, etc.

The project study begins by collecting and processing vast corpus of Federal Reserve speeches through web scraping, extracting valuable insights and performing sentiment analysis using NLP and Deep Learning methods. The semantic richness of these texts is then leveraged to gain a nuanced understanding of the Federal Reserve's sentiments and forward guidance.

Simultaneously, a diverse set of economic indicator data is incorporated, such as GDP, inflation rates, unemployment rate, exchange rates of EUR in USD, to create a comprehensive feature set for modelling the EFR. Furthermore, machine learning algorithms are employed along with industry standard evaluation metricsto investigate the best algorithm for future study and fine tuning.

The project aims to contribute to a better understanding of the relationship between Federal Reserve communications and monetary policy outcomes, potentially providing valuable insights for investors and policymakers. The results of this research demonstrate the potential for improving EFR predictions through the integration of textual data and economic indicators, while also presenting to best models to work with.

Keywords: Federal Reserve, Natural Language Processing, Machine Learning, Prediction, Deep Learning, Sentiment Analysis.

TABLE OF CONTENTS

Abstract 2
INTRODUCTION 5
What is Federal Reserve (Fed) ? 6
The Federal Open Market Committee (FOMC) 6
Federal Funds Rate 7
THE DATA 8
Non-Textual Data 8
Textual Data 10
Word Embeddings 11
SYSTEM AND SOFTWARE 12
Google Colab 12
Python 12
METHODOLOGY 14
Web Scraping 14
Augmented Dicky Fuller Test 14
Correlation and Regression 14
Machine Learning Models 15
FB Prophet 15
Random Forest Regression 16
Extreme Gradient Boosting (XG Boost) 17
Deep Learning 18
Evaluation Metrics 20
ANALYSIS 22
Exploratory Data Analysis 22
Assigning Sentiment 31
Prediction with and without Sentiment 32
Evaluation of Models using economic data 40
Analysis with only Sentiment Data 41
RESULTS 46
References 47

TABLE OF FIGURES

Figure Number	*Description*	*Page Number*
1.1	Project workflow	5
1.2	Federal Reserve system logo	6
4.1	Diagram of RNN, LSTM & GRU	18
4.2	Workings of RNN	18
4.3	Workings of GRU	19
5.1	Economic indicator dataset initial	22
5.2	Economic Indicator dataset after filling null values	22
5.3	Descriptive statistics on Economic indicator dataset	23
5.4	Boxplot of Economic indicators data	23
5.5	Joint plot of Economic indicators data	25
5.6	Line Chart of differences in Effective Funds Rate	28
5.7	Text Analysis of Fed Statements	29
5.8	Word count hisogram of speeches	30
5.9	BankFin Word Embedding Vectors	31
5.10	Deep Network flow for assigning sentiments	32
5.11	FB Prophet (without sentiment) forecast plot	36
5.12	FB Prophet (without sentiment) components plot	36
5.13	FB Prophet (with sentiment) forecast plot	37
5.14	FB Prophet (with sentiment) components plot	37
5.15	FB Prophet(without sentiment) result comparison	38
5.16	FB Prophet (with sentiment) result comparison	38
5.17	XG Boost (without sentiment) model feature importance assessment	39
5.18	XG Boost (with sentiment) model feature importance assessment	40
5.19	FB Prophet (only sentiment) forecast plot	43
5.20	FB Prophet (only sentiment) components plot	43
5.21	FB Prophet (only sentiment) result comparison	44

INTRODUCTION

The ability to anticipate changes in key interest rates can be a huge game-changer for investors, businesses, policymakers, etc., in the landscape of monetary policies and financial markets. Among these interest rates, the Federal Funds Rate, set by the Federal Reserve, holds a key position as it profoundly influences the overall economic environment in the United States. Leveraging current technology to predict its movements could provide a huge advantage to anyone successful in this pursuit. In economics and finance especially, a model that can correctly predict a variable could have a profound effect on the workings of the world.

This project begins at the intersection of finance, economics, and computational linguistics. Its objective is to explore the hypothesis that the sentiment embedded within Federal Reserve speeches and statements can offer valuable insights for predicting future movements of the Federal Funds Rate. With the use of NLP, the aim is to extract nuanced information from the textual data and gauge the sentiment, tone, and potential policy that may not be fully reflected in the traditional economic indicators. Finally, we apply numerous emerging prediction algorithms and compare the results using prediction evaluation metrics.

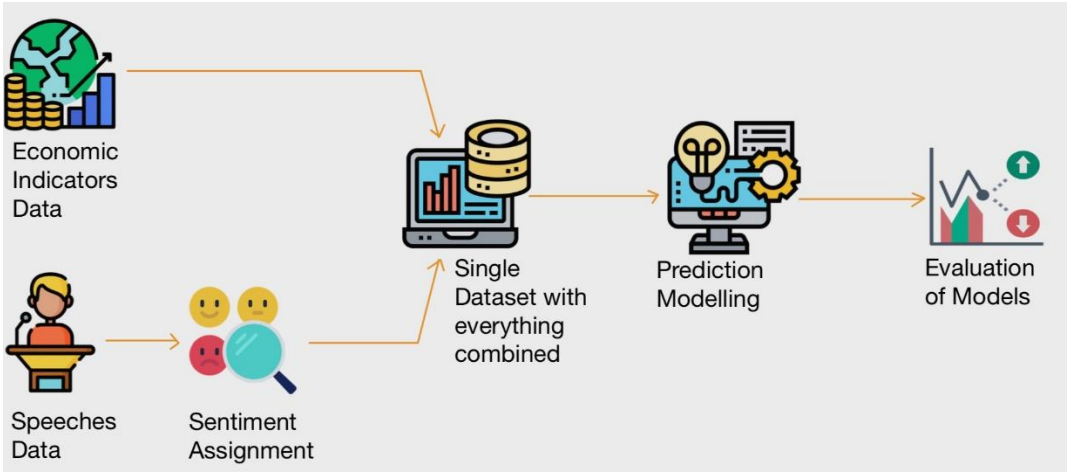


Fig 1.1. Project Workflow Diagram

As illustrated in Figure 1.1, the project's workflo involved several key steps. Initially the inout data included the economic indicators and the speeches data. The speeches data undergo a transformation, converting them into a numerical sentiment score. These sentiment scores are then integrated into the final dataset alongside the economic indicator data. This final dataset serves as the foundation for creating both the training and testing datasets utilized in the predictive modelling phase of the project. The primary objective, ultimately, it

to assess and evaluate various models used in this study to determine their effectiveness and potential to contribute to the Financial and Economic Industry.

Before explaining the details of the project, it is important to gain sufficient prerequisite knowledge regarding the key subjects of the project.

What is Federal Reserve (Fed) ?

The Federal Reserve, often referred to as the "Fed", is the central bank of the United States established in 1913. It is one of the most powerful financial institutions in the world. It plays a crucial role in providing the country with a safe, flexible, and stable monetary and financial system while promoting economic growth. In order to conduct national monetary policy and maintain financial stability the Fed gas a monetary policy-making body, the Federal Open Market Committee, which manages the country's money supply.[1]

Fig 1.2. Federal Reserve System Logo [45]

The Federal Open Market Committee (FOMC)

The FOMC is the branch of the Federal Reserve that is responsible for making decisions regarding the monetary policy, specifically by controlling the interest rates. The committee consists of twelve voting members, which includes seven members of the Board of Governors and 5 presidents of regional Federal Reserve Bank. The President of the Federal Reserve Bank of New York is always a permanent member of the FOMC, while the other regional bank presidents serve on a rotating basis.[2]

These twelve members of the FOMC always meet eight times a year. These are scheduled in advance, while they may have unscheduled meetings if the need arises. During these meetings they discuss the current economic state of the country and discuss whether there should be any changes to near-term monetary policy. A vote to change policy is made and based on the policy, the U.S government would either buy or sell securities to promote economic growth.

Members Sentiments

The FOMC Members are often characterised as being hawkish or dovish based on the type of monetary policy they drive.[3] Hawks are focused on limiting inflation, hence focus short term increase of interest rates by limiting supply of money. Doves focus on increasing money supply and economic growth to drive the interest rates to lower.

Federal Funds Rate

The Federal Funds Rate is the interest rate at which banks and credit unions with excess reserves lend funds overnight to other banks and credit unions. It is a crucial benchmark for short terms interest rates in the United States and has a significant impact on the broader economy. So, these transactions directly impact Eurodollar.

How the Federal Funds Rate works:

- When the FOMC wants to stimulate economic growth, lower unemployment, or combat a recession, it typically lowers the Federal Funds Rate. Lower interest rates encourage borrowing and spending, which can stimulate economic activity.
- When the Fed wants to cool down an overheated economy to prevent inflation, it raises the Federal Funds Rate. Higher interest rates can discourage borrowing and spending, thus slowing down economic growth.

THE DATA

The data used in this project can be divided into textual data and non-textual data. The combination of the textual analysis of Federal reserve Speeches and the non-textual economic indicator data provide a holistic approach to the project. The data used in the project greatly varies in the way it was obtained and application. We will delve into the different types of data used in this project, the reason they were used and they methodology of sourcing them.

Non-Textual Data

To study the effects of textual data on the non-textual data, it is important to source the relevant economic indicators. The FOMC makes its decisions based on key economic indicators that are bound to show signs in inflation, recession, etc.[3] Thus, the economic indicators provide the contextual information about the economic landscape during the periods corresponding to the Federal Reserve Speeches. This contextual information helps us align the speech sentiment with the broader economic trends.

Data Source

All the economic indicator data was sourced from the Federal Reserve Economic Data (FRED) Website. The FRED is an online database consisting of hundreds of thousands of economic data in time series format from various national, international, public, and private sources. It is created and maintained by the Research Department at the Federal Reserve Bank of St. Louis.[6] The data is accessible in various formats (csv, xlsx, etc). All the data in this project sourced from FRED was downloaded in .xlsx format.

Feature Variables(Economic Indicators)

1. *Gross Domestic Product Data of the USA* – This is the market value of the goods and services produced by the labour and property located in the USA. [7]
    a. Source – U.S. Bureau of Economic Analysis
    b. Units - Billions of Dollars, Seasonally Adjusted Annual Rate
    c. Frequency – Quarterly
2. *Unemployment Rate of the USA* – It represents the number of unemployed as a percentage of the labour force.[8]
    a. Source – U.S. Bureau of Labor Statistics
    b. Units – Percent, Seasonally adjusted
    c. Frequency – Monthly
    d. Restrictions –
        i. People aged 16 years or above

ii. People who do not reside in institutions (penal, mental facilities or homes for aged)
iii. People who are not on active duty in the Armed Forces

3. Consumer Price Index for all Urban Consumers: All Items in U.S. City Average – It is the price index of a basket of good and services paid by urban consumers. It can also represent buying habits of urban consumers. It is based on prices for food, clothing, shelter, fuels, etc.[9]
    a. Source – U.S. Bureau of Labor Statistics
    b. Units – Percent, 1982 – 1984 = 100, Seasonally adjusted
    c. Frequency – Monthly
4. U.S. Dollars to Euro Spot Exchange Rate - Noon buying rates in New York City for cable transfers payable in foreign currencies.[10]
    a. Source – Board of Governors of the Federal Reserve System (US)
    b. Units – U.S. Dollars to One Euro, Not Seasonally Adjusted
    c. Frequency – Daily
5. 5-Year Breakeven Inflation Rate – This represents the expected inflation derived from the 5- year Treasure Constant Maturity Securities. The latest value implies what market participants expect inflation to be in the next 5 years.[11]
    a. Source – Federal Reserve Bank of St. Louis
    b. Units – Percent, Not seasonally Adjusted
    c. Frequency – Daily
6. 10-Year Breakeven Inflation Rate - This represents the expected inflation derived from the 10- year Treasure Constant Maturity Securities. The latest value implies what market participants expect inflation to be in the next 10 years.[12]
    a. Source – Federal Reserve Bank of St. Louis
    b. Units – Percent, Not Seasonally Adjusted
    c. Frequency – Daily

Target Variable

1. Effective Federal Funds Rate - The Federal Reserve announces the effective fed funds rate at the end of each trading day, which is the weighted average rate for all transactions in the market that day. It consists of domestic unsecured borrowings between various government sponsored entities.[13]

a. Source – Federal Reserve Bank of New York
b. Units – Percent, Not seasonally Adjusted
c. Frequency – Daily

Textual Data

An indication of the FOMC's economic inclination can be determined through the speeches of the various members; this could be at any events attended by the members where the speeches are recorded and publicly released by the Federal Reserve. On surface level, these speeches appear vague, with the expected appearance of some terms like "inflation", "economy", etc. The job of this model, however, is to determine the hidden sentiment behind these speeches and consequently relate to the expected change in the federal funds rate.

The textual data was obtained through web scraping methods, the code for which is presented in the appendix.

1. FOMC Statements – The FOMC issues statements eight times a year after each scheduled meeting. The main information that the statements convey are the FOMC's monetary policy stance and the outcome of their deliberations on the Federal Funds Rate. This outcome could be whether the FOMC decided to raise, lower, or maintain the Funds Rate.[14]
2. Federal Reserve Speeches – Federal Reserve Speeches refers to the speeches made by the voting members of the FOMC. These are publicly available transcripts of the speeches on the Federal Reserve Website. The speeches typically indicate potential future monetary policy. For this analysis, a large amount of speeches data was required and a simple case of downloading would be tediuous and pedestrian. Hence, I have applied a web scraping script to do this job in second. Through this I was able to obtain 630 speeches.
3. Financial Phrasebank – It is a dataset available online containing 4840 sentences obtained from various financial news. These sentences are pre-labelled with the sentiments positive, negative and neutral. According to hugging face, 59.4% are marked neutral, 28.1 % are marked positive while 12.5% are marked negative. These 4840 sentences were annotated by 16 people in which three were researchers and remaining 13 were students from Aalta University School of Business majoring in finance, accounting and economics.[33]

Word Embeddings

To process the speeches data and to extract meaningful information for each given word and then to convert it into a set of real numbers, it is essential to use word embeddings. Word embeddings provide a method to represent words as continuous vectors in a multi-dimensional space, capturing semantic and syntactic relationships between words.[44] These vector representation of words facilitate machine learning algorithms in understanding the context and meaning of words in a text.[46]

- Semantic Similarity: Word Embeddings facilitate the assessment of semantic relatedness among words. Word that share similar meanings exhibit proximity within the vector spacy. For instance, in the vector space, “king” and “queen” would be positioned closer together than then words like “king” and “car”. [47][48]
- Word Arithmetic: Word embeddings also allow to perform word arithmetic operations such by determining relationships between the vectors. For e.g.: “king – man + woman” would give us “queen”.[48]
- Dimensionality: Dimensionality refers to the number of dimensions in which the vector representation of a word is defined. [49] The dimensionality can vary, but the most common choices are 50,100,200 or 300 dimensions. The choice of dimensionality can impact the quality of representation.

In the words of the developer, BankFin [30] is a “customized word embeddings pre-trained on banking and finance terms which will be helpful in analysing and classifying financial sentiments or stock price sentiment analysis”. As this project is based on Federal Reserve speeches, consisting of financial vocabulary, a word embedding trained on financial text is critical to assign sentiments correctly. Also, the BankFin word embedding contains 100 dimensions of each financial word.

SYSTEM AND SOFTWARE

The Data was collected from various sources through Scraping and download as mentioned previously. In this section, I will describe the software used to build this project.

Google Colab

Google Colab is an interactive Jupyter notebook hosted code editor which is especially beneficial for Deep Learning tasks.[15] It does not require any additional setup, accessed through the browser, and allows access to Google's free computing resources such as GPUs and TPUs.[16]

Python

Various Python libraries have been used throughout the project was various purposes. These will be stated below.

- Data Manipulation (Numpy and Pandas) – NumPy and Pandas as two essential libraries in Python that provide powerful tools for data manipulation, analysis, and computation. Numpy not only offers support for arrays and matrices, but also provides a wide range of mathematical operations to work with them efficiently.[55] Pandas, on the other hand, is built on top of NumPy and provides high-level data structures and functions for data manipulation and analysis.[56]
- Data Visualization (Matplotlib and Seaborn) – Matplotlib and Seaborn are two powerful python libraries for data visualization, widely used for creating a wide range of plots and charts. Matplotlib is versatile and offers comprehensive set of functions for creating various plots and also customizing them.[57] Seaborn built on top of Matplotlib provides a higher-level interface for creating aesthetically please and interactive and informational statistical visualizations.[58]
- Statistical Analysis (statsmodels) – Statsmodels is a Python library for estimating and interpreting statistical models. It provides a wide range of tools for conducting statistical analysis and modelling, making it a valuable resource for data analysts, including linear and non-linear modelling.[59]
- Machine Learning Tasks (Scikit-Learn) – Scikit-learn is often referred to as sklearn. It is a widely used machine learning library in Python. It is renowned for its simplicity, versatility, and robustness,[60] making it a valuable tool for both beginners and experiences data scientists and machine learning

practitioners. It is widely respected library with a user-friendly interface, extensive range of algorithms, and rich documentation.

- Data Scraping (Beautiful Soup) – Beautiful Soup is popular open source python library for web scraping. It provides convenient way to navigate and manipulate web data, making it a valuable tool for extracting information from websites in a structured manner. It provides users with a high-level API to interact with web content.[61]
- NLP Tasks (nltk, genism and spacy) – NLTK, Gensim and spaCy are popular python libraries that are fundamental for various Natural Language Processing (NLP) tasks, such as text analysis, text processing and language modelling.[62]
- Deep Learning Tasks (tensorflow and keras) – Tensorflow and Keras are some fundamental Python libraries for deep learning and neural network development. They play an important role in building and training artificial neural networks, making them essential tools for machine learning practitioners, researchers, etc.[63] They offer a powerful and flexible framework for not only building but also deploying machine learning models.
- Transferring Data and Models between Python files (pickle) – Pickle is a model in Python that provides a method for serializing and deserializing python objects.

METHODOLOGY

Web Scraping

Web scraping is the process of extracting information directly from a website by utilising the html tags that are responsible for building the website. It allows the scraper to collect and store the data in a format convenient to them. The idea is to provide the url to the code and then the parsing code will find the URL and extract the html content on the page. Then within the html content obtained it will look for the chosen html tags and extract the data found within the tags.[17] In this project, for the Speeches and Statements Data, I have extracted elements like the Data, Speaker and the Text. This can be easily coded with Beautiful Soup and inspecting the website that needs to be scraped.

Augmented Dicky Fuller Test

As mentioned earlier, some of the models used in this project require the timeseries data to be stationary. When we say that a data is stationary, we mean that the data shows stationarity. Stationarity is the condition in which the data when plotted in a simple line graph does not show any trend and no periodic fluctuations.[16] These can be hard to spot and identify through graphs alone. Hence, we rely on a standard statistical test to check for Stationarity. This test is the Augmented Dicky-Fuller (ADF) Test. Using the ADF test, there are 2 hypothesis in the test. The null hypothesis is that the data is not stationary, while the alternative hypothesis is that the data is stationary.[18]

When the data is found to be stationary the, the easier method to make it stationary is to apply differencing to remove the trends. By differencing, we take a new value which is the difference between the original value and the previous value. [19]

Correlation and Regression

We say two variables are correlated when there exists a relationship between their values. This relationship could be positive or negative. A correlation coefficient is the numerical denotation of the relationship. The correlation coefficient ranges from +1 to -1 where +1 represents a positive relationship (when value of one variable increases with an increase in the value of the other variable), -1 represents a negative relationship (when value of one variable decreases with the increase in the value of the other variable).[21] A correlation of 0 represents no relationship.

In this project, both Pearson and Spearman Correlation Test have been applied. Pearson Test is used to check if there is statistically significant correlation between the feature and target variables. In a Pearson Correlation Test the Null hypothesis is that there is

no statistically significant correlation.[23] With the Spearman Test I have particularly tried to identify the existence of monotonic relationships between variables using the Spearman correlation. "In a monotonic relationship the variables tend to move with each other, not necessarily at a constant rate."[20] In Spearman Test the null hypothesis is that there is no monotonic association between the feature and target variable.[24]

Regression is used to predict the value of a variable through the values of another variable. Regression provides the numerical context on how much the values of the variables are related to each other. In this project, Linear Regression has been used for multiple feature variables. This is commonly known as Multiple Linear Regression. The general formula is,

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_i + \epsilon$$

Where y_i is the target variable to be predicted, β_0 is the y-intercept, $\beta_1$and $\beta_2$are the regression coefficients for the feature variables x_{i1} and x_{i2} respectively, β_p is the slope for each feature variable and finally, ε is the random error term.

Machine Learning Models

In this project, all the models employed are categorised as supervised learning models. Supervised models belong to the realm of machine learning, where the undergo training using sample data, enabling them to discern underlying patterns and make precise predictions for future data. Further, I explain the models in more detail.

FB Prophet

Facebook Prophet is an open-source forecasting tool developed by Facebook's Core Data Science Team. It is designed to handle time series forecasting, making it a valuable asset for analysts, researchers, etc in diverse fields. It is widely known for its ability to produce accurate forecasts with minimal input from users, making it accessible to both novices and experts in the field of time series forecasting.

FB Prophet employs a blend of regression modelling and Bayesian interference to analyse time series data. To delve into the mathematical underpinnings of this algorithms, it is imperative to dissect it into three key components: trend component ($g(t)$), seasonality component ($s(t)$), and holidays ($h(t)$).

- *Trend Component*: Prophet models the trend using piecewise linear function. At each point t, the trend $g(t)$ is defined as the sum of a slope $k(t)$ and an intercept $m(t)$

$$g(t) = (k(t).t) + m(t)$$

- *Seasonality Component*: Seasonality is modelled using a Fourier Series with a specified number of terms (*N*). The seasonal component ($s(t)$) is expressed as a sum of sine and cosine terms.

$$s(t) = \sum_{n-1}^{N} \left(a_n \,.\cos\left(\frac{2\pi nt}{P}\right) + \; b_n \,.\sin\left(\frac{2\pi nt}{P}\right)\right)$$

  where, P is the period of the seasonality and a_n, b_n are the Fourier coefficients.

- *Holiday Effect*: Holidays or special events can also influence the time series. Indicator variables $h_i(t)$ are used to model the effect of holidays, and a holiday component $h(t)$ as defined by,

$$h(t) = \sum_{i} h_i(t)$$

- *Overall Model*: The final additive model combines these components:

$$y(t) = \; g(t) + \; s(t) + \; h(t) + \epsilon_t$$

  where, $y(t)$is the observed value at time *t* and ϵ_t represents the error term.

In summary, Prophet uses a piecewise linear model for the trend and Fourier series terms to capture seasonality. Bayesian methods estimate parameters while considering uncertainty in the data.

Random Forest Regression

Random Fores is a powerful ensemble machine learning technique which combines predictions from multiple decision trees to provide a robust and accurate model and can be used for both classification and regression tasks. In regression, it predicts continuous values based on characteristics of input features. The algorithm constructs multiple decision trees simultaneously during training and outputs the average prediction from these trees.[36][37][38]

The important elements of Random Forest are detailed below:

- *Decision Trees*: Random Forest consists of an ensemble of decision trees (*T*). Each decision tree is constructed through recursive binary splitting of the data, aiming to minimize the mean squared error (MSE).
- *Bootstrapping*: during training, random subsets of the original dataset are samples with replacement. These subsets, called bootstrap samples are used to train individual decision trees.
- *Feature Subset Selection*: At each node of a decision tree, a random subset of features is considered for splitting. This randomness helps to reduce overfitting and decorrelates the trees.

- *Individual Tree Prediction*: Each decision tree makes a prediction for a given input (x) by traversing through the tree from the root to a leaf node. The prediction of a single tree can be denoted as $T_i(x)$
- *Final Prediction*: The final prediction for the Random Forest is the average of the predictions from all individual decision trees in the forest.

$$\hat{Y}(x) = \frac{1}{N}\sum_{i=1}^{N} T_i(x)$$

where, $\hat{Y}(x)$ is the final predicted output for an input x, N is the number of trees in the forest and finally $T_i(x)$ is the individual prediction of the i^{th} .

Random Forest's capabilities especially lie in its ability to handle complex relationships and noisy data while also providing robust regression results.

Extreme Gradient Boosting (XG Boost)

It is another powerful ensemble machine learning techniques utilising gradient boosting functionality which is widely used for both classification and regression tasks. It builds an ensemble of decision trees sequentially rather than simultaneously like Random Forests mentioned earlier. Each tree corrects the errors of the previous tree. [42]

XG Boost starts with an initial prediction $\hat{y}_0$, this is typically the mean of the target variable. It then fits a decision tree to the residuals (or errors) of the previous prediction. The new tree is then scaled by a learning rate (η) and added to the current model to update the predictions: $\hat{y}_1 = \hat{y}_0 + \eta \, . f_1(x)$.[41]

Some key parameters used in the XG Boost model are explained below.

- Number of estimators ('n_estimators'): It defines the number of boosting rounds or decision trees to be built.[41]
- Regularization with 'reg_lambda' and 'gamma': Regularization encourages simpler models, controlling overfitting. It typically includes complexity of the trees and the leaf scores. 'reg_lambda' (L2 regularization term) and 'gamma' (minimum loss reduction required for further partitioning) help control overfitting.[41]
- Maximum depth ('max_depth'): It limits the depth of each decision tree, preventing overly complex trees. For each tree, the depth also controls the number of splits.[41]
- Feature importance

Deep Learning

The Keras Sequential Model is a powerful tool for building deep learning architectures, especially for NLP tasks as in this Project.

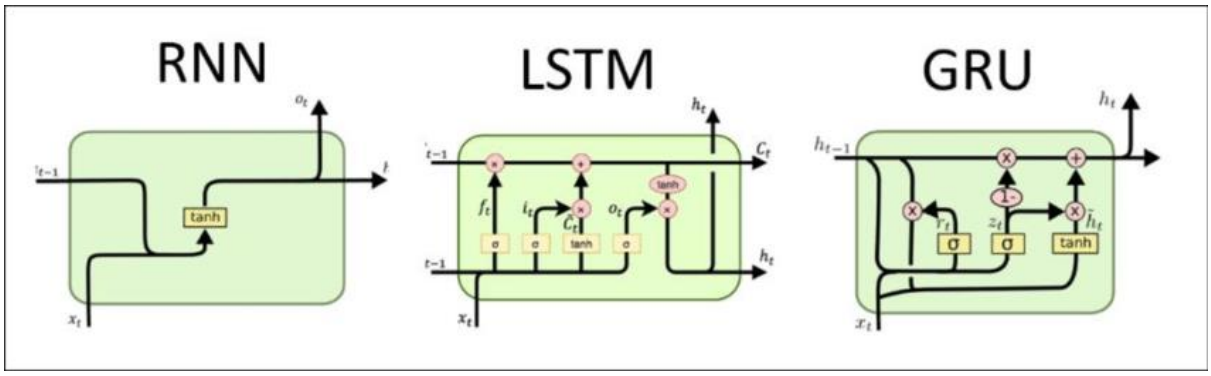


Fig 4.1: Diagram of RNN, LSTM and GRU Networks [50]

Recurrent Neural Networks (RNNs)

Recurrent Neural Networks are a class of deep learning models designed for the sequential data processing. RNNs are equipped with a hidden state that captures information from previous steps and carries it forward to influence the prediction at the current time step.[51]

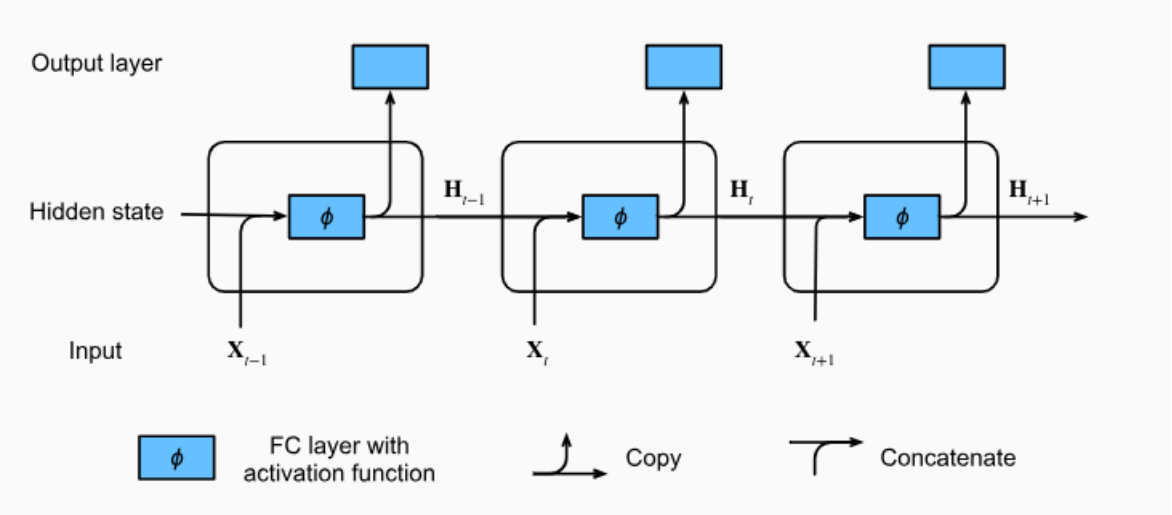


Fig 4.2: Workings of Recurrent Neural Network (RNN) [51]

Mathematically, an RNN computes hidden state h_t at time t as follows:

$$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t + b_h)$$

Where, h_t is the hidden state at time t, x_t is the input, W_{hh} and W_{xh} are weight matrices and b_h is the bias vector.

RNN's do tend to have limitations however, such as vanishing gradient problem and the inability to capture long term dependencies effectively, leading to the development of

more advance models like Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs).

Gated Recurrent Units (GRU)

Gated Recurrent Units are a variant of RNNs designed to address some of the limitations of traditional RNNs. GRUs introduce gating mechanisms that allow network to learn what information to update, store and discard in the hidden state.[52

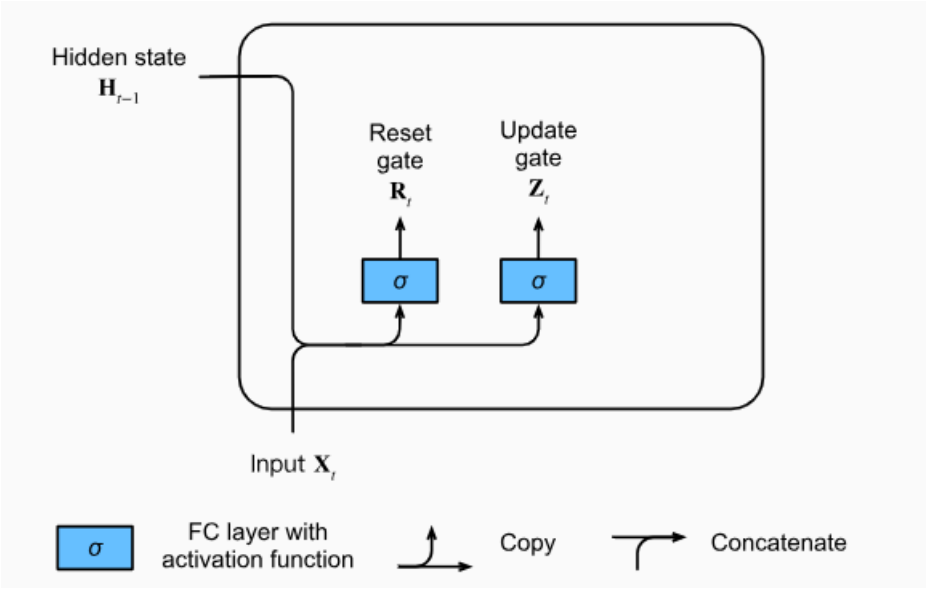


Fig. 4.3: Workings of Gated Recurrent Units (GRU) [52]

Mathematically, a GRU computes the hidden state h_t at time *t* as follows:

$$z_t = \sigma(W_{hz} . h_{t-1} + W_{xz} . x_t + b_z)$$

$$r_t = \sigma(W_{hr} . h_{t-1} + W_{xr} . x_t + b_r)$$

$$\tilde{h}_t = \tanh\big(r_t . (W_{hr} . h_{t-1} + W_{xr} . x_t + b_r)\big)$$

$$h_t = (1 - z_t) . h_{t-1} + z_t . \tilde{h}_t$$

Where, z_t is the update gate, r_t is the reset gate, $\tilde{h}_t$ is the candidate hidden state, and σ is the sigmoid activation function.[52]

GRU's are computationally efficient and can capture dependencies in sequential data more effectively than traditional RNNs.

Embedding Layer

The embedding layer is the initial component of the Sequential Model and is used to transform discrete words or tokens into continuous vector representation. It learns to map words to dense vectors in a lower-dimensional space, where words with similar meaning are closer together.[53]

$$E(x) = word\ embeddings$$

Where, *E(x)* is the embedding representation of input word *x*.

Dense Layer

The Dense Layer is a fully connected layer that follows the GRU layer. It is often used for classification or regression tasks, making predictions based on the features extracted by the previous layers. [54] The Dense layer can be represented as,

$$y = softmax\ (W_y\ .h_t + \ b_y)$$

Where, *y* is the output, *softmax* is the softmax activation function, W_y is the weight matrix, and b_y is the bias vector.

Evaluation Metrics

Evaluation metrics are essential tools for assessing the performance of predictive models. In this project three main metrics have been utilised, mainly Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE).

Root Mean Square Error (RMSE)

Root Mean Square Error (RMSE) is a widely used metric for evaluating the accuracy of prediction models. It quantifies the average deviation of predicted values from actual values when testing the mode. It also determines whether there are any large errors or distances that could be caused if the model overestimated the prediction. The lower the RMSE the better the model's performance.

The formula for RMSE is the square root of the average squared difference between the actual and predicted values.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Where, *n* is the number of data points being tested (or number or rows in the testing dataset), y_i represents the actual values, and $\hat{y}_i$ represents the predicted values.

Mean Absolute Error

Mean Absolute Error (MAE) is another metric for measuring the accuracy of regression models. It calculates the average absolute difference between the predicted and actual values. MAE is less sensitive to outliers as compared to the RMSE. It doesn't penalize the deviations as much as the RMSE.

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Where, *n* is the number of data points, y_i represents the actual values, and $\hat{y}_i$ represents the predicted values.

Mean Absolute Percentage Error (MAPE)

Mean Absolute Percentage Error (MAPE) is a metric that expresses prediction accuracy as a percentage. It measures the average relative error between predicted and actual values. MAPE is particularly useful when dealing with data of varying scales.

The Mathematical Formula for MAPE can be presented as:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{|y_i - \hat{y}_i|}{|y_i|}\right) \times 100\ \%$$

Where, *n* is the number of data points, y_i represents the actual values, and $\hat{y}_i$ represents the predicted values.

ANALYSIS

Exploratory Data Analysis

Initially I imported the numerical data from FRED into the Python file for an initial examination. Ensuring data cleanliness and adhering to a consistent format were essential prerequisite before delving into data analysis. Subsequently, we partitioned the collected data based on the timeline, specifically spanning from January 1, 2012, to June 14, 2023. To enhance data accessibility, column renaming was performed. Additionally, the date column underwent a conversion into a pandas datetime index. Finally, I consolidated all diverse numerical data by merging them into a unified, comprehensive dataframe through an outer join operation on the datetime index.

	key_0	unrate	consum	gdp	frate	xrate	t5yie	t10yie
0	2012-01-01	8.3	227.842	16179.968	0.04	NaN	NaN	NaN
1	2012-02-01	8.3	228.329	NaN	0.11	1.3179	1.89	2.15
2	2012-03-01	8.2	228.807	NaN	0.11	1.3320	1.99	2.26
3	2012-04-01	8.2	229.187	16253.726	0.09	NaN	NaN	NaN
4	2012-05-01	8.2	228.713	NaN	0.16	1.3226	2.06	2.26

Fig 5.1: Initial Merged Dataset of all economic indicators

Upon initial examination of the dataset in Fig 5.1, I observed the presence of null values on select columns. These null values are anticipated to impact our future analysis significantly. Therefore, it is crucial to promptly detect and rectify them. This issue is addresses by employing a two-step approach, initially employing forward fill and then subsequently backward fill methods. Subsequent verification in Fig. 5.2., confirms the successful replacement of NaN values.

key_0	unrate	consum	gdp	frate	xrate	t5yie	t10yie
2012-01-01	8.3	227.842	16179.968	0.04	1.3179	1.89	2.15
2012-02-01	8.3	228.329	16179.968	0.11	1.3179	1.89	2.15
2012-03-01	8.2	228.807	16179.968	0.11	1.3320	1.99	2.26
2012-04-01	8.2	229.187	16253.726	0.09	1.3320	1.99	2.26
2012-05-01	8.2	228.713	16253.726	0.16	1.3226	2.06	2.26

Fig 5.2: Economic indicator dataset after filling null values

Applying descriptive statistics on the data reveals that there are 4183 rows of data in Fig 5.3. However, the range of the data is wide.

	unrate	consum	gdp	frate	xrate	t5yie	t10yie
count	4183.000000	4183.000000	4183.000000	4183.000000	4183.000000	4183.000000	4183.000000
mean	3.660650	302.141451	20312.991712	0.884664	1.174949	1.869460	2.008052
std	0.481748	9.941400	458.807252	1.180648	0.097527	0.495061	0.368558
min	3.400000	227.842000	16179.968000	0.040000	0.961600	0.140000	0.500000
25%	3.600000	303.841000	20386.467000	0.090000	1.104200	1.560000	1.730000
50%	3.600000	303.841000	20386.467000	0.160000	1.143800	1.800000	2.070000
75%	3.600000	303.841000	20386.467000	1.550000	1.239650	2.110000	2.260000
max	14.700000	303.841000	20386.467000	5.080000	1.392700	3.590000	3.020000

Fig. 5.3: Descriptive statistics on economic indicator data

Then, boxplots are utilised to obtain a concise summary on the distribution of the data. Boxplots are a powerful tool and help us visually view the central tendency and spread of the data. They also help identify the existence of potential outliers in the dataset which could skew our results. [25]

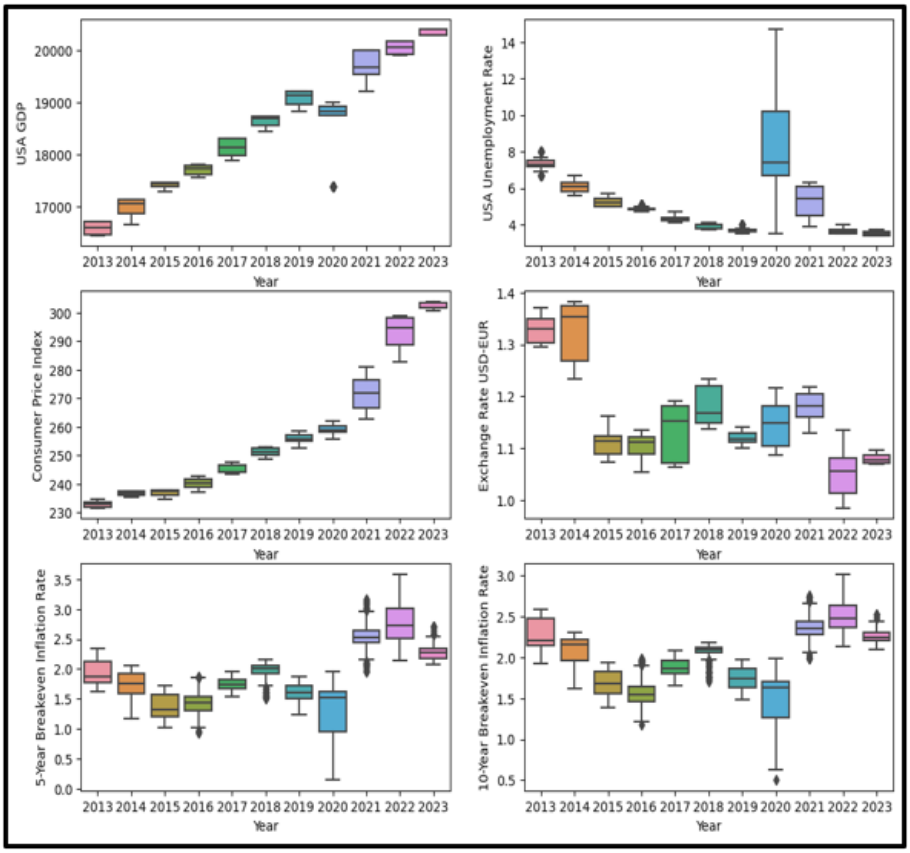


Fig 5.4: Boxplot chart of all economic indicators

The provided box plot chart in Fig. 5.4, illustrated a noteworthy pattern in the behaviour of the US GDP over time. It is evident that the GDP exhibited a consistently expanding range, signifying overall economic growth, with one notable exception in the year 2020. During 2020, the GDP experienced a sharp decline, indicating a significant economic downturn, However, the data also reveals a swift recovery in 2021. This rebound in GDP can be attributed to the implementation of monetary policies formulated by the Federal Open Market Committee (FOMC), which played a pivotal role in stabilizing the economy during the challenging period.

On the other hand, accompanying bar chart depicting the unemployment rate in the USA tells a distinct story. For most of the years within the recorded data span used here, the unemployment rate exhibited a declining trend, indicating a healthier job market. However, a critical anomaly emerged in 2020 when the unemployment rate abruptly surged, reaching its highest level in a decade. This spike in unemployment was likely a direct consequence of the economic disruptions caused by external factors, such as the global pandemic, which resulted in company's being forced to limit the business conducted resulting in a large number of people whose jobs could not be performed remotely, losing their jobs. The chart also offers a glimmer of hope as it shows a slight decrease in the unemployment rate in 2021, followed by a more pronounced dip in the subsequent years. This decline suggests that the labour market began to recover, possibly due to a combination of economic emergency measures, increased vaccination campaign efforts, easing of restrictions, etc.

The box plot representing a consumer price index reveals upward trend over time. Interestingly, there is a sudden surge in the rate of increase starting from 2021. This sudden increase can likely be attributed to a combination of factors. One of the prominent contributing factors is the lingering impact of the global pandemic and the conflict in Ukraine. These events disrupted supply chains, leading to shortages of goods and raw materials, subsequently also driving prices of energy and various other commodities. Some other causes could be the sudden fluctuations in demand or the adjustments to the monetary policy.

In parallel, the data representing exchange rates does not seem to vary heavily over time. However, there is still a discernible trend emerging, showing a gradual decrease in exchange rates. This consistent depreciation in currency exchange rates has the effect of narrowing the gap between the US Dollar and the EURO. The correlation between the declining exchange rates and the trends observed in the GDP box plot are quite notable. It could be suggested that the economic dynamics reflected in the GDP data may be influencing

currency exchange rates. This interconnection underscores the relationship between economic indicators, international events, and the foreign exchange market.

For exploring the relationship between the feature variables and the target variable, I utilized joint plots through the seaborn library. These joint plots are composed of three distinct plots. The first is a bivariate graph illustrating the interaction between the dependent and independent variables. Positioned above the bivariate graph is a plot depicting the distribution of the target variable, while the plot to the right illustrates the distribution of the feature variable.[25] In the Fig 5.5, Some feature variables like Unemployment Rate, Consumer Price Index, GDP, and Exchange rate seem to have a much more clear indication of correlation than others.

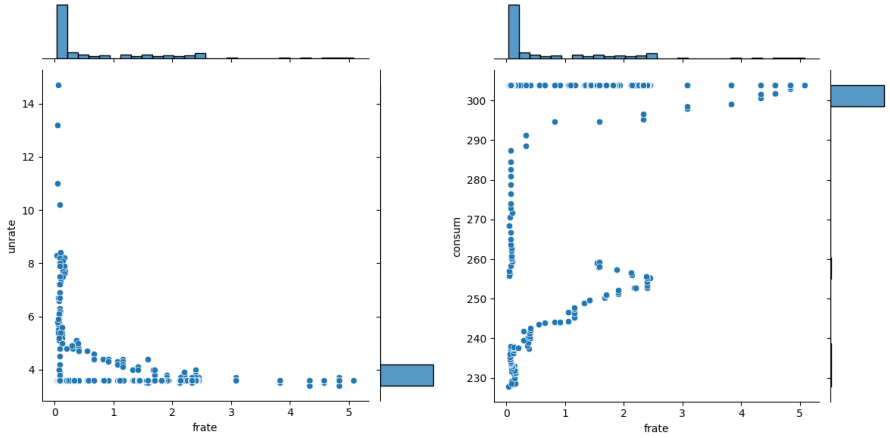


Fig. 5.5: (a) Joint plot between Unemployment Rate in the USA and the Effective Federal Funds Rate, (b) Joint plot between Consumer Price Index in the USA (All States) and the Effective Federal Funds Rate.

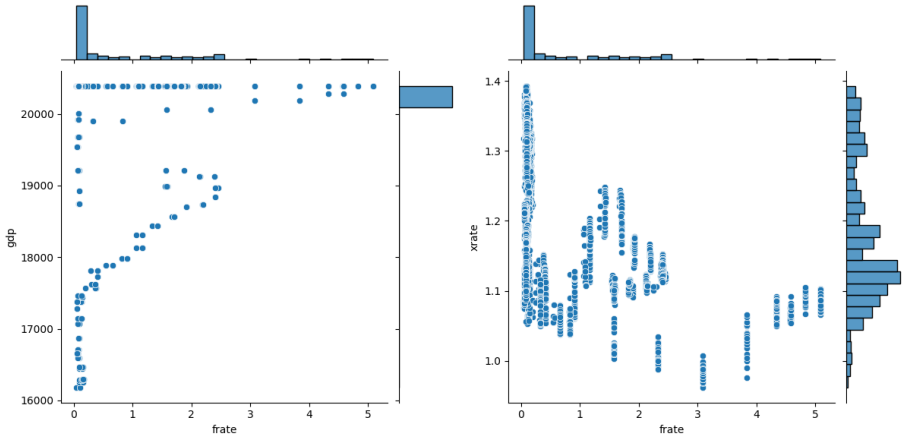


Fig. 5.5: (c) Joint plot between Gross Domestic Product in the USA and the Effective Federal Funds Rate, (d) Joint plot between Exchange Rate of the Euro in US dollars and the Effective Federal Funds Rate.

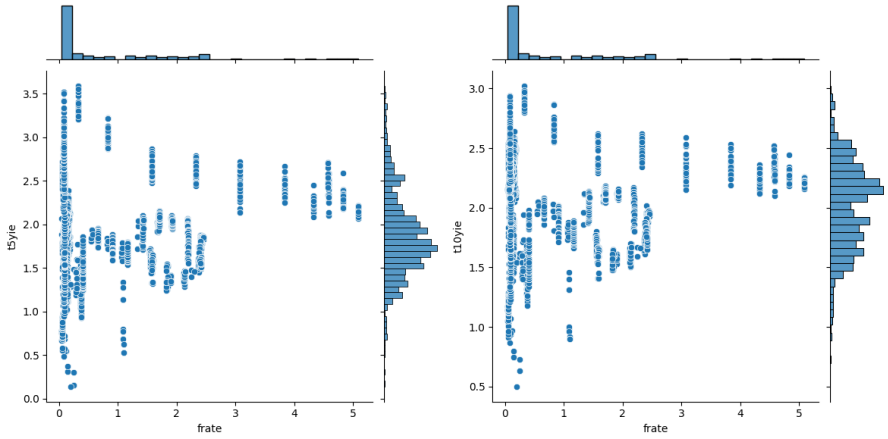


Fig. 5.5: (e) Joint plot between 5-Year Breakeven Inflation Rate in the USA and the Effective Federal Funds Rate, (f) Joint plot between 10-Year Breakeven Inflation rate in the USA and the Effective Federal Funds Rate

As previously detailed in the methodology section, both Pearson and Spearman correlation tests have been employed to assess the association between the feature and target variables. Specifically, the Pearson significance test involves two hypotheses: the Null Hypothesis (H_0) posits that X lacks statistically significant correlation with the target variable, while the Alternate Hypothesis (H_1) suggests that X exhibits a statistically significant correlation with the target variable. In this context, "X" can represent any of the following variables: Unemployment Rate, Consumer Price Index, GDP, Exchange Rate, 5-Year Breakeven Inflation Rate, or 10-Year Breakeven Inflation Rate. The target variable remains the Effective Federal Funds Rate. Upon conducting the Pearson's correlation test, the ensuing results were as follows.

Pearson Correlation Test	
Feature Variable	**p-value**
Unemployment Rate in the USA	7.17e-07
Consumer Price Index (All Items) in all States of USA	0.004211
Gross Domestic Product of the USA	0.0002577
Exchange Rate of the Euro in US Dollars	5.32e-232
5-Year Breakeven Inflation Rate	1.48e-31
10-Year Breakeven Inflation Rate	2.45e-06

Table 5.1: Results of Pearson Correlation Test

Based on the result obtained, it's evident that all p-values are less than the significance level α which is set at 0.05. This signifies that we have ground to reject the Null Hypothesis (H_0). Consequently, it becomes more plausible to assert that there exists a correlation between the feature variables and the target variables.

Subsequently, we examine the outcomes of the Spearman's Correlation test, which assesses the presence of monotonic relationships between the feature variables and the target variables. The Null Hypothesis (H_0) posits that X lacks any monotonic connection with the target variable. Conversely, the Alternate Hypothesis (H_1) suggests that X indeed exhibits a monotonic association with the target variable. The results obtained are as follows:

Spearman Correlation Test		
Feature Variable	**Correlation Coefficient**	**p-value**
Unemployment Rate in the USA	-0.06607	0.00001
Consumer Price Index (All Items) in all States of USA	0.02059	0.18289
Gross Domestic Product of the USA	0.02547	0.09957
Exchange Rate of the Euro in US Dollars	-0.51360	2.19837e-280
5-Year Breakeven Inflation Rate	0.045168	0.0034787
10-Year Breakeven Inflation Rate	-0.11244	3.02086e-13

Table 5.2: Results of Spearman Correlation Test

According to the findings, it appears highly probable that only the Consumer Price Index and the GDP variables exhibit the p-values exceeding the significance threshold. This suggests a likelihood that they do not possess a monotonic association with the Effective Funds Rate. Conversely, the Unemployment Rate, Exchange Rate, 5-year Breakeven Inflation Rate, and 10-Year Breakeven Inflation Rate variables are the ones rejecting the null hypothesis, indicating potential monotonic association with the Effective Funds rate.

Next we obtain the FOMC Meeting Calendar and map the economic indicator data onto the dates of the meetings. We then see how the Effective Federal Funds Rate has fluctuated during the timeperiod of the analysis.

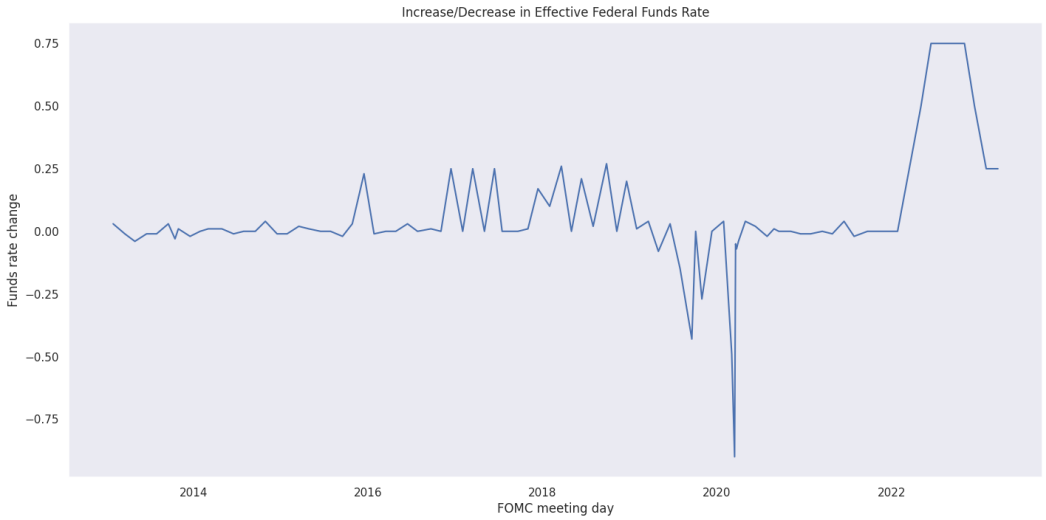


Fig. 5.6: Line Chart of the difference in Effective Federal Funds Rate

This straightforward line graph in Fig. 5.6, illustrates the percentage point deviations by which there has been a difference in the effective funds rate from the rate at the previous meeting. Notable instances of slight spikes are discernible between 2016 to 2019, indicating approximately 0.25 increase in the Effective Funds Rate. Conversely, the most prominent dips are evident during the COVID-19 period, aligning with findings from various articles and research papers.[26][27] An article authored by Ihrig et al., underscores this observation, emphasizing that the monetary policy approach during the COVID-19 period aimed to stabilize the financial market by lowering the rate and maintaining it at that level. This strategy facilitated cash flow during that period and contributed to the nation's post-pandemic recovery.[28]

The economic indicator data is then stored into a binary file using the pickle library in Python. We would then need the pickle library again to read the binary file into a pandas dataframe.

Transitioning to the textual data aspect, we encompass Federal Reserve Statements and Federal Reserve Speeches in our dataset. In the appendix of this report, you can find the web scraping code utilized to collect the Federal Reserve Speeches. Federal Reserve Statements, on the other hand, are typically disseminated on the same day as the FOMC meeting, where monetary policy decisions are made.

In pursuit of acquiring Textual Data, the initial endeavour involved utilization of the FedTools library. FedTools is an open-source Python Library designed for the extraction of FOMC Statements, Beige Books, and Meeting Minutes. Nevertheless, complications arose

concerning the data obtained through this library. Notable inconsistencies within the data became apparent, leading to the development of customer scraping code.

Upon an initial examination of the Statements, employing a highlighting technique reveals limited contextual information regarding the economic conditions or monetary policy inclinations that could contribute to predictive modelling. Merely highlighting common content between two different Statements almost encompasses the entire content, with the exception of numerical adjustments in the federal funds rate target to align with prevailing economic conditions. The image In Fig. 5.7 depicts exactly that.

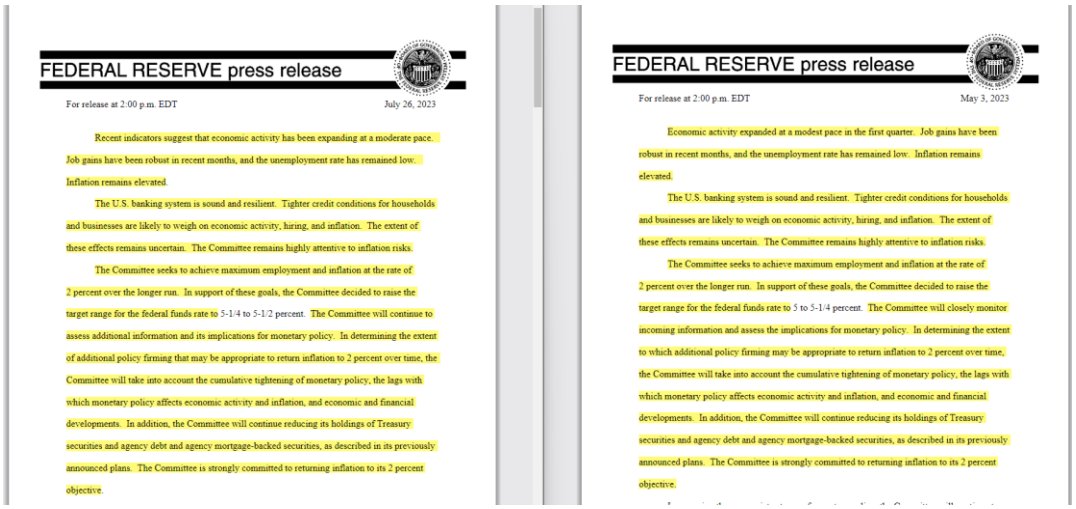


Fig. 5.7: Analysis of text in Federal Reserve Statements

Hence, I opted to proceed by incorporating speeches to provide additional context for the predictive modelling. During the development of the scraping code, a function was established first to generate the URLs. Upon scrutiny, some ULRs were found containing video content with transcripts, which were subsequently excluded. Additionally, non-speech items were identified and removed, discerned by links commencing with '/pubs/feds'. Ultimately, a dictionary comprising dates, speakers and titles was extracted, converted to a pandas dataframe and saved using the pickle library.

Similar to the numerical data, I applied a time-based slicing approach to the speeches, encompassing the period from January 1, 2012, to June 14, 2023. The subsequent data preparation step involved utilizing the date column as the datetime index and arranging the speeches chronologically.

To facilitate text processing, I harnessed the capabilities of established libraries such as NLTK, Gensim, and SpaCy. Employing a loop to iterate through the speech dataframe, I implemented diverse text processing and cleansing techniques on each speech. In each

iteration, the text was transformed into lowercase. Subsequently, I applied the 'en_core_web_sm' SpaCy model to tokenize and lemmatize the text, reducing it to its base form. This process entailed removing all numerical values, stop words, and proper nouns. Finally, the lemmatized tokens from each speech were amalgamated to construct the lemmatized version of the respective speech.

Before proceeding, it is imperative to address the concept of stop words and elucidate why three different libraries with similar functionalities were employed. In the English language, certain words exhibit a higher frequency of occurrence compared to others. These stop words typically encompass terms like "to," "from," "before," and "such," primarily serving as connectors within sentences. Notably, the NLTK library comprises 179 stop words, SpaCy incorporates 326, and Gensim features 337. To maximize the comprehensiveness of stop word elimination, I opted to employ all three libraries.

Subsequently, the lemmatized text underwent a refinement process that involved the removal of escape sequences such as "\r" and "\n." Furthermore, I expanded contracted negations, converting "won't" to "will not" and "can't" to "cannot," among others.

To gain insight into the word count distribution within each speech, I conducted a loop through the words in the original text. Across the 630 speeches included in the analysis, the word count exhibited substantial variation, with an average of approximately 3000 words per speech.

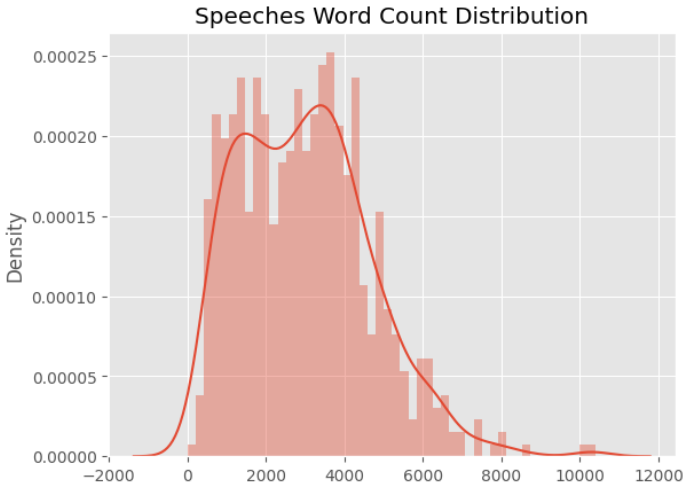


Fig. 5.8: Histogram of Word Count Distribution in Federal Reserve Speeches

Next, using the pickle library, I store the lemmatized text as a binary file to be used as input for applying deep learning techniques to assign sentiments.

Assigning Sentiment

A fundamental stage of the analysis entails attributing sentiment scores to the speeches. Instead of a simple binary classification as positive or negative, the key emphasis lies in assigning a numerical value that quantifies the extent of positivity or negativity. Equally crucial is the accurate capture of tone and meaning. This underscores the significance of employing a robust word embedding technique. Word embedding serves as a textual representation method wherein words with akin meanings share analogous representations, thereby positioning similar words in proximity to one another.

Word embeddings can also be defined as "a methodology in NLP to map words or phrases from vocabulary to a corresponding vector of real numbers which used to find word predictions, word similarities/semantics." [29]

The create a word embedding, trained data is imperative. For this the financial Phrase bank which is labeled with positive, neutral, and negative sentiments for 4840 sentences from financial news was used. First, I encoded the sentiments into a numerical value of 1 for positive, 0 for neutral and -1 for negative. Then I applied Deep Learning Technique [31] with Keras. For this, the phrasebank sentences required preprocessing and cleaning techniques

The Bank Fin Word Embedding refers to a pre-trained word embedding model sourced during my research endeavors.[30] Its particular importance becomes evident in the context of financial documents, owing to its training on an extensive corpus of finance-related books and documents from the banking sector.

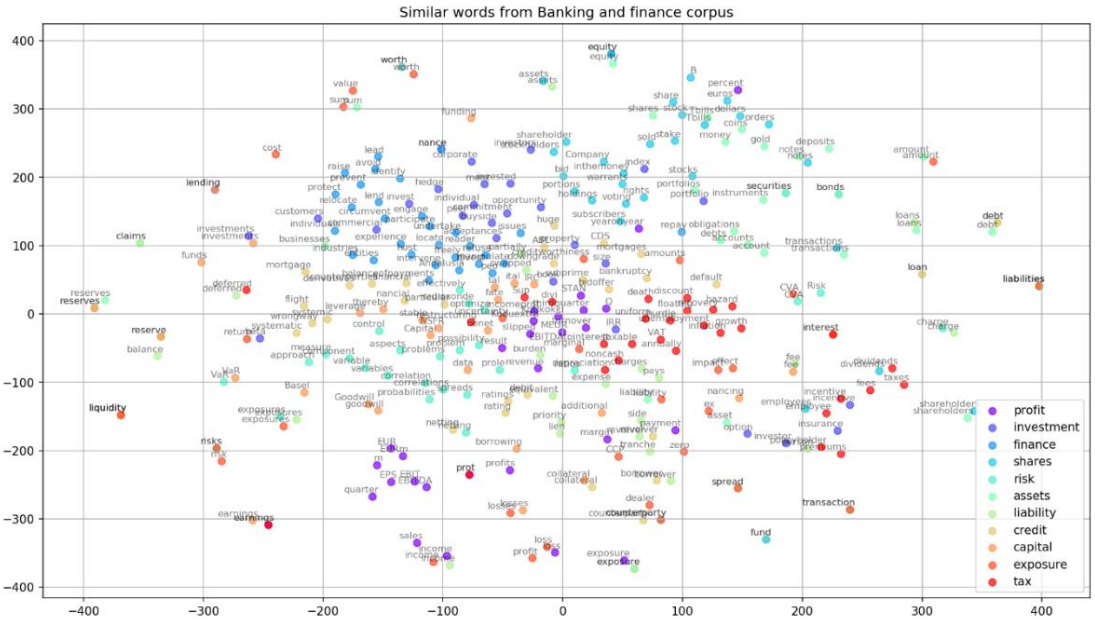


Fig. 5.9: Word vectors positioning in BankFin Word Embedding[30]

To tackle the intricate task of assigning sentiment to the speeches, which constitutes a notably challenging aspect of this project, a method[31] has been adopted. This method leverages deep learning techniques, drawing parallels with a separate project that delves into sentiment analysis on Twitter.[32]

In this project, a deep neural network has been employed. This network ingests a sequence of embedding vectors as input and transforms them into a compressed representation. This compressed form effectively encapsulates all the pertinent information contained within the sequence of words present in the lemmatized speeches.

The specific deep network shown in Fig. 5.10, utilized herein takes the form of Gated Recurrent Unit (GRU). RNNs exhibit a significant limitation when confronted with lengthy text passages, notably their susceptibility to short-term memory constraints. To mitigate this issue, a Gated Recurrent Unit (GRU) network has been integrated, adept at addressing short-term memory limitations. Additionally, an auxiliary layer featuring a sigmoid activation function generates the one-dimensional output. The binary cross-entropy loss function ensures that the assignment of input is limited to two classes.

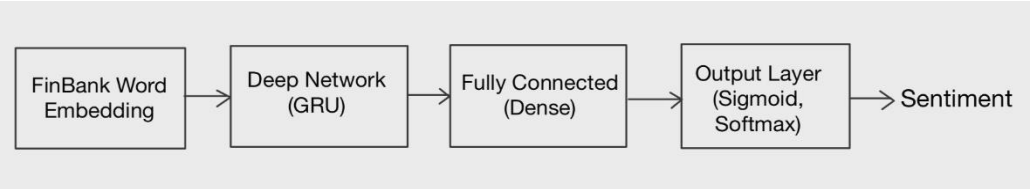


Fig. 5.10: Deep network flow for assigning sentiments to the speeches

Given that we possess a reliable word embedding model that has undergone training on an extensive corpus, the subsequent phase involves the extraction of this word embedding and its transformation into tokenized vectors. To be compatible with the embedding layer, it's imperative that the embedding is encoded into an integer format. The resulting output values fall within the range of 0 to 1, with 0 signifying negativity and 1 denoting positivity. Essentially, the output serves as a prediction of whether the speech leans towards a positive or negative sentiment, with values closer to 0 indicating a stronger negative sentiment, and conversely, values closer to 1 suggesting a more positive sentiment.

Prediction with and without Sentiment

The ultimate phase of the project entails the application of prediction models to the data, both with and without the inclusion of sentiment. The primary distinction between the two scenarios lies in the dimensionality of the independent variables.

A challenge emerges when we possess numerical data available for each day within the specified time period, whereas such consistency cannot be assumed for sentiment data. Consequently, to ensure the preservation of sentiment information, a multi-step approach is employed. Initially, interpolation is performed using the padding method, followed by forward fill, and ultimately, backward fill techniques are applied.

Upon the application of models to both the sentiment-inclusive and sentiment-excluded datasets, the subsequent step involves the assessment of these models using the evaluation metrics explicated in detail within the Methodology Section of this report.

For the regression analysis to be reliable, it results depend on being informed regarding the presence of unit root issues. Unit Root issues deal with the presence of time dependence in the time series data. To ascertain this, the Augmented Dicky Fuller Test is employed to assess stationarity by scrutinizing the null hypothesis, which posits the existence of a unit root i.e., data is not stationary, against the alternative hypothesis of its absence i.e., data is stationary, thereby aiding in the determination of data stationarity.

As is indicated by the test outcomes presented in the table below, it is apparent that the first group of features (comprising of the unemployment rate, consumer price index, GDP, and the sentiment) display stationary attributes. In assessing the null hypothesis, both the test statistic and the p-value are taken into consideration. For these features, the test statistic significantly falls below the critical value. Additionally, the p-value is also below the level of significance α (α = 0.05), signifying the ability to confidently reject the null hypothesis.

ADF Test for all Feature Variables				
No.	**Feature Name**	**ADF Test Statistic**	**p-value**	**Critical Value (1%)**
1.	Unemployment Rate in the USA	-11.36	9.37e-21	-3.43
2.	Consumer Price Index (All Items and All States)	-8.06	1.60e-12	-3.43
3.	GDP	-12.92	3.85e-24	-3.43
4.	Exchange Rate of EUR in USD	-2.21	0.20	-3.43
5.	5-Year Breakeven Inflation Rate	-2.13	0.23	-3.43
6.	10-Year Breakeven Inflation Rate	-2.16	0.22	-3.43
7.	Sentiment	-17.15	6.98e-30	-3.43

Table 5.3: ADF Test Results for all Feature variables

For the second group of variables, encompassing exchange rate, 5-year breakeven inflation rate, and 10-year breakeven inflation rate, the test statistics exhibit value substantially beneath the critical value, while the p-value consistently remains above the predetermined significance level α. Here α is 0.05. To rectify this situation, differencing is employed on the variables of the second group, to render the data stationary. Subsequent to the differencing, the Augmented Dicky Fuller Test is once again conducted exclusively to group 2 variables The Results undergo a significant transformation, with the test statistics for all the variables plummeting well below the critical value. Simultaneously the p-values also indicate that the null hypothesis may be rejected for these variables.

ADF Test after First Differencing				
No.	Feature Name	ADF Test Statistic	p-value	Critical Value (1%)
4.	Exchange Rate of EUR in USD	-15.419	3.04e-28	-3.43
5.	5-Year Breakeven Inflation Rate	-15.87	8.95e-29	-3.43
6.	10-Year Breakeven Inflation Rate	-15.86	9.17e-29	-3.43

Table 5.4: ADF Test after first differencing on feature variables.

Subsequently, a custom function is established to partition the dataset. In the context of timeseries data, it is illogical to employ the conventional random train-test splitting approach. Therefore, the custom function is devised, which calculates the dataset's length and divides it sequentially in a 90:10 ratio of train and test data.

Linear Regression

Model 1: Initially, Linear Regression model 1 is employed on the dataset without sentiment using the 'linear_model' class from the sklearn library. Utilizing the custom function, 3838 instances of training data are obtained along with 425 instances of testing data.

The Linear regression model is then fitted on the training data and subsequently utilised to make predictions on the testing data. The obtained intercept is -1.09, and the corresponding coefficients are:

Feature Variable	Regression Coefficient
Unemployment Rate	-1.58e-01
Consumer Price Index	5.77e-04
GDP	-3.10e-02
Exchange Rate	1.04e+00
5-year Breakeven Inflation Rate	-9.06e-01
10-year Breakeven Inflation Rate	-1.05e-01

Table 5.5: Regression coefficients for Linear Regression model without Sentiment

Next, for model 2, we employ a similar model as Model 1 with the addition of sentiment into the training dataset. The intercept obtained is -0.84 while the coefficients are:

Feature Variable	Regression Coefficient
Unemployment Rate	-1.49e-01
Consumer Price Index	5.84e-04
GDP	-3.14e-02
Exchange Rate	1.12e+00
5-year Breakeven Inflation Rate	-9.32e-01
10-year Breakeven Inflation Rate	-7.32e-02
Sentiment	-6.95e-01

Table 5.6: Regression coefficients for Linear Regression model with Sentiment

Evaluation: Both Model 1 and Model 2 are compared and evaluated on the basis of evaluation metrics like Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). The results are presented below.

Model	MAE	RMSE
Model 1 – Without Sentiment	2.652	3.029
Model 2: With Sentiment	2.658	3.033

Table 5.7: Linear Regression Model Performance Results

As a general guidance, RMSE values falling within the range of 0.2 and 0.5 are indicative of the model relative accurate predictive capability. The outcome displayed indicate that the Model 2 outperforms Model 1 to a slight degree, with this improvement being attributed to the inclusion of sentiment data.

Facebook's Prophet

For providing data to Prophet's model, a new dataset is created stemming from the original dataset. The new dataset consists of just the date columns and the predicted variable (Effective Funds Rate), which are renamed to 'ds' and 'y', respectively. The new dataset is split based on the 90:10 ratio. The feature variables are added into the model as additional regressors. Using the training and predicted data, we can plot the forecast and the components like trend, seasonality, etc.

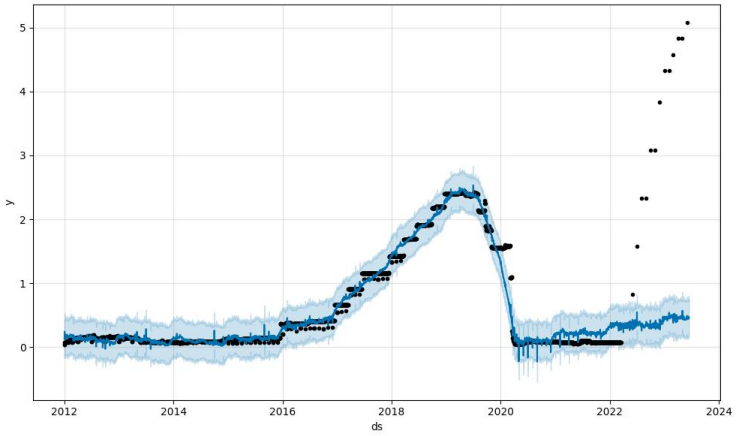


Fig. 5.11: FB Prophet model without sentiment forecast plot

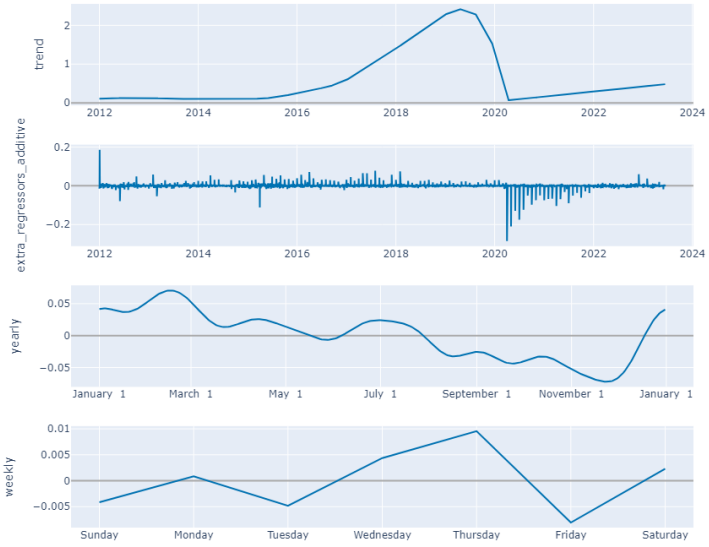


Fig. 5.12: FB Prophet model without sentiment components plot

The forecast plot encompasses a unified graph featuring historical data points, the forecasted curve. Additionally, a light blue shaded region is also incorporated. The date values are indicated as 'ds', while the target values or EFFR is indicated by 'y', pertain to both historical and future dates for which the prediction is done. The black dotted points within the graph correspond to historical trend points, while the blue line encompasses

forecasts for both past and future periods. The presence of the light blue region signifies the extent of uncertainty associated with the forecasts.[43]

On reviewing the graphs, it is evident that there is no discernible trend within the data. Furthermore, the model appears to have adequately incorporated the impacts of the external regressors, as evidenced by the relatively stable lift observed over the duration of time.

In the case of this model, I have implemented analogous modification to those made in Model 1, including the creation of a fresh dataset, renaming of columns, the splitting of data into training and testing sets, and the inclusion of feature variables as additional regressors to the model. However, this model introduces a new regressor, which is Sentiment. Subsequently, forecast and component plots as generated.

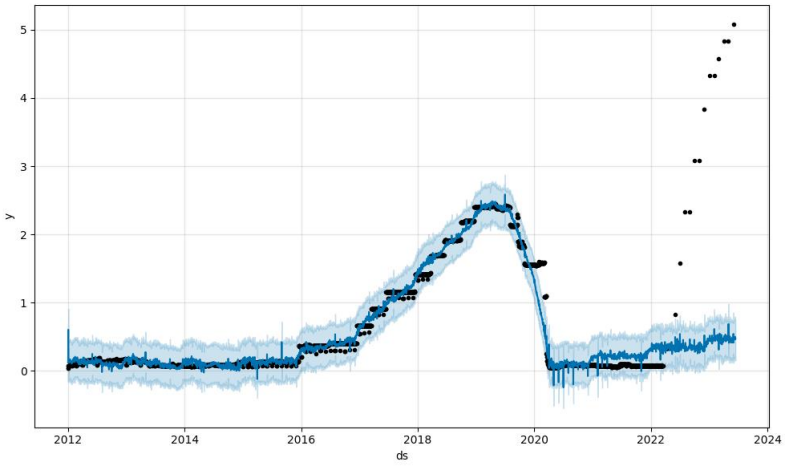


Fig. 5.13: FB Prophet model with sentiment forecast plot

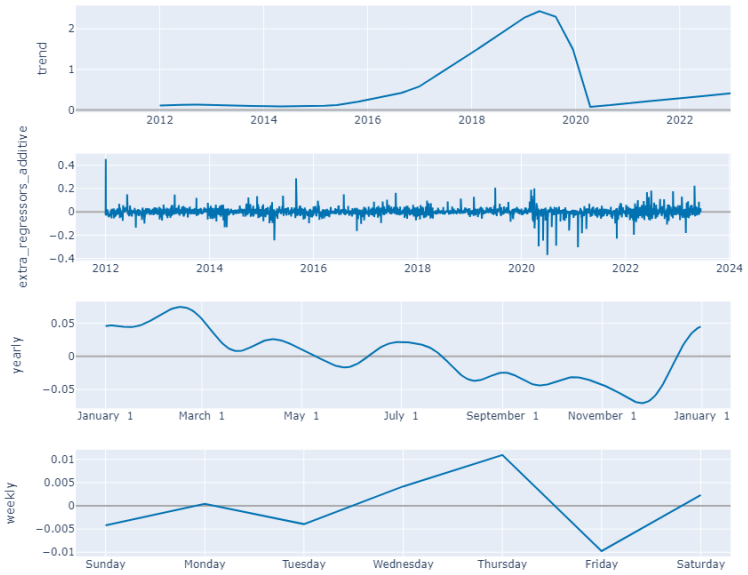


Fig. 5.14: FB Prophet model with sentiment components plot

The forecast and component plots closely resemble those observed in the plots for Model 1. Consequently, the same conclusions can be drawn regarding them.

Evaluation: To evaluate the two models, first comparison plots are generated between the actual and the predicted values. The graphs do not precisely quantify the performance differences. Therefore, to gauge the performances more rigorously, we turn to evaluation metrics such as Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE), yielding the subsequent results.

Model	**MAPE**
Model 1 – without sentiment	81.53%
Model 2 – with sentiment	81.20%

Table 5.8: FB Prophet prediction performance results

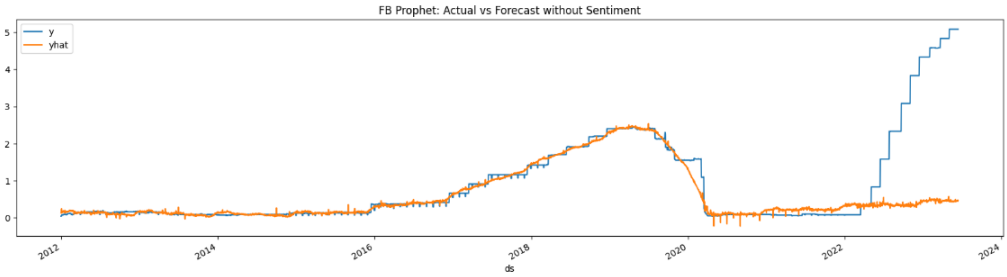


Fig. 5.15: FB Prophet model without Sentiment prediction comparison

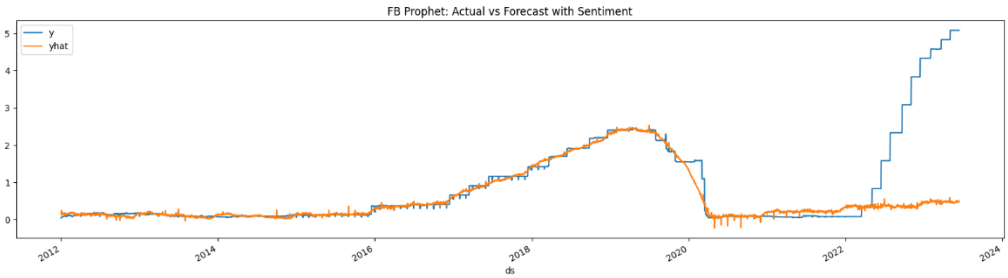


Fig. 5.16: FB Prophet model with Sentiment prediction comparison

Random Forest Regression

The original dataset is split into X and y, which X containing the feature variables and y containing the target variable. For model 1, X will contain all feature variables besides besides. The ‘date’ column was set as the pandas datetime index. X and y were both split into training and testing datasets using the earlier described custom function. The training dataset is fit into the Random Forest Regressor model, defined with 10 decision trees in the forest.

Similarly, operations are conducted on the dataset for model 2 before fitting with the Random Forest Regressor model.

Both models are then evaluated using the Mean Absolute Percentage Error (MAPE) metric. The results clear show that the model with sentiment had better results than the model

without sentiment. This solidifies the study that sentiment can serve as a strong feature in future studies regarding prediction of the effective funds rate.

Model	**MAPE**
Model 1 – without sentiment	80.79 %
Model 2 – with sentiment	80.03 %

Table 5.9: Random forest prediction performance results

Extreme Gradient Boost (XG Boost)

First, a XGB Regressor object is defined with gamma of 0 and max_depth of 3 which restricts the trees to have a maximum of depth of 3 and no minimum loss reduction is required for further partitioning. The parameter n_estimators =100 means 100 decision trees will be built sequentially.

Both the feature variable dataset and the target variable dataset are divided into training and testing subsets, Subsequently, the model is trained through fitting it to the training dataset. This enables us to evaluate the significance of the feature variables in the creation of boosted decision trees. A straightforward bar chart in Fig. 6.17, illustrates that in the Model 1, excluding sentiment, the Unemployment Rate and the Consumer Price Index assume crucial roles in the derivation of the result.

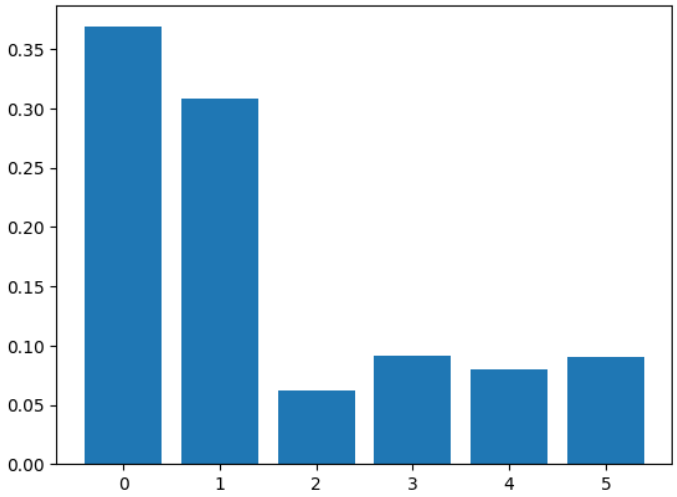


Fig. 5.17: XG Boost Feature Importance assessment for model without sentiment

The Results are then predicted for Model 1.

Subsequently, we proceed with the development of the second model, following a similar approach and employing identical parameters for the XG Boost Regressor Model. However, in this model, sentiments are incorporated, and once again, we evaluate feature importance. A basic bar chart illustrates a significant shift in the outcomes. It becomes

evident that, in addition to the Unemployment Rate and Consumer Price Index, Sentiment assumes a pivotal role in the prediction process.

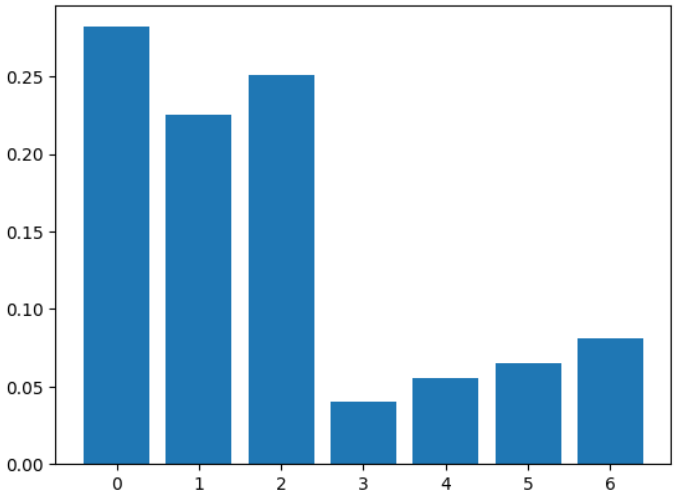


Fig. 5.18: XG Boost Feature Importance assessment for model with sentiment

Evaluation: While assessing the performance of models with and without sentiment, the evaluation relies on Mean Absolute Percentage Error (MAPE). The outcomes consistently indicate that when comparing a model without sentiment to one with sentiment, the model incorporating sentiment consistently demonstrates slightly improved results.

Model	**MAPE**
Model 1 – without sentiment	76.89 %
Model 2 – with sentiment	76.75 %

Table. 5.10: XG Boost model without Sentiment performance results

Evaluation of Models using economic data

In the concluding phase of evaluating all the results derived from diverse models, we have the opportunity to compare these models side by side. This comparative analysis aise in discerning the most suitable model for future studies, allowing for further refinement and the attainment of more precise predictions.

Model	**Evaluation Metric**	**Without Sentiment**	**With Sentiment**
Linear Regression	RMSE	3.029	3.033
	MAE	2.65	2.65
FB Prophet	MAPE	81.53 %	81.20 %
	RMSE	3.22	3.21
Random Forest Regression	MAPE	80.79%	80.03 %

	RMSE	3.1	3.256
XG Boost	MAPE	76.89 %	76.75 %
	RMSE	3.079	3.133

Table 5.11: Evaluation Results of all models with and without sentiment

On careful examination of the results, it becomes evident that XG Boost outperformed both Facebook Prophet and Random forest Regression in terms of Mean Absolute Percentage Error (MAPE). The MAPE scores obtained consistently demonstrated lower error when compared for models both with and without sentiment.

Furthermore, it is discernible from the outcomes that models incorporating sentiment outperformed those that omitted sentiment as a factor.

Analysis with only Sentiment Data

As previously observed with the XG Boost models, the sentiment emerged as a notably vital factor in feature importance assessment, signifying its importance in predictions. Furthermore, throughout the evaluation, we witnessed enhancements in performance among the models that incorporated sentiment within their training datasets. These observations piqued my interest, prompting me to delve into predictive analytics solely dependent on the sentiment data, with the anticipation of uncovering intriguing insights.

Therefore, for this research, I exclusively employed the sentiment scores acquired from the speeches. Then I employed the same predictive algorithms as in previous analyses. To obtain the sentiment dataset, I utilize the pickle library in conjunction with the economic indicators dataset. These two datasets were merged based on the economic indicator dataset. After the merge, only the target variables, which is the effective funds rate was retained, along with the corresponding dates for each data point. The ‘date’ columns was set as the datetime index using pandas library.

A substantial number of missing values were identified in the sentiment data, as not all data points were associated with a speech. To address these null values, I applied interpolation, padding them with existing value, and restricted this padding to a maximum of three consecutive rows. Subsequently, I employed forward fill and backward fill methods to fill the remaining null values effectively.

As mentioned in the previous section, I created a customer function to partition the dataset into training and testing dataset. Additionally, I developed another custom function to separate the dataset into two distinct set: one containing the feature variables and the other containing the target variable.

Following the methodology outlined in the previous section, I initially applied Linear Regression. Upon fitting the model with the training data, I obtained the following intercept and coefficients.

Y-intercept	**Regression coefficient**
1.015	-0.738

Table. 5.12: Linear Regression model with only Sentiment y-intercept and regression coefficient

The predicted values were generated, and a comparative analysis was conducted with the actual values by employing several evaluation metrics, namely Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE).

Model	**RMSE**	**MAE**	**MAPE**
Linear Regression (only sentiment)	3.033	2.659	76.25 %

Fig. 5.13: Linear Regression Model with only Sentiment Performance Results

Then, I applied the FB Prophet Forecasting model to the sentiment data. To make it compatible with FB Prophet's requirements, I created a new dataset in which I renamed the columns 'date' to 'ds' and 'frate' to 'y'. After this transformation, I divided the new dataset into training and testing subsets. For training, I introduced sentiment as a single additional regressor and proceeded to train the model with the training dataset.

Following the model training, a dataframe of future dates was generated and the FB Prophet model was utilized to predict the future values. This forecasting process allowed to generate the Forecast plot, which provides insights into the predicted trends and patterns, as well as the components plot, which visualizes the individual components that contribute to the forecasts such as trend, seasonality, and additional regressors. These plots were important in understanding and interpreting the model's predictions.

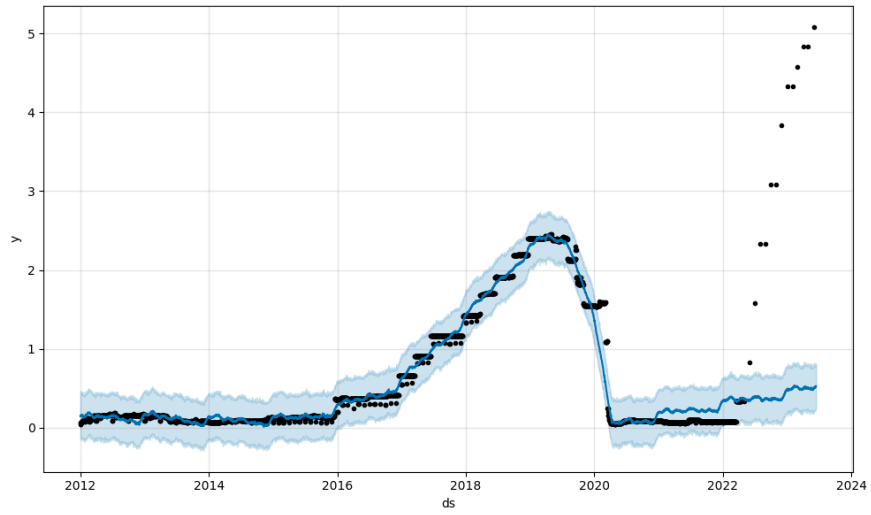


Fig. 5.19: Fb Prophet model with only Sentiment forecast plot

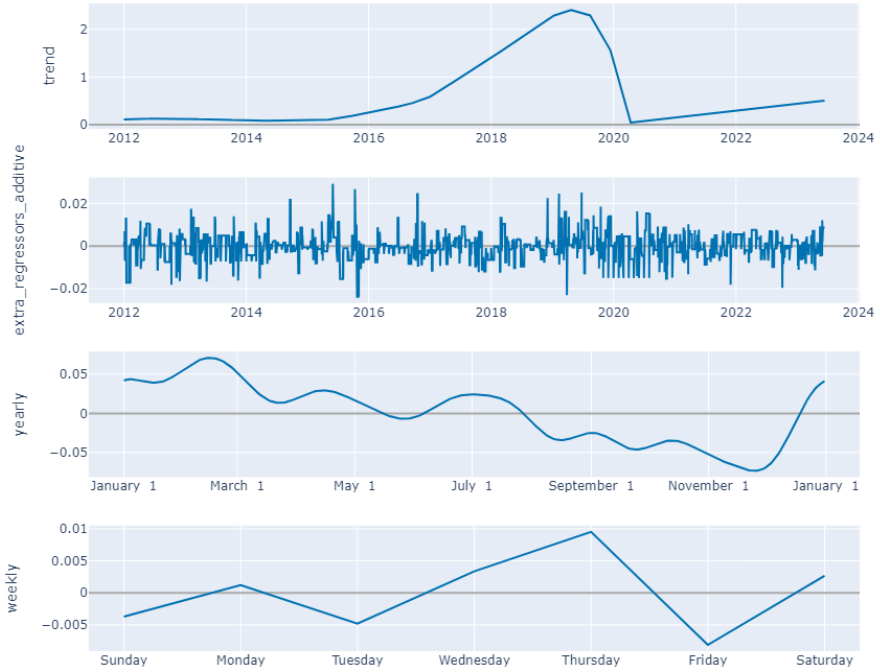


Fig. 5.20: Fb Prophet model with only Sentiment components plot

A comparative plot was generated to visually compare the actual values with the predicted values, revealing an interesting observation. It becomes evident that the predicted values started to deviate from the actual values shortly after the year 2022. This divergence in the two sets of value may be attributed to a distinct interpretation of the trend line by the forecasting model.

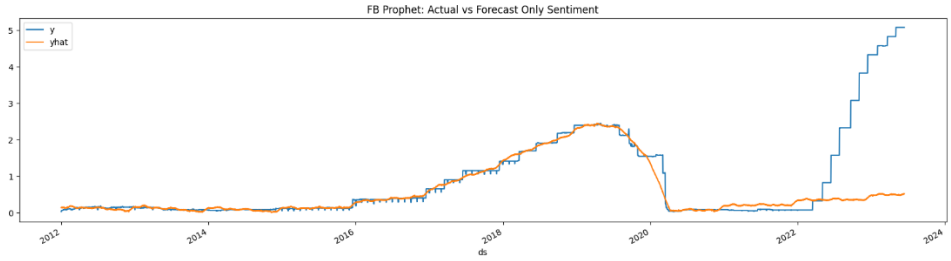


Fig. 5.21: Fb Prophet model with only Sentiment result comparison

The deviation suggests that the model's understanding of the underlying trend shifted or diverged from the actual trend in the dataset after 2022. This could be due to various factors such as changes in data patterns, external events (like the influence of monetary policies made during COVID-19) or shifts in underlying drivers that the model did not adequately capture. Consequently, this discrepancy between the actual and predicted values post-2022 calls for a closer examination and possibly potential adjustments in the modelling approach to improve accuracy and alignment with the true data trends.

To quantify the performance of the FB Prophet model, we use MAPE. The results obtained are as follows:

Model	**MAPE**
Fb Prophet (only sentiment)	80.61 %

Table 5.14: Fb Prophet model with only Sentiment performance results

The third approach involves employing the XG Boost Regressor, configured with specific parameters: 100 estimators, a regularization parameter (reg_lambda) set to 1, a gamma value set to 0, and a maximum depth (max_depth) defined as 3. In this approach the model is again trained on the training dataset, and subsequently, it is used to make predictions on the target values. The quality of these predictions is then assessed by calculating the MAPE.

Model	**MAPE**
XG Boost (only sentiment)	76.55 %

Table 5.15: XG Boost model with only Sentiment performance results

In the final approach, an evaluation is conducted on a Random Forest Regression model which is trained on the same training and testing datasets as previously stated. Following this evaluation, predictions are generated, and the predictive performance is gauged through the calculation of the Mean Absolute Percentage Error (MAPE).

Model	**MAPE**
Random Forest (only sentiment)	79.83 %

Table 6.16: Random Forest model with only Sentiment performance results

The results of this section for all models are in the next section.

RESULTS

While Facebook Prophet and Random Forest Regression are valuable in their own right and applicable in various scenarios, this analysis reveals that XG Boost should be the preferred choice for this specific prediction task. It's ability to minimize prediction errors makes it the optimal choice to further fine-tune and utilize for achieving even more precise forecasts in future studies. This underscores the importance of selecting the right model for the task at hand, as it can greatly influence the quality of prediction and their real-world applications.

The results of the analysis also highlight a significant advantage in including sentiment data within our predictive models. Models that took sentiment into account consistently outperformed those that solely relied on other features. The inclusion of sentiment data added a layer of context and understanding that proved beneficial in capturing nuances and improving the accuracy of the predictions.

Model	**Evaluation Metric**	**Without Sentiment**	**With Sentiment**	**Only Sentiment**
Linear Regression	RMSE	3.029	3.033	3.033
	MAE	2.65	2.65	2.65
FB Prophet	MAPE	81.53 %	81.20 %	80.61 %
	RMSE	3.22	3.21	3.19
Random Forest Regression	MAPE	80.79%	80.03 %	79.83 %
	RMSE	3.1	3.256	3.211
XG Boost	MAPE	76.89 %	76.75 %	76.55 %
	RMSE	3.079	3.133	3.136

Table 6.17: Final Evaluation Results of the Project

References

[1] Hayes, A. (2020). *Federal Reserve System*. [online] Investopedia. Available at: https://www.investopedia.com/terms/f/federalreservebank.asp.

[2] Henricks, M. (2021). *Dovish vs Hawkish: Key Monetary Policy Differences - SmartAsset*. [online] smartasset.com. Available at: https://smartasset.com/blog/financial-advisor/dovish-vs-hawkish/ [Accessed 21 Sep. 2023].

[3] Chen, J. (2022). *Federal Funds Rate*. [online] Investopedia. Available at: https://www.investopedia.com/terms/f/federalfundsrate.asp.

[4] Henricks, M. (2021). *Dovish vs Hawkish: Key Monetary Policy Differences - SmartAsset*. [online] smartasset.com. Available at: https://smartasset.com/blog/financial-advisor/dovish-vs-hawkish/ [Accessed 21 Sep. 2023].

[5] The Investopedia Team and Kelly, R. (2023). *Inflation Hawk: Dovish and Hawkish Monetary Policy Explained*. [online] Investopedia. Available at: https://www.investopedia.com/terms/h/hawk.asp#:~:text=Hawks%20and%20hawkish%20policy%20are [Accessed 21 Sep. 2023].

[6] FRED. (n.d.). *What is FRED? | Getting To Know FRED*. [online] Available at: https://fredhelp.stlouisfed.org/fred/about/about-fred/what-is-fred/.

[7] U.S. Bureau of Economic Analysis (2023). *Gross Domestic Product*. [online] Stlouisfed.org. Available at: https://fred.stlouisfed.org/series/GDP.

[8] FRED (2022). *Unemployment Rate*. [online] Stlouisfed.org. Available at: https://fred.stlouisfed.org/series/UNRATE.

[9] FRED (2019). *Consumer Price Index for All Urban Consumers: All Items*. [online] Stlouisfed.org. Available at: https://fred.stlouisfed.org/series/CPIAUCSL.

[10] Board of Governors of the Federal Reserve System (US) (1999). *U.S. / Euro Foreign Exchange Rate*. [online] FRED, Federal Reserve Bank of St. Louis. Available at: https://fred.stlouisfed.org/series/DEXUSEU.

[11] Federal Reserve Bank of St. Louis (2003). *5-Year Breakeven Inflation Rate*. [online] FRED, Federal Reserve Bank of St. Louis. Available at: https://fred.stlouisfed.org/series/T5YIE.

[12] Federal Reserve Bank of St. Louis (2020). *10-Year Breakeven Inflation Rate*. [online] Stlouisfed.org. Available at: https://fred.stlouisfed.org/series/T10YIE.

[13] Federal Reserve Bank of New York (2000a). *Effective Federal Funds Rate*. [online] FRED, Federal Reserve Bank of St. Louis. Available at: https://fred.stlouisfed.org/series/EFFR.

[14] www.stlouisfed.org. (n.d.). *Understanding an FOMC Statement - The Feducation Video Series*. [online] Available at: https://www.stlouisfed.org/education/feducation-video-series/episode-4-understanding-an-fomc-statement#:~:text=The%20FOMC%2C%20or%20Federal%20Open [Accessed 27 Sep. 2023].

[15] colab.google. (n.d.). *colab.google*. [online] Available at: https://colab.google/.

[16] Yalçın, O.G. (2021). *4 Reasons Why You Should Use Google Colab for Your Next Project*. [online] Medium. Available at: https://towardsdatascience.com/4-reasons-why-you-should-use-google-colab-for-your-next-project-b0c4aaad39ed.

[17] Wu, S. (2021). *Web Scraping Basics*. [online] Medium. Available at: https://towardsdatascience.com/web-scraping-basics-82f8b5acd45c.

[18] Prabhakaran, S. (2019). *Augmented Dickey Fuller Test (ADF Test)*. [online] Machine Learning Plus. Available at: https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/.

[19] Perktold, J., Seabold, S. and Taylor, J. (2023). *Stationarity and detrending (ADF/KPSS) — statsmodels*. [online] www.statsmodels.org. Available at: https://www.statsmodels.org/dev/examples/notebooks/generated/stationarity_detrending_adf_kpss.html.

[20] support.minitab.com. (n.d.). *A comparison of the Pearson and Spearman correlation methods*. [online] Available at: https://support.minitab.com/en-us/minitab/21/help-and-how-

to/statistics/basic-statistics/supporting-topics/correlation-and-covariance/a-comparison-of-the-pearson-and-spearman-correlation-methods/#:~:text=In%20a%20monotonic%20relationship%2C%20the.

[21] ashvika99 (2020). *Difference between Correlation and Regression*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/difference-between-correlation-and-regression/ [Accessed 27 Sep. 2023].

[22] Sereno (2021). *Pearson vs Spearman Correlation | Comparison b/w Spearman & Pearson*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/03/comparison-of-pearson-and-spearman-correlation-coefficients/.

[23] saturncloud.io. (2023). *How to Calculate Correlation between Pandas Columns with Statistical Significance | Saturn Cloud Blog*. [online] Available at: https://saturncloud.io/blog/how-to-calculate-correlation-between-pandas-columns-with-statistical-significance/ [Accessed 28 Sep. 2023].

[24] statistics.laerd.com. (n.d.). *Spearman's Rank-Order Correlation - A guide to how to calculate it and interpret the output.* [online] Available at: https://statistics.laerd.com/statistical-guides/spearmans-rank-order-correlation-statistical-guide-2.php#:~:text=If%20you%20set%20%CE%B1%20%3D%200.05.

[25] www.tutorialspoint.com. (n.d.). *Seaborn.jointplot() method*. [online] Available at: https://www.tutorialspoint.com/seaborn/seaborn_jointplot_method.htm [Accessed 28 Sep. 2023].

[26] Cachanosky, N., Cutsinger, B.P., Hogan, T.L., Luther, W.J. and Salter, A.W. (2021). The Federal Reserve's response to theCOVID-19 contraction: An initial appraisal. *Southern Economic Journal*, 87(4), pp.1152–1174. doi:https://doi.org/10.1002/soej.12498.

[27] Clarida, R.H., Duygan-Bump, B. and Scotti, C. (2021). The COVID-19 Crisis and the Federal Reserve's Policy Response. *www.federalreserve.gov*. [online] Available at: https://www.federalreserve.gov/econres/feds/the-covid-19-crisis-and-the-federal-reserves-policy-response.htm.

[28] Ihrig, J., Weinbach, G. and Wolla, S. (2020). *How the Fed Has Responded to the COVID-19 Pandemic | St. Louis Fed*. [online] www.stlouisfed.org. Available at: https://www.stlouisfed.org/open-vault/2020/august/fed-response-covid19-pandemic.

[29] Prabhu (2019). *Understanding NLP Word Embeddings — Text Vectorization*. [online] Medium. Available at: https://towardsdatascience.com/understanding-nlp-word-embeddings-text-vectorization-1a23744f7223.

[30] Siddhartha, M. (2023). *BankFin Embeddings : Customized word embeddings Pre-Trained on Financial Text corpus for Financial NLP tasks*. [online] GitHub. Available at: https://github.com/sid321axn/bank_fin_embedding [Accessed 28 Sep. 2023].

[31] Nabi, J. (2018). *Machine Learning — Word Embedding & Sentiment Classification using Keras*. [online] Medium. Available at: https://medium.com/towards-data-science/machine-learning-word-embedding-sentiment-classification-using-keras-b83c28087456 [Accessed 28 Sep. 2023].

[32] Sciacovelli, F. (n.d.). *Interest Rate Prediction with Twitter Sentiment*. [online] Available at: https://www.imperial.ac.uk/media/imperial-college/faculty-of-natural-sciences/department-of-mathematics/math-finance/Interest_Rate_Prediction_with_Twitter_Sentiment.pdf [Accessed 28 Sep. 2023].

[33] huggingface.co. (n.d.). *financial_phrasebank · Datasets at Hugging Face*. [online] Available at: https://huggingface.co/datasets/financial_phrasebank.

[34] Prophet. (2019). *Prophet*. [online] Available at: https://facebook.github.io/prophet/.

[35] Khare, P. (2023). *Understanding FB Prophet: A Time Series Forecasting Algorithm*. [online] ILLUMINATION. Available at: https://medium.com/illumination/understanding-fb-prophet-a-time-series-forecasting-algorithm-c998bc52ca10.

[36] IBM (2023). *What is Random Forest? | IBM*. [online] www.ibm.com. Available at: https://www.ibm.com/topics/random-forest.

[37] E R, S. (2021). *Random Forest | Introduction to Random Forest Algorithm*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/.

[38] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), pp.5–32. doi:https://doi.org/10.1023/a:1010933404324.

[39] Yasir, M., Afzal, S., Latif, K., Chaudhary, G.M., Malik, N.Y., Shahzad, F. and Song, O. (2020). An Efficient Deep Learning Based Model to Predict Interest Rate Using Twitter Sentiment. *Sustainability*, 12(4), p.1660. doi:https://doi.org/10.3390/su12041660.

[40] Beheshti, N. (2022). *Random Forest Regression*. [online] Medium. Available at: https://towardsdatascience.com/random-forest-regression-5f605132d19d.

[41] xgboost.readthedocs.io. (n.d.). *XGBoost Parameters — xgboost 1.5.2 documentation*. [online] Available at: https://xgboost.readthedocs.io/en/stable/parameter.html.

[42] Jason Brownlee (2016). *A Gentle Introduction to XGBoost for Applied Machine Learning*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/.

[43] Nihar, P. (2020). *Facebook Prophet For Time Series Forecasting in Python (Part1)*. [online] Medium. Available at: https://towardsdatascience.com/facebook-prophet-for-time-series-forecasting-in-python-part1-d9739cc79b1d#:~:text=Forecast%20Component%20Plot%3A [Accessed 30 Sep. 2023].

[44] Nucleus AI (2023). *Word Embeddings Unveiled: Unlocking the Secrets of Language Representation*. [online] YourStory. Available at: WORD EMBEDDINGS UNVEILED: UNLOCKING THE SECRETS OF LANGUAGE REPRESENTATION Read more at: https://yourstory.com/2023/06/ai-terminology-101-unveiling-power-word-embeddings [Accessed 1 Oct. 2023].

[45] Government, U.S. (n.d.). *Seal of the United States Federal Reserve System.* [online] Wikimedia Commons. Available at: https://commons.wikimedia.org/wiki/File:Seal_of_the_United_States_Federal_Reserve_System.svg [Accessed 1 Oct. 2023].

[46] Word embeddings (2019). *Word embeddings | TensorFlow Core*. [online] TensorFlow. Available at: https://www.tensorflow.org/tutorials/text/word_embeddings.

[47] Arnab (2021). *Word Embeddings*. [online] Analytics Vidhya. Available at: https://medium.com/analytics-vidhya/word-embeddings-aaef8c6bd04a [Accessed 07 Aug. 2023].

[48] Mikolov, T., Chen, K., Corrado, G. and Dean, J. (n.d.). *Distributed Representations of Words and Phrases and their Compositionality*. [online] Available at: https://papers.nips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[49] baeldung (2023). *Dimensionality of Word Embeddings | Baeldung on Computer Science*. [online] www.baeldung.com. Available at: https://www.baeldung.com/cs/dimensionality-word-embeddings [Accessed 1 Oct. 2023].

[50] Toharudin, T., Pontoh, R.S. and Caraka, R.E. (2021). *Employing Long Short-Term Memory and Facebook Prophet Model in Air Temperature Forecasting*. [online] Research Gate. Available at: https://www.researchgate.net/publication/346410173_Employing_Long_Short-Term_Memory_and_Facebook_Prophet_Model_in_Air_Temperature_Forecasting [Accessed 20 Aug. 2023].

[51] d2l.ai. (n.d.). *8.4. Recurrent Neural Networks — Dive into Deep Learning 0.17.0 documentation*. [online] Available at: http://d2l.ai/chapter_recurrent-neural-networks/rnn.html.

[52] d2l.ai. (n.d.). *9.1. Gated Recurrent Units (GRU) — Dive into Deep Learning 0.16.6 documentation*. [online] Available at: https://d2l.ai/chapter_recurrent-modern/gru.html.

[53] Zvornicanin, E. (2023). *What Are Embedding Layers in Neural Networks? | Baeldung on Computer Science*. [online] www.baeldung.com. Available at: https://www.baeldung.com/cs/neural-nets-embedding-layers#:~:text=An%20embedding%20layer%20is%20a [Accessed 1 Oct. 2023].

[54] Haque, N. (2023). *What is Convolutional Neural Network — CNN (Deep Learning)*. [online] www.linkedin.com. Available at: https://www.linkedin.com/pulse/what-convolutional-neural-network-cnn-deep-learning-nafiz-shahriar/ [Accessed 1 Oct. 2023].

[55] NumPy (2022). *Overview — NumPy v1.19 Manual*. [online] numpy.org. Available at: https://numpy.org/doc/stable/.

[56] pandas.pydata.org. (n.d.). *pandas documentation — pandas 1.0.1 documentation*. [online] Available at: https://pandas.pydata.org/docs/.

[57] timhoffm (2021). *matplotlib/matplotlib*. [online] GitHub. Available at: https://github.com/matplotlib/matplotlib.

[58] seaborn (2012). *seaborn: statistical data visualization — seaborn 0.9.0 documentation*. [online] Pydata.org. Available at: https://seaborn.pydata.org/.

[59] Seabold, S. and Perktold, J. (2010). Statsmodels: Econometric and Statistical Modeling with Python. *Proceedings of the 9th Python in Science Conference*, 92-96(9). doi:https://doi.org/10.25080/majora-92bf1922-011.

[60] Tabellout, S. (2023). *Introduction to Machine Learning& Deep Learning & Computer Vision (Part 7)*. [online] Medium. Available at: https://medium.com/@salimtabellout4/introduction-to-machine-learning-deep-learning-computer-vision-part-7-eb7a7b94a291 [Accessed 1 Oct. 2023].

[61] Richardson, L. (2019). *Beautiful Soup Documentation — Beautiful Soup 4.4.0 documentation*. [online] Crummy.com. Available at: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[62] Kozaczko, D. (2018). *8 best Python NLP libraries*. [online] Sunscrapers. Available at: https://sunscrapers.com/blog/8-best-python-natural-language-processing-nlp/.

[63] Galeone, P. (2019). *Hands-On Neural Networks with TensorFlow 2.0*. Packt Publishing Ltd.

APPENDIX

Scraping Functions:

- create_url_list (start_year, end_year, prefix, suffix)
- find_speeches_by_year(host, this_url, print_test=False)
- create_speech_df(host, annual_htm_list)
- retrieve_docs(host, df)

```
def create_url_list(start_year, end_year, prefix, suffix):
    annual_htm_list = []
    for x in range(start_year, end_year+1):
            mid_str = str(x)
            annual_htm_list.append(prefix + mid_str + suffix)
    return annual_htm_list


def find_speeches_by_year(host, this_url, print_test=False):
    conn = HTTPSConnection(host = host)
    conn.request(method='GET', url = this_url)
    resp = conn.getresponse()
    body = resp.read()
    # check that we received the correct response code
    if resp.status != 200:
        print('Error from Web Site! Response code: ', resp.status)
    else:
        soup=BeautifulSoup(body, 'html.parser')
        event_list = soup.find('div', class_='row eventlist')
        # creating the list of dates, titles, speakers and html articles from web p
        date_lst =[]
        title_lst = []
        speaker_lst = []
        link_lst = []

        for row in event_list.find_all('div', class_='row'):
            tmp_date= [x.text for x in row.find_all('time')]
            date_lst.append(tmp_date)

            tmp_speaker = [x.text for x in row.find_all('p', class_='news__speaker'
            speaker_lst.append(tmp_speaker)

            tmp_title = [x.text for x in row.find_all('em')]
            title_lst.append(tmp_title)

        # some of the links include video with the transcript. We are deleteing the
        for link in event_list.find_all('a', href=True, class_ = lambda x: x != 'wa
            link_lst.append(link['href'])

        if print_test:
            print('length of dates: ', len(date_lst))
            print('length of speakers: ', len(speaker_lst))
            print('length of titles: ', len(title_lst))
            print('length of href: ', len(link_lst))

        return date_lst, speaker_lst, title_lst, link_lst


def create_speech_df(host, annual_htm_list):
    all_dates = []
    all_speakers = []
    all_titles = []
    all_links = []
    for item in annual_htm_list:
        date_lst, speaker_lst, title_lst, link_lst = find_speeches_by_year(host,
                                                    item, print_test=False)
        all_dates = all_dates + date_lst
        all_speakers = all_speakers + speaker_lst
        all_titles = all_titles + title_lst
        all_links = all_links + link_lst

    dict1 = {'date': all_dates, 'speaker':all_speakers,
            'title': all_titles, 'link':all_links}
    df = pd.DataFrame.from_dict(dict1)

    #Cleaning up some of the dateframe elements to remove brackets
    df['date']=df['date'].str[0]
    df['date'] = pd.to_datetime(df['date'])
    df['speaker']=df['speaker'].str[0]
    df['title']=df['title'].str[0]

    # creating empty column for documents
```

```
    doc = np.zeros_like(df['date'])
    df['text'] = doc

    # removing items that are not speeches. These contain a link that starts with '/pubs/feds'
    delete_these = df[df['link'].str.match('/pubs/feds')].index
    df = df.drop(delete_these)
    return df



def get_one_doc(host, this_url):
    temp_url = 'https://' + host + this_url
    response = requests.get(temp_url)
    sp = BeautifulSoup(response.text)
    article = sp.find('div', class_='col-xs-12 col-sm-8 col-md-8')

    doc = []
    for p in article.find_all('p'):
        doc.append(p.text)

    return_doc = ''.join(doc)

    return return_doc


def retrieve_docs(host, df):
    for index, row in df.iterrows():
        this_item = df['link'][index]
        print('Scraping text for documents #: ', index)
        doc = get_one_doc(host, this_item)
        df['text'][index] = doc
    return df


if __name__ == '__main__':

    # import functions
    import pandas as pd
    import numpy as np
    from bs4 import BeautifulSoup
    from http.client import HTTPSConnection
    import pickle
    from urllib.request import urlopen
    import requests
    from datetime import datetime, date

    host = 'www.federalreserve.gov'
    prefix = '/newsevents/speech/'
    suffix = '-speeches.htm'
    start_year = 2013
    end_year = 2023

    # create list of web site containing annual speech links
    annual_htm_list =create_url_list(start_year, end_year, prefix, suffix)
    print('Below is the annual_htm_list')
    print(annual_htm_list)

    # create dataframe containing speech information (not yet the text)
    df = create_speech_df(host, annual_htm_list)
    #print(df.info())


    # scrape the text from every speech in the dataframe
    df = retrieve_docs(host, df)
    print(df.info())

    # saving the df to a pickle file
    pickle_out = open('all_fed_speeches', 'wb')
    pickle.dump(df, pickle_out)
    pickle_out.close()
```

```
Below is the annual_htm_list
['/newsevents/speech/2013-speeches.htm', '/newsevents/speech/2014-speeches.htm', '/newsevents/speech/2015-speeches.htm', '/newsev
Scraping text for documents #:  0
<ipython-input-5-1970b0e544c5>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers
  df['text'][index] = doc
Scraping text for documents #:  1
Scraping text for documents #:  2
Scraping text for documents #:  3
Scraping text for documents #:  4
Scraping text for documents #:  5
```

```
Scraping text for documents #:  6
Scraping text for documents #:  7
Scraping text for documents #:  8
Scraping text for documents #:  9
Scraping text for documents #:  10
Scraping text for documents #:  11
Scraping text for documents #:  12
Scraping text for documents #:  13
Scraping text for documents #:  14
Scraping text for documents #:  15
Scraping text for documents #:  16
Scraping text for documents #:  17
Scraping text for documents #:  18
Scraping text for documents #:  19
Scraping text for documents #:  20
Scraping text for documents #:  21
Scraping text for documents #:  22
Scraping text for documents #:  23
Scraping text for documents #:  24
Scraping text for documents #:  25
Scraping text for documents #:  26
Scraping text for documents #:  27
Scraping text for documents #:  28
Scraping text for documents #:  29
Scraping text for documents #:  30
Scraping text for documents #:  31
Scraping text for documents #:  32
Scraping text for documents #:  33
Scraping text for documents #:  34
Scraping text for documents #:  35
Scraping text for documents #:  36
Scraping text for documents #:  37
Scraping text for documents #:  38
Scraping text for documents #:  39
Scraping text for documents #:  40
Scraping text for documents #:  41
Scraping text for documents #:  42
Scraping text for documents #:  43
Scraping text for documents #:  44
Scraping text for documents #:  45
Scraping text for documents #:  46
Scraping text for documents #:  47
Scraping text for documents #:  48
```

```
import tensorflow as tf
tf.test.gpu_device_name()
```

```
''
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import openpyxl

from google.colab import drive
drive.mount('/drive')
```

```
Mounted at /drive
```

```
#Federal Funds Rate Target Range - Upper & Lower Limit
#df_funds = pd.read_excel("/drive/My Drive/Colab Notebooks/data/Fed_funds_target_ra

#Real GDP of USA, Seasonally adjusted , Quarterly
df_gdp = pd.read_excel("/drive/My Drive/Colab Notebooks/data/FRED_RealGDP.xlsx","FR

#Effective Federal funds Rate, Not seasonally adjusted, Daily
df_frate = pd.read_excel("/drive/My Drive/Colab Notebooks/data/FRED_FundsRate.xlsx"

#Unemployment Rate, Percent, Monthly, Seasonally Adjusted
df_unemp = pd.read_excel("/drive/My Drive/Colab Notebooks/data/FRED_UnemploymentRat

#Consumer Price Index for All Urban Consumers: All Items in U.S. City Average, Inde
df_consum = pd.read_excel("/drive/My Drive/Colab Notebooks/data/FRED_Consum_Index.x

#U.S. Dollars to Euro Spot Exchange Rate, U.S. Dollars to One Euro, Monthly, Not Se
df_exuseu = pd.read_excel("/drive/My Drive/Colab Notebooks/data/FOREX_USDEUR.xlsx",

#5-Year Breakeven Inflation Rate, Percent, Daily, Not Seasonally Adjusted
df_t5yie = pd.read_excel("/drive/My Drive/Colab Notebooks/data/T5YIE.xlsx", "FRED G

#10-Year Breakeven Inflation Rate, Percent, Daily, Not Seasonally Adjusted
df_t10yie = pd.read_excel("/drive/My Drive/Colab Notebooks/data/T10YIE.xlsx", "FRED
```

```
from statsmodels.tsa.stattools import adfuller
```

```
from datetime import datetime

start_date = datetime(2012, 1, 1).strftime('%Y-%m-%d')
end_date = datetime(2023,6,14).strftime('%Y-%m-%d')
```

```
df_gdp.rename(columns={"observation_date": "Date", "GDPC1" : "gdp"}, inplace=True)
df_gdp.set_index("Date", inplace=True)
df_gdp.index = pd.to_datetime(df_gdp.index)
df_gdp = df_gdp[(df_gdp.index >= start_date) & (df_gdp.index <= end_date)]
```

```
df_gdp.head()
```

Date	gdp
2012-01-01	16179.968
2012-04-01	16253.726
2012-07-01	16282.151
2012-10-01	16300.035
2013-01-01	16441.485

```
df_frate.rename(columns={"observation_date": "Date", "DFF" : "frate"}, inplace = True)
df_frate.set_index("Date", inplace=True)
df_frate.index = pd.to_datetime(df_frate.index)
df_frate = df_frate[(df_frate.index >= start_date) & (df_frate.index <= end_date)]

df_unemp.rename(columns={"observation_date": "Date", "UNRATE" : "unrate"}, inplace = True)
df_unemp.set_index("Date", inplace=True)
df_unemp.index = pd.to_datetime(df_unemp.index)
df_unemp = df_unemp[(df_unemp.index >= start_date) & (df_unemp.index <= end_date)]
```

```
df_unemp.head()
```

Date	unrate
2012-01-01	8.3
2012-02-01	8.3
2012-03-01	8.2
2012-04-01	8.2
2012-05-01	8.2

```
df_consum.rename(columns={"observation_date": "Date", "CPIAUCSL" : "consum"}, inplace = True)
df_consum.set_index("Date", inplace=True)
df_consum.index = pd.to_datetime(df_consum.index)
df_consum = df_consum[(df_consum.index >= start_date) & (df_consum.index <= end_date)]

df_exuseu.rename(columns={"observation_date": "Date", "DEXUSEU" : "xrate"}, inplace=True)
df_exuseu.set_index("Date", inplace=True)
df_exuseu.index = pd.to_datetime(df_exuseu.index)
df_exuseu = df_exuseu[(df_exuseu.index >= start_date) & (df_exuseu.index <= end_date)]
```

```
df_exuseu.head()
```

Date	xrate
2012-01-02	NaN
2012-01-03	1.3061
2012-01-04	1.2930
2012-01-05	1.2783
2012-01-06	1.2723

```
df_t5yie.rename(columns={"observation_date": "Date", "T5YIE" : "t5yie"}, inplace=True)
df_t5yie.set_index("Date", inplace=True)
df_t5yie.index = pd.to_datetime(df_t5yie.index)
df_t5yie = df_t5yie[(df_t5yie.index >= start_date) & (df_t5yie.index <= end_date)]

df_t10yie.rename(columns={"observation_date": "Date", "T10YIE" : "t10yie"}, inplace=True)
df_t10yie.set_index("Date", inplace=True)
df_t10yie.index = pd.to_datetime(df_t10yie.index)
df_t10yie = df_t10yie[(df_t10yie.index >= start_date) & (df_t10yie.index <= end_date)]
```

```
sf2 = df_unemp.merge(df_consum, how='outer', left_on = df_unemp.index, right_on = df_consum.index)
sf2.set_index("key_0", inplace = True)
sf3 = sf2.merge(df_gdp, how='outer', left_on = sf2.index, right_on = df_gdp.index)
sf3.set_index("key_0", inplace = True)
sf4 = sf3.merge(df_frate, how='outer', left_on = sf3.index, right_on = df_frate.index)
sf4.set_index("key_0", inplace = True)
sf5 = sf4.merge(df_exuseu, how='outer', left_on = sf4.index, right_on = df_exuseu.index)
sf5.set_index("key_0", inplace = True)
sf6 = sf5.merge(df_t5yie, how='outer', left_on = sf5.index, right_on = df_t5yie.index)
sf6.set_index("key_0", inplace = True)
df = sf6.merge(df_t10yie, how='outer', left_on = sf6.index, right_on = df_t10yie.index)
```

```
df.head(5)
```

	key_0	unrate	consum	gdp	frate	xrate	t5yie	t10yie
0	2012-01-01	8.3	227.842	16179.968	0.04	NaN	NaN	NaN
1	2012-02-01	8.3	228.329	NaN	0.11	1.3179	1.89	2.15
2	2012-03-01	8.2	228.807	NaN	0.11	1.3320	1.99	2.26
3	2012-04-01	8.2	229.187	16253.726	0.09	NaN	NaN	NaN
4	2012-05-01	8.2	228.713	NaN	0.16	1.3226	2.06	2.26

```
df.set_index("key_0", inplace = True)
```

```
df.index.rename = "Date"
df.fillna(method = 'ffill', inplace = True)
```

```
df.fillna(method ='backfill', inplace = True)
#


import scipy.stats as stats
import seaborn as sns
df
```

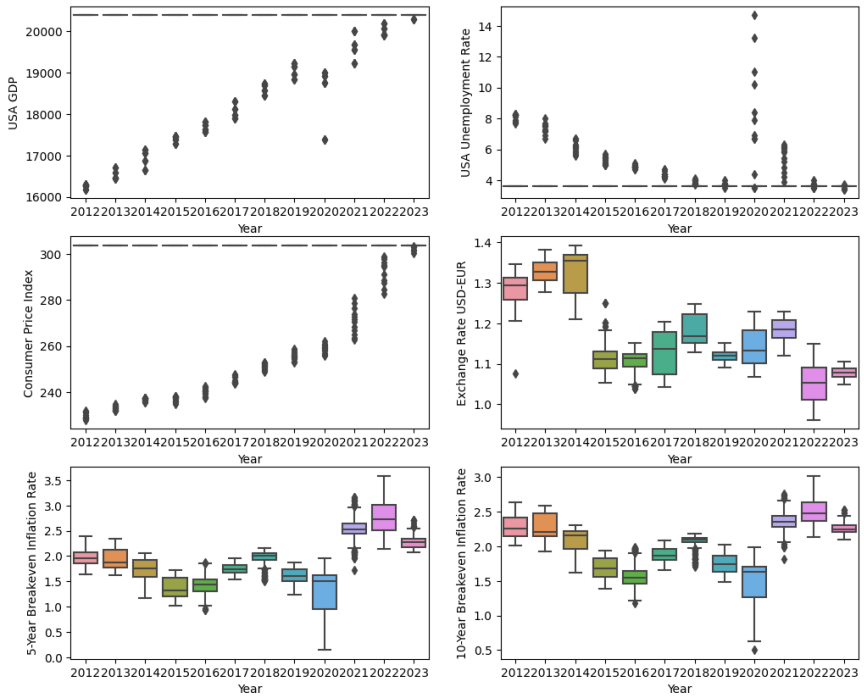
key_0	unrate	consum	gdp	frate	xrate	t5yie	t10yie
2012-01-01	8.3	227.842	16179.968	0.04	1.3179	1.89	2.15
2012-02-01	8.3	228.329	16179.968	0.11	1.3179	1.89	2.15
2012-03-01	8.2	228.807	16179.968	0.11	1.3320	1.99	2.26
2012-04-01	8.2	229.187	16253.726	0.09	1.3320	1.99	2.26
2012-05-01	8.2	228.713	16253.726	0.16	1.3226	2.06	2.26
...	...	...	...	...	...	...	...
2023-06-10	3.6	303.841	20386.467	5.08	1.0749	2.13	2.20
2023-06-11	3.6	303.841	20386.467	5.08	1.0749	2.13	2.20
2023-06-12	3.6	303.841	20386.467	5.08	1.0747	2.09	2.17
2023-06-13	3.6	303.841	20386.467	5.08	1.0792	2.12	2.20
2023-06-14	3.6	303.841	20386.467	5.08	1.0859	2.15	2.21

4183 rows × 7 columns

```
df.describe()
```

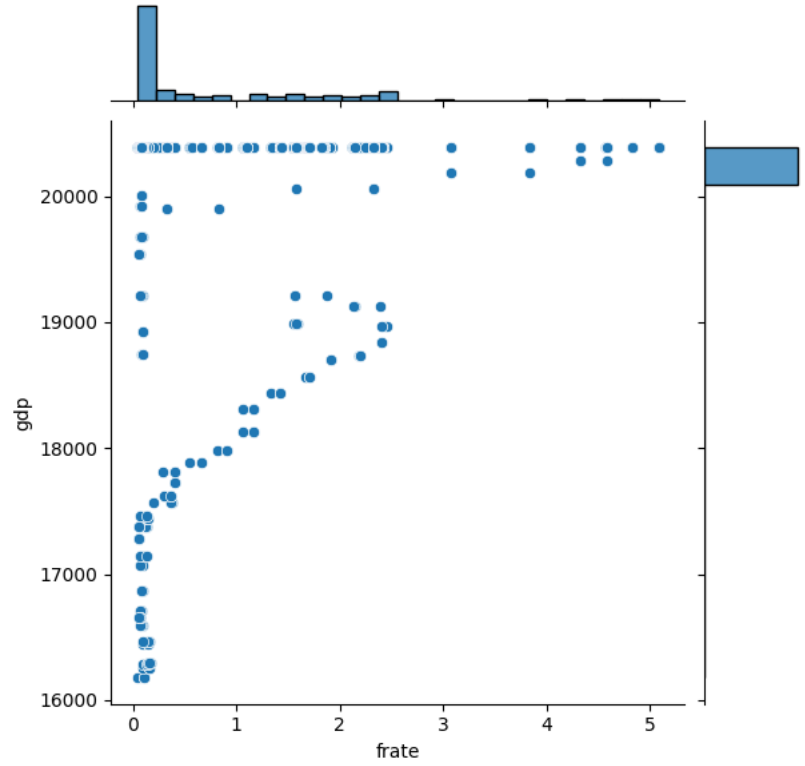
	unrate	consum	gdp	frate	xrate	t5yie	
count	4183.000000	4183.000000	4183.000000	4183.000000	4183.000000	4183.000000	4
mean	3.660650	302.141451	20312.991712	0.884664	1.174949	1.869460	
std	0.481748	9.941400	458.807252	1.180648	0.097527	0.495061	
min	3.400000	227.842000	16179.968000	0.040000	0.961600	0.140000	
25%	3.600000	303.841000	20386.467000	0.090000	1.104200	1.560000	
50%	3.600000	303.841000	20386.467000	0.160000	1.143800	1.800000	
75%	3.600000	303.841000	20386.467000	1.550000	1.239650	2.110000	
max	14.700000	303.841000	20386.467000	5.080000	1.392700	3.590000	

```
plt.figure(figsize = (12, 10))
plt.subplot(3,2,1)
sns.boxplot(x = df.index.year ,y = 'gdp', data = df)
plt.xlabel("Year")
plt.ylabel("USA GDP")
plt.subplot(3,2,2)
sns.boxplot(x = df.index.year , y = 'unrate', data = df)
plt.xlabel("Year")
plt.ylabel("USA Unemployment Rate")
plt.subplot(3,2,3)
sns.boxplot(x = df.index.year ,y = 'consum', data = df)
plt.xlabel("Year")
plt.ylabel("Consumer Price Index")
plt.subplot(3,2,4)
sns.boxplot(x = df.index.year ,y = 'xrate',data = df)
plt.xlabel("Year")
plt.ylabel("Exchange Rate USD-EUR")
plt.subplot(3,2,5)
sns.boxplot(x = df.index.year ,y = 't5yie',data = df)
plt.xlabel("Year")
plt.ylabel("5-Year Breakeven Inflation Rate")
plt.subplot(3,2,6)
sns.boxplot(x = df.index.year ,y = 't10yie',data = df)
plt.xlabel("Year")
plt.ylabel("10-Year Breakeven Inflation Rate")

plt.show()
```



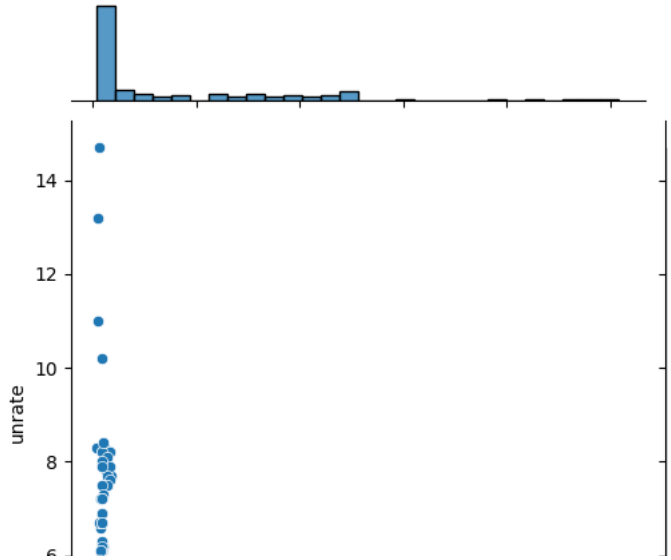

```
plt.figure(figsize = (12, 10))

sns.jointplot(x = 'frate' ,y = 'gdp', data = df)
print(adfuller(df['gdp']))
```

```
(-13.669670567587335, 1.4678523584798227e-25, 29, 4153, {'1%': -3.431925571033693, '5
<Figure size 1200x1000 with 0 Axes>
```



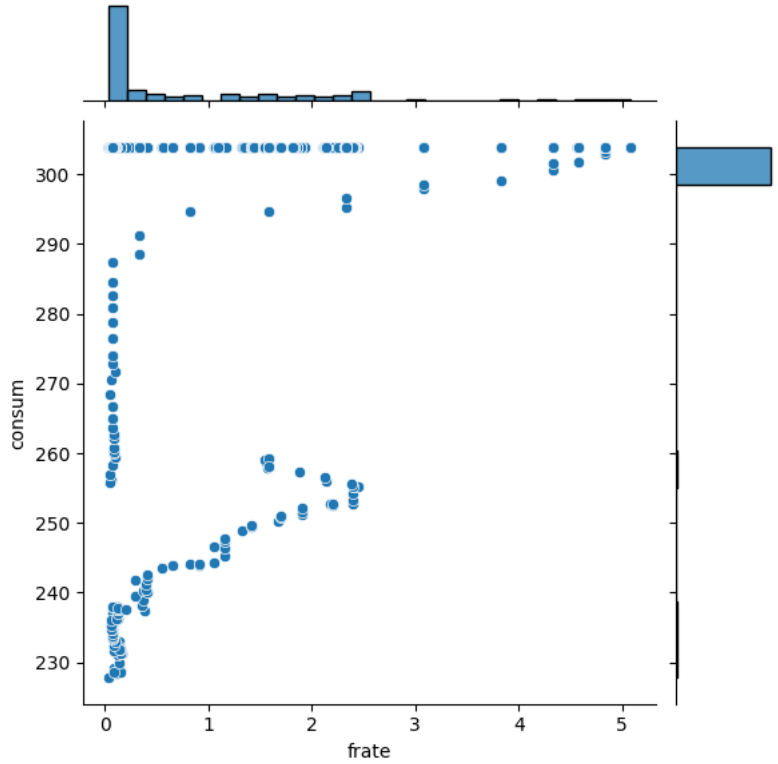

```
sns.jointplot(x = 'frate' , y = 'unrate', data = df)
print(adfuller(df['unrate']))
```

```
(-12.011147843957582, 3.178741406658346e-22, 6, 4176, {'1%': -3.4319168879767545, '5%
```



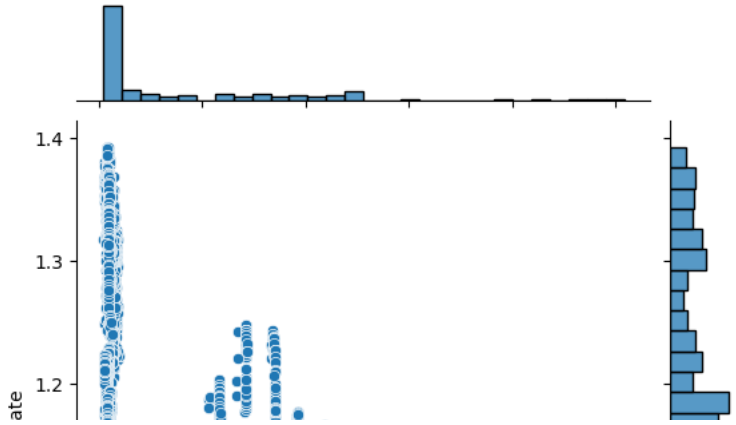

```
sns.jointplot(x = 'frate' ,y = 'consum', data = df)
print(adfuller(df['consum']))
```

```
(-8.523531105692847, 1.0844679042432375e-13, 31, 4151, {'1%': -3.4319263306323293, '5
```



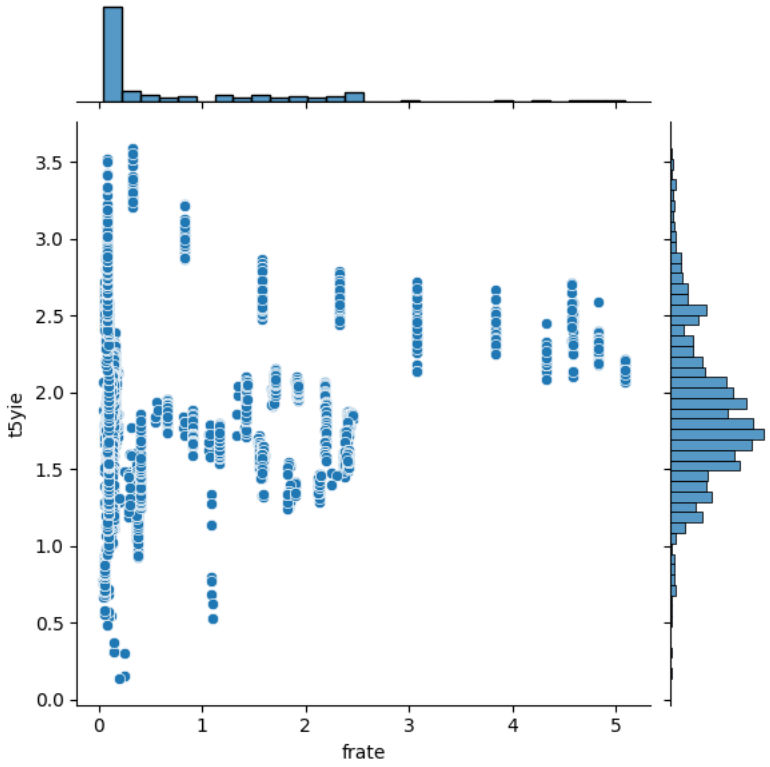

```
sns.jointplot(x = 'frate' ,y = 'xrate',data = df)
print(adfuller(df['xrate']))
```

```
(-2.0638378770113825, 0.2593018861546297, 31, 4151, {'1%': -3.4319263306323293, '5%':
```



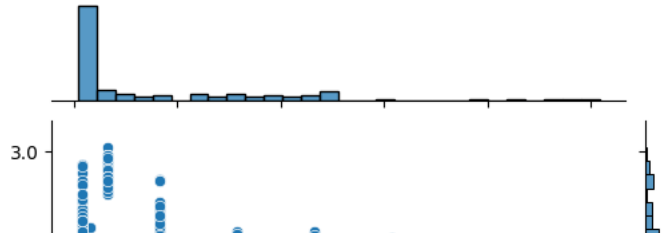

```
sns.jointplot(x = 'frate' ,y = 't5yie',data = df)
print(adfuller(df['t5yie']))
```

```
(-2.0693034311366496, 0.2570355934016994, 31, 4151, {'1%': -3.4319263306323293, '5%':
```




```
sns.jointplot(x = 'frate' ,y = 't10yie',data = df)
print(adfuller(df['t10yie']))
```

```
(-2.155840850182344, 0.2226610557961714, 29, 4153, {'1%': -3.431925571033693, '5%': -
```




```
import pandas as pd
from scipy.stats import pearsonr
```



```
df.head()
```

key_0	unrate	consum	gdp	frate	xrate	t5yie	t10yie
2012-01-01	8.3	227.842	16179.968	0.04	1.3179	1.89	2.15
2012-02-01	8.3	228.329	16179.968	0.11	1.3179	1.89	2.15
2012-03-01	8.2	228.807	16179.968	0.11	1.3320	1.99	2.26
2012-04-01	8.2	229.187	16253.726	0.09	1.3320	1.99	2.26
2012-05-01	8.2	228.713	16253.726	0.16	1.3226	2.06	2.26


```
# Convert dataframe into series
list1 = df['unrate']
list2 = df['frate']

# Apply the pearsonr()
pearsonr(list1, list2)
```

```
PearsonRResult(statistic=-0.0765500977631912, pvalue=7.167936241625508e-07)
```

```
# Convert dataframe into series
list1 = df['consum']
list2 = df['frate']

# Apply the pearsonr()
pearsonr(list1, list2)
```

```
PearsonRResult(statistic=0.044241273642899585, pvalue=0.004211100143942558)
```

```
# Convert dataframe into series
list1 = df['gdp']
list2 = df['frate']

# Apply the pearsonr()
pearsonr(list1, list2)
```

```
PearsonRResult(statistic=0.056475859885841445, pvalue=0.0002577194351199245)
```

```
# Convert dataframe into series
list1 = df['xrate']
list2 = df['frate']

# Apply the pearsonr()
pearsonr(list1, list2)
```

```
PearsonRResult(statistic=-0.47278591891232397, pvalue=5.320565896318147e-232)
```

```
# Convert dataframe into series
list1 = df['t5yie']
list2 = df['frate']

# Apply the pearsonr()
pearsonr(list1, list2)
```

```
PearsonRResult(statistic=0.17392102873096602, pvalue=9.147951431228779e-30)
```

```
# Convert dataframe into series
list1 = df['t10yie']
list2 = df['frate']
```

```
# Apply the pearsonr()
pearsonr(list1, list2)
```

```
PearsonRResult(statistic=0.07278194968716363, pvalue=2.451806006014657e-06)
```

```
list1
```

```
array([2.15, 2.15, 2.26, ..., 2.17, 2.2 , 2.21])
```

T-Test

H0 = none are related H1 = Unemployment rate is related to to frate H2 = Consum H3 = GDP H4 = Xrate H5 = T5yie H6 = T10yie

```
from scipy.stats import ttest_ind
list1 = df['unrate'].values
list2 = df['frate'].values

ttest_ind(list1, list2, equal_var = False)
```

```
TtestResult(statistic=140.79897931674367, pvalue=0.0, df=5536.995707918151)
```

```
from scipy.stats import ttest_ind
list1 = df['consum']
list2 = df['frate']

ttest_ind(list1, list2, equal_var = False)
```

```
TtestResult(statistic=1946.2201805590068, pvalue=0.0, df=4299.943355015108)
```

```
from scipy.stats import ttest_ind
list1 = df['gdp']
list2 = df['frate']

ttest_ind(list1, list2, equal_var = False)
```

```
TtestResult(statistic=2863.30226909957, pvalue=0.0, df=4182.055385281915)
```

```
from scipy.stats import ttest_ind
list1 = df['xrate']
list2 = df['frate']

ttest_ind(list1, list2, equal_var = False)
```

```
TtestResult(statistic=15.847868514202489, pvalue=5.365812555122699e-55, df=4239.068826625521)
```

```
from scipy.stats import ttest_ind
list1 = df['t5yie']
list2 = df['frate']

ttest_ind(list1, list2, equal_var = False)
```

```
TtestResult(statistic=49.75062612033136, pvalue=0.0, df=5608.488029406814)
```

```
from scipy.stats import ttest_ind
list1 = df['xrate']
list2 = df['frate']

ttest_ind(list1, list2, equal_var = False)
```

```
X = df[['unrate', 'consum','gdp','xrate','t5yie','t10yie']]
y = df['frate']
```

```
from sklearn.linear_model import LinearRegression

mlr = LinearRegression()
mlr.fit(X, y)
```

```
LinearRegression
LinearRegression()
```

```
print("Intercept: ", mlr.intercept_)
print("Coefficients:")
list(zip(X, mlr.coef_))
```

```
Intercept:  7.924839983578808
Coefficients:
[('unrate', -0.10289914875014532),
 ('consum', -0.008476757392826141),
 ('gdp', 0.0001552392741163873),
 ('xrate', -8.306879369549405),
 ('t5yie', -1.2245693141902056),
 ('t10yie', 2.3872633666616947)]
```

Results from Joint Plot

- Gross Domestc Product - Somewhat Positive correlation
- Unemployment Rate - Negative Correlation
- Consumer Price Index - Positive Correlation

Feature Engineering

- intended to reduce the number of input variables to those that are believed to be most useful to a model in order to predict the target variable.

```
pip install statsmodels
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (0.14.0)
Requirement already satisfied: numpy>=1.18 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (1.23.5)
Requirement already satisfied: scipy!=1.9.2,>=1.4 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (1.11.2)
Requirement already satisfied: pandas>=1.0 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (1.5.3)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (0.5.3)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (23.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0->statsmodels) (2
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0->statsmodels) (2023.3.post
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.2->statsmodels) (1.16.0)
```

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

model1 = smf.glm(formula = "frate ~ gdp + unrate + consum + xrate + t5yie + t10yie", data = df, family = sm.families.Tweedie())
model2 = smf.glm(formula = "frate ~ unrate + consum +xrate +t5yie +t10yie", data = df, family = sm.families.Tweedie())
result1 = model1.fit()
result2 = model2.fit()

print(result1.summary())

predictions = result1.predict()
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                  frate   No. Observations:                 4183
Model:                            GLM   Df Residuals:                     4176
Model Family:                Tweedie   Df Model:                            6
Link Function:                   Log   Scale:                          0.89622
Method:                         IRLS   Log-Likelihood:                -5177.5
Date:               Fri, 15 Sep 2023   Deviance:                       3492.2
Time:                       13:17:50   Pearson chi2:                 3.74e+03
No. Iterations:                    9   Pseudo R-squ. (CS):             0.3976
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      9.6449      4.634      2.082      0.037       0.563      18.727
gdp            0.0005      0.000      1.471      0.141      -0.000       0.001
unrate        -0.7674      0.219     -3.501      0.000      -1.197      -0.338
consum        -0.0310      0.012     -2.594      0.009      -0.054      -0.008
xrate         -8.6657      0.226    -38.262      0.000      -9.110      -8.222
t5yie         -0.6198      0.119     -5.190      0.000      -0.854      -0.386
t10yie         1.2550      0.169      7.424      0.000       0.924       1.586
==============================================================================
```

```
predictions
```

```
array([0.00696192, 0.00685765, 0.00696694, ..., 1.74691715, 1.71242886,
       1.60609955])
```

p values :

- gdp : 0.978 (97.8% chance that gdp has no effect on frate)

- unrate: 0.00 (0% chance that unrate has no effect on frate)
- consum: 0.021(2.1% chance that consum has no effect on frate)
- xrate: 0.446 (44.6% chance that xrate has no effect on frate)
- t5yie: 0.00
- t10yie: 0.00

- P>|z| — one of the most important statistics in the GLM summary. It uses the z statistics to produce the p-value, a measurement of how likely your coefficient point estimate is measured through the GLM model by chance. The p value of 0% for RiskLevel is saying that there is 0 chance, that RiskLevel has no effect on the dependent Default variable, therefore the results are not produced by chance at all. The p value of e.g. 0.256 for RiskLevel would be saying there is a 25.6% chance the RiskLevel variable has no affect on the dependent variable, Default, and the results are produced by chance. The P>|z| tells, if our point estimate has been calculated “well” to distinguish it from zero. We define “well” using the p<.05, because a common alpha is 0.05. P>|z| will compare the p value to a previously established alpha value, or a z threshold with which you apply significance to your coefficient.
- Revise the AIC and the BIC. AIC (Akaike information criterion) has lower value, if we fit the model better. The same as BIC or MDL, AIC provides the basic scoring to choose the best model. BIC (Bayesian information criterion) has lower value, better the model fit. Calculation is similar to MDL. There is usually high correlation between AIC and BIC. MDL (minimum description length) is another criterion to score a model, lower the value, better the model fit.

```
# Calculate the Akaike criterion
print(result1.aic)
# Calculate the Bayesian information criterion
print(result1.bic)

# Calculate the Akaike criterion
print(result2.aic)
# Calculate the Bayesian information criterion
print(result2.bic)
```

```
10369.091967516522
-31330.571847310428
10025.677478777792
-31337.134176799664
/usr/local/lib/python3.10/dist-packages/statsmodels/genmod/generalized_linear_model.py:1838: FutureWarning: The bic value is comput
  warnings.warn(
```

Kendall - Tau test to check if data is Monotonic

```
# Import required libraries
from scipy.stats import kendalltau

# Calculating Kendall Rank correlation
corr, _ = kendalltau(df['t10yie'], df['frate'])
print('Kendall Rank correlation: %.5f' % corr)
```

```
Kendall Rank correlation: -0.06257
```

Kendall Rank Correlation:

- gdp : 0.40045
- unrate : -0.65527
- consum : 0.32253
- xrate : -0.35413
- t5yie : 0.05601
- t10yie : -0.04086

Spearman Rank Correlation test, and is used to detect the existence of monotonic relationship between variables.

```
from scipy.stats import spearmanr

# calculate Spearman's correlation coefficient and p-value
corr, pval = spearmanr(df['t10yie'], df['frate'])

# print the result
print("Spearman's correlation coefficient:", corr)
print("p-value:", pval)
```

```
Spearman's correlation coefficient: -0.11244485523643455
p-value: 3.020867974154857e-13
```

Results

- gdp: corr coef = 0.42695, p-val = 5.82e-169 < 0.05
- unrate: corr coef = 0.798, p-val = 0.0 < 0.05
- consum: corr coef = 0.318158, p-val = 1.5744e-90 < 0.05
- xrate: corr coef = -0.50616, p-val = 1.600e-247 < 0.05
- t5yie: corr coef = 0.062435, p-val = 0.00011344 < 0.05
- t10yie: corr coef = -0.081261, p-val = 4.97838e-07 < 0.05

All are statistically significant????

```
df.reset_index(inplace = True)
```

```
df.rename(columns = {"key_0":"date"}, inplace = True)
df.set_index('date', inplace = True)
```

```
df
```

date	unrate	consum	gdp	frate	xrate	t5yie	t10yie
2012-01-01	8.3	227.842	16179.968	0.04	1.3179	1.89	2.15
2012-02-01	8.3	228.329	16179.968	0.11	1.3179	1.89	2.15
2012-03-01	8.2	228.807	16179.968	0.11	1.3320	1.99	2.26
2012-04-01	8.2	229.187	16253.726	0.09	1.3320	1.99	2.26
2012-05-01	8.2	228.713	16253.726	0.16	1.3226	2.06	2.26
...	...	...	...	...	...	...	...
2023-06-10	3.6	303.841	20386.467	5.08	1.0749	2.13	2.20
2023-06-11	3.6	303.841	20386.467	5.08	1.0749	2.13	2.20
2023-06-12	3.6	303.841	20386.467	5.08	1.0747	2.09	2.17
2023-06-13	3.6	303.841	20386.467	5.08	1.0792	2.12	2.20
2023-06-14	3.6	303.841	20386.467	5.08	1.0859	2.15	2.21

4183 rows × 7 columns

```
filter = df.iloc[df.shape[0] -1,0].strftime('%Y-%m-%d')
print(filter)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-32-94e79960e241> in <cell line: 1>()
----> 1 filter = df.iloc[df.shape[0] -1,0].strftime('%Y-%m-%d')
      2 print(filter)

AttributeError: 'numpy.float64' object has no attribute 'strftime'
```

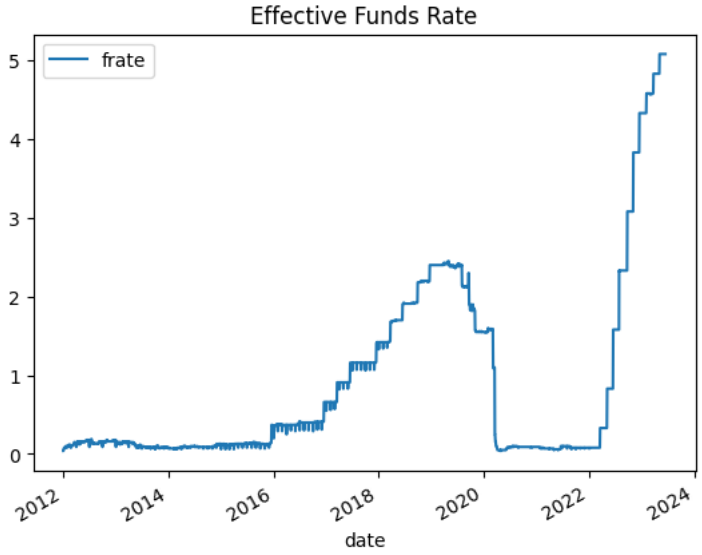
SEARCH STACK OVERFLOW

```
frate_df = df[['frate']]
frate_df.head()
```

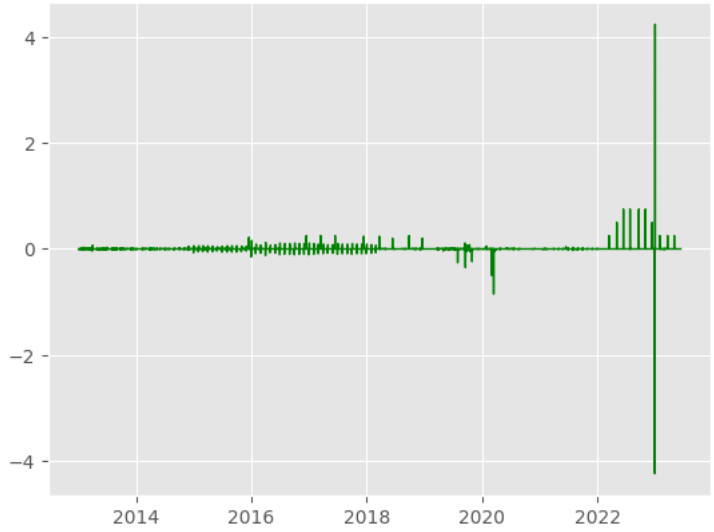
```
                frate
```

```
plt.figure(figsize = (12, 10))
frate_df.plot(label = "Funds Rate")
plt.legend(loc='upper left')
plt.title("Effective Funds Rate")
plt.show()
```

```
<Figure size 1200x1000 with 0 Axes>
```




```
fig, ax = plt.subplots()
ax.plot(final_df['date'], final_df['diff'].values,
         c = 'green',
         linewidth= 1.0)
ax.grid(True)
plt.show()
```



Random Forest

```
l = 0.2 *
```

```
from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n_estimators = 100, random_state = 1)
predictors = ["unrate", "xrate","t5yie","t10yie"]

regressor.fit(train[predictors], train['diff'])
```

```
RandomForestRegressor
RandomForestRegressor(random_state=1)
```

```
from sklearn.metrics import mean_squared_error

y_pred = regressor.predict(test[predictors])
rmse = float(format(np.sqrt(mean_squared_error(test['diff'], y_pred)), '.3f'))
print("\nRMSE: ", rmse)
```

```
RMSE:  0.004
```

```
pip install xgboost
```

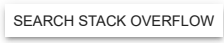
```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (1.7.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.11.2)
```

```
import xgboost as xgb

model = xgb.XGBRegressor(objective = 'reg:squarederror')
model.fit(train[predictors], train['diff'])
print(); print(model)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

```
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_log_error

y_pred = model.predict(test[predictors])

print(metrics.r2_score(test['frate'], y_pred))
#print(metrics.mean_squared_log_error(test['diff'], y_pred))

plt.figure(figsize = (10,10))
sns.regplot(test['diff'], y_pred, fit_reg= True, scatter_kws = {"s":100})
```

```
0.0
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-50-bde3e5e9807f> in <cell line: 12>()
     10
     11 plt.figure(figsize = (10,10))
---> 12 sns.regplot(test['diff'], y_pred, fit_reg= True, scatter_kws = {"s":100})

TypeError: regplot() takes from 0 to 1 positional arguments but 2 positional
arguments (and 2 keyword-only arguments) were given

SEARCH STACK OVERFLOW
```



```
<Figure size 1000x1000 with 0 Axes>
```

```
df
```

date	unrate	frate	xrate	t5yie	t10yie
2012-01-01	8.3	0.04	1.3179	1.89	2.15
2012-02-01	8.3	0.11	1.3179	1.89	2.15

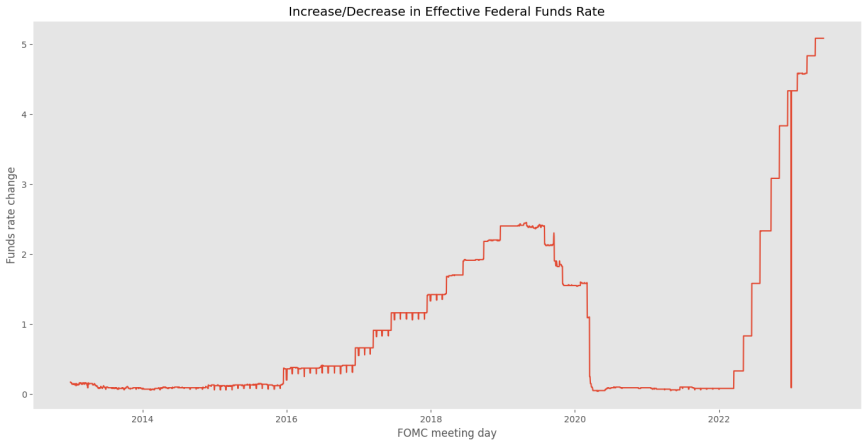
```
import pickle
import os
pickle_out = open('/drive/My Drive/Colab Notebooks/econ_df','wb')
pickle.dump(df, pickle_out)
pickle_out.close()
```

```
df
```

date	unrate	frate	xrate	t5yie	t10yie
2012-01-01	8.3	0.04	1.3179	1.89	2.15
2012-02-01	8.3	0.11	1.3179	1.89	2.15
2012-03-01	8.2	0.11	1.3320	1.99	2.26
2012-04-01	8.2	0.09	1.3320	1.99	2.26
2012-05-01	8.2	0.16	1.3226	2.06	2.26
...	...	...	...	...	...
2023-06-10	3.6	5.08	1.0749	2.13	2.20
2023-06-11	3.6	5.08	1.0749	2.13	2.20
2023-06-12	3.6	5.08	1.0747	2.09	2.17
2023-06-13	3.6	5.08	1.0792	2.12	2.20
2023-06-14	3.6	5.08	1.0859	2.15	2.21

4183 rows × 5 columns

```
plt.figure(figsize=(17, 8))
plt.plot(final_df['date'],final_df['frate'])
plt.title('Increase/Decrease in Effective Federal Funds Rate')
plt.ylabel('Funds rate change')
plt.xlabel('FOMC meeting day')
plt.grid(False)
plt.show()
```




```
final_df
```

	date	Lower_limit	Upper_limit	Avg_limit	unrate	consum	gdp	frate	x
0	2013-01-01	0.0	0.25	0.125	8.0	231.679	16441.485	0.17	1.
1	2013-01-02	0.0	0.25	0.125	8.0	231.679	16441.485	0.17	1.
2	2013-01-03	0.0	0.25	0.125	8.0	231.679	16441.485	0.17	1.
3	2013-01-04	0.0	0.25	0.125	8.0	231.679	16441.485	0.16	1.
4	2013-01-05	0.0	0.25	0.125	8.0	231.679	16441.485	0.16	1.
...	...	...	...	...	...	...	...	...	
3812	2023-06-10	5.0	5.25	5.125	3.6	303.841	20404.088	5.08	1.
3813	2023-06-11	5.0	5.25	5.125	3.6	303.841	20404.088	5.08	1.
3814	2023-06-12	5.0	5.25	5.125	3.6	303.841	20404.088	5.08	1.
3815	2023-	5.0	5.25	5.125	3.6	303.841	20404.088	5.08	1
3816	06-14	5.0	5.25	5.125	3.6	303.841	20404.088	5.08	1.

3817 rows × 13 columns

In this Notebook

- Preprocessing of Speech Data - cleaning it
- Join the Speeches made before an fomc meeting. Append all of the text together.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import nltk
import spacy
import os
import pickle
import datetime
from datetime import datetime
```

```
import datetime
from datetime import datetime

start_date = datetime(2013,1,1).strftime('%Y-%m-%d')
#df_speeches = df_speeches[df_speeches['date'] >= start_date]
```

```
from google.colab import drive
drive.mount('/drive')
```

```
Drive already mounted at /drive; to attempt to forcibly remount, call drive.mount("/drive", force_remount=True).
```

Speech Data - Get and Clean

```
def clean_speech(df_speeches):
    df_speeches["speaker"]= df_speeches["speaker"].str.lower()
    to_remove = ["chairman","governor", "vice", "director", "chair", "division", "of", "monetary" , "pro", "tempore", "affairs", "for", "

    df_speeches["speaker"] = [' '.join([item for item in x.split()
                  if item not in to_remove])
                  for x in df_speeches["speaker"]]

    #f = [datetime.strptime(d, "%m/%d/%Y").strftime("%d/%m/%Y") for d in df_speeches["date"]]

    df_speeches.set_index("date", inplace = True)
    df_speeches = df_speeches.sort_index(ascending = True, inplace = True)
```

```
f = open('/drive/My Drive/Colab Notebooks/all_fed_speeches', 'rb')
df_speeches = pickle.load(f)
clean_speech(df_speeches)
```

```
df_speeches
```

	speaker	title	link	text
date				
2012-01-06	elizabeth a. duke	Economic Developments, Risks to the Outlook, a...	/newsevents/speech/duke20120106a.htm	It is the start of a new year, the traditional...
2012-01-06	sarah bloom raskin	Community Bank Examination and Enforcement	/newsevents/speech/raskin20120106a.htm	\r\n Thank you for the opportunity

Financial Phrasebank get and clean

```
phrasebank = pd.read_csv("/drive/My Drive/Colab Notebooks/data/financial_phrasebank.csv")
```

```
phrasebank
```

	Sentence	Sentiment
0	The GeoSolutions technology will leverage Bene...	positive
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative
2	For the last quarter of 2010 , Componenta 's n...	positive
3	According to the Finnish-Russian Chamber of Co...	neutral
4	The Swedish buyout firm has sold its remaining...	neutral
...	...	...
5837	RISING costs have forced packaging producer Hu...	negative
5838	Nordic Walking was first used as a summer trai...	neutral
5839	According shipping company Viking Line , the E...	neutral
5840	In the building and home improvement trade , s...	neutral
5841	HELSINKI AFX - KCI Konecranes said it has won ...	positive

5842 rows × 2 columns

```
phrasebank["sent_code"] = int()
for i,r in phrasebank.iterrows():
    if phrasebank["Sentiment"][i] == "positive":
        phrasebank["sent_code"][i] = 1
    elif phrasebank["Sentiment"][i] == "negative":
        phrasebank["sent_code"][i] = -1
    else:
        phrasebank["sent_code"][i] = 0
```

```
<ipython-input-44-f4aa9787e90e>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  phrasebank["sent_code"][i] = 1
<ipython-input-44-f4aa9787e90e>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  phrasebank["sent_code"][i] = -1
<ipython-input-44-f4aa9787e90e>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  phrasebank["sent_code"][i] = 0
```

Get FOMC Calendar

```
fomc_calendar = pd.read_csv("/drive/My Drive/Colab Notebooks/data/fomc_calendar.csv")

print(fomc_calendar.shape)
fomc_calendar.head()
```

```
(104, 4)
```

	date	unscheduled	forecast	confcall
0	30/01/2013	False	False	False
1	20/03/2013	False	False	False
2	01/05/2013	False	False	False

```
fomc_calendar['date'] = pd.to_datetime(fomc_calendar['date'] ,format = '%d/%m/%Y')
```

4	31/07/2013	False	False	False

```
fomc_calendar
```

	date	unscheduled	forecast	confcall
0	2013-01-30	False	False	False
1	2013-03-20	False	False	False
2	2013-05-01	False	False	False
3	2013-06-19	False	False	False
4	2013-07-31	False	False	False
...	...	...	...	...
99	2024-06-12	False	True	False
100	2024-07-31	False	False	False
101	2024-09-18	False	True	False
102	2024-11-07	False	False	False
103	2024-12-18	False	True	False

104 rows × 4 columns

```
# split into future and past meeting dates#
current_date = datetime.today().strftime('%Y-%m-%d')

future_calendar = fomc_calendar[(fomc_calendar['date'] >= current_date)]
past_calendar = fomc_calendar[(fomc_calendar['date'] < current_date)]
```

```
import openpyxl

#Federal Funds Rate Target Range - Upper & Lower Limit
df_funds = pd.read_excel("/drive/My Drive/Colab Notebooks/data/Fed_funds_target_range.xlsx")
#Effective Federal funds Rate, Not seasonally adjusted, Daily
df_frate = pd.read_excel("/drive/My Drive/Colab Notebooks/data/FRED_FundsRate.xlsx")


df_funds.rename(columns = {"Date":"date","Lower_limit":"l_limit","Upper_limit":"u_limit"}, inplace = True)
df_funds = df_funds[df_funds["date"] <= current_date]
df_frate.rename(columns = {"observation_date":"date","EFFR":"frate"})
```

	date	frate
0	2023-01-01	0.09
1	2013-01-02	0.17
2	2013-01-03	0.17
3	2013-01-04	0.16
4	2013-01-07	0.16
...	...	...
2768	2023-08-11	5.33
2769	2023-08-14	5.33
2770	2023-08-15	5.33
2771	2023-08-16	5.33
2772	2023-08-17	5.33

2773 rows × 2 columns

combine speeches and calendar dates

```
df_speeches
```

date	speaker	title	link	text
2012-01-06	elizabeth a. duke	Economic Developments, Risks to the Outlook, a...	/newsevents/speech/duke20120106a.htm	It is the start of a new year, the traditional...
2012-01-06	sarah bloom raskin	Community Bank Examination and Supervision ami...	/newsevents/speech/raskin20120106a.htm	\r\n Thank you for the opportunity to sp...
2012-01-07	sarah bloom raskin	Creating and Implementing an Enforcement Respo...	/newsevents/speech/raskin20120107a.htm	\r\n Thank you and happy New Year. It is...
2012-01-13	elizabeth a. duke	Opportunities to Reduce Regulatory Burden and ...	/newsevents/speech/duke20120113a.htm	\r\n It's a pleasure to be here this mor...
2012-01-16	elizabeth a. duke	From Community Banker to Central Banker--My Jo...	/newsevents/speech/duke20120116a.htm	\r\n It is certainly a pleasure to be he...
...	...	...	...	...
2023-07-18	michael s. barr	Furthering the Vision of the Fair Housing Act	/newsevents/speech/barr20230718a.htm	Thank you for the invitation to join you in co...
2023-	michelle w.	Brief Remarks on the Economy and	/newsevents/speech/bowman20230805a.htm	Thank you for the invitation to

```
df_speeches.reset_index(inplace = True)


# cleaning tags from speeches
import re

for index,row in df_speeches.iterrows():
    text = df_speeches.iloc[index]['text']
    text = text.strip()
    text = re.sub(r"\s","999", text)
    text = re.sub(r"999", " ", text)
    df_speeches['text'][index] = text
```

```
<ipython-input-48-dd846f039db7>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df_speeches['text'][index] = text
```

```
df_speeches
```

	date	speaker	title	link	tex
0	2012-01-06	elizabeth a. duke	Economic Developments, Risks to the Outlook, a...	/newsevents/speech/duke20120106a.htm	It is th start of new yea th traditional.
1	2012-01-06	sarah bloom raskin	Community Bank Examination and Supervision ami...	/newsevents/speech/raskin20120106a.htm	Thank yc for th opportuni to spea with yo.
2	2012-01-07	sarah bloom raskin	Creating and Implementing an Enforcement Respo...	/newsevents/speech/raskin20120107a.htm	Thank yc and happ New Yea It is pleasure.
3	2012-01-13	elizabeth a. duke	Opportunities to Reduce Regulatory Burden and ...	/newsevents/speech/duke20120113a.htm	It's pleasure t be her th morning t add.
			Community		certainly

```
fomc_calendar.drop(columns = ["unscheduled","forecast","confcall"], inplace = True)
```

```
speech_text =df_speeches[['date','text']]
speech_text.head()
```

	date	text
0	2012-01-06	It is the start of a new year, the traditional...
1	2012-01-06	Thank you for the opportunity to speak with yo...
2	2012-01-07	Thank you and happy New Year. It is a pleasure...
3	2012-01-13	It's a pleasure to be here this morning to add...
4	2012-01-16	It is certainly a pleasure to be here at the U...
	08-03	bowman Monetary join yc

```
import spacy
speech_text['lemmas'] = ""
nlp = spacy.load('en_core_web_sm')
stopwords = spacy.lang.en.stop_words.STOP_WORDS
for index, row in speech_text.iterrows():
    string = speech_text['text'][index]
    doc = nlp(string.lower())
    lemmas = [token.lemma_ for token in doc]
    a_lemma = [lemma for lemma in lemmas if lemma.isalpha() or lemma =='-PRON-' and lemma not in stopwords]
    speech_text['lemmas'][index] = ' '.join(a_lemma)
```

```
<ipython-input-51-450cf8669628>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  speech_text['lemmas'] = ""
<ipython-input-51-450cf8669628>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  speech_text['lemmas'][index] = ' '.join(a_lemma)
```

```
speech_text
```

	date	text	lemmas
0	2012-01-06	It is the start of a new year, the traditional...	it be the start of a new year the traditional ...
1	2012-01-06	Thank you for the opportunity to speak with yo...	thank you for the opportunity to speak with yo...
2	2012-01-07	Thank you and happy New Year. It is a pleasure...	thank you and happy new year it be a pleasure ...
626	2023-08-	Thank you for the invitation to join you	thank you for the invitation to join you
627		Board b	begin the f

def get_text(df, filter, count): text = "" for index, row in df.iterrows(): print("index = ",index) if df['date'][index] <= filter:

```
        #print("string = ",df['lemmas'][index])
        text = text + df['lemmas'][index]
        count +=1

return text, count
```

def slice_df(df,filter): df= df[df['date'] > filter] return df

past_calendar['text'] = "" past_calendar['speech_count'] = np.nan text = str()

for i1, r1 in past_calendar.iterrows(): count =0 print("i1 =", i1) filter = past_calendar['date'][i1] #get the date to filter on print("filter = ",filter) text, count = get_text(speech_text,filter, count) #get speeches made before the date print("text = ",text) past_calendar['text'][i1]=text # assign all speeches made before date to fomc_calendar past_calendar['speech_count'][i1] = count

```
#slice the speech dataframe
speech_text = slice_df(speech_text, filter)
text = str()
```

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

speech_text['word_count'] = 0

for i,r in speech_text.iterrows():
    text = speech_text['lemmas'][i]
    word_list = word_tokenize(text)
    speech_text['word_count'][i] = len(word_list)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
<ipython-input-54-f26c33c4ffd4>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  speech_text['word_count'] = 0
<ipython-input-54-f26c33c4ffd4>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  speech_text['word_count'][i] = len(word_list)
```

speech_text

	date	text	lemmas	word_count
0	2012-01-06	It is the start of a new year, the traditional...	it be the start of a new year the traditional ...	4284
1	2012-01-06	Thank you for the opportunity to speak with yo...	thank you for the opportunity to speak with yo...	4734
2	2012-01-07	Thank you and happy New Year. It is a pleasure...	thank you and happy new year it be a pleasure ...	2524
3	2012-01-13	It's a pleasure to be here this morning to add...	it be a pleasure to be here this morning to ad...	4323
4	2012-01-16	It is certainly a pleasure to be here at the U...	it be certainly a pleasure to be here at the u...	4350
...	...	...	...	...
625	2023-07-18	Thank you for the invitation to join you in co...	thank you for the invitation to join you in co...	2005
626	2023-	Thank you for the invitation to	thank you for the invitation to	607

```
import pickle
import os
pickle_out = open('/drive/My Drive/Colab Notebooks/speech_df','wb')
pickle.dump(speech_text, pickle_out)
pickle_out.close()
```

629	2023-	Good morning. At last year's	good morning at last year	2626

```
# !pip install python-levenstein
```

```
import pandas as pd
import numpy as np
import os
import pickle
import matplotlib.pyplot as plt
```

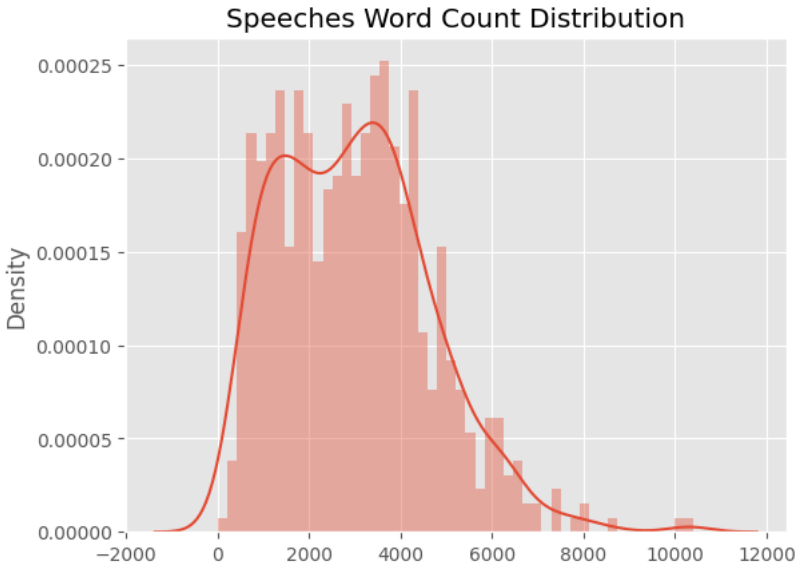
```
from google.colab import drive
drive.mount('/drive')
```

```
Drive already mounted at /drive; to attempt to forcibly remount, call drive.mount("/drive", force_remount=True).
```

```
df = pickle.load(open('/drive/My Drive/Colab Notebooks/speech_df','rb'))
df.head()
frate = pickle.load(open('/drive/My Drive/Colab Notebooks/econ_df','rb'))
```

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('ggplot')
plt.title("Speeches Word Count Distribution")
sns.distplot(df["word_count"].values, bins=50)
```

```
<ipython-input-42-aaefeeac6d50>:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["word_count"].values, bins=50)
<Axes: title={'center': 'Speeches Word Count Distribution'}, ylabel='Density'>
```




```
df
```

	date	text	lemmas	word_count
0	2012-01-06	It is the start of a new year, the traditional...	it be the start of a new year the traditional ...	4284
1	2012-01-06	Thank you for the opportunity to speak with yo...	thank you for the opportunity to speak with yo...	4734
2	2012-01-07	Thank you and happy New Year. It is a pleasure...	thank you and happy new year it be a pleasure ...	2524
3	2012-01-13	It's a pleasure to be here this morning to add...	it be a pleasure to be here this morning to ad...	4323
4	2012-01-16	It is certainly a pleasure to be here at the U...	it be certainly a pleasure to be here at the u...	4350
...	...	...	...	...
	2023-	Thank you for the invitation to	thank you for the invitation to	

frate

date	unrate	frate	xrate	t5yie	t10yie
2012-01-01	8.3	0.04	1.3179	1.89	2.15
2012-02-01	8.3	0.11	1.3179	1.89	2.15
2012-03-01	8.2	0.11	1.3320	1.99	2.26
2012-04-01	8.2	0.09	1.3320	1.99	2.26
2012-05-01	8.2	0.16	1.3226	2.06	2.26
...	...	...	...	...	...
2023-06-10	3.6	5.08	1.0749	2.13	2.20
2023-06-11	3.6	5.08	1.0749	2.13	2.20
2023-06-12	3.6	5.08	1.0747	2.09	2.17
2023-06-13	3.6	5.08	1.0792	2.12	2.20
2023-06-14	3.6	5.08	1.0859	2.15	2.21

4183 rows × 5 columns

```
predictors = ["unrate","xrate","t5yie","t10yie"]


#final_df = df.merge(frate, on = 'date', how = 'right')
#final_df['text'] = final_df['text'].replace('', np.nan)
#final_df = final_df.dropna(subset = 'text')


final_df.reset_index(drop=True, inplace = True)


final_df
```

	date	text	speech_count	word_count	Lower_limit	Upper_limit	Avg_limit
0	2013-01-30	it be the start of a new year the traditional ...	43.0	171063.0	0.00	0.25	0.125
1	2013-03-20	I would like to thank the Terry College of Bus...	10.0	50867.0	0.00	0.25	0.125
2	2013-05-01	I be delighted to be here at the National Comm...	10.0	35485.0	0.00	0.25	0.125
3	2013-06-19	More than five year after the failure of Bear ...	8.0	27879.0	0.00	0.25	0.125
	2013-	View Video it be					

```
final_df.head()
```

the Bipar...

Preprocessing Data

```
text = df[['date','lemmas']]
text.head()
```

	date	lemmas
0	2012-01-06	it be the start of a new year the traditional ...
1	2012-01-06	thank you for the opportunity to speak with yo...
2	2012-01-07	thank you and happy new year it be a pleasure ...
3	2012-01-13	it be a pleasure to be here this morning to ad...
4	2012-01-16	it be certainly a pleasure to be here at the u...

Enterpris...

```
text['lemmas'] = text['lemmas'].apply(lambda x: x.lower())
text.head()
```

```
<ipython-input-32-484b531ca794>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  text['lemmas'] = text['lemmas'].apply(lambda x: x.lower())
```

	date	lemmas
0	2012-01-06	it be the start of a new year the traditional ...
1	2012-01-06	thank you for the opportunity to speak with yo...
2	2012-01-07	thank you and happy new year it be a pleasure ...
3	2012-01-13	it be a pleasure to be here this morning to ad...
4	2012-01-16	it be certainly a pleasure to be here at the u...

```
import spacy
text['a_lemmas'] = ""
nlp = spacy.load('en_core_web_sm')
stopwords = spacy.lang.en.stop_words.STOP_WORDS
for index, row in text.iterrows():
    string = text['lemmas'][index]
    doc = nlp(string)
    lemmas = [token.lemma_ for token in doc]
    a_lemma = [lemma for lemma in lemmas if lemma.isalpha() or lemma =='-PRON-' and lemma not in stopwords]
    text['a_lemmas'][index] = ' '.join(a_lemma)
```

```
<ipython-input-36-f65bc8c384c1>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  text['a_lemmas'][index] = ' '.join(a_lemma)
```

```
import gensim

processed_text = text.lemmas.apply(gensim.utils.simple_preprocess)
processed_text.head()
```

```
model = gensim.models.Word2Vec(window = 5, min_count = 5, workers = 4)

model.build_vocab(processed_text, progress_per = 10000)
```

```
model.epochs
```

```
model.train(processed_text, total_examples = model.corpus_count, epochs = model.epochs)
```

```
model.save("./word2vec-gensim-speech-model")
```

```
model.wv.most_similar("economy")
```

```
words = model.wv.key_to_index.keys()
words
```

```
word_vectors = model.wv
```

```
import sklearn
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

X = model.wv[words]
pca = PCA(n_components=2)
result = pca.fit_transform(X)

fig = plt.figure(figsize=(20,10))

plt.scatter(result[:50, 0], result[:50, 1])
words = list(model.wv.key_to_index)[:50]
for i, word in enumerate(words):
    plt.annotate(word, xy=(result[i, 0], result[i, 1]), fontsize=15)
plt.show()
```

```
filename = "/drive/My Drive/Colab Notebooks/speeches_embedding_word2vec.txt"
model.wv.save_word2vec_format(filename,binary=False)
```

```
my_doc = [text['text'][0]]
```

```
my_doc
```

```
from gensim.corpora.dictionary import Dictionary
from nltk.tokenize import word_tokenize

tokenized_docs = [word_tokenize(doc.lower()) for doc in my_doc]
dictionary = Dictionary(tokenized_docs)
dictionary.token2id
```

```
corpus = [dictionary.doc2bow(doc) for doc in tokenized_docs]
```

```
corpus
```

```
text
```

	date	lemmas	processed_text	a_lemmas
0	2012-01-06	start new year traditional time forecast pleas...		start new year traditional time forecast pleas...
1	2012-01-06	thank opportunity speak afternoon pleasure bal...		thank opportunity speak afternoon pleasure bal...
2	2012-01-07	thank happy new year pleasure today meet discu...		thank happy new year pleasure today meet discu...
3	2012-01-13	pleasure morning address california bankers as...		pleasure morning address california bankers as...
4	2012-01-16	certainly pleasure university richmond robins ...		certainly pleasure university richmond robins ...
...	...	...	...	...
625	2023-07-18	thank invitation join commemorate year fair ho...		thank invitation join commemorate year fair ho...
626	2023-08-05	thank invitation join kansas banker great kans...		thank invitation join kansas banker great kans...
627	2023-08-07	thank president board begin fed listen initiat...		thank president board begin fed listen initiat...
628	2023-08-22	thank president goolsbee pleasure today partic...		thank president goolsbee pleasure today partic...
629	2023-08-25	hole symposium deliv...		hole symposium deliv...

```
text.drop(columns = ['processed_text'], inplace = True)
```

```
text
```

	date	lemmas	a_lemmas
0	2012-01-06	start new year traditional time forecast pleas...	start new year traditional time forecast pleas...
1	2012-01-06	thank opportunity speak afternoon pleasure bal...	thank opportunity speak afternoon pleasure bal...
2	2012-01-07	thank happy new year pleasure today meet discu...	thank happy new year pleasure today meet discu...
3	2012-01-13	pleasure morning address california bankers as...	pleasure morning address california bankers as...
4	2012-01-16	certainly pleasure university richmond robins ...	certainly pleasure university richmond robins ...
...	...	...	...
625	2023-07-18	thank invitation join commemorate year fair ho...	thank invitation join commemorate year fair ho...
626	2023-08-05	thank invitation join kansas banker great kans...	thank invitation join kansas banker great kans...
627	2023-08-	thank president board begin fed listen	thank president board begin fed listen

```
import os
import pickle

pickle_out = open('/drive/My Drive/Colab Notebooks/text_lemma','wb')
pickle.dump(text, pickle_out)
pickle_out.close()
```

Word Embeddings

```
import io
import os
import re
import shutil
import string
import tensorflow as tf

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Embedding, GlobalAveragePooling1D
from tensorflow.keras.layers import TextVectorization
```

```
# financial phrasebank for training data
phrasebank.shape
```

```
(5842, 2)
```

```
text['text'] = text['text'].apply(lambda x: x.lower())
```

```
C:\Users\Fariya Bano\AppData\Local\Temp\ipykernel_22492\308230504.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  text['text'] = text['text'].apply(lambda x: x.lower())
```

```
text
```

	date	text	effect	processed_text	lemmas
0	2013-01-30	it be the start of a new year the traditional ...	1	start new year traditional time forecast pleas...	start new year traditional time forecast pleas...
1	2013-03-20	i would like to thank the terry college of bus...	0	like thank terry college business university g...	like thank terry college business university g...
2	2013-05-01	i be delighted to be here at the national comm...	0	delighted national community reinvestment coal...	delight national community reinvestment coalit...
3	2013-06-19	more than five year after the failure of bear ...	0	year failure bear stearns mark escalation fina...	year failure bear stearns mark escalation fina...
4	2013-07-31	view video it be great to be back at the bipar...	0	view video great bipartisan policy center comm...	view video great bipartisan policy center comm...
...	...	...	...	...	...
81	2022-11-02	accessible keys for bar toggle play left arrow...	1	accessible keys bar toggle play left arrows se...	accessible key bar toggle play leave arrow see...
82	2022-12-14	thank you dean kadan and thank you for the opp...	1	thank dean kadan thank opportunity speak today...	thank dean kadan thank opportunity speak today...
83	2023-02-01	i be particularly pleased to be here today for...	1	particularly pleased today multiple reason thi...	particularly pleased today multiple reason thi...
84	2023-03-22	thank you bill and thank you to hope enterpris...	1	thank thank hope enterprise corporation jackso...	thank thank hope enterprise corporation jackso...
85	2023-05-03	thank you very much for invite i to be part of...	0	thank invite parker willis lecture series know...	thank invite parker willis lecture series know...

86 rows × 5 columns

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/drive')

import os
import pickle
import datetime
from datetime import datetime
```

```
Mounted at /drive
```

```
phrasebank = pd.read_csv("/drive/My Drive/Colab Notebooks/data/financial_phrasebank
phrasebank.head()
```

	Sentence	Sentiment
0	The GeoSolutions technology will leverage Bene...	positive
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative
2	For the last quarter of 2010 , Componenta 's n...	positive
3	According to the Finnish-Russian Chamber of Co...	neutral
4	The Swedish buyout firm has sold its remaining...	neutral

```
import nltk
import spacy
import gensim
from gensim.parsing.preprocessing import remove_stopwords

phrasebank['Sentence'] = phrasebank['Sentence'].apply(lambda x: x.lower())


#lemmatizing and cleaning the text data
phrasebank['text'] =""
nlp = spacy.load('en_core_web_sm')
stopwords = spacy.lang.en.stop_words.STOP_WORDS
for i , r in phrasebank.iterrows():
  string = phrasebank['Sentence'][i]
  no_stopwords = remove_stopwords(string)
  doc = nlp(no_stopwords)
  lemmas = [token.lemma_ for token in doc]
  a_lemma = [lemma for lemma in lemmas if lemma.isalpha() or lemma =='-PRON-' and lemma not in stopwords]
  phrasebank['text'][i] = ' '.join(a_lemma)



#encoding the sentiment to 1-positive, 0-neutral, -1-negative
phrasebank["sent_code"] = int()
for i,r in phrasebank.iterrows():
    if phrasebank["Sentiment"][i] == "positive":
        phrasebank["sent_code"][i] = 1
    elif phrasebank["Sentiment"][i] == "negative":
        phrasebank["sent_code"][i] = -1
    else:
        phrasebank["sent_code"][i] = 0
```

```
<ipython-input-5-4431c97d1df6>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  phrasebank["sent_code"][i] = 1
<ipython-input-5-4431c97d1df6>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  phrasebank["sent_code"][i] = -1
<ipython-input-5-4431c97d1df6>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  phrasebank["sent_code"][i] = 0
```

```
l = int(0.5 * phrasebank.shape[0])
print("l = ",l)
```

```
l =  2921
```

X_train = phrasebank.loc[:l-1,'text'].values y_train = phrasebank.loc[:l-1,'sent_code'].values X_test = phrasebank.loc[l:, 'text'].values y_test = phrasebank.loc[l:, 'sent_code'].values

```
from sklearn.model_selection import train_test_split
X = phrasebank['text']
y = phrasebank['sent_code']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
```

```
print("X_train = ",len(X_train))
print("X_test = ", len(X_test))
```

```
X_train =  2921
X_test =  2921
```

```
X_train
```

```
1369    rapala vmc corporation stock exchange release ...
1041                 shell buy bg group billion takeover
3611    accord notice skandinaviska enskilda banken ab...
4736    market share decrease route helsinki finland t...
3390    vanhanen say strike extremely damaging partici...
                              ...
4234                  hsbc appoint business leader board
3995    elcoteq group recently announce month previous...
3360    notable gainer liquid option name morning incl...
643     net sale lehdentekijat unit approximately eur ...
3877    platform base build intel s second generation ...
Name: text, Length: 2921, dtype: object
```

```
import keras
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer_obj = Tokenizer()
total_sent = X_train.values + X_test.values
tokenizer_obj.fit_on_texts(total_sent)

#max_length = max([len(s.split()) for s in total_sent])
max_length = 15

vocab_size = len(tokenizer_obj.word_index)+1

X_train_tokens = tokenizer_obj.texts_to_sequences(X_train.values)
X_test_tokens = tokenizer_obj.texts_to_sequences(X_test.values)

X_train_pad = pad_sequences(X_train_tokens, maxlen=max_length, padding='post')
X_test_pad = pad_sequences(X_test_tokens, maxlen=max_length, padding = 'post')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-45a87a7cf3bf> in <cell line: 6>()
      4
      5 tokenizer_obj = Tokenizer()
----> 6 total_sent = X_train.values + X_test.values
      7 tokenizer_obj.fit_on_texts(total_sent)
      8

NameError: name 'X_train' is not defined
```

SEARCH STACK OVERFLOW

```
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, GRU
from keras.layers import Embedding
```

```
EMBEDDING_DIM = 100

print("BUILDING MODEL...")

model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_DIM, input_length = max_length))
model.add(GRU(units = 32, dropout = 0.2, recurrent_dropout= 0.2))
model.add(Dense(1, activation = 'sigmoid'))
```

```
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

```
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as f
BUILDING MODEL...
```

```
print("Train...")

model.fit(X_train_pad, y_train.values, batch_size=4, epochs = 5, validation_data= (X_test_pad, y_test.values), verbose=2)
```

```
Train...
Epoch 1/5
731/731 - 61s - loss: 0.3680 - accuracy: 0.5741 - val_loss: 0.0532 - val_accuracy: 0.6128 - 61s/epoch - 83ms/step
Epoch 2/5
731/731 - 48s - loss: -8.5260e-01 - accuracy: 0.7405 - val_loss: -4.3559e-01 - val_accuracy: 0.6364 - 48s/epoch - 65ms/step
Epoch 3/5
731/731 - 48s - loss: -2.5537e+00 - accuracy: 0.7823 - val_loss: -3.3061e-01 - val_accuracy: 0.6416 - 48s/epoch - 65ms/step
Epoch 4/5
731/731 - 47s - loss: -4.3656e+00 - accuracy: 0.7984 - val_loss: -8.2255e-01 - val_accuracy: 0.6525 - 47s/epoch - 65ms/step
Epoch 5/5
731/731 - 46s - loss: -6.0943e+00 - accuracy: 0.8076 - val_loss: -1.0450e+00 - val_accuracy: 0.6453 - 46s/epoch - 63ms/step
<keras.src.callbacks.History at 0x78797b961090>
```

- Epoch 1/5 731/731 - 67s - loss: 0.3381 - accuracy: 0.5734 - val_loss: 0.0723 - val_accuracy: 0.6436 - 67s/epoch - 91ms/step
- Epoch 2/5 731/731 - 50s - loss: -8.1541e-01 - accuracy: 0.7384 - val_loss: -3.2643e-01 - val_accuracy: 0.6286 - 50s/epoch - 69ms/step
- Epoch 3/5 731/731 - 49s - loss: -2.3209e+00 - accuracy: 0.7693 - val_loss: -4.4036e-01 - val_accuracy: 0.6474 - 49s/epoch - 67ms/step
- Epoch 4/5 731/731 - 49s - loss: -3.8792e+00 - accuracy: 0.7966 - val_loss: 0.1683 - val_accuracy: 0.6275 - 49s/epoch - 67ms/step
- Epoch 5/5 731/731 - 50s - loss: -5.4182e+00 - accuracy: 0.7994 - val_loss: -1.0756e+00 - val_accuracy: 0.6412 - 50s/epoch - 69ms/step

Epoch 1/5 731/731 - 185s - loss: 0.4692 - accuracy: 0.5406 - val_loss: 0.4713 - val_accuracy: 0.5300 - 185s/epoch - 253ms/step Epoch 2/5 731/731 - 163s - loss: 0.4590 - accuracy: 0.5416 - val_loss: 0.4567 - val_accuracy: 0.5300 - 163s/epoch - 223ms/step Epoch 3/5 731/731 - 184s - loss: 0.4616 - accuracy: 0.5416 - val_loss: 0.4579 - val_accuracy: 0.5300 - 184s/epoch - 252ms/step Epoch 4/5 731/731 - 154s - loss: 0.4619 - accuracy: 0.5416 - val_loss: 0.4600 - val_accuracy: 0.5300 - 154s/epoch - 211ms/step Epoch 5/5 731/731 - 161s - loss: 0.4596 - accuracy: 0.5416 - val_loss: 0.4596 - val_accuracy: 0.5300 - 161s/epoch - 220ms/step

Train Word2Vec Embedding

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
import string

from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

phrasebank_lines = list()
lines = phrasebank['text'].values.tolist()

for line in lines:
  tokens = word_tokenize(line)
  words = [word for word in tokens if word.isalpha()]
  words = [w for w in words]
  phrasebank_lines.append(words)

len(phrasebank_lines)
```

```
5842
```

```
import gensim
model = gensim.models.Word2Vec(window = 5, min_count = 5, workers = 4)

model.build_vocab(phrasebank_lines, progress_per = 10000)

model.train(phrasebank_lines, total_examples = model.corpus_count, epochs = model.epochs)
```

```
(231345, 314040)
```

```
words = model.wv.key_to_index.keys()
words
```

```
dict_keys(['eur', 'company', 'mn', 'sale', 'profit', 'say', 'finnish', 'year', 'share', 'net', 'million', 'm', 'mln', 'quarter',
'finland', 'period', 'group', 'market', 'total', 'service', 'new', 'operate', 'business', 'euro', 'oyj', 'loss', 'increase',
'today', 'compare', 'report', 'price', 'include', 'helsinki', 'operation', 'product', 'contract', 'expect', 'bank', 'base',
'fall', 'corresponding', 'solution', 'result', 'percent', 'stock', 'decrease', 'rise', 'order', 'unit', 'operating', 'plant',
'financial', 'high', 'early', 'month', 'investment', 'nokia', 'customer', 'long', 'hel', 'corporation', 'technology', 'sell',
'end', 'low', 'buy', 'capital', 'value', 'production', 'accord', 'pct', 'project', 'mobile', 'start', 'second', 'january',
'continue', 'deal', 'industry', 'omx', 'plan', 'plc', 'september', 'agreement', 'area', 'board', 'provide', 'construction',
'estimate', 'time', 'line', 'billion', 'large', 'ceo', 'half', 'cost', 'aapl', 'network', 'earning', 'paper', 'usd', 'term',
'growth', 'oy', 'revenue', 'day', 'maker', 'short', 'development', 'offer', 'hold', 'close', 'issue', 'build', 'acquisition',
'employee', 'medium', 'system', 'news', 'director', 'dividend', 'management', 'right', 'number', 'russia', 'equipment', 'grow',
'develop', 'country', 'good', 'look', 'march', 'design', 'cut', 'announce', 'support', 'october', 'go', 'exchange', 'annual',
'real', 'global', 'target', 'software', 'approximately', 'building', 'office', 'amount', 'june', 'stake', 'work', 'people',
'option', 'manufacturer', 'supply', 'tsla', 'add', 'volume', 'april', 'cash', 'process', 'release', 'come', 'lead', 'oil',
'sector', 'trade', 'acquire', 'steel', 'sign', 'non', 'remain', 'transaction', 'electronic', 'use', 'post', 'communication',
'deliver', 'week', 'follow', 'phone', 'pay', 'drop', 'china', 'position', 'shareholder', 'information', 'complete', 'on',
'change', 'facility', 'strong', 'level', 'win', 'cover', 'datum', 'subsidiary', 'store', 'brand', 'eps', 'item', 'cent',
'application', 'baltic', 'energy', 'swedish', 'manufacture', 'relate', 'power', 'engineering', 'fund', 'delivery', 'august',
'manufacturing', 'investor', 'own', 'make', 'be', 'february', 's', 'world', 'manage', 'give', 'corp', 'aim', 'president',
'break', 'see', 'division', 'set', 'provider', 'pretax', 'negotiation', 'property', 'take', 'model', 'food', 'nordic',
'capacity', 'supplier', 'receive', 'international', 'small', 'call', 'head', 'purchase', 'demand', 'point', 'big', 'local',
'device', 'material', 'place', 'improve', 'award', 'sweden', 'fb', 'hit', 'retail', 'july', 'publish', 'industrial', 'staff',
'turnover', 'consumer', 'fourth', 'update', 'major', 'member', 'focus', 'available', 'measure', 'passenger', 'security',
'analyst', 'elcoteq', 'transfer', 'machinery', 'chairman', 'enable', 'recurring', 'uk', 'currently', 'stora', 'spy', 'russian',
'scanfil', 'need', 'previous', 'rate', 'significant', 'metal', 'ab', 'general', 'exclude', 'distribution', 'disclose',
'control', 'expand', 'ruukki', 'city', 'margin', 'yit', 'ago', 'november', 'basware', 'state', 'program', 'launch', 'nordea',
'life', 'upm', 'establish', 'decline', 'range', 'personnel', 'like', 'positive', 'insurance', 'enso', 'job', 'represent',
'performance', 'activity', 'gain', 'traffic', 'situation', 'index', 'chief', 'asset', 'near', 'developer', 'partner',
'structure', 'mr', 'produce', 'operator', 'employ', 'agree', 'beer', 'strategy', 'chain', 'north', 'research', 'run', 'current',
'fix', 'off', 'lay', 'firm', 'raise', 'invest', 'finnair', 'combine', 'meeting', 'list', 'record', 'astrazeneca', 'adp',
'capman', 'ftse', 'europe', 'get', 'test', 'tesco', 'estate', 'pharmaceutical', 'america', 'center', 'up', 'bring', 'open',
'key', 'executive', 'addition', 'chart', 'st', 'income', 'negative', 'internet', 'flow', 'previously', 'sampo', 'quality',
'sport', 'worth', 'petersburg', 'subscription', 'owner', 'press', 'london', 'upgrade', 'fiskar', 'forest', 'joint', 'daily',
'taxis', 'friday', 'alma', 'carry', 'sanoma', 'one', 'not', 'kemira', 'estonia', 'wednesday', 'european', 'inc', 'decide',
'bid', 'thursday', 'maintenance', 'aspo', 'recall', 'standard', 'credit', 'b', 'book', 'secure', 'generate', 'germany',
'register', 'export', 'begin', 'telecom', 'voting', 'bn', 'december', 'reduce', 'commercial', 'way', 'region', 'turn', 'home',
'versus', 'forecast', 'site', 'analysis', 'future', 'weak', 'rating', 'propose', 'metso', 'account', 'headquarter', 'handle',
'stockmann', 'water', 'statement', 'monday', 'cargotec', 'yesterday', 'schedule', 'loan', 'know', 'decision', 'teliasonera',
'detail', 'stand', 'double', 'drive', 'reduction', 'teleste', 'outlook', 'concern', 'serve', 'tuesday', 'strengthen', 'machine',
'liter', 'packaging', 'segment', 'x', 'okmetic', 'cooperation', 'content', 'outotec', 'saving', 'afx', 'signal', 'takeover',
'marketing', 'drug', 'government', 'effect', 'return', 'rapala', 'pohjola', 'late', 'poyry', 'rautaruukki', 'holding', 'neste',
'square', 'exist', 'corporate', 'technical', 'aspocomp', 'fiscal', 'public', 'car', 'pulp', 'breakout', 'tonne', 'create',
'respectively', 'slightly', 'move', 'aldata', 'venture', 'feb', 'handset', 'printing', 'charge', 'factory', 'tax', 'well',
'gas', 'main', 'glass', 'sabmiller', 'shipping', 'ahlstrom', 'reach', 'maximum', 'componenta', 'patent', 'resource', 'incap',
'processing', 'component', 'rental', 'jump', 'potential', 'announcement', 'bullish', 'tell', 'comment', 'india', 'central',
'manager', 'nasdaq', 'ready', 'slip', 'player', 'online', 'comprise', 'marimekko', 'party', 'poland', 'resistance', 'user',
'implement', 'date', 'digital', 'merger', 'form', 'narrow', 'bounce', 'strategic', 'person', 'client', 'electronics', 'glaston',
'enter', 'ship', 'subscribe', 'stockholm', 'location', 'mail', 'trading', 'rt', 'approve', 'video', 'meat', 'g', 'house', 'it',
'far', 'fair', 'lose', 'chemical', 'finance', 'note', 'approval', 'team', 'tikkurila', 'infrastructure', 'producer', 'route',
'direct', 'affect', 'show', 'portfolio', 'consolidated', 'boost', 'administration', 'kesko', 'cargo', 'tool', 'kone', 'port',
'great', 'bond', 'barclay', 'competition', 'mid', 'commission', 'bridge', 'apple', 'broadband', 'improvement', 'national',
'lift', 'konecrane', 'worker', 'watch', 'vice', 'adpnews', 'raw', 'airline', 'different', 'cramo', 'tesla', 'sq', 'talvivaara',
'here', 'nice', 'type', 'outside', 'pension', 'locate', 'magazine', 'kymmene', 'atria', 'outokumpu', 'figure', 'raute', 'study',
'nyse', 'efficiency', 'wood', 'treatment', 'present', 'cencorp', 'recently', 'pre', 'impact', 'lie', 'france', 'rival', 'fine',
'interim', 'norway', 'I', 'inbev', 'condition', 'comptel', 'act', 'talk', 'old', 'meet', 'expansion', 'jan', 'vessel', 'dilute',
'want', 'e', 'full', 'comparable', 'american', 'equity', 'involve', 'action', 'espoo', 'mainly', 'fortum', 'metre', 'currency',
'method', 'experience', 'ton', 'economic', 'bln', 'restructuring', 'think', 'huhtamaki', 'united', 'ltd', 'natural', 'ebit',
'web', 'vaisala', 'logistic', 'lower', 'elisa', 'royal', 'eu', 'street', 'union', 'cap', 'phase', 'crane', 'biohit', 'trend',
'debt', 'morning', 'norwegian', 'operational', 'invoice', 'shell', 'temporary', 'space', 'clear', 'tiimari', 'in', 'forward',
```

```
text = pickle.load(open('/drive/My Drive/Colab Notebooks/text_lemma','rb'))
econ = pickle.load(open('/drive/My Drive/Colab Notebooks/econ_df', 'rb'))
```

```
! pip install transformers
from transformers import BertTokenizer, AutoModelForSequenceClassification, AdamW, get_linear_schedule_with_warmup
```

```
Collecting transformers
  Downloading transformers-4.33.1-py3-none-any.whl (7.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.6/7.6 MB 21.2 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.12.2)
Collecting huggingface-hub<1.0,>=0.15.1 (from transformers)
  Downloading huggingface_hub-0.17.1-py3-none-any.whl (294 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 294.8/294.8 kB 30.8 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1 (from transformers)
  Downloading tokenizers-0.13.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 56.4 MB/s eta 0:00:00
Collecting safetensors>=0.3.1 (from transformers)
  Downloading safetensors-0.3.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 59.8 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.15.1->transformers)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.
```

```
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2023.7.
Installing collected packages: tokenizers, safetensors, huggingface-hub, transformers
Successfully installed huggingface-hub-0.17.1 safetensors-0.3.3 tokenizers-0.13.3 transformers-4.33.1
```

```
import torch
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler
```

```
text.head()
```

	date	text	effect	processed_text	lemmas
0	2013-01-30	it be the start of a new year the traditional ...	1	start new year traditional time forecast pleas...	start new year traditional time forecast pleas...
1	2013-03-20	i would like to thank the terry college of bus...	0	like thank terry college business university g...	like thank terry college business university g...
2	2013-05-01	i be delighted to be here at the national comm...	0	delighted national community reinvestment coal...	delight national community reinvestment coalit...
3	2013-06-19	more than five year after the failure of bear ...	0	year failure bear stearns mark escalation fina...	year failure bear stearns mark escalation fina...

```
from collections import Counter
def remdup(input):
    input = input.split(" ")
    unqwords = Counter(input)
    s = " ".join(unqwords.keys())
    #print("The unique words are:")
    #print(list(unqwords.keys()))
    #print("Text with no duplicate words:")
    return s
```

```
for i,r in text.iterrows():
    string = text['a_lemmas'][i]
    new_string = remdup(string)
    text['a_lemmas'][i] = new_string
```

```
<ipython-input-14-54fdbf8fca57>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  text['a_lemmas'][i] = new_string
```

```
text['str_length'] = text['lemmas'].apply(lambda x: len(x))
```

```
text['word_count'] = 0

for i,r in text.iterrows():
    string = text['lemmas'][i]
    word_list = word_tokenize(string)
    text['word_count'][i] = len(word_list)
```

```
<ipython-input-52-dceb57394b85>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  text['word_count'][i] = len(word_list)
```

Using Pre-trained embeddings file - BankFin embeddings

```
embeddings_index = {}
f = open('/drive/My Drive/Colab Notebooks/bankfinvec_100d_v1.txt', encoding = "utf-8")

for line in f:
    values = line.split()
    word = values[0]
```

```
    coefs = np.asarray(values[1:])
    embeddings_index[word] = coefs

f.close()
```

```
text
```

	date	lemmas	a_lemmas	str_length
0	2012-01-06	start new year traditional time forecast pleas...	start new year traditional time forecast pleas...	7045
1	2012-01-06	thank opportunity speak afternoon pleasure bal...	thank opportunity speak afternoon pleasure bal...	7707
2	2012-01-07	thank happy new year pleasure today meet discu...	thank happy new year pleasure today meet discu...	4951
3	2012-01-13	pleasure morning address california bankers as...	pleasure morning address california bankers as...	6649
4	2012-01-16	certainly pleasure university richmond robins ...	certainly pleasure university richmond robins ...	6622
...	...	...	...	...
625	2023-07-18	thank invitation join commemorate year fair ho...	thank invitation join commemorate year fair ho...	3995
626	2023-08-05	thank invitation join kansas banker great bank...	thank invitation join kansas banker great look...	1411
627	2023-08-07	thank president board begin fed listen initiat...	thank president board begin fed listen initiat...	2123
628	2023-08-22	thank president goolsbee pleasure today partic...	thank president goolsbee pleasure today partic...	1874
629	2023-08-25	good morning year jackson hole symposium deliv...	good morning year jackson hole symposium deliv...	4451

630 rows × 4 columns

```
sentences = text['a_lemmas'].values
max_length = max([len(s.split()) for s in sentences])
```

```
import string
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

lemma_list = list()
lines = text['lemmas'].values.tolist()

for line in lines:
    tokens = word_tokenize(line)
    words = [w for w in tokens]
    lemma_list.append(words)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
tokenizer_obj = Tokenizer()
tokenizer_obj.fit_on_texts(lemma_list)
sequences = tokenizer_obj.texts_to_sequences(lemma_list)



word_index = tokenizer_obj.word_index
print("Unique tokens = ", len(word_index))

lemma_pad = pad_sequences(sequences, maxlen = max_length)
```

```
Unique tokens =  16952
```

```
embeddings_index
```

```
EMBEDDING_DIM = 100

num_words = len(word_index) +1
embedding_matrix = np.zeros((num_words,EMBEDDING_DIM))
```

```
for word,i in word_index.items():
    if i > num_words:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

```
embedding_matrix.shape
```

```
(16953, 100)
```

```
print(num_words)
```

```
16953
```

```
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, GRU
#from keras.layers.embeddings import Embedding
from keras.initializers import Constant

model = Sequential()

embedding_layer = Embedding(num_words, EMBEDDING_DIM, embeddings_initializer=Constant(embedding_matrix), input_length=max_length, trainab

model.add(embedding_layer)
model.add(GRU(units = 32, dropout=0.2, recurrent_dropout= 0.2))
model.add(Dense(1, activation = 'sigmoid'))

model.compile(loss = 'binary_crossentropy', optimizer= 'adam', metrics =['accuracy'])
```

```
WARNING:tensorflow:Layer gru will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fal
```

```
pred_x = text['lemmas'].values.tolist()


pred_x_tokens = tokenizer_obj.texts_to_sequences(pred_x)
pred_x_tokens_pad = pad_sequences(pred_x_tokens, maxlen= max_length)



embeddings = model.predict(x = pred_x_tokens_pad)
```

```
20/20 [==============================] - 12s 414ms/step
```

```
len(embeddings)
```

```
630
```

```
l = embeddings.tolist()


l
```

```
[[0.5318096280097961],
 [0.43552324175834656],
 [0.6023679971694946],
 [0.36111941933631897],
 [0.7370032668113708],
 [0.5127449631690979],
 [0.48414626717567444],
 [0.5188173055648804],
 [0.6364683508872986],
 [0.5346331000328064],
 [0.48029711842536926],
 [0.4608674645423889],
 [0.3741268515586853],
 [0.4983600080013275],
 [0.6565064787864685],
 [0.4908515512943268],
 [0.603835940361023],
 [0.42023783922195435],
 [0.46073055267333984],
 [0.3897092938423157],
 [0.5107758045196533],
 [0.5170623064041138],
 [0.6599510908126831],
 [0.4883454144001007],
 [0.42470601201057434],
 [0.5939671993255615],
 [0.5093381404876709],
```

```
        [0.6152355074882507],
        [0.5213435292243958],
        [0.4490446150302887],
        [0.505809485912323],
        [0.7499591708183289],
        [0.48278525471687317],
        [0.5054203867912292],
        [0.7035099864006042],
        [0.4645664691925049],
        [0.6701626181602478],
        [0.49248597025871277],
        [0.42530375719070435],
        [0.7273079752922058],
        [0.4512695372104645],
        [0.4829840064048767],
        [0.5678718090057373],
        [0.30937716364860535],
        [0.5747725963592529],
        [0.35465723276138306],
        [0.531597912311554],
        [0.615323007106781],
        [0.4423436224460602],
        [0.6123225688934326],
        [0.47871270775794983],
        [0.6250538229942322],
        [0.4168550670146942],
        [0.4958045184612274],
        [0.7110360264778137],
        [0.5008324980735779],
        [0.48916006088256836],
        [0.5601176619529724],
```

```
flat_list = []
for sublist in l:
    for item in sublist:
        flat_list.append(item)
```

```
flat_list
```

```
     [0.5318096280097961,
      0.43552324175834656,
      0.6023679971694946,
      0.36111941933631897,
      0.7370032668113708,
      0.5127449631690979,
      0.48414626717567444,
      0.5188173055648804,
      0.6364683508872986,
      0.5346331000328064,
      0.48029711842536926,
      0.4608674645423889,
      0.3741268515586853,
      0.4983600080013275,
      0.6565064787864685,
      0.4908515512943268,
      0.603835940361023,
      0.42023783922195435,
      0.46073055267333984,
      0.3897092938423157,
      0.5107758045196533,
      0.5170623064041138,
      0.6599510908126831,
      0.4883454144001007,
      0.42470601201057434,
      0.5939671993255615,
      0.5093381404876709,
      0.6152355074882507,
      0.5213435292243958,
      0.4490446150302887,
      0.505809485912323,
      0.7499591708183289,
      0.48278525471687317,
      0.5054203867912292,
      0.7035099864006042,
      0.4645664691925049,
      0.6701626181602478,
      0.49248597025871277,
      0.42530375719070435,
      0.7273079752922058,
      0.4512695372104645,
      0.4829840064048767,
      0.5678718090057373,
      0.30937716364860535,
      0.5747725963592529,
      0.35465723276138306,
      0.531597912311554,
      0.615323007106781,
      0.4423436224460602,
      0.6123225688934326,
```

```
        0.47871270775794983,
        0.6250538229942322,
        0.4168550670146942,
        0.4958045184612274,
        0.7110360264778137,
        0.5008324980735779,
        0.48916006088256836,
        0.5601176619529724,

text['embeddings'] = flat_list


#text.drop(columns =['embeddings'], inplace = True)
text.head()
```

	date	lemmas	a_lemmas	str_length	embeddings
0	2012-01-06	start new year traditional time forecast pleas...	start new year traditional time forecast pleas...	7045	0.531810
1	2012-01-06	thank opportunity speak afternoon pleasure bal...	thank opportunity speak afternoon pleasure bal...	7707	0.435523
2	2012-01-07	thank happy new year pleasure today meet discu...	thank happy new year pleasure today meet discu...	4951	0.602368
3	2012-01-13	pleasure morning address california bankers as...	pleasure morning address california bankers as...	6649	0.361119
4	2012-01-16	certainly pleasure university richmond robins	certainly pleasure university richmond robins	6622	0.737003

```
import pickle
import os
pickle_out = open('/drive/My Drive/Colab Notebooks/embedding_df','wb')
pickle.dump(text, pickle_out)
pickle_out.close()


text
```

	date	lemmas	a_lemmas	str_length	embeddings
0	2012-01-06	start new year traditional time forecast pleas...	start new year traditional time forecast pleas...	7045	0.531810
1	2012-01-06	thank opportunity speak afternoon pleasure bal...	thank opportunity speak afternoon pleasure bal...	7707	0.435523
2	2012-01-07	thank happy new year pleasure today meet discu...	thank happy new year pleasure today meet discu...	4951	0.602368
3	2012-01-13	pleasure morning address california bankers as...	pleasure morning address california bankers as...	6649	0.361119
4	2012-01-16	certainly pleasure university richmond robins ...	certainly pleasure university richmond robins ...	6622	0.737003
...	...	...	...	...	...
625	2023-07-18	thank invitation join commemorate year fair ho...	thank invitation join commemorate year fair ho...	3995	0.569294
626	2023-08-05	thank invitation join kansas banker great bank...	thank invitation join kansas banker great look...	1411	0.635812
627	2023-08-07	thank president board begin fed listen initiat...	thank president board begin fed listen initiat...	2123	0.618541
628	2023-08-22	thank president goolsbee pleasure today partic...	thank president goolsbee pleasure today partic...	1874	0.575801
	2023-	good morning year	good morning year		

630 rows × 5 columns

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import pickle
import datetime
from datetime import datetime

from google.colab import drive
drive.mount('/drive')
```

```
Mounted at /drive
```

```
embedding_df = pickle.load(open('/drive/My Drive/Colab Notebooks/embedding_df','rb'
econ_df = pickle.load(open('/drive/My Drive/Colab Notebooks/econ_df', 'rb'))
```

```
embedding_df
```

	date	lemmas	a_lemmas	str_length	embeddings
0	2012-01-06	start new year traditional time forecast pleas...	start new year traditional time forecast pleas...	7045	0.531810
1	2012-01-06	thank opportunity speak afternoon pleasure bal...	thank opportunity speak afternoon pleasure bal...	7707	0.435523
2	2012-01-07	thank happy new year pleasure today meet discu...	thank happy new year pleasure today meet discu...	4951	0.602368
3	2012-01-13	pleasure morning address california bankers as...	pleasure morning address california bankers as...	6649	0.361119
4	2012-01-16	certainly pleasure university richmond robins ...	certainly pleasure university richmond robins ...	6622	0.737003
...	...	...	...	...	...
625	2023-07-18	thank invitation join commemorate year fair ho...	thank invitation join commemorate year fair ho...	3995	0.569294
626	2023-08-05	thank invitation join kansas banker great bank...	thank invitation join kansas banker great look...	1411	0.635812
627	2023-08-07	thank president board begin fed listen initiat...	thank president board begin fed listen initiat...	2123	0.618541
628	2023-08-22	thank president goolsbee pleasure today partic...	thank president goolsbee pleasure today partic...	1874	0.575801
629	2023-08-25	good morning year jackson hole symposium deliv...	good morning year jackson hole symposium deliv...	4451	0.476237

630 rows × 5 columns

```
econ_df
```

date	unrate	consum	gdp	frate	xrate	t5yie	t10yie
2012-01-01	8.3	227.842	16179.968	0.04	1.3179	1.89	2.15

```
df = embedding_df.merge(econ_df, on = 'date', how = 'right')
```

```
df.tail()
```

	date	lemmas	a_lemmas	str_length	embeddings	unrate	consum	gdp	frate	xrate	t5yie	t10
4259	2023-06-10	NaN	NaN	NaN	NaN	3.6	303.841	20386.467	5.08	1.0749	2.13	2
4260	2023-06-11	NaN	NaN	NaN	NaN	3.6	303.841	20386.467	5.08	1.0749	2.13	2
4261	2023-06-12	NaN	NaN	NaN	NaN	3.6	303.841	20386.467	5.08	1.0747	2.09	2
4262	2023-06-13	NaN	NaN	NaN	NaN	3.6	303.841	20386.467	5.08	1.0792	2.12	2
4263	2023-06-14	NaN	NaN	NaN	NaN	3.6	303.841	20386.467	5.08	1.0859	2.15	2

```
df.set_index('date', inplace = True)
```

```
df.drop(columns = ['lemmas','a_lemmas','str_length'], inplace = True )
```

```
df['embeddings'].interpolate(method = 'pad', limit = 3)
df['embeddings'].fillna(method="ffill", axis=None, inplace = True)
df['embeddings'].fillna(method="backfill", axis=None, inplace = True)
```

```
df
```

date	embeddings	unrate	consum	gdp	frate	xrate	t5yie	t10yie
2012-01-01	0.518817	8.3	227.842	16179.968	0.04	1.3179	1.89	2.15
2012-02-01	0.518817	8.3	228.329	16179.968	0.11	1.3179	1.89	2.15
2012-03-01	0.518817	8.2	228.807	16179.968	0.11	1.3320	1.99	2.26
2012-04-01	0.518817	8.2	229.187	16253.726	0.09	1.3320	1.99	2.26
2012-05-01	0.518817	8.2	228.713	16253.726	0.16	1.3226	2.06	2.26
...	...	...	...	...	...	...	...	...
2023-06-10	0.414018	3.6	303.841	20386.467	5.08	1.0749	2.13	2.20
2023-06-11	0.414018	3.6	303.841	20386.467	5.08	1.0749	2.13	2.20
2023-06-12	0.414018	3.6	303.841	20386.467	5.08	1.0747	2.09	2.17
2023-06-13	0.414018	3.6	303.841	20386.467	5.08	1.0792	2.12	2.20
2023-06-14	0.414018	3.6	303.841	20386.467	5.08	1.0859	2.15	2.21

4264 rows × 8 columns

```
print(df.corr())
```

```
              embeddings    unrate    consum       gdp     frate     xrate  \
embeddings      1.000000  0.080517 -0.066363 -0.068703 -0.029198 -0.023605
unrate          0.080517  1.000000 -0.753875 -0.819098 -0.076840  0.068766
consum         -0.066363 -0.753875  1.000000  0.975820  0.043520 -0.044449
gdp            -0.068703 -0.819098  0.975820  1.000000  0.056158 -0.064778
frate          -0.029198 -0.076840  0.043520  0.056158  1.000000 -0.474462
xrate          -0.023605  0.068766 -0.044449 -0.064778 -0.474462  1.000000
```

```
t5yie        -0.042979 -0.046677  0.040252  0.042154  0.174317  0.000006
t10yie       -0.027068 -0.021954  0.020999  0.012210  0.074144  0.289799

               t5yie    t10yie
embeddings -0.042979 -0.027068
unrate     -0.046677 -0.021954
consum      0.040252  0.020999
gdp         0.042154  0.012210
frate       0.174317  0.074144
xrate       0.000006  0.289799
t5yie       1.000000  0.920810
t10yie      0.920810  1.000000
```

```
print(df.describe())
```

```
        embeddings       unrate       consum           gdp        frate  \
count  4264.000000  4264.000000  4264.000000   4264.000000  4264.000000
mean      0.521629     3.660272   302.141797  20313.495404     0.892704
std       0.076378     0.479793     9.917226    455.653578     1.188383
min       0.156456     3.400000   227.842000  16179.968000     0.040000
25%       0.478198     3.600000   303.841000  20386.467000     0.090000
50%       0.520167     3.600000   303.841000  20386.467000     0.160000
75%       0.564155     3.600000   303.841000  20386.467000     1.550000
max       0.819841    14.700000   303.841000  20386.467000     5.080000

             xrate        t5yie       t10yie
count  4264.000000  4264.000000  4264.000000
mean      1.174237     1.869780     2.007709
std       0.097489     0.495264     0.368576
min       0.961600     0.140000     0.500000
25%       1.103900     1.560000     1.730000
50%       1.142650     1.800000     2.070000
75%       1.239000     2.110000     2.262500
max       1.392700     3.590000     3.020000
```

The Augmented Dicky Fuller Test

```
from statsmodels.tsa.stattools import adfuller

print("Unrate: ",adfuller(df['unrate']))
print("GDP: ",adfuller(df['gdp']))
print("Consum: ",adfuller(df['consum']))
print("Xrate: ", adfuller(df['xrate']))
print("t5yie: ", adfuller(df['t5yie']))
print("t10yie: ", adfuller(df['t10yie']))
print("Embedding: ", adfuller(df['embeddings']))
```

```
Unrate:  (-11.362977947855217, 9.374279390000756e-21, 30, 4233, {'1%': -3.4318957760316944, '5%': -2.8622230386181973, '10%': -2.56
GDP:  (-12.92260032192169, 3.84644000048785475e-24, 31, 4232, {'1%': -3.4318961415125915, '5%': -2.8622232000728345, '10%': -2.56713
Consum:  (-8.0647541330645, 1.6074142528009642e-12, 30, 4233, {'1%': -3.4318957760316944, '5%': -2.8622230386181973, '10%': -2.5671
Xrate:  (-2.2068201095100597, 0.20377946923540863, 28, 4235, {'1%': -3.4318950455880155, '5%': -2.8622227159377456, '10%': -2.56713
t5yie:  (-2.134413579436135, 0.23090370559183382, 29, 4234, {'1%': -3.431895410723543, '5%': -2.862222877239852, '10%': -2.56713350
t10yie:  (-2.1627410639404094, 0.2200450340846105, 29, 4234, {'1%': -3.431895410723543, '5%': -2.862222877239852, '10%': -2.5671335(
Embedding:  (-17.147011473658203, 6.983134532060371e-30, 2, 4261, {'1%': -3.4318856122567074, '5%': -2.8622185486666925, '10%': -2.
```

Make Data Stationary

```
df['xrate'] = df['xrate'].diff()
```

```
df['t5yie'] = df['t5yie'].diff()
df['t10yie'] = df['t10yie'].diff()
```

```
df.fillna(method="backfill", axis=None, inplace = True)
```

```
from statsmodels.tsa.stattools import adfuller

print("Unrate: ",adfuller(df['unrate']))
print("Xrate: ", adfuller(df['xrate']))
print("t5yie: ", adfuller(df['t5yie']))
print("t10yie: ", adfuller(df['t10yie']))
print("Embedding: ", adfuller(df['embeddings']))
```

```
Unrate:  (-11.362977947855217, 9.374279390000756e-21, 30, 4233, {'1%': -3.4318957760316944, '5%': -2.8622230386181973, '10%': -2.56
Xrate:  (-15.41994185384844, 3.044002146997704e-28, 27, 4236, {'1%': -3.4318946806249886, '5%': -2.8622225547118227, '10%': -2.5671
```

```
t5yie:  (-15.872520694251739, 8.94764353957158e-29, 28, 4235, {'1%': -3.4318950455880155, '5%': -2.8622227159377456, '10%': -2.5671
t10yie:  (-15.86251597291833, 9.176963496052847e-29, 28, 4235, {'1%': -3.4318950455880155, '5%': -2.8622227159377456, '10%': -2.567
Embedding:  (-17.147011473658203, 6.983134532060371e-30, 2, 4261, {'1%': -3.4318856122567074, '5%': -2.8622185486666925, '10%': -2.
```

```
df.head()
```

date	embeddings	unrate	consum	gdp	frate	xrate	t5yie	t10yie
2012-01-01	0.518817	8.3	227.842	16179.968	0.04	0.0000	0.00	0.00
2012-02-01	0.518817	8.3	228.329	16179.968	0.11	0.0000	0.00	0.00
2012-03-01	0.518817	8.2	228.807	16179.968	0.11	0.0141	0.10	0.11
2012-04-01	0.518817	8.2	229.187	16253.726	0.09	0.0000	0.00	0.00
2012-05-01	0.518817	8.2	228.713	16253.726	0.16	-0.0094	0.07	0.00

```
df.tail()
```

date	embeddings	unrate	consum	gdp	frate	xrate	t5yie	t10yie
2023-06-10	0.414018	3.6	303.841	20386.467	5.08	0.0000	0.00	0.00
2023-06-11	0.414018	3.6	303.841	20386.467	5.08	0.0000	0.00	0.00
2023-06-12	0.414018	3.6	303.841	20386.467	5.08	-0.0002	-0.04	-0.03
2023-06-13	0.414018	3.6	303.841	20386.467	5.08	0.0045	0.03	0.03
2023-06-14	0.414018	3.6	303.841	20386.467	5.08	0.0067	0.03	0.01

Define Functions

```
def split_train_test(X,y):
  l = int(0.1 *X.shape[0])
  X_train = X.iloc[:len(X)-l]
  X_test = X.iloc[len(X)-l:]
  y_train = y.iloc[:len(y)-l]
  y_test = y.iloc[len(y)-l:]
  y_test = y_test[:-1]
  X_test = X_test[:-1]
  return X_train, y_train, X_test, y_test
```

```
def get_xy_without(df):
  X = df[['unrate','gdp','consum','xrate','t5yie','t10yie']]
  y = df[['frate']]
  return X,y
```

```
def get_xy_with(df):
  X = df[['unrate','gdp','consum','xrate','t5yie','t10yie','embeddings']]
  y = df[['frate']]
  return X,y
```

Multiple Linear Regression - Without Sentiment

```
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split, cross_val_score
```

```
print(df.corr())
```

```
                embeddings    unrate    consum       gdp     frate     xrate  \
embeddings    1.000000  0.080517 -0.066363 -0.068703 -0.029198  0.012244
unrate        0.080517  1.000000 -0.753875 -0.819098 -0.076840 -0.003754
consum       -0.066363 -0.753875  1.000000  0.975820  0.043520  0.036565
gdp          -0.068703 -0.819098  0.975820  1.000000  0.056158  0.032377
frate        -0.029198 -0.076840  0.043520  0.056158  1.000000  0.011306
xrate         0.012244 -0.003754  0.036565  0.032377  0.011306  1.000000
t5yie        -0.002933 -0.020230 -0.011378  0.005775 -0.036383  0.035582
t10yie        0.002937 -0.013268 -0.000482  0.011359 -0.030785  0.081507

                t5yie    t10yie
embeddings -0.002933  0.002937
unrate     -0.020230 -0.013268
consum     -0.011378 -0.000482
gdp         0.005775  0.011359
frate      -0.036383 -0.030785
xrate       0.035582  0.081507
t5yie       1.000000  0.848830
t10yie      0.848830  1.000000
```

```
print(df.describe())
```

```
        embeddings       unrate       consum           gdp        frate  \
count  4264.000000  4264.000000  4264.000000   4264.000000  4264.000000
mean      0.521629     3.660272   302.141797  20313.495404     0.892704
std       0.076378     0.479793     9.917226    455.653578     1.188383
min       0.156456     3.400000   227.842000  16179.968000     0.040000
25%       0.478198     3.600000   303.841000  20386.467000     0.090000
50%       0.520167     3.600000   303.841000  20386.467000     0.160000
75%       0.564155     3.600000   303.841000  20386.467000     1.550000
max       0.819841    14.700000   303.841000  20386.467000     5.080000

             xrate        t5yie       t10yie
count  4264.000000  4264.000000  4264.000000
mean     -0.000054     0.000061     0.000014
std       0.007873     0.045837     0.034404
min      -0.172200    -0.940000    -0.680000
25%      -0.001600    -0.010000    -0.010000
50%       0.000000     0.000000     0.000000
75%       0.001500     0.010000     0.010000
max       0.230900     0.480000     0.390000
```

```
X,y = get_xy_without(df)
```

```
X_train, y_train, X_test, y_test = split_train_test(X,y)
```

```
X_train.shape
```

```
(3838, 6)
```

```
X_test.shape
```

```
(425, 6)
```

```
#distribution of diff
import seaborn as sns
sns.distplot(y)
```

```
<ipython-input-29-667cf3724e2c>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y)
<Axes: ylabel='Density'>
```

```
X_train
```

date	unrate	gdp	consum	xrate	t5yie	t10yie
2012-01-01	8.3	16179.968	227.842	0.0000	0.00	0.00
2012-02-01	8.3	16179.968	228.329	0.0000	0.00	0.00
2012-03-01	8.2	16179.968	228.807	0.0141	0.10	0.11
2012-04-01	8.2	16253.726	229.187	0.0000	0.00	0.00
2012-05-01	8.2	16253.726	228.713	-0.0094	0.07	0.00
...	...	...	...	...	...	...
2022-04-10	3.6	20386.467	303.841	0.0000	0.00	0.00
2022-04-11	3.6	20386.467	303.841	0.0001	0.04	0.04
2022-04-12	3.6	20386.467	303.841	-0.0023	-0.06	-0.05
2022-04-13	3.6	20386.467	303.841	0.0022	-0.06	-0.06
2022-04-14	3.6	20386.467	303.841	-0.0078	0.08	0.09

3838 rows × 6 columns

```
reg_model = linear_model.LinearRegression()

reg_model = LinearRegression().fit(X_train, y_train)

print("Intercept: ", reg_model.intercept_)

list(zip(X, reg_model.coef_))
```

```
Intercept:  [-1.13179913]
[('unrate',
  array([-1.58188455e-01,  5.76893018e-04, -3.10378232e-02,  1.04525323e+00,
         -9.06294076e-01, -1.05217126e-01]))]
```

```
y_pred = reg_model.predict(X_test)
print("Prediction for test set: {}".format(y_pred))
```

```
[0.57673556]
[0.62227315]
[0.60522329]
[0.62896969]
[0.62896969]
[0.6366738 ]
[0.59601814]
[0.69210362]
[0.64364624]
[0.62075679]
[0.62896969]
[0.62896969]
[0.62896969]
[0.64002553]
[0.67138536]
[0.62896969]
[0.58190882]
[0.62896969]
[0.62896969]
[0.60947806]
[0.6342628 ]
[0.61958626]
[0.64723778]
[0.63479235]
[0.62896969]
[0.62896969]
[0.66816892]
[0.603328  ]]
```

```
y_test['pred'] = y_pred
```

```
y_test
```

date	frate	pred
2022-04-15	0.33	0.630642
2022-04-16	0.33	0.628970
2022-04-17	0.33	0.628970
2022-04-18	0.33	0.595802
2022-04-19	0.33	0.628336
...	...	...
2023-06-09	5.08	0.634792
2023-06-10	5.08	0.628970
2023-06-11	5.08	0.628970
2023-06-12	5.08	0.668169
2023-06-13	5.08	0.603328

425 rows × 2 columns

```
mae = metrics.mean_absolute_error(y_test['frate'], y_test['pred'])
mse = metrics.mean_squared_error(y_test['frate'], y_test['pred'])
r2 = np.sqrt(metrics.mean_squared_error(y_test['frate'], y_test['pred']))

print('Mean Absolute Error:', mae)
print('Mean Square Error:', mse)
print('Root Mean Square Error:', r2)
```

```
Mean Absolute Error: 2.6526253319152526
Mean Square Error: 9.17751125142895
Root Mean Square Error: 3.0294407489549866
```

```
y_test
```

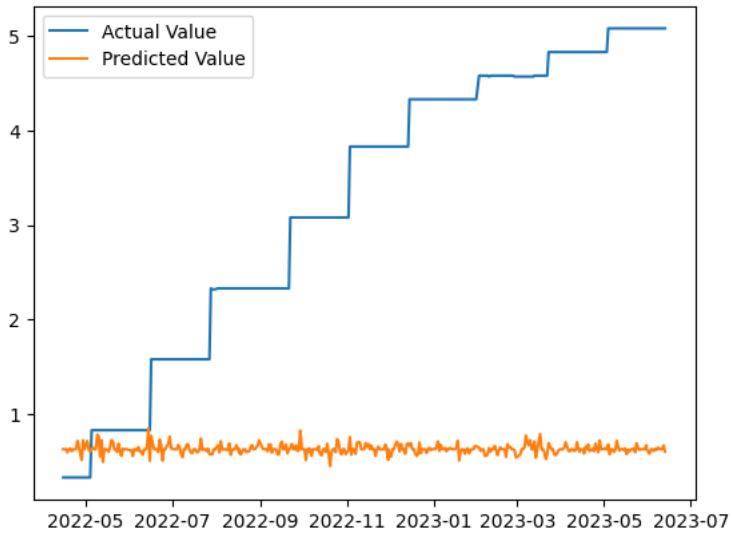
date	frate	pred
2022-04-15	0.33	0.630642
2022-04-16	0.33	0.628970
2022-04-17	0.33	0.628970
2022-04-18	0.33	0.595802
2022-04-19	0.33	0.628336

```
reg_model_diff = pd.DataFrame({'Actual value': y_test['frate'], 'Predicted value':
reg_model_diff
```

date	Actual value	Predicted value
2022-04-15	0.33	0.630642
2022-04-16	0.33	0.628970
2022-04-17	0.33	0.628970
2022-04-18	0.33	0.595802
2022-04-19	0.33	0.628336
...	...	...
2023-06-09	5.08	0.634792
2023-06-10	5.08	0.628970
2023-06-11	5.08	0.628970
2023-06-12	5.08	0.668169
2023-06-13	5.08	0.603328

425 rows × 2 columns

```
plt.plot(y_test['frate'], label="Actual Value")
plt.plot(y_test['pred'], label = "Predicted Value")
plt.legend(loc = 'upper left')
plt.show()
```



Multiple Linear Regression - With Sentiment

```
df
```

date	embeddings	unrate	consum	gdp	frate	xrate	t5yie	t10yie
2012-01-01	0.518817	8.3	227.842	16179.968	0.04	0.0000	0.00	0.00
2012-02-01	0.518817	8.3	228.329	16179.968	0.11	0.0000	0.00	0.00
2012-03-01	0.518817	8.2	228.807	16179.968	0.11	0.0141	0.10	0.11
2012-04-01	0.518817	8.2	229.187	16253.726	0.09	0.0000	0.00	0.00
2012-05-01	0.518817	8.2	228.713	16253.726	0.16	-0.0094	0.07	0.00
...	...	...	...	...	...	...	...	...
2023-06-10	0.414018	3.6	303.841	20386.467	5.08	0.0000	0.00	0.00
2023-06-11	0.414018	3.6	303.841	20386.467	5.08	0.0000	0.00	0.00
2023-06-12	0.414018	3.6	303.841	20386.467	5.08	-0.0002	-0.04	-0.03
2023-06-13	0.414018	3.6	303.841	20386.467	5.08	0.0045	0.03	0.03
2023-06-14	0.414018	3.6	303.841	20386.467	5.08	0.0067	0.03	0.01

4264 rows × 8 columns

```
X,y = get_xy_with(df)

X_train, y_train, X_test, y_test = split_train_test(X,y)

reg_model = linear_model.LinearRegression()

reg_model = LinearRegression().fit(X_train, y_train)

print("Intercept: ", reg_model.intercept_)

list(zip(X, reg_model.coef_))
```

```
Intercept:  [-0.84290095]
[('unrate',
  array([-1.49191787e-01,  5.84211178e-04, -3.13957066e-02,  1.11938449e+00,
         -9.31734362e-01, -7.32088682e-02, -6.94589375e-01]))]
```

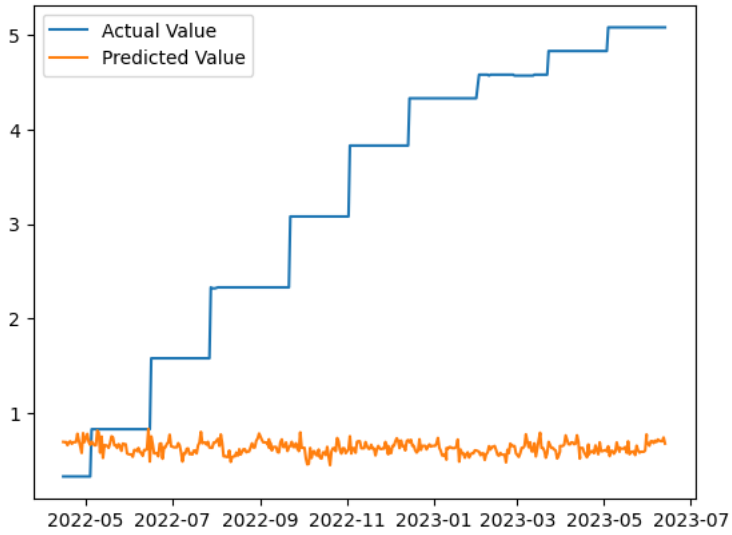
```
y_pred = reg_model.predict(X_test)
print("Prediction for test set: {}".format(y_pred))
```

```
[0.66646161]
[0.58226386]
[0.59146029]
[0.59146029]
[0.59146029]
[0.60251717]
[0.7749257 ]
[0.70313483]
[0.65506646]
[0.70313483]
[0.70313483]
[0.68343223]
[0.70880968]
[0.69386896]
[0.72191546]
[0.70898208]
[0.70313483]
[0.70313483]
[0.74237659]
[0.67802376]]
```

```
reg_model_diff = y_test
reg_model_diff['pred'] =y_pred


mae = metrics.mean_absolute_error(reg_model_diff['frate'], reg_model_diff['pred'])
mse = metrics.mean_squared_error(reg_model_diff['frate'], reg_model_diff['pred'])
r2 = np.sqrt(metrics.mean_squared_error(reg_model_diff['frate'], reg_model_diff['pred']))

print('Mean Absolute Error:', mae)
print('Mean Square Error:', mse)
print('Root Mean Square Error:', r2)
```

```
Mean Absolute Error: 2.6581809144099786
Mean Square Error: 9.202956094785208
Root Mean Square Error: 3.0336374362776457
```

```
plt.plot(reg_model_diff['frate'], label="Actual Value")
plt.plot(reg_model_diff['pred'], label = "Predicted Value")
plt.legend(loc = 'upper left')
plt.show()
```



FB Prophet - without embeddings

```
df.drop(columns = ['level_0', 'index'], inplace = True)
```

```
df
```

	date	embeddings	unrate	consum	gdp	frate	xrate	t5yie	t10yie
0	2012-01-01	0.518817	8.3	227.842	16179.968	0.04	0.0000	0.00	0.00
1	2012-02-01	0.518817	8.3	228.329	16179.968	0.11	0.0000	0.00	0.00
2	2012-03-01	0.518817	8.2	228.807	16179.968	0.11	0.0141	0.10	0.11
3	2012-04-01	0.518817	8.2	229.187	16253.726	0.09	0.0000	0.00	0.00
4	2012-05-01	0.518817	8.2	228.713	16253.726	0.16	-0.0094	0.07	0.00
...	...	...	...	...	...	...	...	...	...
4259	2023-06-10	0.414018	3.6	303.841	20386.467	5.08	0.0000	0.00	0.00
4260	2023-06-11	0.414018	3.6	303.841	20386.467	5.08	0.0000	0.00	0.00
4261	2023-06-12	0.414018	3.6	303.841	20386.467	5.08	-0.0002	-0.04	-0.03
4262	2023-06-13	0.414018	3.6	303.841	20386.467	5.08	0.0045	0.03	0.03
4263	2023-06-14	0.414018	3.6	303.841	20386.467	5.08	0.0067	0.03	0.01

4264 rows × 9 columns

```
from prophet.plot import plot_plotly, plot_components_plotly
from prophet import Prophet


fb_data = df.reset_index(inplace = True)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-139-44ce9b53660d> in <cell line: 1>()
----> 1 fb_data = df.reset_index(inplace = True)

                                 2 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in insert(self, loc, column, value,
allow_duplicates)
   4815         if not allow_duplicates and column in self.columns:
   4816             # Should this be a different kind of error??
-> 4817             raise ValueError(f"cannot insert {column}, already exists")
   4818         if not isinstance(loc, int):
   4819             raise TypeError("loc must be int")

ValueError: cannot insert level_0, already exists
```

SEARCH STACK OVERFLOW

```
fb_data = df.rename(columns ={"date":"ds","frate":"y"})
l = int(0.1 *fb_data.shape[0])
train = fb_data.iloc[:len(fb_data)-l]
test = fb_data.iloc[len(fb_data)-l:]


m = Prophet()
m.add_regressor('unrate')
m.add_regressor('gdp')
m.add_regressor('consum')
#m.add_regressor('xrate')
m.add_regressor('t10yie')
m.fit(train)
future = m.make_future_dataframe(periods=86,freq='MS') #MS for monthly, H for hourly
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp5rwejppp/5intw210.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp5rwejppp/ufi3c7dl.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=7416
15:31:43 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
15:31:48 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

```
future.shape
```

```
(3857, 1)
```

```
n_df=train
n_df=n_df.append(test)
```

```
future=n_df
forecast = m.predict(future)
```

```
<ipython-input-262-74b86f31db5d>:2: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
```

```
n_df.shape
```

```
(4264, 9)
```

```
forecast
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lo
0	2012-01-01	0.113419	-0.156806	0.426440	0.113419	0.113419	0.017180	0.017
1	2012-01-02	0.113507	-0.109324	0.441589	0.113507	0.113507	0.046737	0.046
2	2012-01-03	0.113596	-0.038675	0.536486	0.113596	0.113596	0.130149	0.130
3	2012-01-04	0.113684	-0.175783	0.416345	0.113684	0.113684	0.010280	0.010
4	2012-01-05	0.113773	-0.135277	0.424918	0.113773	0.113773	0.032365	0.032
...	...	...	...	...	...	...	...	
4259	2023-06-10	0.457748	0.166013	0.759619	0.457748	0.457748	0.002311	0.002
4260	2023-06-11	0.458074	0.147858	0.743625	0.458074	0.458074	-0.002459	-0.002
4261	2023-06-12	0.458399	0.172211	0.805445	0.458399	0.458399	0.020697	0.020
4262	2023-06-13	0.458725	0.143789	0.712066	0.458725	0.458725	-0.015074	-0.015
4263	2023-06-14	0.459051	0.187433	0.758610	0.459051	0.459051	0.004607	0.004

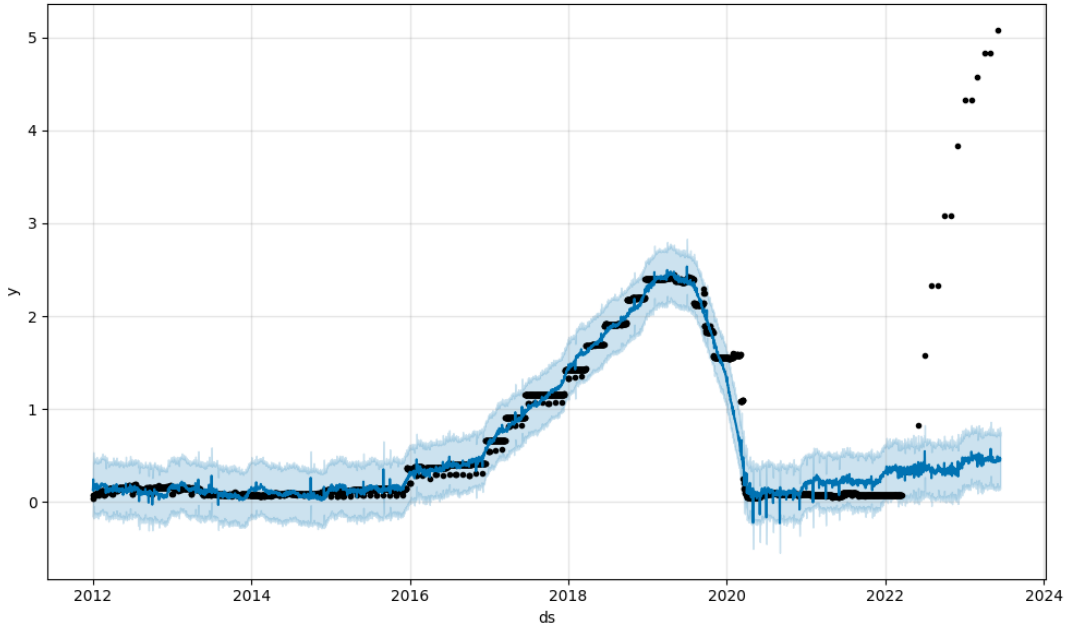
4264 rows × 34 columns

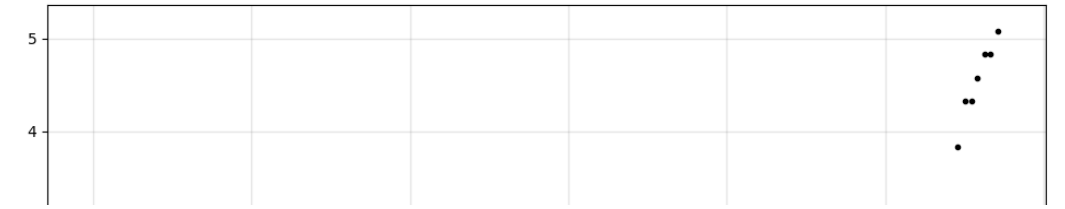
```
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']]
```

	ds	yhat	yhat_lower	yhat_upper
0	2012-01-01	0.130598	-0.156806	0.426440
1	2012-01-02	0.160245	-0.109324	0.441589
2	2012-01-03	0.243744	-0.038675	0.536486
3	2012-01-04	0.123965	-0.175783	0.416345
4	2012-01-05	0.146138	-0.135277	0.424918
...	...	...	...	...
4259	2023-06-10	0.460059	0.166013	0.759619
4260	2023-06-11	0.455614	0.147858	0.743625
4261	2023-06-12	0.479097	0.172211	0.805445
4262	2023-06-13	0.443651	0.143789	0.712066
4263	2023-06-14	0.463657	0.187433	0.758610

4264 rows × 4 columns

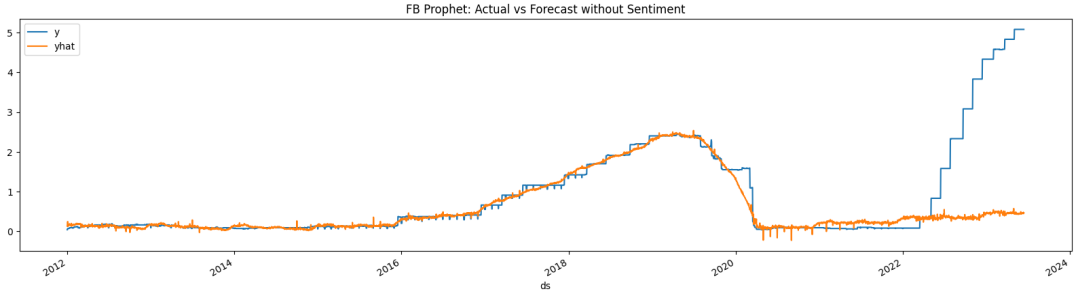
```
m.plot(forecast)
```



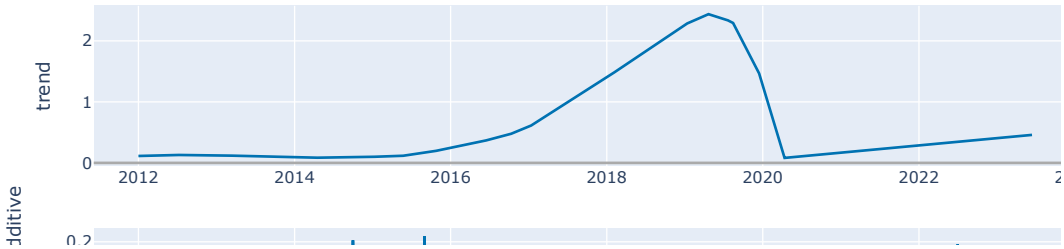



```
ax = (fb_data.plot(x='ds',y='y',figsize=(20,5),title='Actual Vs Forecast'))
forecast.plot(x='ds',y='yhat',figsize=(20,5),title='FB Prophet: Actual vs Forecast without Sentiment', ax=ax)
```

```
<Axes: title={'center': 'FB Prophet: Actual vs Forecast without Sentiment'}, xlabel='ds'>
```




```
plot_components_plotly(m, forecast)
```




```
from statsmodels.tools.eval_measures import rmse
predictions = forecast.iloc[-1:]['yhat']
print("Root Mean Squared Error between actual and  predicted values: ",rmse(predictions,test['y']))
print("Mean Value of Test Dataset:", test['y'].mean())
```

```
Root Mean Squared Error between actual and  predicted values:  3.2204820565034327
Mean Value of Test Dataset: 3.2615258215962437
```

Based on a rule of thumb, it can be said that RMSE values between 0.2 and 0.5 shows that the model can relatively predict the data accurately

```
def smape_kun(y_true, y_pred):
    mape = np.mean(abs((y_true-y_pred)/y_true))*100
    smape = np.mean((np.abs(y_pred - y_true) * 200/ (np.abs(y_pred) + np.abs(y_true))).fillna(0))
    print('MAPE: %.2f\nSMAPE: %.2f'% (mape,smape), "%")
```

```
import numpy as np

tt=test['y']

smape_kun(tt,predictions)
```

```
MAPE: 81.53
SMAPE: 143.60 %
```

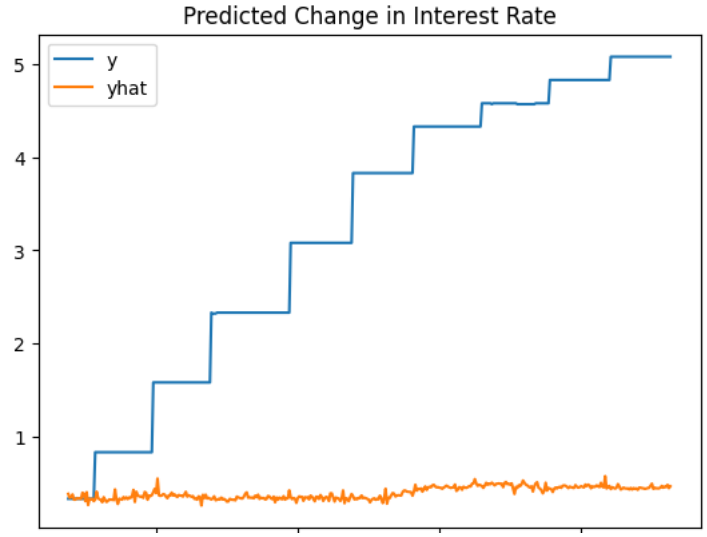
```
test
```

	ds	embeddings	unrate	consum	gdp	y	xrate	t5yie	t10yie
3838	2022-04-15	0.427774	3.6	303.841	20386.467	0.33	0.0016	0.00	0.00
3839	2022-04-16	0.427774	3.6	303.841	20386.467	0.33	0.0000	0.00	0.00
3840	2022-04-17	0.427774	3.6	303.841	20386.467	0.33	0.0000	0.00	0.00
3841	2022-04-18	0.427774	3.6	303.841	20386.467	0.33	-0.0027	0.03	0.03
3842	2022-04-19	0.427774	3.6	303.841	20386.467	0.33	0.0004	0.00	0.01
...	...	...	...	...	...	...	...	...	...
4259	2023-06-10	0.414018	3.6	303.841	20386.467	5.08	0.0000	0.00	0.00
4260	2023-06-11	0.414018	3.6	303.841	20386.467	5.08	0.0000	0.00	0.00
4261	2023-06-12	0.414018	3.6	303.841	20386.467	5.08	-0.0002	-0.04	-0.03
4262	2023-06-13	0.414018	3.6	303.841	20386.467	5.08	0.0045	0.03	0.03
4263	2023-06-14	0.414018	3.6	303.841	20386.467	5.08	0.0067	0.03	0.01

426 rows × 9 columns

```
tt.plot(legend=True)
predictions.plot(title='Predicted Change in Interest Rate',legend=True)
```

```
<Axes: title={'center': 'Predicted Change in Interest Rate'}>
```




```
predictions
```

```
3838    0.382085
3839    0.339494
3840    0.345585
3841    0.376063
3842    0.320670
          ...
4259    0.460059
4260    0.455614
4261    0.479097
4262    0.443651
4263    0.463657
Name: yhat, Length: 426, dtype: float64
```

```
tt
```

```
3838    0.33
3839    0.33
3840    0.33
3841    0.33
3842    0.33
        ...
4259    5.08
4260    5.08
4261    5.08
4262    5.08
4263    5.08
Name: y, Length: 426, dtype: float64
```

FB Prophet - with Embeddings

```
train = fb_data.iloc[:len(fb_data)-l]
test = fb_data.iloc[len(fb_data)-l:]
```

```
m = Prophet()
m.add_regressor('unrate')
m.add_regressor('gdp')
m.add_regressor('consum')
#m.add_regressor('xrate')
m.add_regressor('embeddings')
m.add_regressor('t10yie')
m.fit(train)
future = m.make_future_dataframe(periods=4,freq='MS') #MS for monthly, H for hourly
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp5rwejppp/kk_elwb3.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp5rwejppp/6qsospbg.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=9588
15:36:37 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
15:36:39 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

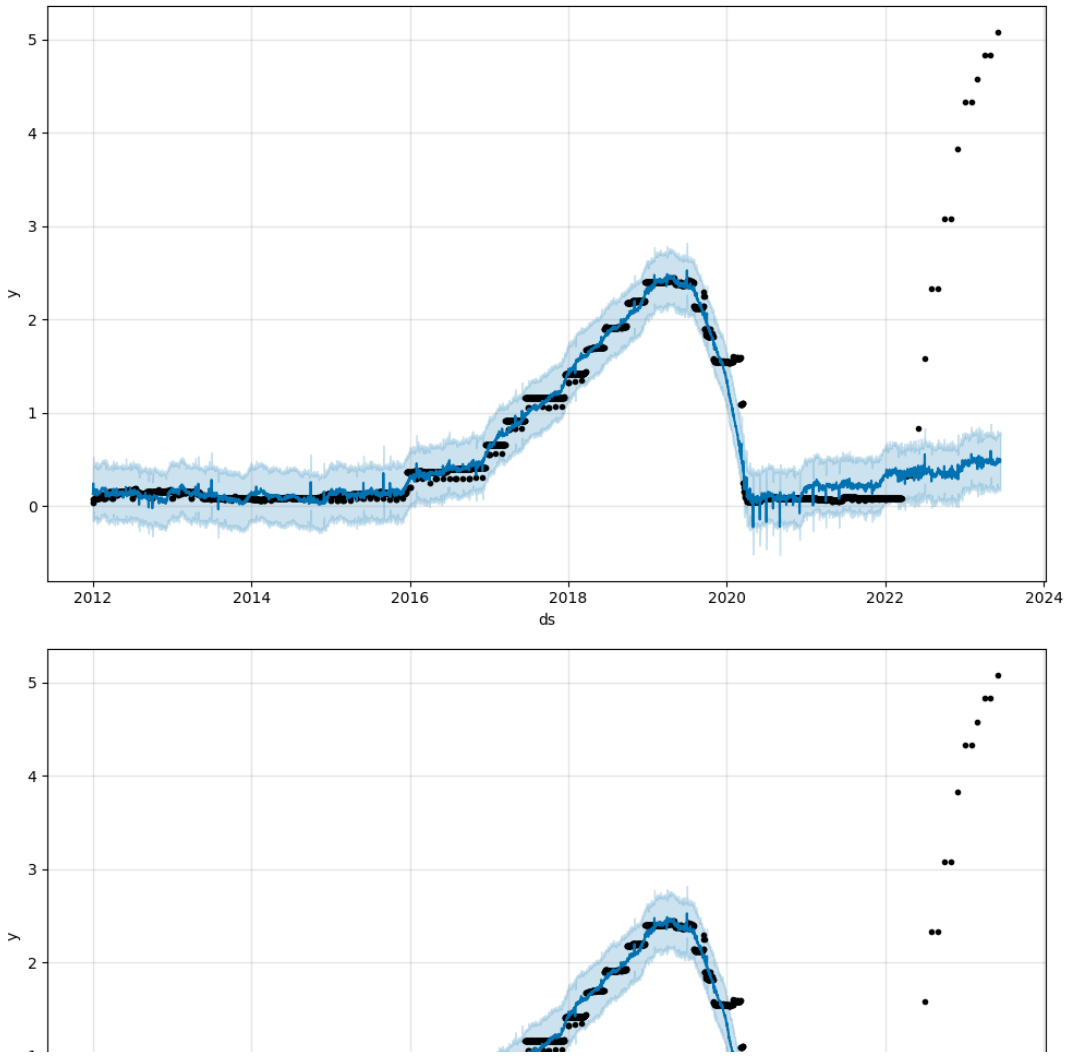
```
n_df=train
n_df=n_df.append(test)

future=n_df
forecast = m.predict(future)
```

```
<ipython-input-277-74b86f31db5d>:2: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
```

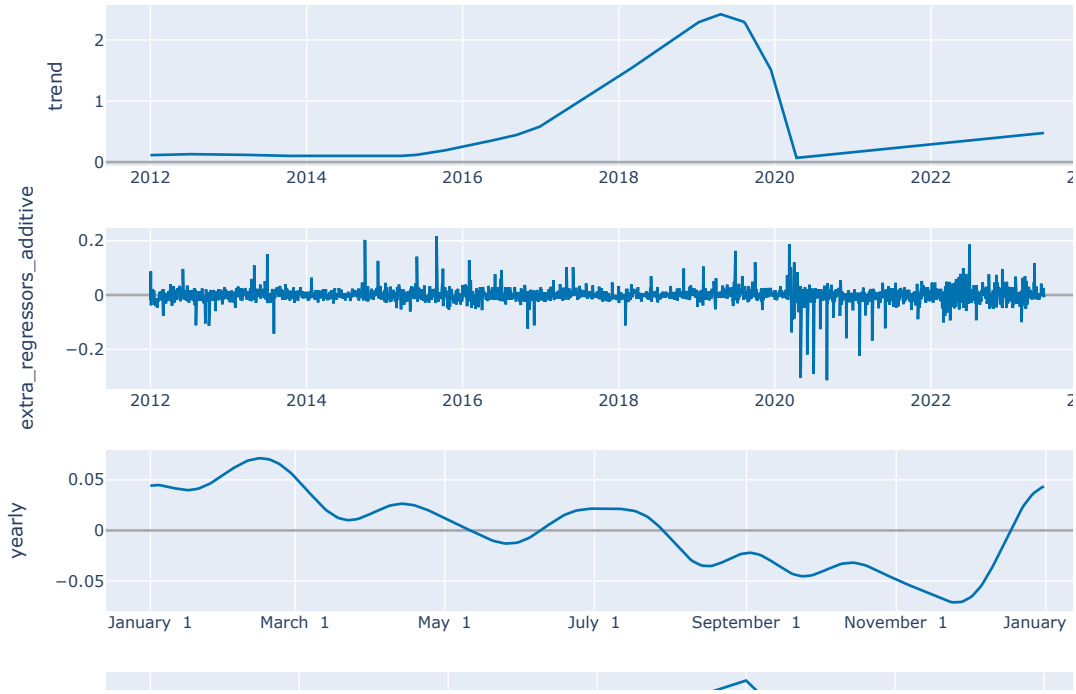
```
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

	ds	yhat	yhat_lower	yhat_upper
4259	2023-06-10	0.487633	0.194241	0.788553
4260	2023-06-11	0.483197	0.184112	0.774354
4261	2023-06-12	0.506521	0.204475	0.792925
4262	2023-06-13	0.471080	0.167909	0.769208
4263	2023-06-14	0.491202	0.214862	0.792398

```
m.plot(forecast)
```

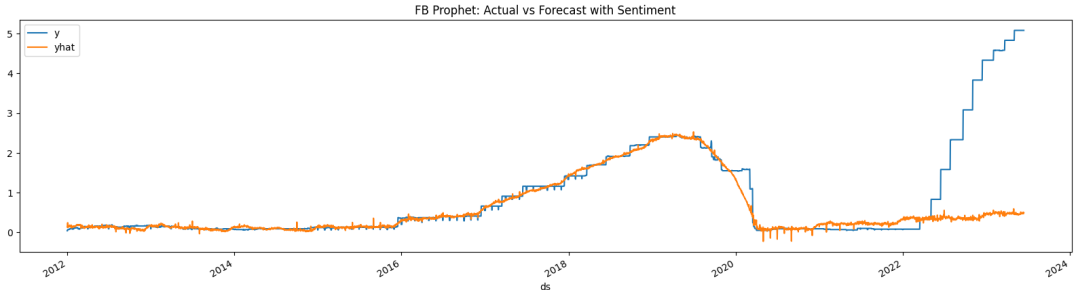



```
plot_components_plotly(m, forecast)
```




```
ax = (fb_data.plot(x='ds',y='y',figsize=(20,5),title='Actual Vs Forecast'))
forecast.plot(x='ds',y='yhat',figsize=(20,5),title='FB Prophet: Actual vs Forecast with Sentiment', ax=ax)
```

```
<Axes: title={'center': 'FB Prophet: Actual vs Forecast with Sentiment'}, xlabel='ds'>
```




```
from statsmodels.tools.eval_measures import rmse
predictions = forecast.iloc[-1:]['yhat']
print("Root Mean Squared Error between actual and  predicted values: ",rmse(predictions,test['y']))
print("Mean Value of Test Dataset:", test['y'].mean())
```

```
Root Mean Squared Error between actual and  predicted values:  3.2087084491919353
Mean Value of Test Dataset: 3.2615258215962437
```

```
tt=test['y']
```

```
smape_kun(tt,predictions)
```

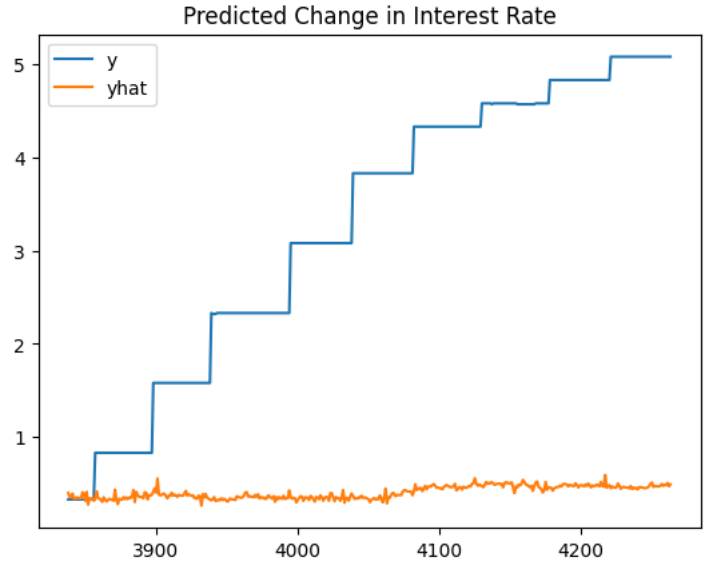
```
MAPE: 81.20
SMAPE: 142.50 %
```

```
predictions
```

```
3838    0.398510
3839    0.356271
3840    0.352029
3841    0.392102
3842    0.336680
          ...
4259    0.487633
4260    0.483197
4261    0.506521
4262    0.471080
4263    0.491202
Name: yhat, Length: 426, dtype: float64
```

```
tt.plot(legend=True)
predictions.plot(title='Predicted Change in Interest Rate',legend=True)
```

```
<Axes: title={'center': 'Predicted Change in Interest Rate'}>
```



Random ForestRegression without embedding

```
from sklearn.model_selection import train_test_split # for splitting the data
from sklearn.metrics import mean_squared_error # for calculating the cost function
from sklearn.ensemble import RandomForestRegressor # for building the model
```

```
X = df.drop(['frate','embeddings'], axis = 1)
y = df['frate']
```

```
X_train, y_train,X_test, y_test = split_train_test(X,y)
```

```
X_train.set_index("date",inplace = True)
#y_train.set_index("date",inplace = True)
X_test.set_index("date",inplace = True)
#y_test.set_index("date",inplace = True)
```

```
type(y_train)
```

```
pandas.core.series.Series
```

```
model = RandomForestRegressor(n_estimators = 10, random_state = 0)
model.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(n_estimators=10, random_state=0)
```

```
X_test
```

	unrate	consum	gdp	xrate	t10yie

```
y_pred = model.predict(X_test)

rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse)
```

```
RMSE:  3.1
```

```
def smape_kun(y_true, y_pred):
    mape = np.mean(abs((y_true-y_pred)/y_true))*100
    smape = np.mean((np.abs(y_pred - y_true) * 200/ (np.abs(y_pred) + np.abs(y_true))))
    print('MAPE: %.2f\nSMAPE: %.2f'% (mape,smape), "%")

smape_kun(y_test.values, y_pred)
```

```
MAPE: 80.79
SMAPE: 131.28 %
```

2023-06-13	3.6	303.841	20386.467	0.0045	0.03

Random Forest Regression with embedding

```
X = df.drop('frate', axis = 1)
y = df['frate']

X_train, y_train,X_test, y_test = split_train_test(X,y)

X_train.set_index("date",inplace = True)
#y_train.set_index("date",inplace = True)
X_test.set_index("date",inplace = True)
#y_test.set_index("date",inplace = True)

model = RandomForestRegressor(n_estimators = 10, random_state = 0)
model.fit(X_train, y_train)
```

```
RandomForestRegressor(n_estimators=10, random_state=0)
```

```
X_test
```

date	embeddings	unrate	consum	gdp	xrate	t10yie
2022-04-15	0.427774	3.6	303.841	20386.467	0.0016	0.00
2022-04-16	0.427774	3.6	303.841	20386.467	0.0000	0.00
2022-04-17	0.427774	3.6	303.841	20386.467	0.0000	0.00
2022-04-18	0.427774	3.6	303.841	20386.467	-0.0027	0.03
2022-04-19	0.427774	3.6	303.841	20386.467	0.0004	0.01
...	...	...	...	...	...	...
2023-06-09	0.414018	3.6	303.841	20386.467	-0.0031	0.00
2023-06-10	0.414018	3.6	303.841	20386.467	0.0000	0.00
2023-06-11	0.414018	3.6	303.841	20386.467	0.0000	0.00
2023-06-12	0.414018	3.6	303.841	20386.467	-0.0002	-0.03
2023-06-13	0.414018	3.6	303.841	20386.467	0.0045	0.03

425 rows × 6 columns

```
y_pred = model.predict(X_test)

rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse)
```

```
RMSE:  3.256
```

```
def smape_kun(y_true, y_pred):
    mape = np.mean(abs((y_true-y_pred)/y_true))*100
```

```
    smape = np.mean((np.abs(y_pred - y_true) * 200/ (np.abs(y_pred) + np.abs(y_true))))
    print('MAPE: %.2f\nSMAPE: %.2f'% (mape,smape), "%")

smape_kun(y_test.values, y_pred)
```

```
MAPE: 80.03
SMAPE: 141.56 %
```

XG Boost without Sentiment

```
import xgboost as xgb
from sklearn.metrics import mean_squared_error

regressor = xgb.XGBRegressor(
    n_estimators=100,
    reg_lambda=1,
    gamma=0,
    max_depth=3
)

X = df.drop(['frate','embeddings'], axis = 1)
y = df['frate']

X_train, y_train, X_test, y_test = split_train_test(X,y)

X_train.set_index("date",inplace = True)
#y_train.set_index("date",inplace = True)
X_test.set_index("date",inplace = True)
#y_test.set_index("date",inplace = True)

X_train
```

date	unrate	consum	gdp	xrate	t5yie	t10yie
2012-01-01	8.3	227.842	16179.968	0.0000	0.00	0.00
2012-02-01	8.3	228.329	16179.968	0.0000	0.00	0.00
2012-03-01	8.2	228.807	16179.968	0.0141	0.10	0.11
2012-04-01	8.2	229.187	16253.726	0.0000	0.00	0.00
2012-05-01	8.2	228.713	16253.726	-0.0094	0.07	0.00
...	...	...	...	...	...	...
2022-04-10	3.6	303.841	20386.467	0.0000	0.00	0.00
2022-04-11	3.6	303.841	20386.467	0.0001	0.04	0.04
2022-04-12	3.6	303.841	20386.467	-0.0023	-0.06	-0.05
2022-04-13	3.6	303.841	20386.467	0.0022	-0.06	-0.06
2022-04-14	3.6	303.841	20386.467	-0.0078	0.08	0.09

3838 rows × 6 columns

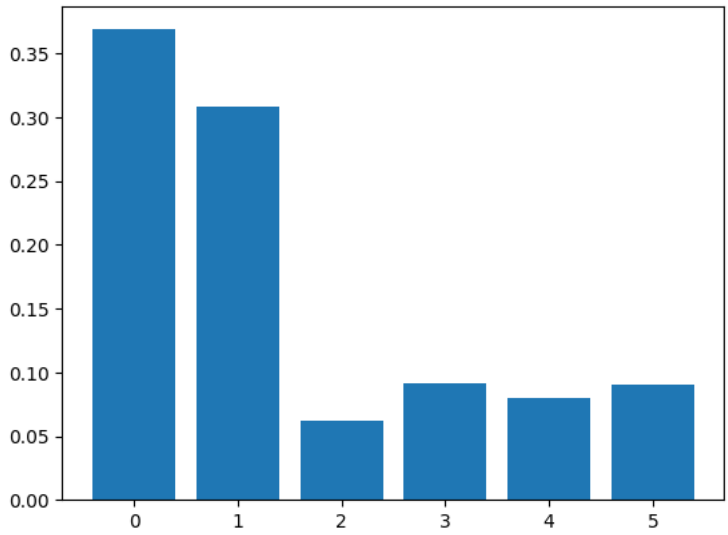
```
regressor.fit(X_train, y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

```
pd.DataFrame(regressor.feature_importances_.reshape(1, -1), columns=X_train.columns)
```

	unrate	consum	gdp	xrate	t5yie	t10yie
0	0.36855	0.307924	0.061747	0.091113	0.079833	0.090835

```
from matplotlib import pyplot

pyplot.bar(range(len(regressor.feature_importances_)), regressor.feature_importances_)
pyplot.show()
```



Consum & Unemployment is the greatest predictor

```
y_pred = regressor.predict(X_test)
```

```
print(mean_squared_error(y_test, y_pred))
rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse)
```

```
9.479747725514049

RMSE:  3.079
```

```
def smape_kun(y_true, y_pred):
    mape = np.mean(abs((y_true-y_pred)/y_true))*100
    smape = np.mean((np.abs(y_pred - y_true) * 200/ (np.abs(y_pred) + np.abs(y_true))))
    print('MAPE: %.2f\nSMAPE: %.2f'% (mape,smape), "%")

smape_kun(y_test.values,y_pred)
```

```
MAPE: 76.89
SMAPE: 127.75 %
```

XG Boost with Sentiment

```
regressor = xgb.XGBRegressor(
    n_estimators=100,
    reg_lambda=1,
    gamma=0,
    max_depth=3
)
```

```
X = df.drop(['frate'], axis = 1)
y = df['frate']
```

```
X_train, y_train, X_test, y_test = split_train_test(X,y)
```

```
X_train.set_index("date",inplace = True)
#y_train.set_index("date",inplace = True)
X_test.set_index("date",inplace = True)
#y_test.set_index("date",inplace = True)
```

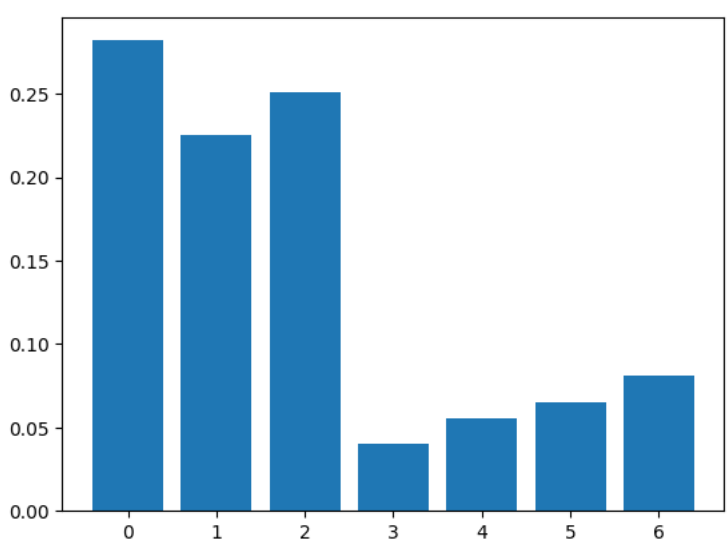
```
regressor.fit(X_train, y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

```
pd.DataFrame(regressor.feature_importances_.reshape(1, -1), columns=X_train.columns)
```

	embeddings	unrate	consum	gdp	xrate	t5yie	t10yie
0	0.281925	0.225174	0.251265	0.040146	0.055219	0.065425	0.080845

```
from matplotlib import pyplot

pyplot.bar(range(len(regressor.feature_importances_)), regressor.feature_importance
pyplot.show()
```




```
y_pred = regressor.predict(X_test)
```

```
print(mean_squared_error(y_test, y_pred))
rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse)
```

```
9.818174294659492

RMSE:  3.133
```

```
smape_kun(y_test.values,y_pred)
```

```
MAPE: 76.75
SMAPE: 128.14 %
```

Support Vector Regression - without Embedding

```
from sklearn.linear_model import LinearRegression # for building a linear regression model
from sklearn.svm import SVR # for building SVR model
from sklearn.preprocessing import MinMaxScaler

# Visualizations
import plotly.graph_objects as go # for data visualization
import plotly.express as px # for data visualization
```

```
df.head()
```

	date	embeddings	unrate	frate	xrate	t10yie
0	2012-01-01	0.518817	8.3	0.04	0.0000	0.00
1	2012-02-01	0.518817	8.3	0.11	0.0000	0.00
2	2012-03-01	0.518817	8.2	0.11	0.0141	0.11
3	2012-04-01	0.518817	8.2	0.09	0.0000	0.00
4	2012-05-01	0.518817	8.2	0.16	-0.0094	0.00

```
scaler=MinMaxScaler()

df['unrate(scaled)']=scaler.fit_transform(df[['unrate']])
df['xrate(scaled)']=scaler.fit_transform(df[['xrate']])
df['t10yie(scaled)']=scaler.fit_transform(df[['t10yie']])
df['embeddings(scaled)']=scaler.fit_transform(df[['embeddings']])

df
```

	date	embeddings	unrate	frate	xrate	t10yie	unrate(scaled)	xrate(scaled)	t10yie(scaled)	emb
0	2012-01-01	0.518817	8.3	0.04	0.0000	0.00	0.433628	0.427189	0.635514	
1	2012-02-01	0.518817	8.3	0.11	0.0000	0.00	0.433628	0.427189	0.635514	
2	2012-03-01	0.518817	8.2	0.11	0.0141	0.11	0.424779	0.462168	0.738318	
3	2012-04-01	0.518817	8.2	0.09	0.0000	0.00	0.424779	0.427189	0.635514	
4	2012-05-01	0.518817	8.2	0.16	-0.0094	0.00	0.424779	0.403870	0.635514	
...	...	...	...	...	...	...	...	...	...	
4259	2023-06-10	0.414018	3.6	5.08	0.0000	0.00	0.017699	0.427189	0.635514	
4260	2023-06-11	0.414018	3.6	5.08	0.0000	0.00	0.017699	0.427189	0.635514	
4261	2023-06-12	0.414018	3.6	5.08	-0.0002	-0.03	0.017699	0.426693	0.607477	
4262	2023-06-13	0.414018	3.6	5.08	0.0045	0.03	0.017699	0.438353	0.663551	
4263	2023-06-14	0.414018	3.6	5.08	0.0067	0.01	0.017699	0.443810	0.644860	

4264 rows × 10 columns

```
X = df[['unrate(scaled)','xrate(scaled)','t10yie(scaled)']]
y = df['frate'].values


model1 = LinearRegression()
model2 = SVR(kernel='rbf', C=100, epsilon=1)

# Fit the two models
lr = model1.fit(X, y)
svr = model2.fit(X, y)

mesh_size = 0.05

# Identify min and max values for input variables
x_min, x_max = X['unrate(scaled)'].min(), X['unrate(scaled)'].max()
y_min, y_max = X['xrate(scaled)'].min(), X['xrate(scaled)'].max()
z_min, z_max = X['t10yie(scaled)'].min(), X['t10yie(scaled)'].max()


# Return evenly spaced values based on a range between min and max
xrange = np.arange(x_min, x_max, mesh_size)
yrange = np.arange(y_min, y_max, mesh_size)
zrange = np.arange(z_min, z_max, mesh_size)


# Create a meshgrid
xx, yy ,zz= np.meshgrid(xrange, yrange, zrange)
```

```
# Use models to create a prediciton plane --- Linear Regression
pred_LR = model1.predict(np.c_[xx.ravel(), yy.ravel(), zz.ravel()])
pred_LR = pred_LR.reshape(xx.shape)

# Use models to create a prediciton plane --- SVR
pred_svr = model2.predict(np.c_[xx.ravel(), yy.ravel(),zz.ravel()])
pred_svr = pred_svr.reshape(xx.shape)

# Note, .ravel() flattens the array to a 1D array,
# then np.c_ takes elements from flattened xx and yy arrays and puts them together,
# this creates the right shape required for model input

# prediction array that is created by the model output is a 1D array,
# Hence, we need to reshape it to be the same shape as xx or yy to be able to display it on a graph
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:

X does not have valid feature names, but LinearRegression was fitted with feature names

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:

X does not have valid feature names, but SVR was fitted with feature names
```

```
pred_svr
```

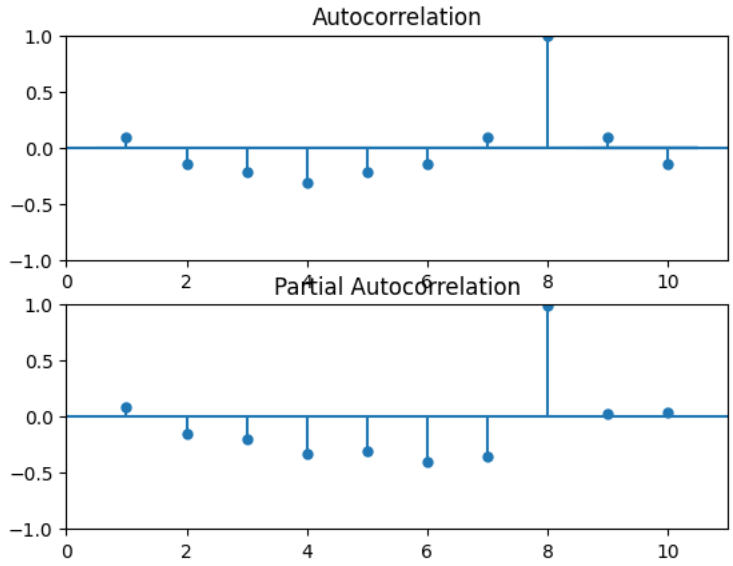
```
array([[[ 0.57430732,  0.71948905,  0.91468586, ..., -0.32590376,
         -0.56416784, -0.68732258],
        [ 0.43123373,  0.5618826 ,  0.74657873, ..., -0.42767007,
         -0.67275184, -0.79997202],
        [ 0.28673955,  0.39843858,  0.56692903, ..., -0.5040214 ,
         -0.74912325, -0.87670843],
        ...,
        [ 0.53443365,  0.5256191 ,  0.52135841, ...,  0.54614919,
          0.53652222,  0.5321126 ],
        [ 0.57257161,  0.56639167,  0.56332437, ...,  0.5815667 ,
          0.57508955,  0.5721157 ],
        [ 0.60087868,  0.59674599,  0.59468646, ...,  0.60729529,
          0.60299369,  0.60099584]],

       [[ 0.44020506,  0.57156712,  0.75620793, ..., -0.22764274,
         -0.48342889, -0.62566145],
        [ 0.26129139,  0.37503705,  0.54736434, ..., -0.33795984,
         -0.60367347, -0.7525786 ],
        [ 0.0831942 ,  0.17498528,  0.32927041, ..., -0.4259337 ,
         -0.69403649, -0.84518828],
        ...,
        [ 0.49735228,  0.48456305,  0.47732518, ...,  0.53809259,
          0.52691347,  0.52110511],
        [ 0.54685429,  0.53792382,  0.53279902, ...,  0.57568366,
          0.56815463,  0.56424219],
        [ 0.58359014,  0.5776182 ,  0.57418789, ...,  0.60317397,
          0.59816927,  0.59555082]],

       [[ 0.30953497,  0.42007662,  0.58403348, ..., -0.07658794,
         -0.33735084, -0.49244539],
        [ 0.09075294,  0.18029783,  0.3300092 , ..., -0.19240263,
         -0.46671859, -0.63163157],
        [-0.12520879, -0.06078304,  0.06914137, ..., -0.29051005,
         -0.57036678, -0.74005957],
        ...,
        [ 0.4561422 ,  0.43867438,  0.42777446, ...,  0.53265923,
          0.52020029,  0.51301921],
        [ 0.51817876,  0.50600427,  0.49834653, ...,  0.57149754,
          0.56309772,  0.55826426],
        [ 0.56424358,  0.55609544,  0.55097258, ...,  0.60011915,
          0.59452943,  0.59130251]],

       ...,

       [[ 4.96952392,  5.66873968,  6.27218056, ..., -0.14137036,
         -0.06927937,  0.05653033],
        [ 4.69160343,  5.35871822,  5.93742861, ..., -0.16454984,
         -0.10078618,  0.01599066],
        [ 4.30086207,  4.91719088,  5.45477128, ..., -0.17217568,
         -0.11706298, -0.01120332],
        ...,
        [ 0.54400467,  0.53388539,  0.52597517, ...,  0.5796801 ,
          0.58408405,  0.58863475],
        [ 0.56975445,  0.56131133,  0.55457022, ...,  0.60302588,
          0.6062313 ,  0.60943636],
        [ 0.59287963,  0.5863306 ,  0.58103483, ...,  0.62048026,
          0.62272505,  0.62491081]],
```

ARIMA - without embedding

```
df.set_index('date', inplace = True)
```

```
from statsmodels.tsa.arima.model import ARIMA

without = df.drop('embeddings', axis=1)
model = ARIMA(without.values.reshape(-1).tolist(), order=(2,1,2))
```

```
from statsmodels.tsa.arima_process import arma_generate_sample
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf

fig, (ax1,ax2) = plt.subplots(2,1)

plot_acf(without.values.reshape(-1).tolist(),lags=10, zero=False, ax=ax1)

plot_pacf(without.values.reshape(-1).tolist(), lags=10, zero=False, ax=ax2)

plt.show()
```



1,1

```
model = ARIMA(withoutm order = (1,0,1))
results = model.fit()
```

```
------------------------------------------------------------------------
SyntaxError                               Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/IPython/core/compilerop.py in ast_parse(self, source, filename,
symbol)
     99         Arguments are exactly the same as ast.parse (in the standard library),
    100         and are passed to the built-in compile function."""
--> 101         return compile(source, filename, symbol, self.flags | PyCF_ONLY_AST, 1)
    102
    103     def reset_compiler_flags(self):

SyntaxError: invalid syntax. Perhaps you forgot a comma? (<ipython-input-905-b0b7a6d6f221>, line 1)
```

SEARCH STACK OVERFLOW

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import pickle
import datetime
from datetime import datetime

from google.colab import drive
drive.mount('/drive')
```

```
Mounted at /drive
```

```
embedding_df = pickle.load(open('/drive/My Drive/Colab Notebooks/embedding_df','rb'))
econ_df = pickle.load(open('/drive/My Drive/Colab Notebooks/econ_df', 'rb'))
```

```
df = embedding_df.merge(econ_df, on = 'date', how = 'right')
```

```
df.drop(columns = ['lemmas','a_lemmas','str_length', 'unrate','xrate','t5yie','t10yie'], inplace = True)
```

```
df.set_index('date', inplace = True)
```

```
df['embeddings'].interpolate(method = 'pad', limit = 3)
df['embeddings'].fillna(method="ffill", axis=None, inplace = True)
df['embeddings'].fillna(method="backfill", axis=None, inplace = True)
```

```
print(df.corr())
```

```
                embeddings    consum       gdp     frate
embeddings    1.000000 -0.066363 -0.068703 -0.029198
consum       -0.066363  1.000000  0.975820  0.043520
gdp          -0.068703  0.975820  1.000000  0.056158
frate        -0.029198  0.043520  0.056158  1.000000
```

```
print(df.describe())
```

```
         embeddings       consum           gdp        frate
count  4264.000000  4264.000000   4264.000000  4264.000000
mean      0.521629   302.141797  20313.495404     0.892704
std       0.076378     9.917226    455.653578     1.188383
min       0.156456   227.842000  16179.968000     0.040000
25%       0.478198   303.841000  20386.467000     0.090000
50%       0.520167   303.841000  20386.467000     0.160000
75%       0.564155   303.841000  20386.467000     1.550000
max       0.819841   303.841000  20386.467000     5.080000
```

```
def split_train_test(X,y):
  l = int(0.1 *X.shape[0])
  X_train = X.iloc[:len(X)-l]
  X_test = X.iloc[len(X)-l:]
  y_train = y.iloc[:len(y)-l]
  y_test = y.iloc[len(y)-l:]
  y_test = y_test[:-1]
  X_test = X_test[:-1]
  return X_train, y_train, X_test, y_test
```

```
def get_xy(df):
  X = df[['embeddings']]
  y = df['frate']
  return X,y
```

Linear Regression

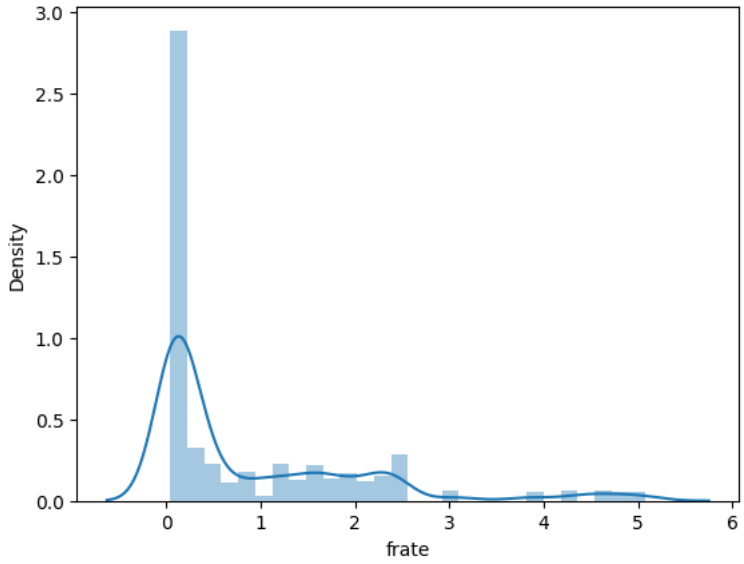
```
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split, cross_val_score
```

```
X,y = get_xy(df)
```

```
X_train, y_train, X_test, y_test = split_train_test(X,y)
```

```
import seaborn as sns
sns.distplot(y)
```

```
<ipython-input-14-578d37c7dc05>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y)
<Axes: xlabel='frate', ylabel='Density'>
```




```
reg_model = linear_model.LinearRegression()

reg_model = LinearRegression().fit(X_train, y_train)

print("Intercept: ", reg_model.intercept_)

list(zip(X, reg_model.coef_))
```

```
Intercept:  1.0150340084584408
[('embeddings', -0.7383700552448077)]
```

```
y_pred = reg_model.predict(X_test)
print("Prediction for test set: {}".format(y_pred))
```

```
Prediction for test set: [0.6991784  0.6991784  0.6991784  0.6991784  0.6991784  0.6991784
 0.6991784  0.6991784  0.6991784  0.6991784  0.6991784  0.6991784
 0.6991784  0.6991784  0.6991784  0.6991784  0.6991784  0.6991784
 0.6991784  0.6991784  0.66345188 0.66345188 0.66345188 0.66345188
 0.66345188 0.66345188 0.66345188 0.66345188 0.66345188 0.66345188
 0.66345188 0.66345188 0.66345188 0.66345188 0.66345188 0.66345188
 0.66345188 0.66345188 0.61360102 0.67589223 0.67589223 0.67589223
 0.67589223 0.67589223 0.57121537 0.57121537 0.57121537 0.6107848
 0.6107848  0.6107848  0.6107848  0.6107848  0.6107848  0.6107848
 0.6107848  0.6107848  0.6107848  0.6107848  0.6107848  0.6107848
 0.6107848  0.61409424 0.57280133 0.57280133 0.57280133 0.57280133
 0.57280133 0.64110542 0.64110542 0.64110542 0.64110542 0.64110542
 0.64110542 0.64110542 0.64110542 0.64110542 0.64110542 0.64110542
 0.64110542 0.64110542 0.64110542 0.57080951 0.57080951 0.57080951
 0.57080951 0.57080951 0.57080951 0.60365025 0.60365025 0.60365025
 0.60365025 0.60365025 0.69382957 0.69382957 0.69382957 0.69382957
 0.69382957 0.69382957 0.69382957 0.69382957 0.69382957 0.69382957
 0.69382957 0.69382957 0.69382957 0.69382957 0.69382957 0.69382957
 0.69382957 0.53916814 0.53916814 0.53916814 0.53916814 0.53916814
 0.53916814 0.53916814 0.53916814 0.53916814 0.53916814 0.53916814
 0.61372491 0.61885175 0.61885175 0.61885175 0.61885175 0.61885175
 0.61885175 0.61885175 0.61885175 0.61885175 0.63876891 0.63876891
 0.63876891 0.69564723 0.69564723 0.69564723 0.69564723 0.69564723
 0.69564723 0.69564723 0.69564723 0.67536017 0.63553015 0.63553015
 0.64407334 0.64407334 0.64407334 0.64407334 0.64407334 0.64407334
 0.64407334 0.64407334 0.64407334 0.64407334 0.64407334 0.64407334
 0.64407334 0.64407334 0.64407334 0.64407334 0.64407334 0.64407334
 0.64407334 0.62810954 0.5981781  0.5981781  0.59856776 0.6351268
 0.6351268  0.6351268  0.44787268 0.44787268 0.67029007 0.59591029
 0.59591029 0.59591029 0.59591029 0.55562684 0.55562684 0.58178713
 0.57533069 0.57533069 0.58316245 0.58316245 0.58316245 0.58316245
 0.58316245 0.58316245 0.62998847 0.62998847 0.62998847 0.62998847
```

```
0.62998847 0.62998847 0.62998847 0.62998847 0.62998847 0.62998847
0.62998847 0.62998847 0.62998847 0.62998847 0.62998847 0.62998847
0.62998847 0.62998847 0.62998847 0.62998847 0.62998847 0.62998847
0.62998847 0.62998847 0.62998847 0.62998847 0.56869151 0.61084434
0.62619183 0.62619183 0.62619183 0.62619183 0.62619183 0.62619183
0.62619183 0.62619183 0.62619183 0.62619183 0.62619183 0.64887667
0.64887667 0.68140117 0.6567717  0.6567717  0.6567717  0.6567717
0.6567717  0.6567717  0.6567717  0.6567717  0.6567717  0.6567717
0.6567717  0.6567717  0.6567717  0.6567717  0.6567717  0.6567717
0.6567717  0.6567717  0.6567717  0.6567717  0.6567717  0.6567717
0.6567717  0.6567717  0.6567717  0.6567717  0.6567717  0.6567717
0.6567717  0.54874557 0.54874557 0.54874557 0.54874557 0.52593614
0.63159141 0.63159141 0.63159141 0.63159141 0.63159141 0.63159141
0.63159141 0.63159141 0.63159141 0.60658859 0.63517869 0.57239868
0.57239868 0.57239868 0.57239868 0.57239868 0.57239868 0.57239868
0.57239868 0.57239868 0.57239868 0.57239868 0.57239868 0.57239868
0.57239868 0.57239868 0.57239868 0.64005727 0.63985975 0.63985975
0.64558809 0.64558809 0.64558809 0.61925911 0.61925911 0.62412
0.55731006 0.55731006 0.55731006 0.55731006 0.55731006 0.55731006
0.55731006 0.55731006 0.65699973 0.65699973 0.65699973 0.6666199
0.6666199  0.60668889 0.63402505 0.63402505 0.63402505 0.63402505
0.63402505 0.63402505 0.59579859 0.59579859 0.59579859 0.59579859
0.59579859 0.63380077 0.63380077 0.63380077 0.63380077 0.63380077
0.63380077 0.63380077 0.63380077 0.63380077 0.63380077 0.63380077
0.63380077 0.63380077 0.67247842 0.67247842 0.55746387 0.55746387
```

```
X_test
```

date	embeddings
2022-04-15	0.427774
2022-04-16	0.427774
2022-04-17	0.427774
2022-04-18	0.427774
2022-04-19	0.427774
...	...
2023-06-09	0.414018
2023-06-10	0.414018
2023-06-11	0.414018
2023-06-12	0.414018
2023-06-13	0.414018

425 rows × 1 columns

```
reg_model_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred})
reg_model_diff
```

date	Actual value	Predicted value
2022-04-15	0.33	0.699178
2022-04-16	0.33	0.699178
2022-04-17	0.33	0.699178
2022-04-18	0.33	0.699178
2022-04-19	0.33	0.699178
...	...	...
2023-06-09	5.08	0.709335
2023-06-10	5.08	0.709335
2023-06-11	5.08	0.709335
2023-06-12	5.08	0.709335
2023-06-13	5.08	0.709335

425 rows × 2 columns

```
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
```

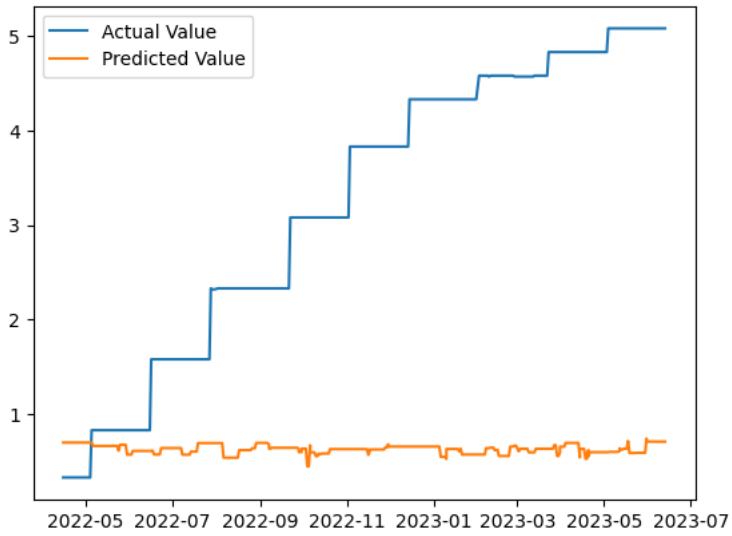
```
print('Mean Absolute Error:', mae)
print('Mean Square Error:', mse)
print('Root Mean Square Error:', r2)
```

```
Mean Absolute Error: 2.659636474355128
Mean Square Error: 9.205104649531853
Root Mean Square Error: 3.0339915374852073
```

```
def smape_kun(y_true, y_pred):
    mape = np.mean(abs((y_true-y_pred)/y_true))*100
    smape = np.mean((np.abs(y_pred - y_true) * 200/ (np.abs(y_pred) + np.abs(y_true))))
    print('MAPE: %.2f\nSMAPE: %.2f'% (mape,smape), "%")

smape_kun(y_test,y_pred)
```

```
MAPE: 76.25
SMAPE: 122.59 %
```

```
plt.plot(reg_model_diff['Actual value'], label="Actual Value")
plt.plot(reg_model_diff['Predicted value'], label = "Predicted Value")
plt.legend(loc = 'upper left')
plt.show()
```



FB Prophet

```
from prophet.plot import plot_plotly, plot_components_plotly
from prophet import Prophet
```

```
df.reset_index(inplace = True)
```

```
fb_data = df.rename(columns ={"date":"ds","frate":"y"})
l = int(0.1 *fb_data.shape[0])
train = fb_data.iloc[:len(fb_data)-l]
test = fb_data.iloc[len(fb_data)-l:]
```

```
train
```

	index	ds	embeddings	consum	gdp	y
0	0	2012-01-01	0.518817	227.842	16179.968	0.04
1	1	2012-02-01	0.518817	228.329	16179.968	0.11

```
m = Prophet()
m.add_regressor('embeddings')
m.fit(train)
future = m.make_future_dataframe(periods=86,freq='MS') #MS for monthly, H for hourly
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpsfj_5zoi/9hn9gtiy.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpsfj_5zoi/jllf9jqc.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=5743
10:19:05 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:19:07 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

```
n_df=train
n_df=n_df.append(test)

future=n_df
forecast = m.predict(future)
```

```
<ipython-input-34-74b86f31db5d>:2: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future
  n_df=n_df.append(test)
```

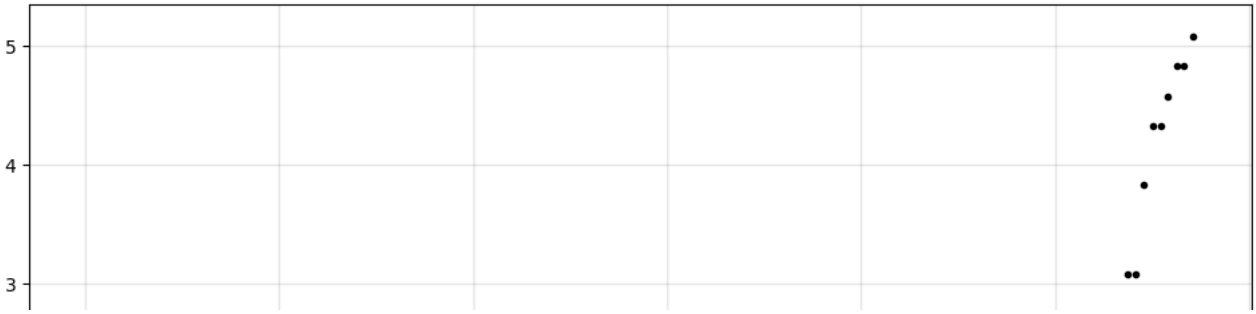
```
from statsmodels.tools.eval_measures import rmse
predictions = forecast.iloc[-l:]['yhat']
print("Root Mean Squared Error between actual and  predicted values: ",rmse(predictions,test['y']))
print("Mean Value of Test Dataset:", test['y'].mean())
```

```
Root Mean Squared Error between actual and  predicted values:  3.1905152995032067
Mean Value of Test Dataset: 3.2615258215962437
```

```
import numpy as np

tt=test['y']

smape_kun(tt,predictions)
```

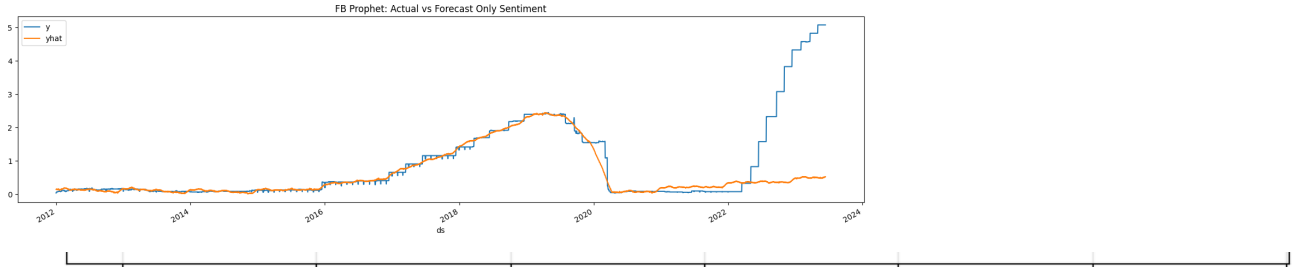
```
MAPE: 80.61
SMAPE: 140.84 %
```

```
m.plot(forecast)
```

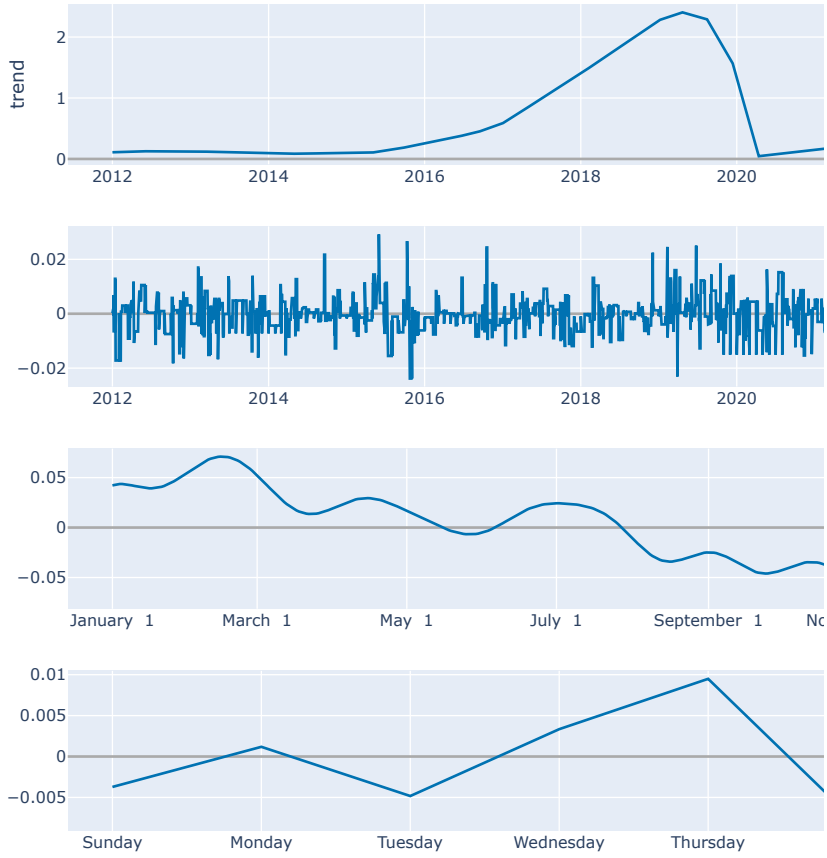



```
ax = (fb_data.plot(x='ds',y='y',figsize=(20,5),title='Actual Vs Forecast'))
forecast.plot(x='ds',y='yhat',figsize=(20,5),title='FB Prophet: Actual vs Forecast
```

```
<Axes: title={'center': 'FB Prophet: Actual vs Forecast Only Sentiment'},
xlabel='ds'>
```




```
plot_components_plotly(m, forecast)
```



XG Boost

```
import xgboost as xgb
from sklearn.metrics import mean_squared_error


regressor = xgb.XGBRegressor(
    n_estimators=100,
    reg_lambda=1,
    gamma=0,
    max_depth=3
)


regressor.fit(X_train, y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

```
y_pred = regressor.predict(X_test)


rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse)
```

```
RMSE:  3.136
```

```
mean_squared_error(y_test, y_pred)
```

```
9.831968148882039
```

```
y_pred
```

```
array([0.5485488 , 0.16209045, 0.9584034 , 0.9584034 , 0.39525563,
       0.39525563, 0.39525563, 0.39525563, 0.39525563, 0.39525563,
       0.39525563, 0.39525563, 0.39525563, 0.39525563, 0.39525563,
       0.39525563, 0.39525563, 0.39525563, 0.39525563, 0.39525563,
       0.39525563, 0.39525563, 0.39525563, 0.7044581 , 0.49644572,
       0.49644572, 0.49644572, 0.9584034 , 0.9584034 , 0.9584034 ,
       0.18429607, 0.18429607, 1.1662041 , 0.29636937, 0.29636937,
       0.29636937, 0.29636937, 0.36030212, 0.36030212, 0.3220686 ,
       0.43654183, 0.43654183, 0.3220686 , 0.3220686 , 0.3220686 ,
       0.3220686 , 0.3220686 , 0.3220686 , 1.0620013 , 1.0620013 ,
       1.0620013 , 1.0620013 , 1.0620013 , 1.0620013 , 1.0620013 ,
       1.0620013 , 1.0620013 , 1.0620013 , 1.0620013 , 1.0620013 ,
       1.0620013 , 1.0620013 , 1.0620013 , 1.0620013 , 1.0620013 ,
       1.0620013 , 1.0620013 , 1.0620013 , 1.0620013 , 1.0620013 ,
       1.0620013 , 1.0620013 , 1.0620013 , 1.0620013 , 0.25839144,
```

```
            0.5485488 , 0.5485488 , 0.5485488 , 0.5485488 , 0.5485488 ,
            0.5485488 , 0.5485488 , 0.4928609 , 0.23348889, 0.23348889,
            0.23348889, 0.23348889, 0.48923686, 0.48923686, 0.78205216,
            0.36030212, 0.49644572, 0.49644572, 0.49644572, 0.49644572,
            0.49644572, 0.49644572, 0.49644572, 0.49644572, 0.49644572,
            0.49644572, 0.49644572, 0.49644572, 0.49644572, 0.49644572,
            0.49644572, 0.49644572, 0.49644572, 0.49644572, 0.49644572,
            0.9253004 , 0.2907782 , 0.31763035, 0.3200491 , 0.3200491 ,
            0.41649282, 0.41649282, 0.41649282, 0.9267996 , 0.22030717,
            0.22030717, 0.22030717, 0.22030717, 0.22030717, 0.53071815,
            0.53071815, 0.53071815, 0.53071815, 0.53071815, 0.53071815,
            0.53071815, 0.54013944, 0.23150863, 0.23150863, 0.23150863,
            0.23150863, 0.23150863, 0.23150863, 0.23150863, 0.23150863,
            0.23150863, 0.23150863, 0.23150863, 0.23150863, 0.23150863],
           dtype=float32)
```

```
def smape_kun(y_true, y_pred):
    mape = np.mean(abs((y_true-y_pred)/y_true))*100
    smape = np.mean((np.abs(y_pred - y_true) * 200/ (np.abs(y_pred) + np.abs(y_true))))
    print('MAPE: %.2f\nSMAPE: %.2f'% (mape,smape), "%")

smape_kun(y_test.values,y_pred)
```

```
MAPE: 76.55
SMAPE: 127.94 %
```

Random Forest Regression

```
from sklearn.metrics import mean_squared_error # for calculating the cost function
from sklearn.ensemble import RandomForestRegressor # for building the model


model = RandomForestRegressor(n_estimators = 10, random_state = 0)
model.fit(X_train, y_train)
```

```
RandomForestRegressor(n_estimators=10, random_state=0)
```

```
y_pred = model.predict(X_test)

rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse)
```

```
RMSE:  3.211
```

```
smape_kun(y_test.values, y_pred)
```

```
MAPE: 79.83
SMAPE: 141.72 %
```

Support Vector Regression

```
X = df['embeddings'].values
y = df['frate'].values



X.reshape(-1, 1)
y.reshape(-1, 1)
```

```
array([[0.04],
       [0.11],
       [0.11],
       ...,
       [5.08],
       [5.08],
       [5.08]])
```

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
#X = sc_X.fit_transform(X)
#y = sc_y.fit_transform(y)


y
```

```
array([0.04, 0.11, 0.11, ..., 5.08, 5.08, 5.08])
```

ARIMA

```
df
```

	date	embeddings	frate
0	2012-01-01	0.518817	0.04
1	2012-02-01	0.518817	0.11
2	2012-03-01	0.518817	0.11
3	2012-04-01	0.518817	0.09
4	2012-05-01	0.518817	0.16
...	...	...	...
4259	2023-06-10	0.414018	5.08
4260	2023-06-11	0.414018	5.08
4261	2023-06-12	0.414018	5.08
4262	2023-06-13	0.414018	5.08
4263	2023-06-14	0.414018	5.08

4264 rows × 3 columns

```
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.arima_process import arma_generate_sample
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf

fig, (ax1,ax2) = plt.subplots(2,1)

plot_acf(df,lags=10, zero=False, ax=ax1)

plot_pacf(df, lags=10, zero=False, ax=ax2)

plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-47-8b775a25d31f> in <cell line: 7>()
      5 fig, (ax1,ax2) = plt.subplots(2,1)
      6
----> 7 plot_acf(df,lags=10, zero=False, ax=ax1)
      8
      9 plot_pacf(df, lags=10, zero=False, ax=ax2)
```

```
speech = pickle.load(open('/drive/My Drive/Colab Notebooks/speech_df','rb'))
```

```
speech
```

```
speech['word_count'].mean()
```

```
import seaborn as sns
#distribution of diff
sns.distplot(speech['word_count'])
```

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
from collections import Counter
text_list = speech['lemmas'].values.tolist()

Counter(word_tokenize(text_list))
```

```
from nltk.probability import FreqDist
fdist = FreqDist(text_list)
top_ten = fdist.most_common(10)
```