FIN42110: Financial Data Science Project

Sofia Emelianova - 22205187
Ajit Nambiyar - 22200172
Fanis-Filippos Papanikolaou - 22200286

April 2023

1 Executive Summary

The objective of this project was to examine the predictive capabilities of the underlying sentiment within the Beige Book that is released 2 weeks prior to the FOMC meeting, in predicting the committee's decisions to maintain, increase or decrease the Target Federal Funds Rate using a supervised learning model called Random Forest Classifier.

The sentiment analysis was carried out using an NLP package called FinBERT, which is a pre-trained NLP model specialised in understanding financial and economical textual sentiment.We paired the sentiment analysis with the other other quantitative indicators like consumer price index and existing target federal funds rate. The results indicated that the sentiment underlying in the beige book has been insignificant in improving the model's performance vis-a-vis using only the quantitative metrics mentioned above.

The trends arising in the sentiment within the sectors and regions covered in the Beige Book are also explored within this report. Based on this analysis it was observed that the average sentiment within the Beige Book of the overall economy of USA has been relatively negative since 2018.

2 Introduction

The Federal Reserve plays a critical role in managing the US economy through its monetary policy decisions. The Federal Open Market Committee (FOMC) hold eight regularly scheduled meetings per year during which members review economic and financial conditions, determine the appropriate course of monetary policy and set interest rates. One important tool that the FOMC uses in making its decisions is the beige book.

The beige book is a report published by the Federal Reserve eight times a year, two weeks prior to each FOMC meeting. The report provides a summary of economic conditions in each of the twelve Federal Reserve districts. It is based on information gathered through interviews with business contacts, economists, and other sources. The report covers a wide range of economic indicators, including consumer spending, employment, manufacturing, and real estate. At the top, it has a summary section that provides the holistic view of the economy, both by sector and by region. The purpose of the beige book is to provide the FOMC with a snapshot of the economy, helping them to make informed decisions about monetary policy.

The federal funds target rate is decided upon and set by the FOMC and is defined as the interest rate at which depository institutions lend and borrow funds overnight from each other to meet their reserve requirements. The target rate is determined based on the assessment of current economic conditions, in addition to the current maximum employment and price stability goals. The goal of this research is to extract the underlying sentiment of the beige book reports and examine if it can be used to predict future movement in the Fed Fund Target Rate, as well as how it relates to the economy's response to interest rate changes.

The text from the beige books will be analysed, using the specialised Natural Language Processing model, *FinBERT*, to understand underlying financial and economic sentiment. Finally, in order to forecast future FOMC decisions on interest rate movements, appropriate machine learning predictive modelling will be used. Due to computational limitations, 206 beige books were examined, from the end of 1996 to the end of 2022.

3 Methodology

3.1 Data Collection

The URLs were scraped from the Fed website.Appendix A.1
The textual data from the beige books and the FOMC meeting dates were collected through web scraping techniques. Using the Python module *BeautifulSoup*, the text from the Federal Reserve website was scraped. The text of the beige books was essential in order to do the sentiment analysis. Appendix A.2
The FOMC meeting dates were used to assure if the correct target rate was assigned to the corresponding beige book. There was perfect matching between the two so no further action was needed and the target rates were plugged in accordingly.Appendix A.4

The website structures of both the FOMC meeting and the beige book text URLs, differed throughout the periods, so specific scraping strategies had to be deployed in order to cover all periods. Appendix A.3
The FRED API was used to extract the Fed Fund Target Rate, Consumer Price, Confidence Indices and the Producer Price index. *Yfinance* was used to get the Adjusted Close prices of the S&P500 since the FRED API did not provide values before 2013. Appendix A.6

For the purposes of this report, the summary section of the beige books was assigned the column name *Overall* for both the regions and the sectors.

The macro data we chose to examine are representative of the US economy, like the market return and the Consumer Price Index (CPI), which is a measure of Inflation. Also, consumer confidence was examined as perhaps another metric of overall sentiment. Finally, the Fed Fund Target Rate was extracted as its movement is what we will try to predict.

3.2 Data Cleaning

After scraping the data, the sectors had to be cleaned. When looking at unique sectors, an inconsistency was determined between the sector nomenclature during multiple beige book reports. Hence, standardisation was considered and a sector key map was created, tagging the various inconsistencies to a standardised naming system.Appendix A.5
Secondly, inconsistencies were noticed again, this time in the region nomenclature. The same process was followed, by creating the region key map. Appendix A.6 Further work was done to drop any null values and unnecessary HTML elements in the text. Finally, for the FOMC meeting dates, the website for the meetings of the past 5 years differed to the older ones, so the dates had to be adjusted to match the overall formatting. There were slight inconsistencies of FOMC meetings throughout the years where the number of meetings exceeded the number of beige books released during the year due to unscheduled meetings during Covid and the missing beige book for September 2003.

3.3 Textual Analysis

A finance-specific BERT model, *FinBERT* was deployed to analyse the sentiment of the beige book reports. FinBERT is a language model that is specifically trained on financial text. It is based on the BERT architecture and is capable of processing large volumes of financial data to extract insights, sentiment, and relationships between entities. FinBERT has been shown to outperform traditional NLP models on a variety of financial tasks, including stock price prediction and sentiment analysis.

The limitation was that FinBERT only allows 512 tokens at a time. To counter that, the text tokens were split into two, where the sentiment analysis was done on each part, and a weighted average based on the number of tokens of each, was used to derive an overall text sentiment.

Three sentiment scores; positive, negative and neutral were the desired output of the model for which values ranged from 0 to 1 for each sentiment score. The NLP model gave three outputs in the form of positive, negative, and neutral. The sum of all three is equal to 1 and the values range from 0 to 1. To evaluate the overall sentiment, a sentiment index was developed where the index was formed by taking the average of the positive and negative scores.Appendix A.5

3.4 Database Creation and Querying

Before querying, the database was created using the scraped and cleaned data. The plan was to calculate the summary statistics for sectors, regions and for the sentiment scores. For the summary statistics of sectors and regions, the sentiment index was calculated as the difference of the positive and negative sentiment score and

divided by two. For the summary statistics of the positive, negative and neutral sentiment scores, the full data set was used.

Since the sentiment scores were inputted into the database as columns, the general outline of summary statistics queries included selecting all three scores into one query and applying the inbuilt function. The results were then collected in a data frame for each calculated statistic. In the case of the median and standard deviation, the query was set up to extract and prepare the data. The scores were then sliced outside of the query where the median and standard deviation were calculated for each score, after which the results were appropriately stored into a data frame. Finally, the summary matrix was concatenated using the results from the calculated statistics.

In the cases of sector and region, the general outline of the summary statistics queries was also true with the exception of sentiment index being used to compute the statistics. The group by statement was used within the query to group the sentiment index by sector and later on by region. For median and standard deviation calculations, querying was used to slice out and arrange the needed data for the calculation, which was then performed outside of the query. We grouped the data and calculated the standard deviations and the medians of the sector and region groups. Appendix A.7

3.5 Data visualisation

The box plot and scatter plots were computed. The box plot is a great tool to analyse the polarity of the sentiment and get an understanding of how the observations are distributed. To compute the box plot, the three sentiment scores were extracted individually from the *beigebook* data frame and then inserted into separate *go.Box* methods. For graphing the scatter plots, the date column was extracted from the *beigebook* data frame in addition to the three sentiment scores, sentiment index and target rates which were then added to the *go.Scatter* method. Each variable was plotted on the y-axis while the date was plotted on the x-axis. Appendix A.8

3.6 Predictive/Explanatory Modelling using Machine Learning

3.6.1 Feature Selection

The feature selection was primarily carried out by evaluating the correlation matrix of the variables we have gathered in our data set. We aimed to select variables that displayed low correlation with other variables. Based on this selection criteria we decided to drop the following features.
1. Producer Price Index (PPI) as it displayed a perfect positive correlation with Consumer Confidence.
2. S&P500 was also excluded from the model since it displayed a high correlation with CPI.
3. Delta was excluded from the model since it showcased a perfect positive correlation with the Predicted movement of the Target Fed Funds rate.
4. *Positive* and *Negative* sentiment scores we also excluded in favour of the *Sentiment index* and *Neutral* sentiment scores. Appendix A.9

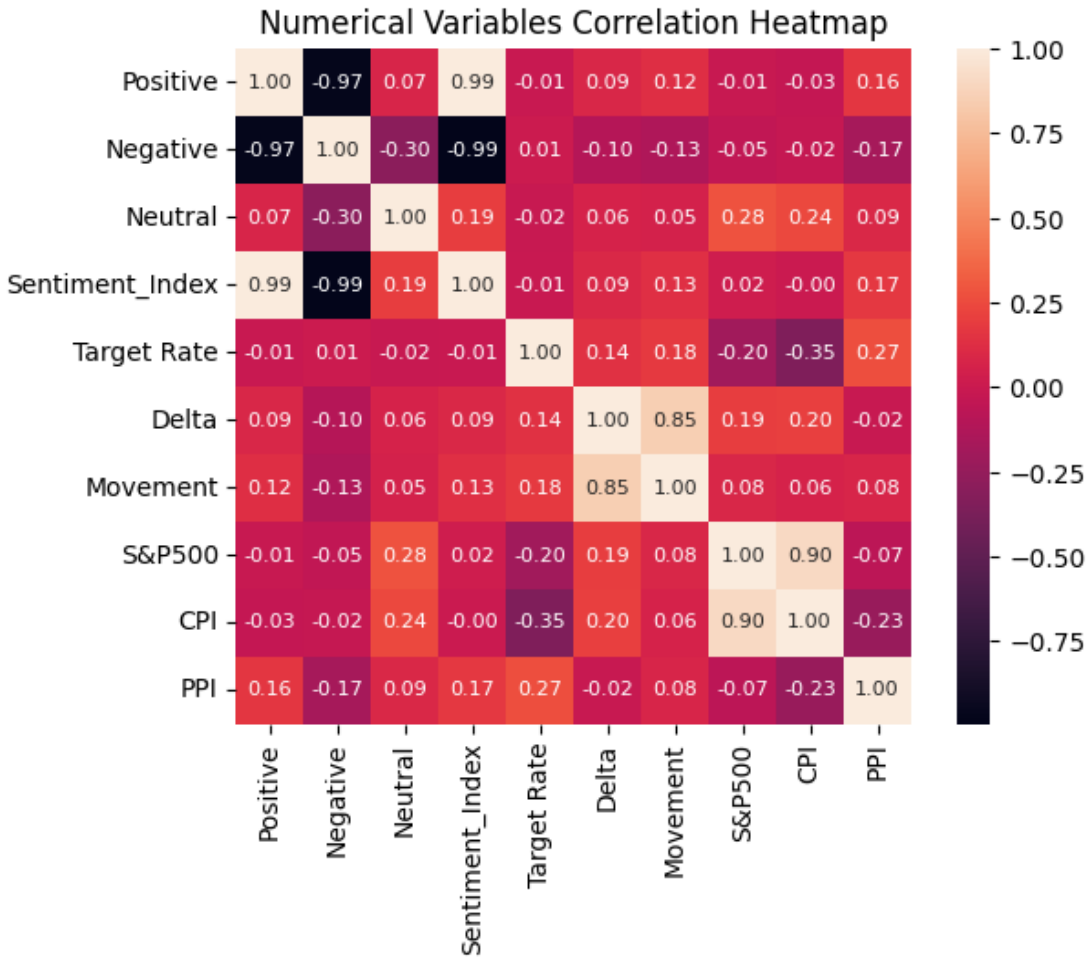

Figure 1: Correlation matrix of the features.

3.6.2 Data Pre-processing

Initially the sector and region data were transformed to feature by assigning dummy variables. But we observed that it did not add significant explanatory value to the models based on our trials. Hence, we decided to exclude it to rationalise dimensionality of the model. The train and test split of the model weighed in the fact that the data set has recurring macro data mapped against the sentiment scores. So, in order to avoid the test data being polluted with any training data we have employed a strategy to split the training and test data based on date. The training data has been taken up to the period of 2012 and the test data has been from 2012-2022.

3.6.3 Random Forest Model

Random Forest Classifier is an ensemble learning algorithm that combines multiple decision trees to create a model. It is a powerful and popular machine learning algorithm that can be used for various classification tasks. One of the significant advantages of Random Forest Classifier is that it can handle multi-class classification problems with ease. The model here is designed to classify the outcome of the FOMC meeting in terms of whether the Target Federal Funds Rate will *increase*, *decrease* or be held *constant*. We deployed hyper-parameter tuning technique to find the optimal parameters to input into the model based on an objective score to maximise F1-Macro. F1 macro is preferred over other metrics like *Accuracy* for multi-class classification because it gives equal weight to each class, regardless of its frequency in the data set.

We decided to run two Random Forest models and compare them. In both of them, the dependent variable was the Movement of the interest rate. The first model had the all the macro data that we kept, the Target Rate and the Neutral and Sentiment Index. The second model had the macro data and the Target Rate without the Sentiment Index and the Neutral. Appendix A.9

4 Results

4.1 Summary Statistics of the Sentiment

The results for summary statistics of the sentiment scores for the data set with 16,479 observations are represented in Table 1, while Figure 2 illustrates these results as box plots.

Mean: The mean is the average sentiment score for each category. The mean score for *positive* sentiment is 0.299, which indicates that the sentiment of the observations is slightly positive on average. The mean score for *negative* sentiment is 0.647, indicating that the sentiment of the observations is mostly negative. The mean score for *neutral* sentiment is 0.054, which shows that the sentiment of the observations is relatively neutral.

Median: The median is the middle score in the data set. The median score for *positive* sentiment is 0.125, which is lower than the mean score, indicating that the distribution of *positive* sentiment scores is skewed towards the lower end of the range. The median score for *negative* sentiment is 0.817, which is higher than the mean score, indicating that the distribution of *negative* sentiment scores is skewed towards the higher end of the range. The median score for *neutral* sentiment is 0.032, which is also lower than the mean score, indicating that the distribution of *neutral* sentiment scores is skewed towards the lower end of the range.

Standard deviation: The standard deviation is a measure of the spread of the data. The higher the standard deviation, the more spread out the data is. The standard deviation for *positive* sentiment is 0.325, for *negative* sentiment it is 0.340, and for *neutral* sentiment it is 0.077. These values show that the sentiment scores in all three categories are quite spread out, with *negative* sentiment having the highest standard deviation.

Maximum and Minimum: These values indicate the range of sentiment scores in each category. The maximum and minimum scores for *positive* sentiment are 0.961 and 0.006, respectively. For *negative* sentiment, the maximum and minimum scores are 0.976 and 0.011, respectively, while for *neutral* sentiment, they are 0.931 and 0.010, respectively. These values show that there is a wide range of sentiment scores in each category.

Table 1: Summary Statistics matrix for the sentiment scores.

	Positive	Negative	Neutral
Observations	16479	16479	16479
Mean	0.299006	0.647352	0.053642
Median	0.125314	0.816504	0.031778
Std. Dev.	0.325211	0.339828	0.077267
Maximum	0.960600	0.976279	0.930647
Minimum	0.006355	0.010825	0.010152

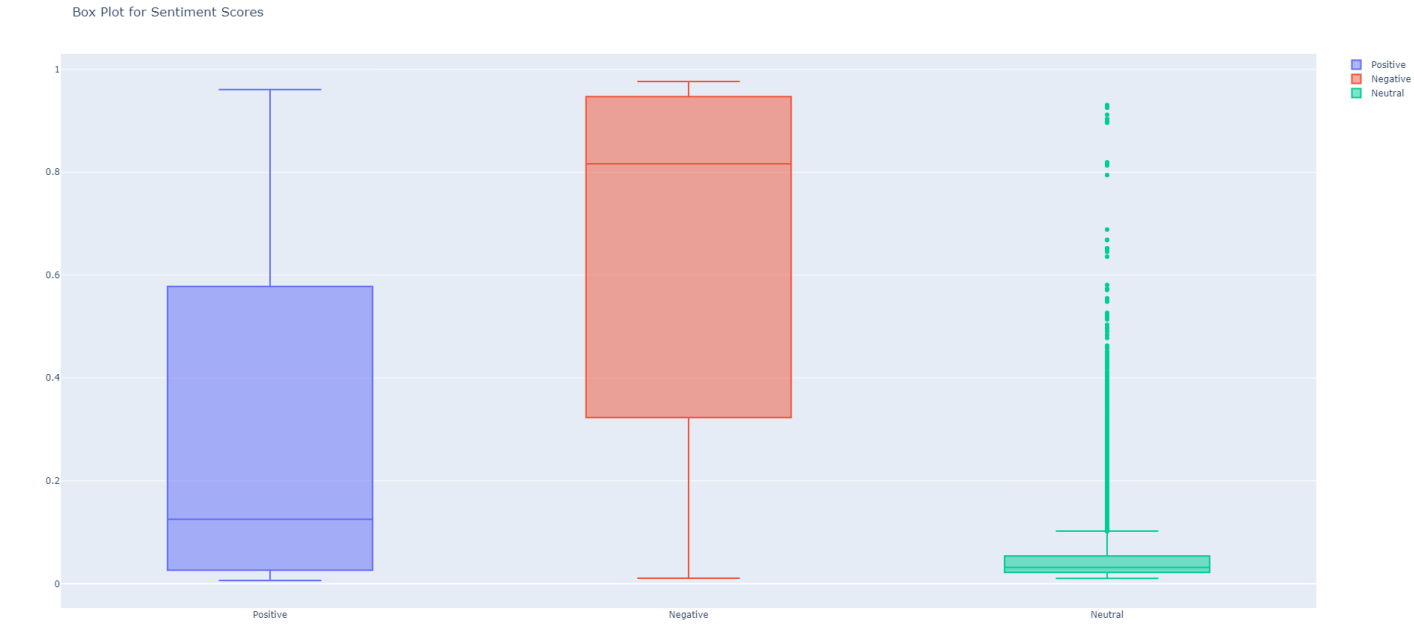

Figure 2: Box plots for positive, negative and neutral Sentiment Scores.

The sentiment index is a numerical score that represents the overall sentiment or emotional tone of a piece of text. Thus, a low sentiment index can mean that the overall sentiment or attitude towards was more negative. This can indicate a loss of confidence or optimism among market participants, investors, or consumers, which can have a negative impact. It could also indicate a shift in market or economic conditions that is influencing people's perceptions and attitudes. Figure 3 is a graph of the calculated sentiment index over the period 1996-2022. Between 1996 and 2008 the sentiment index is consistent with some fairly usual yearly fluctuations (as expected). This can mean that the overall sentiment or attitude was not changing greatly over time. This could indicate a stable level of confidence or optimism among market participants reflecting a stable or predictable market or economic environment. However, a constant sentiment index does not necessarily mean that everything is positive or favorable. It could simply mean that the *positive* and *negative* factors influencing sentiment are canceling each other out, resulting in a *neutral* sentiment overall. In 2009 the sentiment index peaks downwards in reference to the 2009 financial crisis where there was an overall loss of confidence and the economy suffered. The 2018-2022 during the Covid period, the sentiment index is once again seen dipping downwards indicating negative attitude of the market participants.

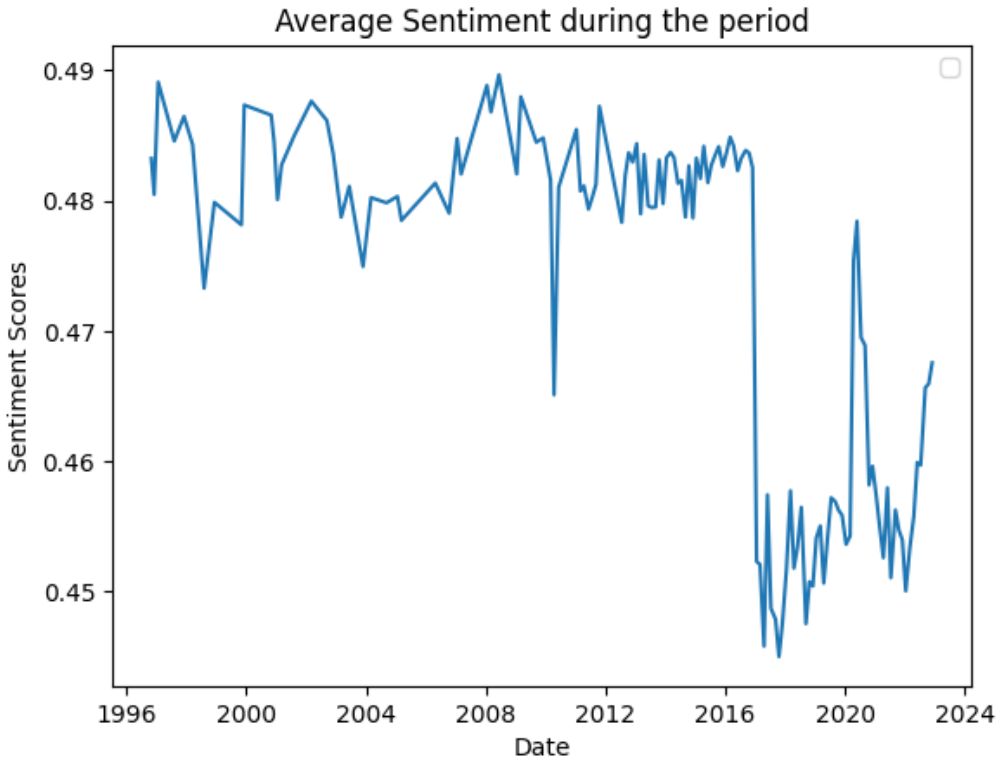

Figure 3: The average Sentiment index over time (1996-2022).

Table 2 represents the summary statistics for each sector while Figure 5 illustrates the sentiment index for each sector. The mean sentiment index for the *Overall* sector is 0.438, indicating an overall negative sentiment (since the mean is less than 0.5 benchmark for neutrality used in sentiment analysis). However, the median sentiment index is 0.488, which suggests that the distribution is skewed towards positive sentiment for some sectors. The sector with the highest mean sentiment index is *Retail*, with a value of 0.486, while the sector with the lowest mean sentiment index is *Banking and Financial Services*, with a value of 0.471. The standard deviation for sentiment index varies across sectors, ranging from 0.006 for *Business Spending* to 0.091 for the *Overall*. This suggests that the distribution of sentiment index scores is relatively tight for some sectors and more spread out for others. In terms of minimum and maximum sentiment index values, the *Business Spending* sector has the highest minimum value of 0.464 and has the lowest maximum value of 0.492. Figure 4 plots the sentiment index over time for each sector. The same trend as in Figure 3 can be observed for the sectors.

Table 2: Summary Statistics matrix per sector.

Sector	Mean	Median	Std. Dev.	Maximum	Minimum
Agriculture and Natural Resources	0.472962	0.488843	0.040462	0.494844	0.044203
Banking and Financial Services	0.470667	0.487529	0.029391	0.493945	0.238493
Business Spending	0.483901	0.483158	0.006199	0.491979	0.463853
Consumer Spending and Tourism	0.482394	0.478923	0.013874	0.494891	0.375051
Energy	0.474377	0.485894	0.030844	0.494355	0.093490
Manufacturing and Transportation	0.480061	0.475933	0.018947	0.494924	0.287907
Overall	0.438035	0.487559	0.090509	0.492484	0.034677
Prices and Labor Markets	0.467888	0.487616	0.034205	0.494530	0.165753
Real Estate and Construction	0.479407	0.490220	0.019366	0.494455	0.173922
Retail	0.486229	0.478651	0.007540	0.494687	0.443133
Selected Business Services	0.479701	0.474990	0.023052	0.494715	0.182103

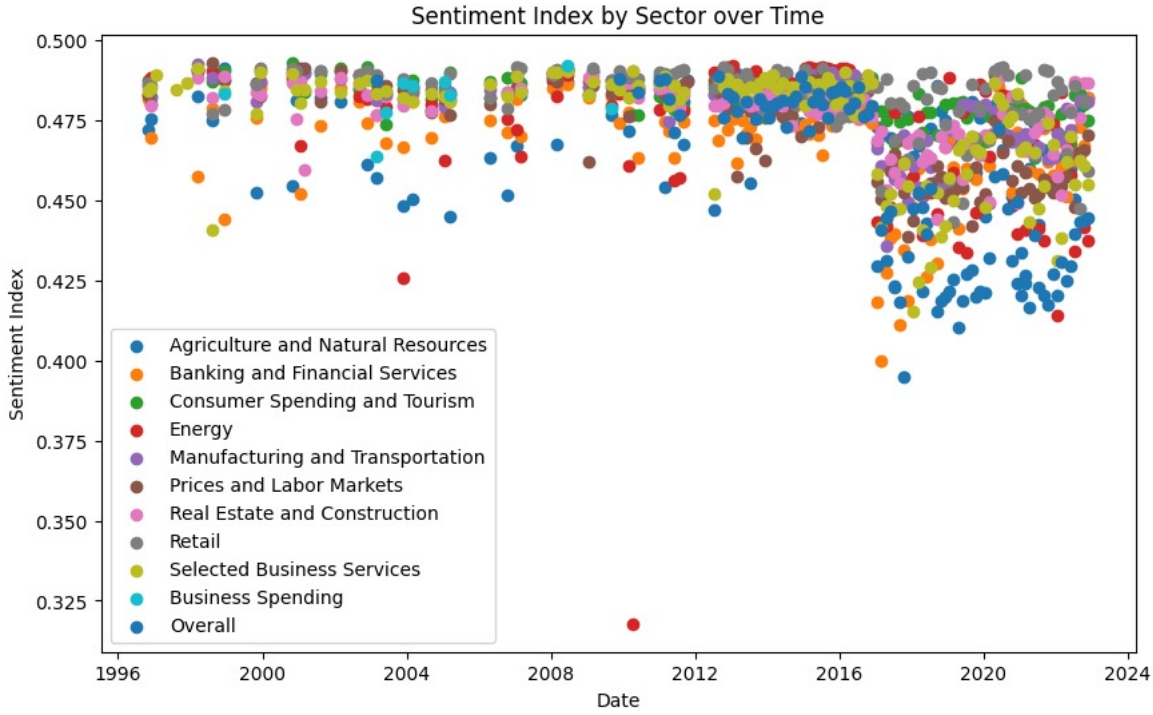

Figure 4: Scatter plot of the Sentiment index for each sector over time (1996-2022).

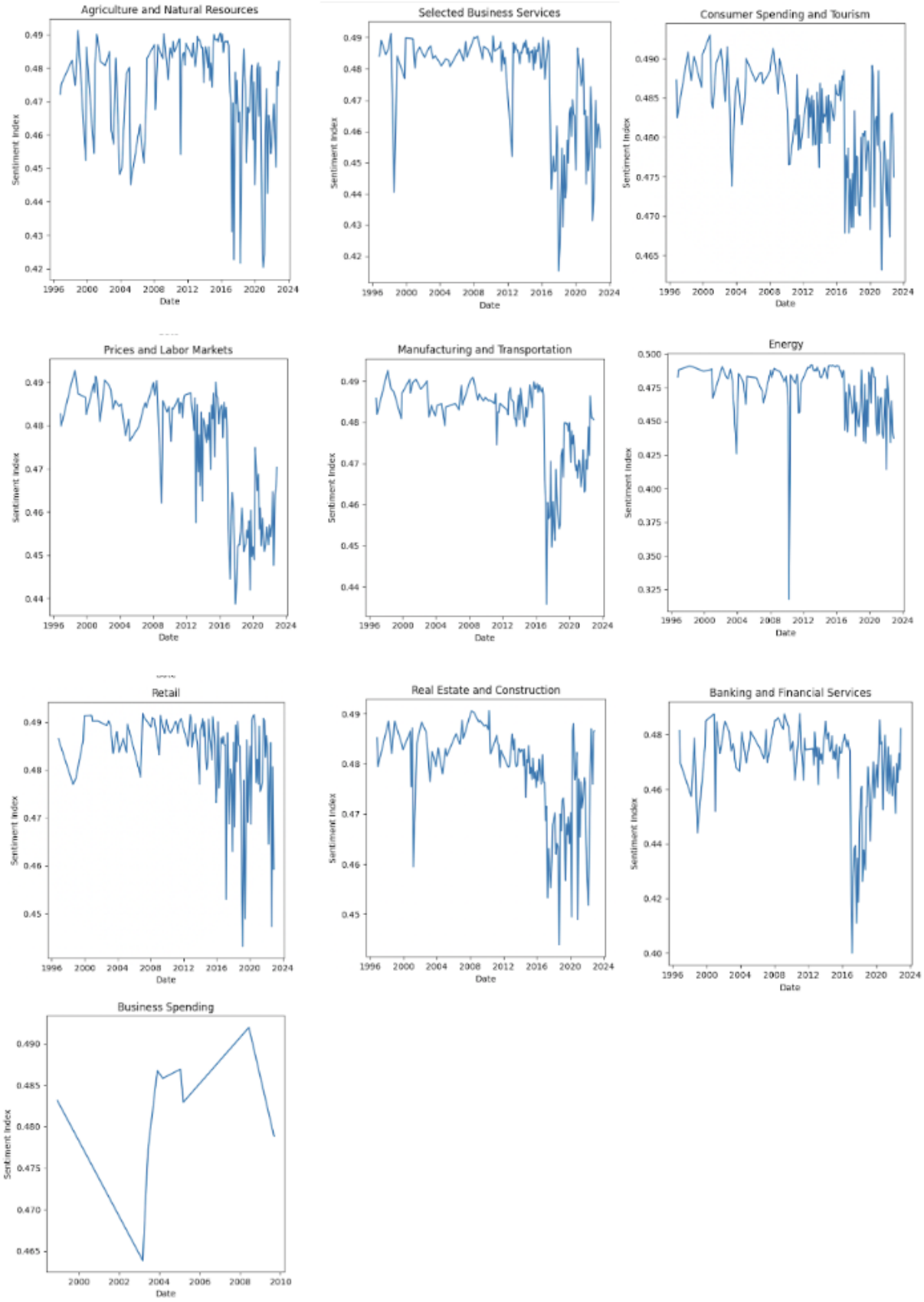

Figure 5: Graphs for the Sentiment index for individual sectors over time (1996-2022).

Table 3 summarises the calculated statistics for the regions using the sentiment index. The mean values range from 0.471 for *St Louis* to 0.480 for *New York*, with the *Overall* mean value being 0.453. The median values range from 0.469 for *Boston* to 0.491 for *Atlanta*, with the *Overall* sectors' median value being 0.475. The standard deviation values range from 0.0215 for *San Francisco* to 0.083 for the *Overall* sector, indicating that there is a large variability in the data. The maximum values range from 0.4934 for *Overall* to 0.4949 for *Dallas*, while the minimum values range from 0.0347 for the *Overall* to 0.294 for *San Francisco*. *New York* has the highest mean values, while *St Louis* has the lowest mean value. Additionally, *Atlanta* has the highest median and *Boston* has the lowest median. The *Overall* data has the largest standard deviation, indicating more variability compared to individual regions. Figure 6 plots a scatter plot of the sentiment index by region over time while Figure 7 illustrates the trend for regions independently of one another. The same rationale can be applied to analyse Figure 7. For the period 2016 on-wards, the difficult economic conditions of that time are illustrated. The data in general appears to be stable for the years prior to 2016 thus indicating that the average sentiment index does not vary too much throughout the years except for some outliers.

Table 3: Summary Statistics matrix per region.

Region	Mean	Median	Std. Dev.	Maximum	Minimum
Atlanta	0.474760	0.491190	0.030033	0.494517	0.044203
Boston	0.471445	0.469350	0.037478	0.494496	0.165753
Chicago	0.472400	0.483836	0.030838	0.494844	0.238608
Cleveland	0.473742	0.475881	0.030051	0.493715	0.093490
Dallas	0.478744	0.486953	0.022800	0.494924	0.176866
Kansas	0.477308	0.479421	0.023195	0.494380	0.254872
Minneapolis	0.477601	0.481633	0.026268	0.494557	0.182103
New York	0.480476	0.487481	0.021731	0.494891	0.274300
Overall	0.453296	0.474990	0.083202	0.493436	0.034677
Philadelphia	0.476707	0.489009	0.024853	0.494715	0.224042
Richmond	0.474543	0.484753	0.028922	0.494001	0.102755
San Fransicsco	0.478724	0.482634	0.021472	0.494527	0.293509
St Louis	0.470660	0.488448	0.032402	0.493760	0.155777

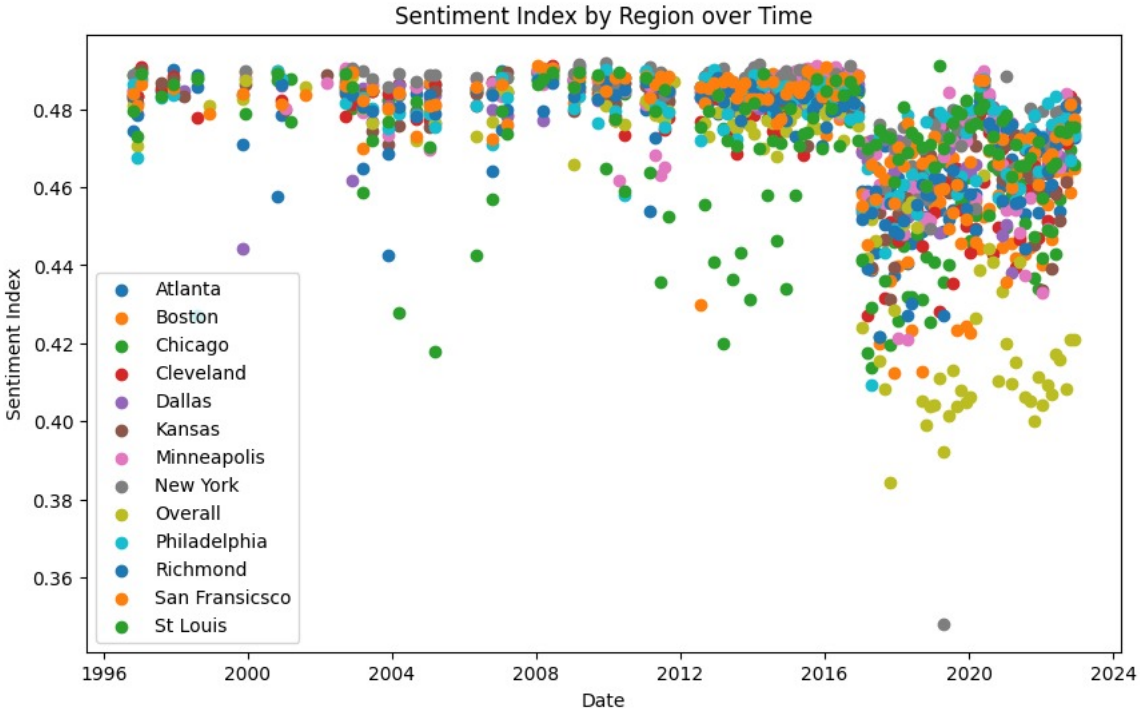

Figure 6: Scatter plot of the Sentiment index for each region over time (1996-2022).

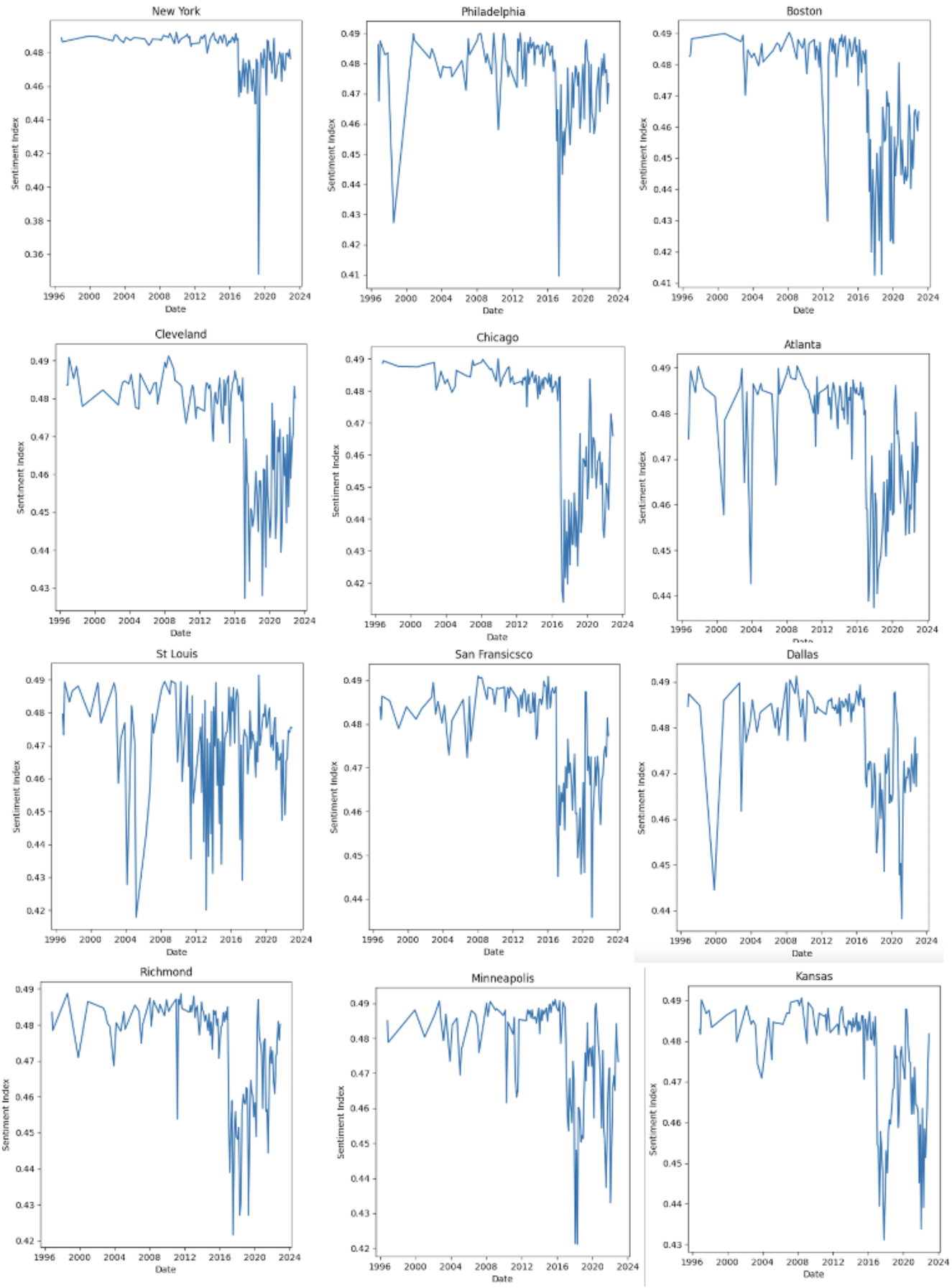

Figure 7: Graphs for the Sentiment index for individual regions over time (1996-2022).

It can be inferred from Figure 8, that the rate has varied over time in response to economic conditions and Federal Reserve policy decisions, ranging from a high of 5.5% in the 1990s to a low of 0% during the 2008-2009 financial crisis. In recent years, the Fed has gradually increased the rate before lowering it to near-zero levels in response to the economic impact of the COVID-19 pandemic. The Fed Funds Rate is a critical tool for the Federal Reserve to manage monetary policy and can impact the broader economy by affecting borrowing costs for businesses and individuals. The current interest rate hikes can be seen at their beginning, in early 2022. The volatility of the target rate that can be observed in this chart, is in line with the volatile nature of global and US economy.

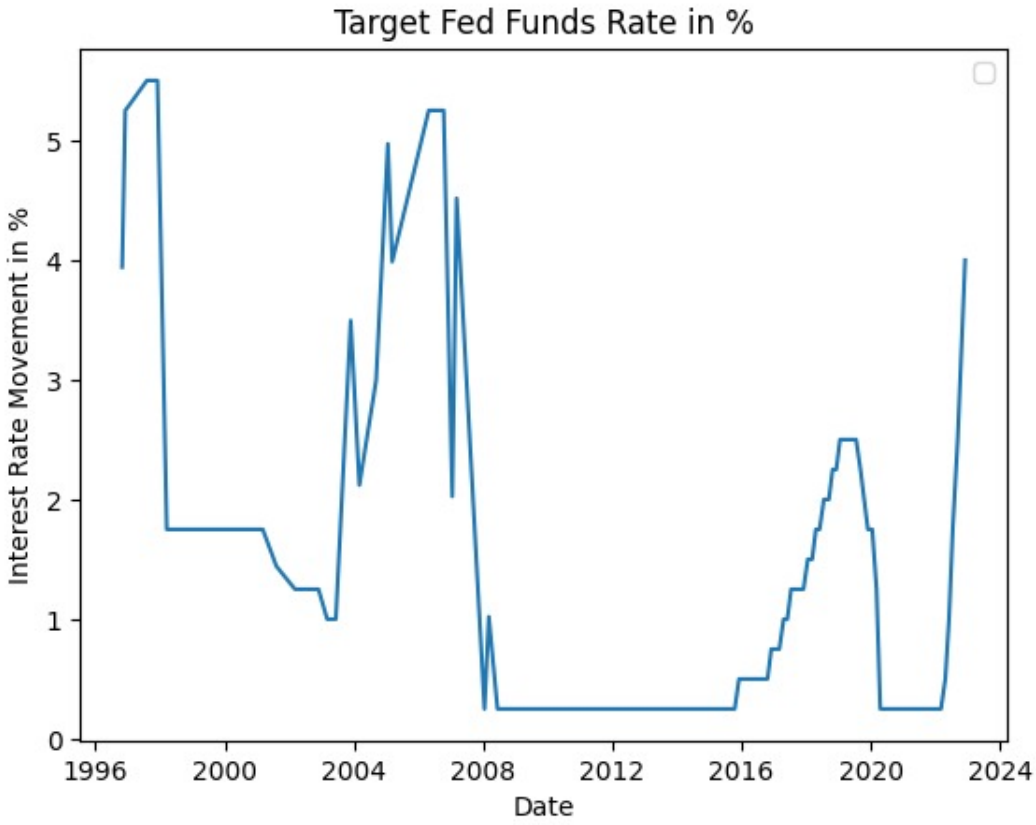

Figure 8: The Target Fed Fund Rate in % over time (1996-2022).

4.2 Machine Learning

In order to study the effectiveness of the sentiment analysis related features' explanatory power in interest rate movement, we decided to build two Random Forest models, one with sentiment related features and one without.

4.2.1 Random Forest

The two models that were deployed gave us different and interesting results. Both models were recommended Gini for the recursive splitting criteria and a maximum tree depth of 5. While the first model appears to have a similar performance to the second one, when looking at the 0.9 F1 Macro score, the n-estimators tell us that the second model required 100 trees in the forest while the first model required 20, possibly making it more computationally efficient and less probable to over-fit.

Table 4: The 4 Random Forest Models

	Macro Features		**Sentiment Features**	**Hyper-parameter tuning based results**			**Performance Score**	
	CPI	**Target Rate**	**Sentiment_Index**	**Neutral**	**Criterion**	**Max_Depth**	**n_Estimators**	**F1_Macro Score**
RF -Model 1	Yes	Yes	Yes	Yes	Gini	5	20	0.906
RF-Model 2	Yes	Yes	No	No	Gini	5	100	0.903

4.2.2 Feature Importance

We evaluated the feature importance and observed that the sentiment related features i.e 'Sentiment Index' and 'Neutral' contributed hardly any additional information in the prediction of the movement of the target federal

reserve rate. The CPI and the Target Rate contributed most of the information for the model.

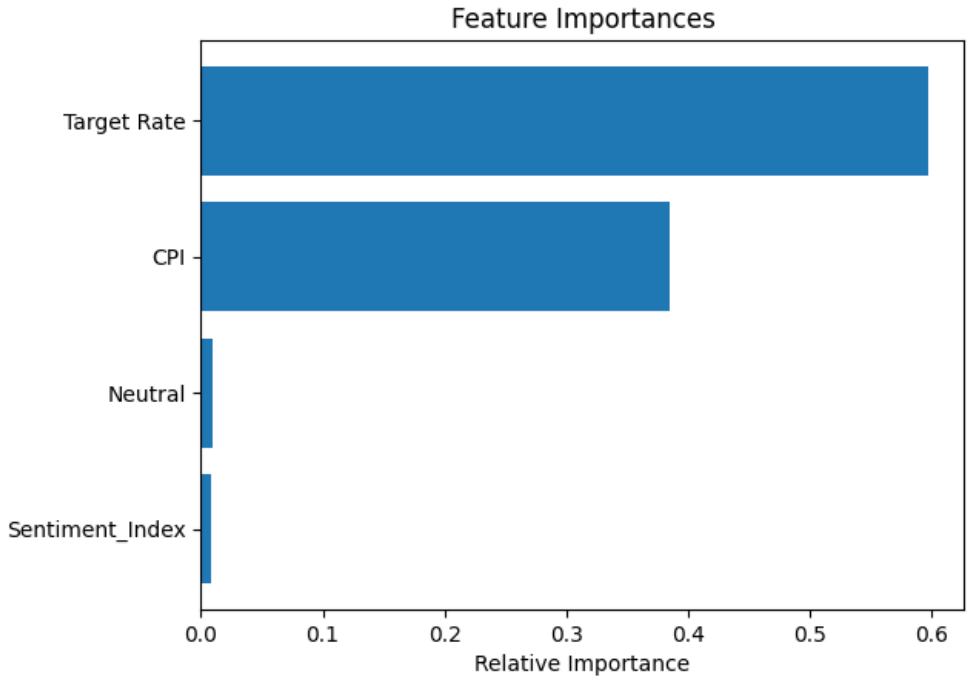

Figure 9: Relative Feature Importance Results Model 1.

4.2.3 Prediction Results

The results of the model's forecasting capabilities can be analysed with 3x3 confusion matrices. The results of the model's performance in classifying the outcome are as follows:

We took into account the number of instances that were correctly classified (true positives) and incorrectly classified (false positives, false negatives, and true negatives) for three classes: *increase*, *constant*, and *decrease*. For the *increase* label, the model predicted 389 positives(TP) out of a total of 3252, showcasing a low True Positive Rate (TPR), a recall of 0.1196, a precision of 0.645 and an F1 Score of 0.2018, all quite low metrics.

For the *constant* label, the model had 4966 TPs in a total of 5577 positives, performing a lot better in predicting the non-movement of the Target Rate. This results in a precision of 0.768, recall of 0.8904, and F1-score of 0.8246. For the *decrease* label, the model falsely predicted 1760 cases of a decrease, whereas there were none in that time period. Overall, the model performs well for the *constant* label, but performs poorly for the *increase* and *decrease* labels.

Table 5: Confusion Matrix of the First Model

Predicted			
Increase	389	214	0
Constant	1500	4966	0
Decrease	1363	397	0
Actual	Increase	Constant	Decrease

Table 6: Performance metrics Model 1.

Label	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1-score**
Increase	389	2863	214	0.6451	0.1196	0.2018
Constant	4966	611	1500	0.7680	0.8904	0.8247
Decrease	0	1760	0	0	0	0

From this model's confusion matrix, we notice a better performance in the increase label and a similar performance for the constant level. Overall the model performs slightly better performance than the first one, but they are both very close.

Table 7: Confusion Matrix of the Second Model

Predicted			
Increase	242	361	0
Constant	720	5746	0
Decrease	741	1019	0
Actual	Increase	Constant	Decrease

Table 8: Performance metrics Model 2.

Label	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1-score**
Increase	242	1461	361	0.4013	0.1421	0.2098
Constant	5746	1380	720	0.8886	0.8063	0.8454
Decrease	0	1760	0	0	0	0

5 Discussion

5.1 Business Value

Predicting FOMC (Federal Open Market Committee) monetary policy decisions can provide significant business value for a range of stakeholders, including investors, financial institutions, and corporations. Following are some of the business uses cases:

1. Aiding Investment Strategy: Investors can use predictions of FOMC policy decisions to adjust their investment strategies. For example, if the market anticipates a rate cut, investors may shift their investments towards sectors that are likely to benefit from lower interest rates, such as housing, consumer spending, or stocks. Similarly, if the market anticipates a rate hike, investors may shift towards bonds or sectors that are less sensitive to interest rates.

2. Cost of Capital: Financial institutions can use predictions of FOMC policy decisions to manage their cost of capital. A change in interest rates can impact borrowing costs and lending rates, which can affect a bank's profitability. By predicting FOMC decisions, financial institutions can adjust their lending and borrowing rates accordingly to remain competitive and maintain their profitability.

3. Currency Trading: Predicting FOMC policy decisions can also provide value for currency traders. Changes in interest rates can affect currency values, so traders who are able to anticipate the FOMC's actions may be able to take positions that can lead to profit.

4. Corporate Planning: Corporations can use predictions of FOMC policy decisions to inform their financial planning. For example, if a company has plans to issue debt or raise capital, it may want to time these actions around expected interest rate changes to minimise borrowing costs.

5.2 Limitations

Token Size Limitation: The *FinBERT* model has a maximum limitation of only 512 tokens which effects the maximum sequence length that the model can handle.

Lag in information: The beige book is published two weeks before the Federal Reserve's policy meetings, which means that the information it provides may not be as up-to-date as other sources of economic information. Furthermore, beige books measure historical performance and do not hint on any possible future fluctuations in economic performance.

Subjectivity of the Beige Book: The beige book relies on subjective reports from business contacts and other sources, which can be biased or incomplete.

Lack of the Beige Book content: The information provided by the beige books is very balanced and incomplete. This is done on purpose by the Fed as to not cause any excess positive or negative speculation that can have an effect on the market.

Information of a few districts but not of the overall economy: The information content of the beige books is of the current economic conditions across the 12 Federal Reserve District rather than of the overall economy.

6 Conclusion

Based on the findings of the classification random forest model and sentiment analysis using FinBERT on the beige book, it can be concluded that while the models was successful in predicting a constant interest rate policy decision by the FOMC, its performance was weaker in predicting the *increase* and *decrease* interest rate classifications.

The sentiment analysis using FinBERT did not provide much additional explanatory power to the classification problem, suggesting that sentimental inference may not be as helpful in explaining monetary policy decisions as other macro indicators such as CPI and the prevailing Target rate.

However, incorporating additional features and macro indicators may help to improve the model's performance in predicting the *increase* and *decrease* interest rate classifications.

Furthermore, the limitations of the Beige book, including its qualitative nature and limited frequency, may pose challenges for predicting monetary policy decisions accurately. Incorporating additional macroeconomic indicators and data sources may help to overcome these limitations and improve the performance of the model.

Overall, the findings suggest that while the classification random forest model and sentiment analysis using FinBERT can provide valuable insights into economic conditions and trends, incorporating multiple sources of data and indicators is essential for accurate prediction of monetary policy decisions.

7 References

1. Beige Book Sentiment Text Analysis with BERT

2. Board of Governors of the Federal Reserve System

3. Federal Reserve Bank of St. Louis

4. Yahoo! Finance

5. How to get all 3 labels' sentiment from finbert instead of the most likely label's?

6. Financial News Sentiment Analysis using FinBERT

7. yiyanghkust/finbert-tone

8. FinBERT: financial sentiment analysis with BERT

9. An Introduction to Statistical Learning with Applications in R

10. gav-conn

A Appendices - Python Code

Import appropriate libraries

```
import numpy as np
import pandas as pd
import requests
from bs4 import BeautifulSoup
import sqlite3 as lite
import datetime as dt
from datetime import datetime
from datetime import timedelta
from datetime import date
import time as time
import matplotlib.pyplot as plt
import seaborn as sb
import fredapi as fa
import re
import csv
import yfinance as yf
from transformers import AutoTokenizer, AutoModelForSequenceClassification, pipeline
import torch
import statistics
import plotly.graph_objects as go
```

A.1

Beige Book URL Scraping

```
html_link = pd.DataFrame(columns=["Date", "URL"])
x = 0
for year in range(1996, 2023):
    master_url = "https://www.federalreserve.gov/monetarypolicy/beigebook" + str(year) + ".htm"
    print("Processing Beige Book Year: " + str(year) + " via " + master_url)
    time.sleep(1 + 0.06 * np.random.rand())

    page = requests.get(master_url)
    soup = BeautifulSoup(page.content, 'html.parser')
    results = soup.find('table', class_='table table-layout')
    hrefs = results.find_all('a', href=True)
    for href in hrefs:
        if href.has_attr('href') and ('.htm' in href['href']):

            date_url = ((re.findall('([0-9]{6,})', href['href']))[0] + "01")[:8]
            date_report = datetime.strptime(date_url, '%Y%m%d').date()

            if 1996 <= year <= 2010:
                html_link.loc[x, ['Date', 'URL']] =
                [date_report, str(href['href']).replace("default", "FullReport")]
            elif 2011 <= year <= 2016:
                html_link.loc[x, ['Date', 'URL']] = [date_report, href['href']]
            if year >= 2017:
                html_link.loc[x, ['Date', 'URL']] = [date_report, str("https://www.federalreserve.gov")
                                                     + str(href['href'])]

            x = x + 1
```

Return to Report

A.2

Scraping functions for 1996-2010 beige books

```
def get_paragraph_text(bs_tag):
    parent_body = str(bs_tag.parent)
```

```
    result = ''
    if parent_body.find(str(bs_tag)):
        block_after_strong = parent_body[parent_body.find(str(bs_tag))
         + len(str(bs_tag)):]
        result = BeautifulSoup(block_after_strong
        [:block_after_strong.find('<strong>')],
                               'html.parser').get_text().strip()

    return result


def parse_and_strip(line_to_strip):
    try:
        soup_response = BeautifulSoup(' '.
                                      join(line_to_strip.split()), 'html.parser')
    except:
        soup_response = BeautifulSoup(' '.join(line_to_strip.split()).
                                      replace('/', ' '), 'html.parser')
    return soup_response.get_text().strip()


def save_line(url_date, region, sector, comment):
    return [url_date, region.replace(" ", "_").replace(".", ""), sector, comment]


def process_url_1996_2010(df, process_year, report_date, url):
    print(url)
    url_date = (re.findall('([0-9]{8})', url))[0]
    full_report = requests.get(url)
    soup_response = BeautifulSoup(full_report.content, 'html.parser')
    if 1996 <= process_year < 2011:
        tables = soup_response.find_all('table')
        for table in tables:
            tds = table.find_all('td')
            if len(tds) == 1:
                scope = ''
                topic_name = ''
                paragraph = ''

                strongs = str(tds).split('<strong>')
                for strong in strongs:
                    str_strong = str(strong)
                    str_strip = parse_and_strip(str_strong)
                    if '<td>' in str_strong:
                        pass
                    elif '<font size="+1">' in str_strong:
                        if scope != '':
                            df.loc[len(df), ["Date", "Region", "Sector", "Comments"]] = \
                                save_line(report_date, "_" + scope, topic_name, paragraph)

                        district_pos = str_strong[:str_strong.find('</strong>')].find('District')
                        if district_pos > 0:
                            scope = parse_and_strip(str_strong[district_pos
                                                               + 8:str_strong.find('</strong>')]
                                                               ).replace('-', '')
                        else:
                            scope = str_strip  # parse_and_strip(strong)
                        topic_name = 'Summary'
                        paragraph = parse_and_strip(str_strong[str_strong.find('</strong>') + 9:])
                    elif str_strong.find('</strong>') > 100:
                        paragraph = paragraph + ' ' + ' '.join(str_strip.split())
                    else:
                        df.loc[len(df), ["Date", "Region", "Sector", "Comments"]] = \
```

```
                        save_line(report_date, "_" + scope, topic_name, paragraph)
                    end_strong = str_strong.find('</strong>')
                    topic_name = ' '.join(str_strong[:end_strong].split())
                    paragraph = ' '.join(parse_and_strip(str_strong[end_strong:]).split())

            if scope != '':
                df.loc[len(df), ["Date", "Region", "Sector", "Comments"]] = \
                    save_line(report_date, "_" + scope, topic_name, paragraph)
```

Return to Report

A.3

Scraping of the beige book text

```
#Region mapping
page_count = 0
beigebook = pd.DataFrame(columns=["Date", "Region", "Sector", "Comments"])

count = 0
region_list = ["div_summary", "div_boston", "div_new_york", "div_philadelphia",
               "div_cleveland", "div_richmond", "div_atlanta", "div_chicago",
         "div_st_louis", "div_minneapolis", "div_kansas_city", "div_dallas", "div_san_francisco"]

for index, item in html_link.iterrows():
    date = item["Date"]
    date_object = datetime.strptime(str(date), "%Y-%m-%d")
    year = date_object.year

    response1 = requests.get(item["URL"])
    soup2 = BeautifulSoup(response1.content, 'html.parser')
    region_count = 0
    # Code to scrape data for period between 2011-2017
    # Creating a loop to iterate over the different region sections in the HTML
    if 1996 <= year <= 2010:
        process_url_1996_2010(beigebook, year, item["Date"], item["URL"])

    elif 2011 <= year < 2017:
        for region in region_list:
            # time.sleep(1+0.06*np.random.rand())
            region_count = 0
            print(region_count)
            print("Processing Beige Book for :" + str(region) + " " + str(date))
            location = soup2.find('div', {'id': region})
            ptags = location.find_all('p')

            comms = ""
            sector_name = "Overall"

            strong_status = 0  # Binary toggle to assess if the p tags are within a strong tag

            # Creating a loop to iterate over the elements in the Div Tag / Region section
            if ptags is not None:
                for elements in ptags:

                    strong_tag = elements.find("strong")
                    # h_tags = elements.find("h4")

                    if strong_tag is not None:
                        row = row + 1
                        strong_status = 1
                        sector_name = strong_tag.text
                        comms = elements.text
                    else:
```

```
                    strong_status = 0
                    comms = comms + elements.text + " "
                beigebook.loc[len(beigebook), ["Date", "Region", "Sector", "Comments"]] =
                [date, str(region), sector_name,comms]

#         print(beigebook)
#         beigebook.to_csv('beigebook.csv', index=False)
    elif year >= 2017:
        location = soup2.find('div', {'id': "article"})
        tags = ['p', 'a', "h4"]
        ptags = location.find_all(tags)

        comms = ""
        sector_name = "Overall"
        region = region_list[region_count]
        # year_list_item = year_list_item +1

        print("Processing Beige Book for :" + str(region) + " " + str(date))

        for elements in ptags:

            # print("Processing Beige Book for :" +str(region)+" "+str(date))
            strong_tag = elements.find("strong")
            a_tag = elements.find("h4")

            #  To run a check on h4 tag so that the region variable is updated in the dataframe

            if strong_tag is not None:
                row = row + 1
                strong_status = 1
                sector_name = strong_tag.text
                comms = elements.text

            if elements.name == "h4":
                if elements.text == "Highlights by Federal Reserve District":
                    region = region_list[0]
                if elements.text == "Federal Reserve Bank of Boston":
                    region = region_list[0]
                    region_count = 1
                if elements.text == "Federal Reserve Bank of New York":
                    region = region_list[1]
                    region_count = 2
                if elements.text == "Federal Reserve Bank of Philadelphia":
                    region = region_list[2]
                    region_count = 3
                if elements.text == "Federal Reserve Bank of Cleveland":
                    region = region_list[3]
                    region_count = 4
                if elements.text == "Federal Reserve Bank of Richmond":
                    region = region_list[4]
                    region_count = 5
                if elements.text == "Federal Reserve Bank of Atlanta":
                    region = region_list[5]
                    region_count = 6
                if elements.text == "Federal Reserve Bank of Chicago":
                    region = region_list[6]
                    region_count = 7
                if elements.text == "Federal Reserve Bank of St. Louis":
                    region = region_list[7]
                    region_count = 8
                if elements.text == "Federal Reserve Bank of Minneapolis":
                    region = region_list[8]
                    region_count = 9
```

```
                if elements.text == "Federal Reserve Bank of Kansas City":
                    region = region_list[9]
                    region_count = 10
                if elements.text == "Federal Reserve Bank of Dallas":
                    region = region_list[10]
                    region_count = 11
                if elements.text == "Federal Reserve Bank of San Francisco":
                    region = region_list[11]
                    region_count = 12
                print(elements.text)
                print(region)

            else:
                strong_status = 0
                comms = comms + elements.text + " "
                region = region_list[region_count]

            beigebook.loc[len(beigebook), ["Date", "Region", "Sector", "Comments"]] =
                                [date, str(region), sector_name, comms]

#Data Cleaning
beigebook["Region"] = beigebook["Region"].str.split("_").apply(lambda x: x[1:])
beigebook["Region"] = beigebook["Region"].apply(lambda x: [s.capitalize() for s in x])
beigebook["Region"] = beigebook["Region"].apply(lambda x: " ".join(x))
```

Return to Report

A.4

Scraping FOMC Meeting Dates

```
fomc= pd.DataFrame(columns=["Date"])

row = 0

for  year in range(1996, 2018):

     master_url = "https://www.federalreserve.gov/monetarypolicy/fomchistorical" + str(year)
                                                          + ".htm"
     time.sleep(1+0.06*np.random.rand())
     page = requests.get(master_url)
     soup = BeautifulSoup(page.content, 'html.parser')
     h5 = soup.find_all('h5')

     if year>=2011:

        master_div = soup.find_all('div',class_='row divided-row')

     else:

        master_div = soup.find_all('div',class_='row divided-row panel-body')

     for y in master_div:

        results = y.find_all('p' )

        for x in results:

          atags=x.a

          if year<2008:

               if year == 2007:
```

```
                for htags in h5:

                    if "Meeting" in htags.text:

                            if "meeting.pdf" in atags['href']:
                                dates = [atags['href'].split("meeting")[0][-8:]]
                                fomc.loc[row,"Date"] = datetime.strptime(dates[0],
                                "%Y%m%d").date()
            else:

                if atags.text =="Minutes":

                    dates = [atags['href'].split(".htm")[0][-8:]]
                    dates[0].strip('[]')

                    fomc.loc[row,"Date"] = datetime.strptime(dates[0], "%Y%m%d").date()

        elif year>=2008:

            if "Minutes" and "Released" in x.text:

                    dates = [atags['href'].split(".htm")[0][-8:]]
                    dates[0].strip('[]')
                    fomc.loc[row,"Date"] = datetime.strptime(dates[0], "%Y%m%d").date()

        row = row + 1

url = "https://www.federalreserve.gov/monetarypolicy/fomccalendars.htm"
time.sleep(1+0.06*np.random.rand())
page = requests.get(url)
soup = BeautifulSoup(page.content, 'html.parser')
new = soup.find_all('div',class_='col-xs-12 col-md-4 col-lg-2')

for n in new:

   atagz = n.find_all('a', href = True)

   for a in atagz:

      if a.text == 'PDF':
         datez = [a['href'].split("a1")[0][-8:]]
         fomc.loc[row,"Date"] = datetime.strptime(datez[0].strip('[]'), "%Y%m%d").date()
         row += 1
fomc = fomc.drop_duplicates(subset = "Date") #Drops the duplicates
#Sorts the dates, as from 2018 onwards, dates were descending
fomc = fomc.sort_values('Date', ascending=True)

#Drop FOMC Meeting Dates that don't have a corresponding beige book or
beige book doesn't give sentiment score

fomc['Date'] = pd.to_datetime(fomc['Date'])

drop_dates = ['2020-03-03', '2020-03-23', '2003-09-16', '2019-10-11', '2000-08-22',
'2001-11-06', '2000-06-28']

fomc = fomc[~fomc['Date'].isin(drop_dates)]
#drop dates to match end 1996-end of 2022 range
fomc = fomc.drop(index=fomc.index[:6].union(fomc.index[-2:]))
```

Return to Report

A.5

Sentiment Analysis

```
model = AutoModelForSequenceClassification.from_pretrained('ProsusAI/finbert', num_labels=3)
tokenizer = AutoTokenizer.from_pretrained('ProsusAI/finbert')
text = beigebook.iloc[:, 3]
for index, r in text.iteritems():
    try:
        sentences = str(r)
        inputs = tokenizer(r, return_tensors="pt")
        tokens = tokenizer.tokenize(sentences)
        print('Carrying out sentiment analysis of row no :'index)

        if len(tokens) > 512:
            words = sentences.split()
            mid = len(words)//2
            sent_1 = ' '.join(words[:mid])
            sent_2 = ' '.join(words[mid:])
            input_sent_1 = tokenizer(sent_1,return_tensors="pt")
            input_sent_2 = tokenizer(sent_2,return_tensors="pt")

            tokens1 = tokenizer.tokenize(sent_1)
            tokens2 = tokenizer.tokenize(sent_2)

            with torch.no_grad():
                logits1 = model(**input_sent_1).logits
            scores1 = {k: v for k, v in zip(model.config.id2label.values(),
                                scipy.special.softmax(logits1.numpy().squeeze()))}
            with torch.no_grad():
                logits2 = model(**input_sent_2).logits
            scores2 = {k: v for k, v in zip(model.config.id2label.values(),
                                scipy.special.softmax(logits2.numpy().squeeze()))}

            avg_pos = (((len(tokens1)/(len(tokens1)+len(tokens2)))*scores1['positive'])
            + ((len(tokens2)/(len(tokens1)+len(tokens2)))*scores2['positive']))
            avg_neg = (((len(tokens1)/(len(tokens1)+len(tokens2)))*scores1['negative'])
            + ((len(tokens2)/(len(tokens1)+len(tokens2)))*scores2['negative']))
            avg_neu = (((len(tokens1)/(len(tokens1)+len(tokens2)))*scores1['neutral'])
            + ((len(tokens2)/(len(tokens1)+len(tokens2)))*scores2['neutral']))
            beigebook.loc[index,['Positive','Negative','Neutral']]=[avg_pos,avg_neg,avg_neu]

        else:
            with torch.no_grad():
                logits = model(**inputs).logits
            scores = {k: v for k, v in zip(model.config.id2label.values(), scipy.special.
                                softmax(logits.numpy().squeeze()))}
            beigebook.loc[index,['Positive','Negative','Neutral']]=[scores['positive'],
                                scores['negative'],scores['neutral']]

    except Exception as e:
        print(f"Error occurred at row {index}: {e}")
        continue
# Adds a column with the Sentiment_index
beigebook['Sentiment_index'] = ((beigebook['Positive'] - beigebook['Negative'])/2)
 # Sentiment analysis i.e. Positive, Negative and Neutral
# beigebook.to_csv('beigebook_alpha.csv', index = False)
```

Return to Report

A.5.1

Sentiment Analysis Data Cleaning

```
#Cleaning up the sector column
sector_keys= pd.read_csv("Sector_keys.csv")
for index , i in sector_keys.iterrows():
    beigebook['Sector'] = beigebook['Sector'].replace(i[0],i[1])

beigebook['Sector'] = beigebook['Sector'].str.replace("\xa0",'')
beigebook['Sector'] = beigebook['Sector'].str.replace("\n",'')
beigebook = beigebook.dropna()
beigebook = beigebook[beigebook['Sector'] != 'Null']

# Cleaning up the region data
reg_keys= pd.read_csv("Region Keys.csv")
for index , i in reg_keys.iterrows():
    beigebook['Region'] = beigebook['Region'].replace(i[0],i[1])

beigebook['Region'] = beigebook['Region'].str.replace("_",'')
beigebook['Region'] = beigebook['Region'].str.replace("Summary",'')
beigebook = beigebook[beigebook['Sector'] != 'Consumer spending']
```

A.6

Macro Data Collection, Cleaning and Processing

```
fred = fa.Fred(api_key='76c20b6988354509c6574aee17017680')
cpi = fred.get_series('CPIAUCSL', observation_start='1996-01-01',
                      observation_end='2023-04-01')
consconf = fred.get_series('CSCICP03USM665S', observation_start='1996-01-01',
                      observation_end='2023-04-01')
ppi = fred.get_series('CSCICP03USM665S', observation_start='1996-01-01',
                      observation_end='2023-04-01')
targetnew = pd.DataFrame(fred.get_series('DFEDTARU', observation_start='1996-01-01',
                      observation_end='2023-04-01'))
targetold = pd.DataFrame(fred.get_series('DFEDTAR', observation_start='1996-01-01',
                      observation_end='2023-04-01'))
sp500 = yf.download("^GSPC", start="1996-01-01", end="2023-04-18")
sp500.index = sp500.index.date
df1 = pd.concat([targetold, targetnew], axis = 0 ) #Merge older and new Target Rate tickers
df1.columns = ['Target Rate']
df1 = df1.rename_axis('Date')
df1['S&P500'] = sp500['Adj Close']
df2 = pd.DataFrame({'CPI': cpi, 'PPI': ppi, 'Consumer Confidence': consconf})
df2.index.name = 'Date'

fomc['Date'] = pd.to_datetime(fomc['Date'])
fomc.index = fomc['Date']
#Match Fed Target Rate to the corresponding FOMC meeting date
fomc['Date'] = pd.to_datetime(fomc['Date'])
fomc.index = fomc['Date']
for index1, row1 in fomc.iterrows():
    if fomc.loc[index1, 'Date'] in df1.index:
        fomc.loc[index1, 'Target Rate'] = df1.loc[fomc.loc[index1, 'Date'] , 'Target Rate']
#Create column w the delta of the interest rate
fomc['Delta'] = fomc['Target Rate'].diff()

#Match target rate with the rows with the corresponding beige book dates
html_link = pd.concat([html_link, fomc[['Target Rate', 'Delta']].reset_index(drop = True)], axis = 1)
html_link['Date'] = pd.to_datetime(html_link['Date'])
html_link.index = html_link['Date']

#Match target rate with the rows with the corresponding beige book dates
for index, row in beigebook.iterrows():
```

```
    if beigebook.loc[index, 'Date'] in html_link.index:
        # print(beigebook.loc[index, 'Date'])
        beigebook.loc[index, 'Target Rate'] = html_link.loc
        [beigebook.loc[index, 'Date'], 'Target Rate']
        beigebook.loc[index, 'Delta'] = html_link.loc
        [beigebook.loc[index, 'Date'], 'Delta']
print(beigebook)

beigebook['Date'] = pd.to_datetime(beigebook['Date'])

# Match macro data with the rows with the corresponding beige book dates
for index, row in beigebook.iterrows():
    date = row['Date']
    if beigebook.loc[index, 'Date'] in df1.index:
        #Code to give next working day if beigebook is released on a weekday
        if beigebook.loc[index,'Date'].weekday() == 5:
            beigebook.loc[index, 'S&P500'] = df1.loc[beigebook.loc[index, 'Date'] +
                                timedelta(days=2), 'S&P500']
            if np.isnan(df1.loc[beigebook.loc[index, 'Date'] + timedelta(days=1), 'S&P500']):
                beigebook.loc[index, 'S&P500'] = df1.loc[beigebook.loc[index, 'Date'] +
                                                timedelta(days=3), 'S&P500']
        elif beigebook.loc[index,'Date'].weekday() == 6:
            beigebook.loc[index, 'S&P500'] = df1.loc[beigebook.loc[index, 'Date'] +
                                            timedelta(days=1), 'S&P500']
        elif np.isnan(df1.loc[beigebook.loc[index, 'Date'], 'S&P500']):
                beigebook.loc[index, 'S&P500'] = df1.loc[beigebook.loc[index, 'Date']
                                                    + timedelta(days=1), 'S&P500']
            #in case there is a two day gap that's due to a weekend
        else:
            beigebook.loc[index, 'S&P500'] = df1.loc[beigebook.loc[index, 'Date'], 'S&P500']
    mask = np.logical_and(df2.index.year == date.year, df2.index.month == date.month)
    if mask.any():

        beigebook.loc[index, 'CPI'] = df2.loc[mask, 'CPI'].values[0]
        beigebook.loc[index, 'PPI'] = df2.loc[mask, 'PPI'].values[0]
        beigebook.loc[index, 'Consumer Confidence'] = df2.loc[mask, 'Consumer Confidence'].values[0]

#Creates column 'Movement' for the movement of the Target Rate
c = [(beigebook['Delta'] > 0),
     (beigebook['Delta'] < 0),
     (beigebook['Delta'] == 0)]
values = [1, -1, 0]
beigebook['Movement'] = np.select(c, values)
```

Return to Report

A.7

Database Creation, Querying and Summary Statistics

```
con = lite.connect('Beigebook.db')
beigebook.to_sql('Beigebook', con, if_exists='replace', index=False)
cur = con.cursor()
cur.execute("""CREATE TABLE Beigebook (Date date, Region TEXT, Sector TEXT,
Comments TEXT, Positive INT, Negative INT, Neutral INT, Sentiment_index INT,
Target Rate INT, Delta INT, Movement INT,
            S&P500 INT, CPI INT, PPI INT, Consumer Confidence INT)""");
            cur.execute("INSERT OR IGNORE INTO Beigebook VALUES (?, ?, ?, ?
            , ?, ?,  ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);", beigebook)

# ------------------------------------------------------------------------------------------------------------
# SUMMARY STATS + BOX PLOT OF THE POSITIVES, NEGATIVES AND NEUTRALS ALONE
# Count
```

```
query_count = """SELECT COUNT(Positive) AS Total_pos, COUNT(Negative)
        AS Total_neg, COUNT(Neutral) AS Total_neu
                FROM Beigebook"""
query_observations = pd.read_sql_query(query_count, con)
obs_df = pd.DataFrame({'Positive': query_observations['Total_pos'], 'Negative':
                    query_observations['Total_neg'], 'Neutral': query_observations['Total_neu']})

# Mean: For Positives, Negatives and Neutrals
query_average = """SELECT AVG(Positive) AS Avg_pos, AVG(Negative) AS Avg_neg, AVG(Neutral) AS Avg_neu
                FROM Beigebook"""
query_mean = pd.read_sql_query(query_average, con)
mean_df = pd.DataFrame({'Positive': query_mean['Avg_pos'], 'Negative':
                    query_mean['Avg_neg'], 'Neutral': query_mean['Avg_neu']})

# Median: For Positives, Negatives and Neutrals
query_med = """SELECT Positive, Negative, Neutral
            FROM Beigebook
            ORDER BY Positive, Negative, Neutral ASC"""
query_median = pd.read_sql_query(query_med, con)
positive_median = query_median['Positive'].median()
negative_median = query_median['Negative'].median()
neutral_median = query_median['Neutral'].median()
median_df = pd.DataFrame({'Positive': [positive_median], 'Negative':
            [negative_median], 'Neutral': [neutral_median]})

# Minimum and maximum: For the Positives, Negatives and Neutrals
query_maxi = """SELECT MAX(Positive) AS Max_pos, MAX(Negative) Max_neg, MAX(Neutral) AS Max_neu
            FROM Beigebook"""
query_max = pd.read_sql_query(query_maxi, con)
max_df = pd.DataFrame({'Positive': query_max['Max_pos'], 'Negative': query_max['Max_neg'].
                                    'Neutral': query_max['Max_neu']})

query_mini = """SELECT MIN(Positive) AS Min_pos, MIN(Negative) AS Min_neg, MIN(Neutral) AS Min_neu
            FROM Beigebook"""
query_min = pd.read_sql_query(query_mini, con)
min_df = pd.DataFrame({'Positive': query_min['Min_pos'],
                'Negative': query_min['Min_neg'], 'Neutral': query_min['Min_neu']})

# Standard deviation: For the Positives, Negatives and Neutrals
query_stand_dev = """SELECT Positive, Negative, Neutral
                FROM Beigebook"""
query_std = pd.read_sql_query(query_stand_dev, con)
positive_std = query_std['Positive'].std()
negative_std = query_std['Negative'].std()
neutral_std = query_std['Neutral'].std()
std_df = pd.DataFrame({'Positive': [positive_std], 'Negative':
                    [negative_std], 'Neutral': [neutral_std]})

# Summary Table
sum_mat = pd.concat([obs_df, mean_df, median_df, std_df, max_df, min_df], axis = 0)
sum_mat.index = ['Observations', 'Mean', 'Median', 'Std. Dev.', 'Maximum', 'Minimum']
sum_mat.columns = ['Positive', 'Negative', 'Neutral']
lat_summat = sum_mat.to_latex()
print(lat_summat)

# Section 4: Data visualisation - include nice charts and plots of the
#data set and, if relevant, some visual analysis.
Positive = beigebook['Positive']
Negative = beigebook['Negative']
Neutral = beigebook['Neutral']

fig = go.Figure()
fig.add_trace(go.Box(y=Positive, name = 'Positive'))
```

```
fig.add_trace(go.Box(y=Negative, name = 'Negative'))
fig.add_trace(go.Box(y=Neutral, name = 'Neutral'))
fig.update_layout(title='Box Plot for Sentiment Scores')
fig.show()
# ------------------------------------------------------------------------
# SUMMARY STATS + BOX PLOT OF THE SECTORS ALONE
# Mean: For Sectors
query_average_sec = """SELECT Sector, AVG(Sentiment_Index) AS Mean_sector
                FROM Beigebook
                GROUP BY Sector"""
query_mean_sec = pd.read_sql_query(query_average_sec, con)
query_mean_sec.columns = ['Mean_sector']
query_mean_sec = query_mean_sec.set_index('Sector')

# Median: For Sectors
query_med_sec = """SELECT Sector, Sentiment_Index AS Median_sector
            FROM Beigebook
            GROUP BY Sector"""
query_median_sec = pd.read_sql_query(query_med_sec, con)
med_sec = query_median_sec.groupby('Sector')['Median_sector'].median().to_frame()
med_sec.columns = ['Median_sector']

# Minimum and maximum: For Sectors
query_maxi_sec = """SELECT Sector, MAX(Sentiment_Index) AS Maximum_sector
            FROM Beigebook
            GROUP BY Sector"""
query_max_sec = pd.read_sql_query(query_maxi_sec, con)
query_max_sec.columns = ['Maximum_sector']
query_max_sec = query_max_sec.set_index('Sector')

query_mini_sec = """SELECT Sector, MIN(Sentiment_Index) AS Minimum_sector
            FROM Beigebook
            GROUP BY Sector"""
query_min_sec = pd.read_sql_query(query_mini_sec, con)
query_min_sec.columns = ['Minimum_sector']
query_min_sec = query_min_sec.set_index('Sector')

# Standard deviation: For Sectors
query_stdev_sec = """SELECT Sector, Sentiment_Index AS Standard_Deviation_sector
        FROM Beigebook"""
query_std_sec = pd.read_sql_query(query_stdev_sec, con)
std_sector = query_std_sec.groupby('Sector')['Standard_Deviation_sector']
                        .std().to_frame()
std_sector.columns = ['Standard_Deviation_sector']

# Summary Table for Sector Statistics
sum_mat_sec = pd.concat([query_mean_sec, med_sec, std_sector,
                    query_max_sec, query_min_sec], axis=1)
sum_mat_sec.index = ['Region']
sum_mat_sec.columns = ['Mean', 'Median', 'Std. Dev.', 'Maximum', 'Minimum']
lat_summatsec = sum_mat_sec.to_latex()
print(lat_summatsec)
# ------------------------------------------------------------------------
# SUMMARY STATS + BOX PLOT OF THE REGIONS ALONE
# Mean: For Regions
query_average_reg = """SELECT Region, AVG(Sentiment_Index) AS Mean_region
                FROM Beigebook
                GROUP BY Region"""
query_mean_reg = pd.read_sql_query(query_average_reg, con)
query_mean_reg.columns = ['Mean_region']
query_mean_reg = query_mean_reg.set_index('Region')

# Median: For Regions
```

```
query_med_reg = """SELECT Region, Sentiment_Index AS Median_region
            FROM Beigebook
            GROUP BY Region"""
query_median_reg = pd.read_sql_query(query_med_reg, con)
med_reg = query_median_reg.groupby('Region')['Median_region'].median().to_frame()
med_reg.columns = ['Median_region']

# Minimum and maximum: For Regions
query_maxi_reg = """SELECT Region, MAX(Sentiment_Index) AS Maximum_region
            FROM Beigebook
            GROUP BY Region"""
query_max_reg = pd.read_sql_query(query_maxi_reg, con)
query_max_reg.columns = ['Maximum_region']
query_max_reg = query_max_reg.set_index('Region')

query_mini_reg = """SELECT Region, MIN(Sentiment_Index) AS Minimum_region
            FROM Beigebook
            GROUP BY Region"""
query_min_reg = pd.read_sql_query(query_mini_reg, con)
query_min_reg.columns = ['Minimum_region']
query_min_reg = query_min_reg.set_index('Region')

# Standard deviation: For Regions
query_stdev_reg = """SELECT Region, Sentiment_Index AS Standard_Deviation_region
        FROM Beigebook"""
query_std_reg = pd.read_sql_query(query_stdev_reg, con)
std_region = query_std_reg.groupby('Region')['Standard_Deviation_region'].std().to_frame()
std_region.columns = ['Standard_Deviation_region']

# Summary Table for Region Statistics
sum_mat_reg = pd.concat([query_mean_reg, med_reg, std_region, query_max_reg, query_min_reg], axis=1)
sum_mat_reg.index = ['Region']
sum_mat_reg.columns = ['Mean', 'Median', 'Std. Dev.', 'Maximum', 'Minimum']
lat_summatreg = sum_mat_reg.to_latex()
print(lat_summatreg)
```

Return to Report

A.8

Data Visualization

```
# Box plot for positive, negative, neutral
Positive = beigebook['Positive']
Negative = beigebook['Negative']
Neutral = beigebook['Neutral']

fig = go.Figure()
fig.add_trace(go.Box(y=Positive, name = 'Positive'))
fig.add_trace(go.Box(y=Negative, name = 'Negative'))
fig.add_trace(go.Box(y=Neutral, name = 'Neutral'))
fig.update_layout(title='Box Plot for Sentiment Scores')
fig.show()
# ------------------------------------------------------------------------------------------------------------
# Sentiment index plot from 1996 to 2022
import plotly.graph_objs as go
import pandas as pd
sentiment_index = beigebook['Sentiment Index']
date = beigebook['Date']
fig1 = go.Figure()
fig1.add_trace(go.Scatter(x=date, y=sentiment_index, mode='markers',
                          marker=dict(color=sentiment_index, colorscale='Viridis', line_width=1)))
fig1.update_layout(title='Sentiment Index over Time',
                   xaxis_title='Date',
```

```
                  yaxis_title='Sentiment Index')
fig1.show()
# ------------------------------------------------------------------------------------------------------------
# Target rates plot from 1996 to 2022
target_rate = beigebook.iloc[:, 8]
date = beigebook.iloc[:, 0]
fig2 = go.Figure()
fig2.add_trace(go.Scatter(x=date, y=target_rate, mode='lines'))
fig2.update_layout(title='Target Rate over Time',
                  xaxis_title='Date',
                  yaxis_title='Target Rate')
fig2.show()
# ------------------------------------------------------------------------------------------------------------
# Positive plot from 1996 to 2022
sent_pos = beigebook['Positive']
date = beigebook['Date']
fig3 = go.Figure()
fig3.add_trace(go.Scatter(x=date, y=sent_pos, mode='markers',
                          marker=dict(color=sent_pos, colorscale='inferno', line_width=1)))
fig3.update_layout(title='Positive Sentiment Score over Time',
                  xaxis_title='Date',
                  yaxis_title='Positive Sentiment Score')
fig3.show()
# Negative plot from 1996 to 2022
sent_neg = beigebook['Negative']
date = beigebook['Date']
fig4 = go.Figure()
fig4.add_trace(go.Scatter(x=date, y=sent_neg, mode='markers',
                          marker=dict(color=sent_neg, colorscale='blues', line_width=1)))
fig4.update_layout(title='Negative Sentiment Score over Time',
                  xaxis_title='Date',
                  yaxis_title='Negative Sentiment Score')
fig4.show()

# Neutral plot from 1996 to 2022
sent_neu = beigebook['Neutral']
date = beigebook['Date']
fig5 = go.Figure()
fig5.add_trace(go.Scatter(x=date, y=sent_neu, mode='markers',
                          marker=dict(color=sent_neu, colorscale='greys', line_width=1)))
fig5.update_layout(title='Neutral Sentiment Score over Time',
                  xaxis_title='Date',
                  yaxis_title='Neutral Sentiment Score')
fig5.show()
```

Return to Report

A.9

Machine Learning

```
#Correlation heatmap plotting
# Creating training and test data
df= beigebook
print(df.dtypes)


df['Date'] = pd.to_datetime(df['Date'])
df = df.sort_values(by='Date')
split_date = '2012-01-01'

train_data = df[df['Date'] < split_date]
test_data = df[df['Date'] >= split_date]
```

```
# One hot encoding of regions
# one_hot_region = pd.get_dummies(df['Region'])
# df=pd.concat([df,one_hot_region],axis=1)

# One hot encoding of sectors
# one_hot_sector = pd.get_dummies(df['Sector'])
# df=pd.concat([df,one_hot_sector],axis=1)

exclude_cols= ['Movement','Comments','Unnamed: 0','Sector','Region','Date','Delta','PPI','S&P500','Cons
predictors = [column for column in df.columns if column != 'Movement']
predictors =[column for column in df.columns if column not in exclude_cols]


# Splitting the training and testing data
# df_train_x, df_test_x, df_train_y, df_test_y = train_test_split(df[predictors], df['Movement'], test_


#Splitting the train and test data based on date
df_train_x=train_data[predictors]
df_train_y=train_data['Movement']
df_test_x=test_data[predictors]
df_test_y=test_data['Movement']

# Splitting the training and testing data
# df_train_x, df_test_x, df_train_y, df_test_y = train_test_split(df[predictors], df['Movement'], test_


# Random Forest model creation
RF = RandomForestClassifier()
paramgrid = {
             'n_estimators': [20,100,150,250,300,500],
             'criterion': ['gini', 'entropy'],
             'max_depth': [3, 5, 7, 10, 15],
             }
G_CV = GridSearchCV(RF, param_grid=paramgrid, scoring='f1_macro', n_jobs=-1, cv=5)
G_CV.fit(df_train_x, df_train_y)

# neg_mean_absolute_error
best_params = G_CV.best_params_

print("Best parameters:", best_params)
print("Best Score:",{G_CV.best_score_})

rfc_best = RandomForestClassifier(n_estimators=best_params['n_estimators'],
                                  criterion=best_params['criterion'],
                                  max_depth=best_params['max_depth'])

rfc_best.fit(df_train_x, df_train_y)

# Testing the model
results = rfc_best.predict(df_test_x)
RF_prob_est = rfc_best.predict_proba(df_test_x)
print(results)
cm = confusion_matrix(df_test_y, results)
cm_df = pd.DataFrame(cm,
                     index = ['Increase','Constant','Decrease'],
                     columns = ['Increase','Constant','Decrease'])

# Plotting the feature importance
features = df_train_x.columns

importances = rfc_best.feature_importances_
```

```
indices = np.argsort(importances)
plt.title('Feature Importances')
indices = indices[:20]
plt.barh(range(len(indices)), importances[indices], align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()


#Plotting the confusion matrix
plt.figure(figsize=(5,4))
sns.heatmap(cm_df, annot=True, fmt='d')
plt.title('Confusion Matrix')
```

Return to Report