mv3p

December 14, 2023

0.0.1 Overview

The Federal Reserve (Fed) is a United States governmental institution that serves to keep the national economy stable and under control. By far the most powerful and influntial economic organization in the nation, the Fed is tasked with promoting "maximum employment, stable prices, and moderate long-term interest rates" [1]. Significantly, while this purpose statement contains three distinct items, the Fed's mission is commonly referred to as the "dual mandate". This is because moderating long-term interest rates actually forms the mechanism by which employment is maximized and inflation is curved.

In particular, since long-term interest rates essentially represent the cost of borowing money, a change in the prevailing interest rate significantly impacts levels of personal expenditures as well as corporate investment. For example, when interest rates are relatively high, consumers have greater incentive to save money (and earn interest) rather than spend money, while companies have less incentive to invest (since they must pay a higher interest rate on their loans). This leads to an overall cooling in the economy. Conversely, lower interest rates increase personal consumption and corporate investment, causing the economy to heat up.

Long-term interest rates also have important implications for personal and institutional investment. For example, since bond prices are known to have an inverse relationship with bond prices, portfolios that are heavy in bonds lose value quickly when interest rates spike, leading to a painful loss in returns for investors [2]. In addition, even equity portfolios are negatively impacted by higher interest rates; since companies will spend less on new projects when the cost of borrowing is high, corporate profits taper, resulting in lower stock prices.

In the United States, the most important long-term interest rate is the federal funds rate (FFR), a target interest rate set by the Fed. When the Fed sets a new FFR, the Fed moves the prevailing interest rate observed in the economy towards this target interest rate through means such as requiring banks to keep more or less money in reserve or adjusting the number of federal bonds available on the market (thus increasing or decreasing the supply of money available for borrowing) [3]. As a result, the FFR almost always becomes the prevailing interest rate in the economy. Thus the FFR, through its effect on each of the personal and corporate spending and investment decisions discussed previously, has dramatic effects on a myriad of important economic indicators, including inflation and unemployment.

0.0.2 Research Objectives

Due to the power that the FFR wields in the U.S. economy, the ability to predict changes in the FFR would allow consumers, corporations, and investment managers to make optimal financial decisions. For this reason, much effort has been expended to predict changes in the FFR. The Fed has shared

some indicators that it considers in its decisions, including various price indices and labor market indicators, such as the annual change in the price index for personal consumption expenditures [1]. However, it does not explicitly share the way it considers or weights these metrics. Other groups that have attempted prediction, such as the Blue Chip consensus, are relatively successful at predicting changes in the FFR when most or all economic indicators are optimal (or adverse), but are inconsistent in accuracy when economic conditions are mixed [4]. We seek to build upon this work.

Our objective here is to use the techniques of regression and classification to develop a more robust model for predicting changes in the FFR. We are also concerned with finding features that best predict FOMC decisions to raise, lower, or hold constant the FFR. In short, we seek to answer these three key questions: 1) What features have the most predictive power for forecasting FOMC decisions? 2) What regression and classification methods are best suited to forecasting these decisions? 3) How accurately can we predict FOMC decisions?

0.0.3 Our Data

We started our search for data on key economic indicators that might influence the FFR by looking to the St. Louis Federal Reserve (FRED) website [5]. This site hosts an enormous quantity of time series data on economic indicators. This data has been carefully and methodically tracked, and is a gold standard for economic data. Many reputable news sources, including the Wall Street Journal, frequently cite FRED data. In selecting features, we focused on features with data continuously tracked since 1989 (the year our FOMC decisions data starts) to the present day. We also chose features that we felt were most likely to have predictive power in determining the FOMC's decision to raise or lower interest rate. With these two considerations in mind, we selected the several features which are summarized in the table below.

Feature	Variable	Frequency	Description
Bank Prime Loan Rate Changes	`loan`	daily	Rate charged by banks for short-term loans to creditworthy debtors
Exports of Goods and Services	`export`	quarterly	Total dollar value of goods and services exported in the quarter
Personal Consumption Expenditures Rate	`pce`	monthly	Measure of core inflation for personal expenditures
Unemployment Rate	`ue`	monthly	Number of unemployed as a percentage of the labor force
Change in Real GDP	`rgdp`	quarterly	Quarterly change in inflation-adjusted GDP
Total Vehicle Sales	`cars`	monthly	Total number of vehicle sales in millions

Feature	Variable	Frequency	Description
Recession Indicator	`recess`	monthly	Binary variable indicating whether the US is in a recession
GDP Deflator	`gdpd`	daily	Price index given by ratio of nominal GDP to real GDP
Velocity of M1 Money	`veloc`	quarterly	Ratio of nominal GDP to the quarterly average of M1 money stock
New Private Housing Units Started	`house`	monthly	Number of new housing units beginning construction in millions

In addition, we hypothesized that stock market performance and the strength of the dollar might be important factors in the FOMC's decision. The stock market acts as a barometer for the health of the economy, and the strength of the dollar indicates the strength of the economy relative to other countries. Thus, we used daily closing price data from Yahoo! Finance for the S&P 500 (`spx`, a proxy for total stock market) and US Dollar (`usd`, the value of the USD versus a basket of foreign currencies) Indices.

Considering the differences in opinion on economic policy that undoubtedly factor into the FOMC's decisions to raise or lower interest rates, we also generated binary variables for the political party of the Fed Chair (`fed_party`) and President of the United States (`potus_party`) [6], [7]. These variables are equal to 1 when the position was held by a Republican and 0 when held by a Democrat. Since this data is undisputed and readily accessible from several sources, we used a dataset available on Wikipedia after double-checking its veracity.

Our final explanatory feature, `cli`, was pulled from The Organisation for Economic Co-operation and Development (OECD) [8]. This organization is a multinational group founded in 1961 for the purpose of stimulating economic progress and world trade [9]. Given its status as a United Nations observer, we are confident in the validity of its data. The composite leading indicator (CLI) is a statistic created to "provide early signals of turning points in business cycles". Hence, the Fed would likely care about proactively adjusting interest rates in response to changes in this indicator.

Finally, our data on the FFR (`ffr`) came from the research of an economics professor at Williams University. Kenneth Kuttner is a leading researcher in monetary policy, and the data we downloaded from his website was the backbone of a paper he published in the Journal of Finance (the top journal in academic finance) [10]. It contains information on the percentage point change (`change`) at each announcement and the direction of each change (`direction`). The direction is indicated by -1 for a decrease, 0 for no change, and 1 for an increase in the FFR.

0.0.4 Data Cleaning and Feature Engineering

The data is pulled from five primary sources: FRED, Yahoo! Finance, OECD, Wikipedia, and Dr. Kuttner's website. FRED and Yahoo! Finance data is easily pulled using their API by entering a specific data series identifier. The OECD data was manually downloaded as a CSV and loaded. Wikipedia data on political parties was inputted by hand and also loaded from CSV. The FFR data from Dr. Kuttner's research page was downloaded as a CSV and loaded using pandas. The details of this process, including merging the data with different frequences, is demonstrated and explained in the Appendix. Given that our data comes from publicly accessible websites, we were able to use it without any legal problems.

The datasets we obtained were well-maintained, containing no missing data or invalid values. However, our data did not agree in frequency. For example, some variables including GDP and exports are reported quarterly, while others such as personal expenditures and the composite leading indicator are reported monthly. Further, a couple features (stock closing price and dollar index) change daily.

In addition, we realized that using the current day's values to predict changes in the federal funds interest rate would result in "look-ahead" bias. For example, it is unrealistic to believe that the closing stock price at 4:00 PM in New York on a given day would have any effect on an FOMC decision in Washington D.C. made at noon earlier that day. More likely, today's decisions are influenced by data ranging from one to several days prior.

To resolve these issues, we first filled in all empty days between reports of monthly and quarterly reported variables with the value from the previous report so we had values for all features for each day. We then used our `collapse()` function, which took our forward filled dataframe of over 12000 rows, shifted the data by a few days (to avoid look-ahead bias), and returned an aggregated dataframe with just 275 rows corresponding to scheduled FOMC announcement days. (We exclude unscheduled FOMC announcements because people do not know they are happening until after they have been announced, and because nearly every unscheduled FOMC meeting has led to a change in rates. They are often held for emergency reasons, such as in the case of COVID-19.)

To bolster our model, we introduced new features coresponding to the change from the previous announcement period to the current period for all previous noncategorical features, allowing regression and classification on not only the current point estimate but also the period trend for each numerical feature. (Our Categorical features were excluded because they change only over longer time horizons.) This essentially doubles the number of features in the dataset. This trend feature is obtained by running a linear regression of the feature on time over the period. We generate these features as we believe that decisions to change the FFR are driven by not only the curent economic outlook, but how the outlook has changed since the last decision.

0.0.5 Robustness

The data cleaning and feature engineering code found above is flexible and simple to modify. One can easily add not only new features but also recent FOMC decisions. Adding new data series from FRED and Yahoo! Finance is the easiest modification. Simply add the identifying name of the feature as well as the series code for FRED or the ticker for Yahoo! Finance, and the for loops will handle the rest. Adding data from outside CSV files only requires adding the name of the new dataframe to the list when creating `df_features`. Thus, any number of new features (columns) from a variety of sources can be quickly added. By virtue of the `ffill()` function, any data pulled

using the FRED or Yahoo! Finance API is properly extended to the last row of the dataframe, which corresponds to the most recent FOMC announcement we have data for.

New FOMC announcement data can be appended by modifying the ffr_clean.csv file. This is done by adding the new target Fed Funds Rate under the `ffr` column and the change in basis points under the `change` column. In addition, the `scheduled` column should be entered as 1 if the meeting was scheduled and 0 if it was a surprise. The columns `decision` and `mom` are calculated automatically. It is critical to note that new FOMC announcements should be added sequentially and even if the meeting was unscheduled in order to keep the data consistent. Modifying this data is more of a hassle than adding new features, but meetings generally only occur every six weeks and require the addition of only three data points to the CSV file. All things considered, this is an insignificant burden to adding more data points. In addition, when new data points are added, the data coming from FRED and Yahoo! Finance automatically pull from 1989 until the present day. It is the responsibility of the researcher to ensure any data coming from CSV files is still accurate when adding new FOMC announcement data, as this will not automatically refresh, but autofill from the most recent valid entry using the `ffill()` function.

### 0.0.6 Data Visualization and Basic Analysis

The outcomes we are seeking to explain and predict are change in FFR and the direction of that change. After an initial pass through the data, we found that for the 257 FFR announcements since 1989, the FFR was increased 53 times, was decreased 33 times, and was not changed 188 times (68% of all instances). Since the naive prediction method of always predicting no change (the most common outcome) yields an accuracy score of 68% historically, this score serves as a baseline by which we can measure the success of our models and methods. In short, any method that has accuracy equal to or lower than 68% would be no better of a predictor than predicting no change every time.

A plot of change in FFR over time is plotted below. As can be seen, the FFR has varied between 0% and 10% during the entire period of interest, with changes typically no greater than 1% for each announcement.

Federal Funds Rate
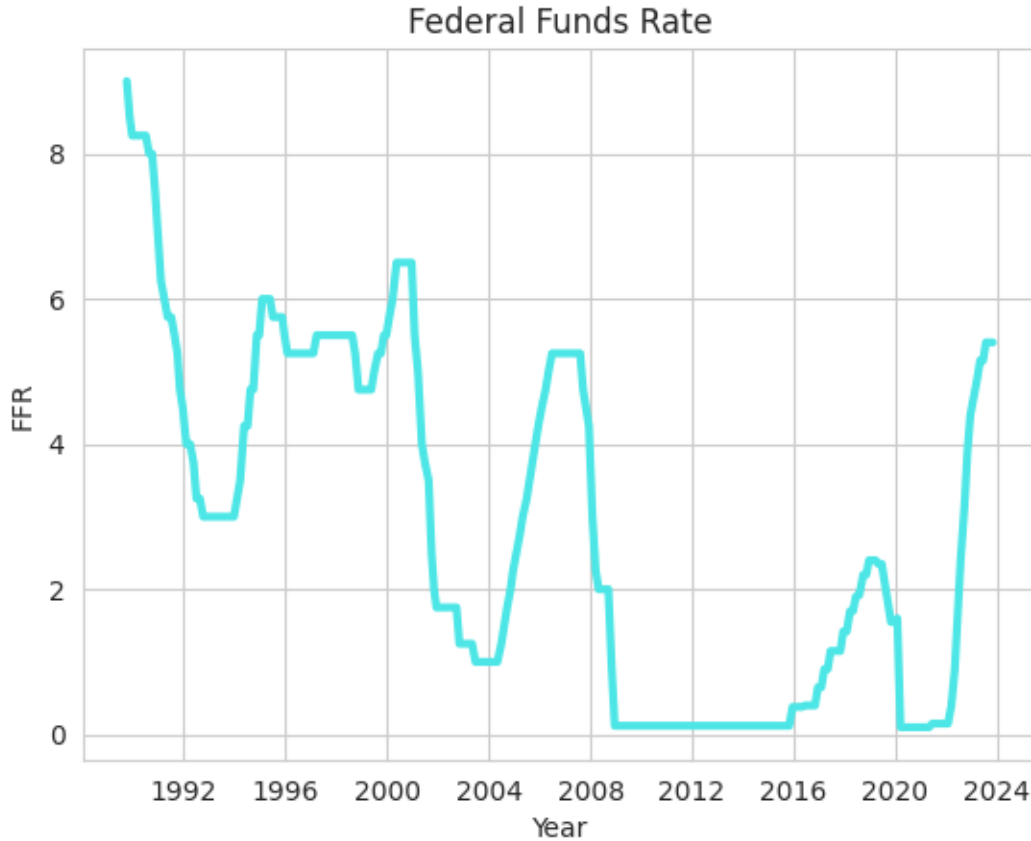
As previously mentioned, we generated additional features to account for trends in the economic data over time. Generating these extra features likely introduced high autocorrelation into our dataset. To solve this problem, We perfored Factor Analysis of Mixed Data (FAMD), an analogue to PCA for both categorical and continuous data, in order to only highlight the most important variation of our collapsed dataset.

Below we plot the original dataset and the collapsed dataset in two dimensions using the factorial axes generated by using FAMD to reduce dimensionality while minimizing information loss.

Factor Analysis of Original Data   Factor Analysis of Collapsed Data
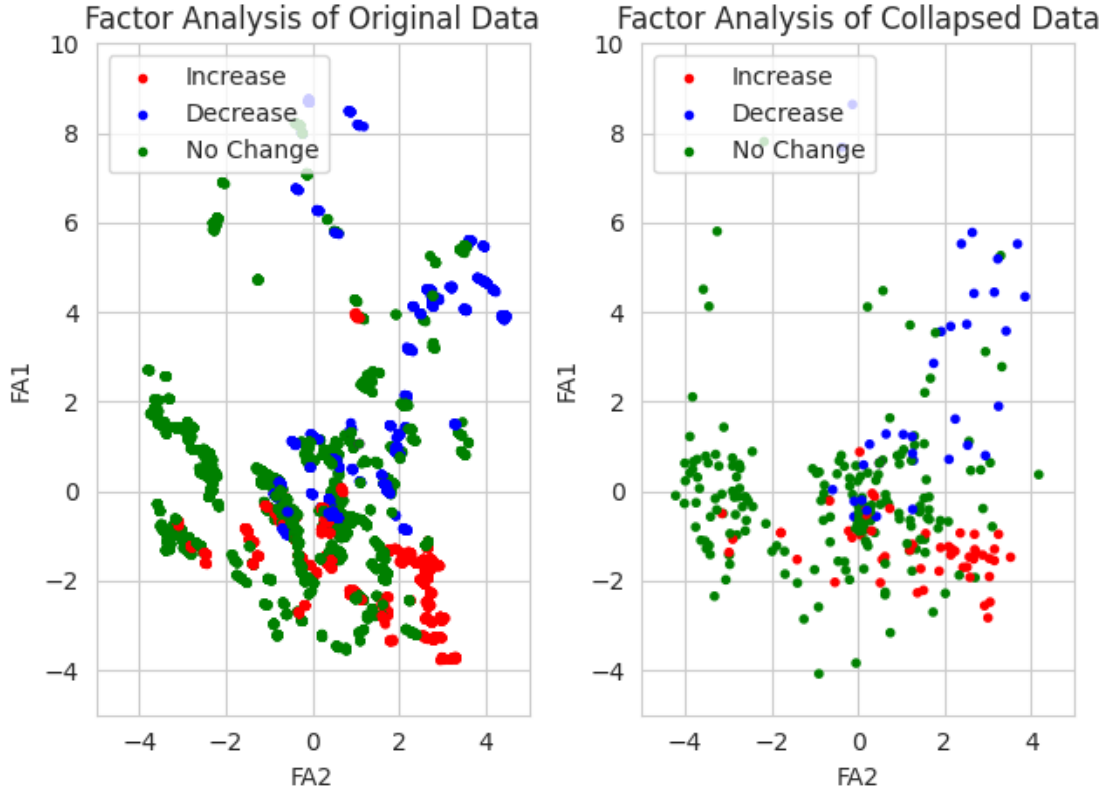
As can be seen, while the data from the original dataset (left) is noisy, the data from our collapsed dataset (right) is cleaner while still preserving the same basic structure of the original data. Importantly, clear clusters corresponding to an increase in the FFR and to a decrease in the FFR are visible, giving hope that our learning algorithms can predict changes in the FFR with moderate success.

0.0.7   Learning Algorithms and In-Depth Analysis

Our primary, naive attempt was to do a XGBoost regression. Accommodating to the datetime nature of the data required we split our training and testing sets on a specific date. However, as seen below, this failed to properly model the changes. The first portion of the data until around 2012 was used for training, and the highlighted region shows the results with the test data.

XGBoost Regression Model
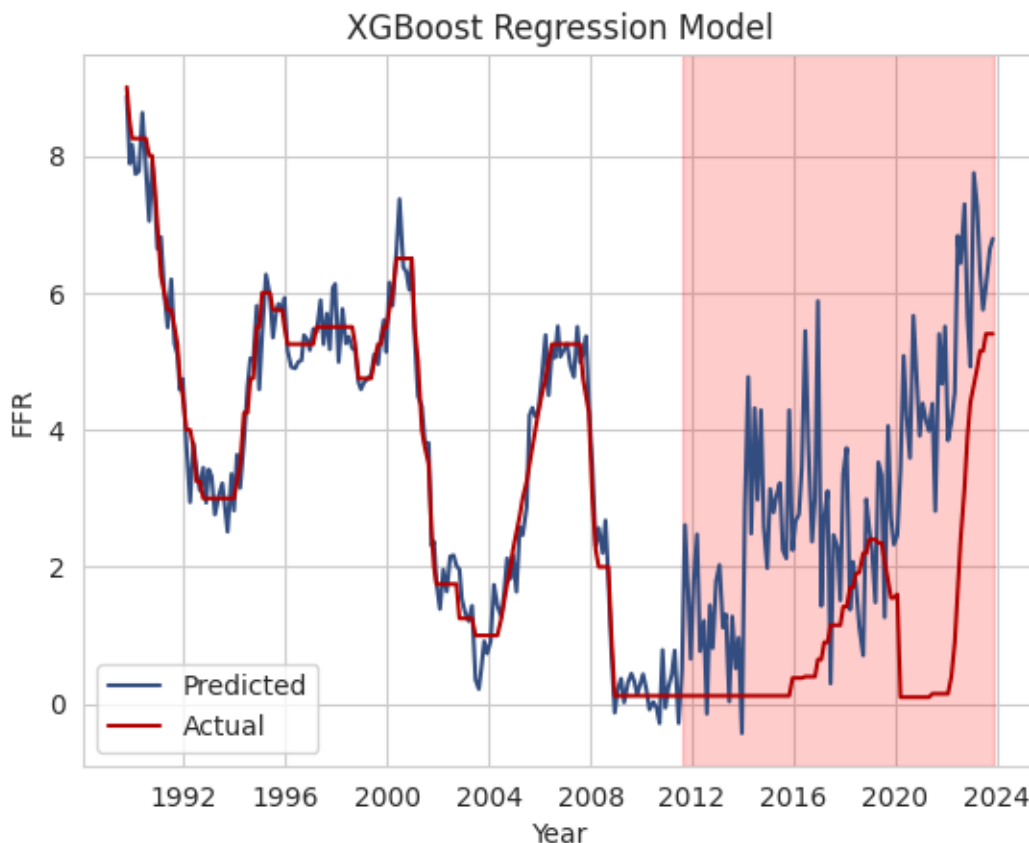
Recognizing that predicting the exact interest rate would be difficult, we decided classifying an increase, decrease, or no change in the rate rather than regression was a better approach. For our classification problem, we chose 4 different modeling algorithms. First, the RandomForestClassifier for its proficiency in handling diverse data types and its resilience against overfitting, making it ideal for capturing subtle patterns in the decision-making process of the FOMC. XGBClassifier was selected for its effectiveness in dealing with structured data and its high performance in both speed and accuracy. Softmax Logistic Regression offered a more straightforward approach, beneficial for interpreting the relationships between variables and the decisions made. Lastly, Gaussian Discriminant Analysis (GDA) provided a contrastive approach by assuming a Gaussian distribution of the data, offering insights where other models might overlook. Each tool was chosen not just for its individual merits but for how their collective insights could offer a balanced set of approaches for classifying the data.

After using this `collapse()` function on the original data we were now able to train a basic `RandomForestClassifier()` model to find the approximate feature importances which are seen below.

We use a rolling window of 2 to 42 days in our collapse method, as it incorporates the most current information of 2 days before the decision is made, up to 6 weeks (42 days) right after the previous FOMC decision. This window captures the most possible information in the typical 6 week cycle between decision meetings. Empirical testing of different windows consistently supports using this window as it prroduces our most robust and accurate models.

8

Feature Importance
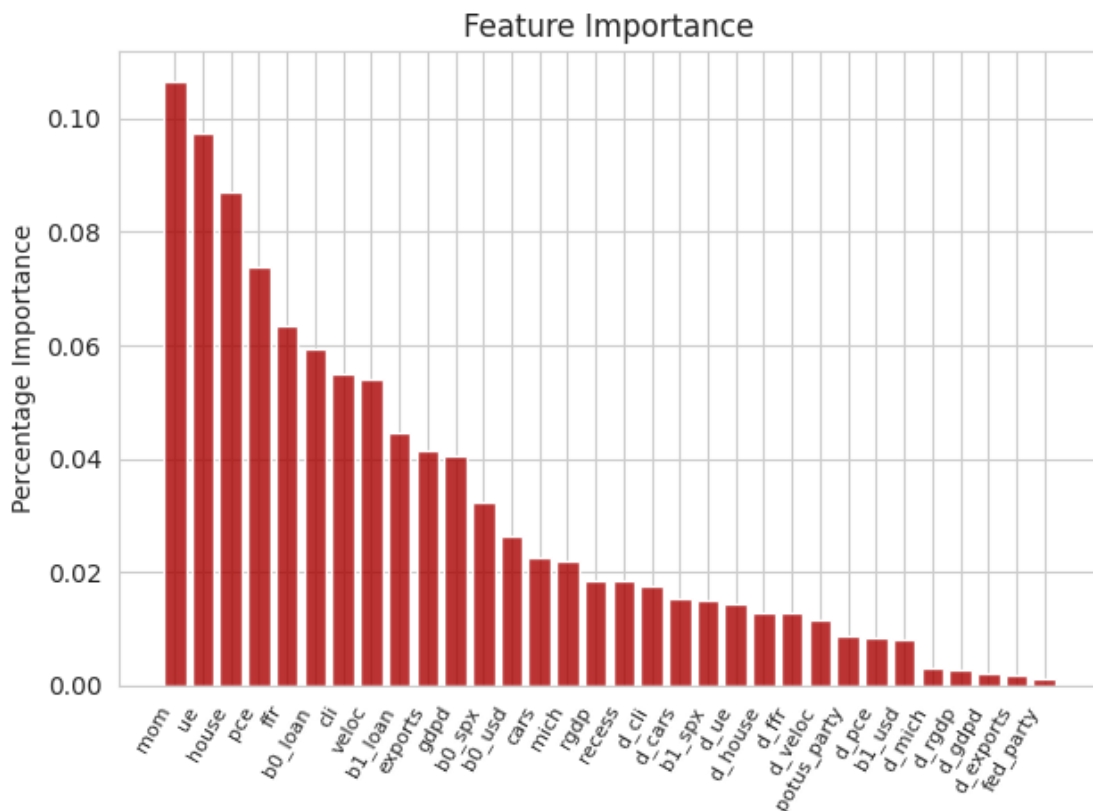
Since the beginning of our data, there have been 275 scheduled FOMC decisions. With this limited selection of data, we use Stratified 5-Fold Cross Validation for testing our model, averaging the score across each group. To optimally tune the hyperparameters, we utilize the library `Bayesian Optimization`, which uses bayesian inference and gaussian processes to optimally iterate through hyperparameters. See the appendix for our implementation of this code. After tuning hyperparameters, we test each model, using KFold validation in the cell below:

0.0.8 Average Accuracy Results:

- **RandomForestClassifier** - Average Accuracy: 0.8028
- **XGBClassifier** - Average Accuracy: 0.7921
- **Softmax Logistic Regression** - Average Accuracy: 0.7519
- **GDA** - Average Accuracy: 0.6497

0.0.9 Classification Reports:

RandomForestClassifier - Classification Report:

	precision	recall	f1-score	support
0.0	0.78	0.42	0.55	33
1.0	0.80	0.96	0.87	188
2.0	0.87	0.49	0.63	53

XGBClassifier - Classification Report:

	precision	recall	f1-score	support
0.0	0.76	0.39	0.52	33
1.0	0.78	0.96	0.86	188
2.0	0.88	0.43	0.58	53

Softmax Logistic Regression - Classification Report:

	precision	recall	f1-score	support
0.0	0.71	0.45	0.56	33
1.0	0.78	0.88	0.83	188
2.0	0.61	0.47	0.53	53

GDA - Classification Report:

	precision	recall	f1-score	support
0.0	0.44	0.55	0.49	33
1.0	0.81	0.66	0.73	188
2.0	0.45	0.66	0.53	53

Interestingly, the Random Forest Classifier emerges as the top-performing model with an average accuracy of approximately 80%. This superior performance can be attributed to the inherent strengths of Random Forests in handling many data types, its robustness to overfitting, and its ability to model complex interactions and non-linear relationships. The classification report shows high precision and recall for class 1.0 (no change in interest rate), indicating strong performance in identifying the most prevalent class. However, it's notable that the recall for rate increases and decreases is comparatively lower, suggesting some challenges in correctly classifying the minority classes.

XGBClassifier, with an average accuracy of about 79%, is a close second. The classification report for XGBoost mirrors that of the Random Forest, with a slight decrease in the recall for interest rate increases and decreases. This indicates a similar trend in the models' ability to handle minority classes.

Softmax Logistic Regression shows an average accuracy of around 75%. Its linearity likely provides limitations compared to ensemble methods (Random Forest and XGBoost), which might not capture complex economic decision-making scenarios. The classification report reflects a balanced performance across classes, albeit with lower precision and recall values than the ensemble methods, especially for the minority classes.

Gaussian Discriminant Analysis exhibits the lowest average accuracy at approximately 65%. GDA makes assumptions about the distribution of data (i.e., Gaussian), which likely does not hold true for our dataset, possibly leading to its lower performance. Interestingly, the classification report shows a relatively higher recall for the minority classes compared to the other models, indicating its potential in identifying these classes, albeit at the cost of precision.

Returning to the fundamental questions of our project:

What regression and classification methods are best suited to forecasting FOMC decisions? Among the methods analyzed, the RandomForestClassifier is best suited for forecasting these decisions. Its ability to handle complex data structures and its robustness against overfitting make it the most effective choice for this classification task.

How accurately can we predict FOMC decisions? The RandomForestClassifier, as the most effective model in our analysis, predicts FOMC decisions with an average accuracy of approximately 80%. This level of accuracy indicates a high degree of reliability in the model's predictions. However, it's important to note that while this accuracy is notable, there remains a degree of uncertainty, particularly in predicting less frequent decision outcomes. Therefore, while predictions are generally reliable, they should be contextualized within the inherent unpredictability of FOMC decisions.

Final Model Iteration for Industrial Applications In validating our models, we ran k-fold cross-validation across time, randomly assigning different FOMC decisions from any time period with a mix of past and future decisions. Interpreting our model with such training methods implies that future conditions surrounding FOMC decisions are being used to predict current decisions. This may violate causality and raises significant issues with the validity of this initial model. In practice, the proposed model will use past data to predict future FOMC decisions, a significant deviation from how our current model was trained.

To combat this potential misuse of our model, we employ a new implementation of a new sequential jackknife strategy. We start by training our model on the first 137 FOMC decisions (the first half of our data), and train on all previous decisions to predict only the very next decision. After obtaining this prediction, we wait for the next period and repeat by training on all available data (including the previous decision that is now available in our dataset) and predict the following decision once more. Continuing this pattern, we calculate predictions for every sequential decision akin to how this model would be practically applied, as described in the motivation section.

While holding all other previously established components of our model constant, we replace k-fold cross-validation with this 'sequential jackknife' approach for both Random Forest and XGBoost. As can be seen in our results below, our overall accuracy does not deviate from the previous analysis; it remains at roughly 81% for Random Forest, while XGBoost hovers at 79%. Given the similarity in results, the same conclusions from our previous analysis can be directly applied to this final rendition of the model with 'sequential jackknife' training. In practice this method of sampling will be used in predicting the next FOMC decision.

```
[9]:
```

```
Random Forest Accuracy: 0.8102189781021898
XGB Accuracy: 0.7883211678832117

Random Forest Classification Report:
              precision    recall  f1-score   support

         0.0       0.71      0.38      0.50        13
         1.0       0.81      0.97      0.88       101
         2.0       0.89      0.35      0.50        23

    accuracy                           0.81       137
```

```
   macro avg      0.80      0.57      0.63        137
weighted avg       0.81      0.81      0.78        137


XGB Classification Report:
              precision   recall  f1-score   support

         0.0      0.57      0.31      0.40         13
         1.0      0.80      0.95      0.87        101
         2.0      0.80      0.35      0.48         23

    accuracy                          0.79        137
   macro avg      0.72      0.54      0.58        137
weighted avg       0.78      0.79      0.76        137
```

0.0.10 Model Accuracies:

- **Random Forest Accuracy**: 0.8102
- **XGB Accuracy**: 0.7883

0.0.11 Classification Reports:

Random Forest Classification Report:

```
        precision   recall  f1-score   support

0.0        0.71      0.38      0.50         13
1.0        0.81      0.97      0.88        101
2.0        0.89      0.35      0.50         23
```

XGB Classification Report:

```
        precision   recall  f1-score   support

0.0        0.57      0.31      0.40         13
1.0        0.80      0.95      0.87        101
2.0        0.80      0.35      0.48         23
```

0.0.12 Ethical Implications

We have attempted to develop a model for predicting changes in the federal funds rate based on available economic indicators. As mentioned previously, the ability to predict changes to this prevailing interest rate before FOMC announcement allows for making strategical decisions to minimize or eliminate losses. For example, a prior prediction of what will happen to interest rates might motivate an individual to move from a low-security job to a high-security job, choose to wait or to rush to get a loan, or sell and buy stocks prior to a big move in the stock market.

Our intent and methodology raise a number of ethical issues. First, attempting to obtain an advantage through information about the future inherently leads to others who do not share that information facing a disadvantage. For example, someone who has highly credible evidence to

believe that the Federal Reserve will decrease the federal funds rate has incentive to sell stocks while keeping this information private so that others do not push the stock price down. Yet by not sharing that evidence, such an individual puts other investors in a more difficult place. Such moral grounds are the basis for material nonpublic information laws that prohibit using, sharing, or selling information about future company decisions that will affect stock prices to turn a profit (#A). While the information obtained through analysis such as that performed in this portfolio is not nearly as certain or exclusive (as the work we have done is highly reproducible), it is still important to acknowledge that successful attempts to predict the market in order to profit will inevitably result in someone losing in a zero-sum, competitive financial world.

Another ethical consideration is the potential loss that could be incurred when too much trust is placed in the prediction of the change in the federal fund rate generated by a model such as ours. With so many factors being weighed and considered by the Federal Reserve, and with economic indicators being so numerous and complex, it is inconceivable that any model used to predict changes in the federal funds rate could have near-perfect accuracy. Those that do not understand the extreme uncertainty involved in both the inputs and outputs to the model are likely to pivot too far, opening themselves up to loss incurred when the actual Federal Reserve decision flies in the face of predictions. In making and sharing predictions, therefore, it is important that equal emphasis and care is placed to express the uncertainty of the outcome, as well as to acknowledge that despite warnings, many will take risks based on the predictions that may have severe economic consequences.

Finally, it is important to note that our aim and methodology of predicting economic outcomes, when implemented and replicated on a large scale, may actually influence the outcomes themselves. Several times throughout history, the mere expectation of economic outcomes or a lack of faith in economic indicators have created economic disasters. For instance, in the events of Black Monday on October 19, 1987, the Dow Jones Industrial Average fell twenty percent in one day. The initial trigger was nervousness about possible overvaluation of stocks and the decline of the dollar, but the crisis was intensified by panic due to a lack of information as to what was happening ("B"). This episode and many others demonstrate that expectations or predictions of what will happen, whether grounded or totally unfounded, can often become a self-fulfilling prophecy. Thus it is possible that initially spurious relationships between economic indicators and interest rates mistakenly included or even emphasized in a model like this one might actually generate an unintended (and perhaps undesired) causal relationship, especially when machine learning is involved. Thus care must be taken to critically analyze how features are included, removed, or split on in such models, especially when destructive self-fulfilling feedback loops can have dramatic financial effects on millions across the country.

0.0.13   Citations:

[1] "Monetary Policy: What Are Its Goals? How Does It Work?" Federal Reserve Board - Monetary Policy: What Are Its Goals? How Does It Work?, www.federalreserve.gov/monetarypolicy/monetary-policy-what-are-its-goals-how-does-it-work.htm.

[2] Dziubinski, Susan. "How to Invest Your Money as the Fed Raises Interest Rates." Morningstar, Inc., 27 July 2022, www.morningstar.com/portfolios/how-invest-your-money-fed-raises-interest-rates.

[3] "Federal Open Market Committee." The Fed - Federal Open Market Committee,

www.federalreserve.gov/monetarypolicy/fomc.htm.

[4] Gavin, William T, and Rachel J Mandal. "The Art of Predicting the Federal Reserve: St. Louis Fed." Saint Louis Fed Eagle, Federal Reserve Bank of St. Louis, 9 Dec. 2021, www.stlouisfed.org/publications/regional-economist/july-2000/inside-the-briefcase-the-art-of-predicting-the-federal-reserve.

[5] FRED, Federal Reserve Bank of St. Louis, fred.stlouisfed.org/.

[6] "Chair of the Federal Reserve." Wikipedia, Wikimedia Foundation, 19 Nov. 2023, en.wikipedia.org/wiki/Chair_of_the_Federal_Reserve.

[7] "List of Presidents of the United States." Wikipedia, Wikimedia Foundation, 3 Dec. 2023, simple.wikipedia.org/wiki/List_of_presidents_of_the_United_States.

[8] "Leading Indicators - Composite Leading Indicator (CLI) - OECD Data." theOECD, data.oecd.org/leadind/composite-leading-indicator-cli.htm.

[9] "OECD." Wikipedia, Wikimedia Foundation, 12 Dec. 2023, en.wikipedia.org/wiki/OECD.

[10] "More on Ken's Research." Economics, econ.williams.edu/faculty-pages/research/.

[11] Chen, James. "Material Nonpublic Information (MNPI)? Definition and Laws." Investopedia, Investopedia, www.investopedia.com/terms/m/materialinsiderinformation.asp.

[12] Carlson, Mark A. "A brief history of the 1987 stock market crash with a discussion of the Federal Reserve Response." Finance and Economics Discussion Series, vol. 2006, no. 13, 2006, pp. 1–24, https://doi.org/10.17016/feds.2007.13.stopedia, Investopedia, www.investopedia.com/terms/m/materialinsiderinformation.asp.

1 Appendix

The appendix contains both our code to pull the data pull and code to clean the data.

2 Imports

```
[ ]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from datetime import date, timedelta
import fredapi as fd
from prince import FAMD
from sklearn.preprocessing import StandardScaler
from matplotlib.colors import LinearSegmentedColormap
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score,␣
 ↪classification_report, confusion_matrix, roc_curve
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪GridSearchCV, RandomizedSearchCV, StratifiedKFold
```

```python
from bayes_opt import BayesianOptimization
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from xgboost import XGBClassifier
from prince import FAMD
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
import matplotlib.pyplot as plt
```

2.0.1 Data Gathering

```python
[ ]: ### Data Pulls

# specify what data to pull from FRED
fred = fd.Fred(api_key = 'bde7928ce1d3cc555b5d2fb725f0ef4b')
start_dt = '1/1/1989'
name_id = [('loan', 'PRIME'), ('exports', 'A020RL1Q158SBEA'), ('pce',␣
 ↪'PCETRIM12M159SFRBDAL'),
          ('ue', 'UNRATE'), ('rgdp', 'A191RL1Q225SBEA'), ('cars', 'TOTALSA'),␣
 ↪('recess', 'USREC'),
          ('gdpd', 'A191RI1Q225SBEA'), ('veloc', 'M1V'), ('house', 'HOUST'),␣
 ↪('mich', 'MICH')]
fred_data = []

for x in name_id:
    # get FRED series using the proper id
    df = pd.DataFrame(fred.get_series(x[1], observation_start=start_dt))
    # make column name identifiable
    df.rename(columns={0: x[0]}, inplace=True)
    fred_data.append(df)

# aggregate FRED data series
df_fred = pd.concat(fred_data, axis=1)

# specify what data to pull from Yahoo! Finance
tickers = [('spx', '^SPX'), ('usd', 'DX-Y.NYB')]
start_dt = '1989-01-01'
end_dt = str(date.today() - timedelta(1))
interval = '1d'
stock_data = []

# access data and display
for t in tickers:
    # get price data using ticker
    df = yf.download(t[1], start = start_dt, end = end_dt, interval= interval)
    # make column name identifiable
    df = df[['Adj Close']].rename(columns={'Adj Close': t[0]})
```

```python
    stock_data.append(df)

# aggregate stock data
df_stock = pd.concat(stock_data, axis=1)

# read in cli data from csv file downloaded off OECD website
df_cli = pd.read_csv('cli.csv', index_col=0)
df_cli.index = pd.to_datetime(df_cli.index)

# read in manually gathered political party data
df_party = pd.read_csv('party.csv', index_col=0)
df_party.index = pd.to_datetime(df_party.index)

# create feature dataset
df_features = pd.concat([df_fred, df_stock, df_cli, df_party], axis=1)

# pull in FOMC data
df_fomc = pd.read_csv('ffr_clean.csv', index_col=0)
df_fomc.index = pd.to_datetime(df_fomc.index)

# create decisions and momentum column
decision = df_fomc['change'].values.copy()
down, up = tuple([decision < 0]), tuple([decision > 0])
decision[down], decision[up] = -1., 1.

# create new columns
df_fomc['decision'] = decision
df_fomc['mom'] = decision

# consider only scheduled meetings
df_fomc = df_fomc.query('scheduled == 1')
df_fomc.drop('scheduled', axis=1, inplace=True)

# get start and end dates
start_dt = df_features.index[0]
end_dt = df_fomc.index[-1]

# get every day between start and end date
delta = end_dt - start_dt
daily = [start_dt + timedelta(days=i) for i in range(delta.days + 1)]

# make dataframe of dates
df_days = pd.DataFrame(daily)
df_days.index = pd.to_datetime(df_days.values[:,0])

# initialize master dataframe
df = pd.concat([df_days, df_fomc, df_features], axis=1)
```

```python
# drop irrelevant columns and rows
df.drop(0, axis=1, inplace=True)
last_ix = df_days.index.get_loc(end_dt) + 1
df.drop(df.index[last_ix:], inplace=True)

# find start and stop columns
cols = df.columns.to_list()
start_ix, end_ix = cols.index('mom'), cols.index('fed_party')

# shift necessary columns
df_shift = df.iloc[:, start_ix:end_ix].copy()
df.iloc[:, start_ix:end_ix] = df_shift.shift()

# fill in NaN entries
df_fill = df.iloc[:, start_ix:].copy()
df.iloc[:, start_ix:] = df_fill.ffill()

# save master dataframe
df.to_csv('master_data.csv')
df.tail()
```

2.0.2 Data Cleaning and Feature Engineering

```python
# Load in our dataset, set the index to the date column, and take only values
 ↪after 1989-08-23
df = pd.read_csv('master_data.csv', index_col=0)
df.index = pd.to_datetime(df.index)
df = df.loc['1989-08-23':]

# Combine the mom_u and mom_d columns into a single column, dorp them, and then
 ↪make an indicator column for the event days
df['mom'] = -df['mom_d'] + df['mom_u']
df = df.drop(columns=['mom_u', 'mom_d'])
df['label_day'] = ~df['decision'].isna()

# Forward fill in the NaN values, drop the first row, and save the data
df = df.fillna(method='ffill')
df = df.iloc[1:]
df.to_csv('master_data_clean.csv')

# define the types of columns to be used in our preprocessing
monthly = ['pce','ue','cars','house','cli','exports','rgdp','gdpd','veloc',
 ↪'ffr','mich']
labels = ['decision', 'ffr', 'change']
daily = ['spx','usd', 'loan']
categorical = ['fed_party','potus_party','recess', 'mom']
```

```python
# define the function to collapse the data to the event days
def collapse(df, key = 'label_day', start = 7, end = 60, cat_col = [],␣
 ↪daily_col = [], other = [], labels = []):
    """A function to collapse the data into a single row for each event,␣
 ↪capturing trends and point in time data.
    df: dataframe
    key: column name of the event day
    start: number of days before the event to capture
    end: number of days after the event to capture"""

    # Get our different dataframes
    categorical = df[cat_col]
    daily = df[daily_col]
    monthly = df[other]
    keep = df[key]
    labels = df[labels]

    # create the windows
    event_days = df[df[key]].index

    ################## Handle Categorical and Label Data ##################
    cat_collapsed = categorical[keep]
    y_labels = labels[keep]

    ################## Handle Monthly Data ##################
    # Create event windows
    windows = []
    for day in event_days:
        # Calculate start and end days of the window
        start_day = day - pd.Timedelta(days=start)
        end_day = day - pd.Timedelta(days=end)

        # Ensure start_day is not before the start of the dataset
        if end_day < df.index[0]:
            end_day = df.index[0]

        windows.append((start_day, end_day))

    # Initialize a list to store aggregated data
    aggregated_data = []

    for (start_day, end_day) in windows:
        # Select data within the window
        window_data = monthly.loc[end_day:start_day]

        # Apply your custom aggregation function
```

```python
        final_vals = window_data.loc[start_day]
        change = final_vals - window_data.iloc[0]
        # relabel the change columns to include a d_ prefix
        change = change.rename(lambda x: 'd_' + x)

        # concatenate the data horizontally
        aggregated_window = pd.concat([final_vals, change], axis=0)
        aggregated_data.append(aggregated_window)

# Combine aggregated data
aggregated_monthly = pd.concat(aggregated_data, axis=1).T
aggregated_monthly.index = event_days

################## Handle Daily Data ##################
# define the model
model = linear_model.LinearRegression()
daily_agg = []

# Get the names of the features
daily_feat = daily.columns
b0_name = ['b0_' + feat for feat in daily_feat]
b1_name = ['b1_' + feat for feat in daily_feat]

for (start_day, end_day) in windows:
    # Select data within the window
    window_data = daily.loc[end_day:start_day]
    time = np.arange(len(window_data)).reshape(-1, 1)

    # Initialize lists to store betas for each feature
    beta0 = []
    beta1 = []

    # Loop through each feature and fit the model
    for feat in daily_feat:
        model.fit(time, window_data[feat])
        beta0.append(model.intercept_)
        beta1.append(model.coef_[0])

    # Create DataFrames for betas
    beta0_df = pd.DataFrame([beta0], columns=b0_name)
    beta1_df = pd.DataFrame([beta1], columns=b1_name)

    # concatenate the data horizontally
    aggregated_window = pd.concat([beta0_df, beta1_df], axis=1)
    daily_agg.append(aggregated_window)

# Combine aggregated data
```

```
        aggregated_daily = pd.concat(daily_agg, axis=0)
        aggregated_daily.index = event_days


        ################ Combine Data ################
        # Combine all data
        combined = pd.concat([cat_collapsed, aggregated_monthly, aggregated_daily],␣
    ↪axis=1)
        return combined, y_labels
```

2.0.3 FAMD, Model Fitting, and Further Analysis

```python
# Define the function to load our data and get it ready for FAMD
def encoder(df, col_split):
    """
    Prepares the data for the model by standardizing the continuous features,
    converting the categorical features to strings,
    and converting sparse columns to a dense format for FAMD.
    """
    categorical_cols = df.columns[:col_split]
    # converting the categorical features to strings
    df[categorical_cols] = df[categorical_cols].astype(str)
    return df

def load_data(data_address='beta_dates/beta_data_2_42.csv',␣
  ↪label_address='beta_dates/true_labels.csv'):
    df = pd.read_csv(data_address, index_col=0)
    y = pd.read_csv(label_address, index_col=0).values[:,0]
    y = y + 1
    X = encoder(df, 4)
    return X, y

# Load your dataset
X, y = load_data()

# Apply FAMD for dimensionality reduction
famd = FAMD(n_components=13)
famd.fit(X)
X_transformed = famd.transform(X)

# Create a pipeline with Random Forest
pipe = Pipeline([
    ('rf', RandomForestClassifier())
])

# Function to optimize with cross-validation
def rf_evaluate(n_estimators, max_depth, min_samples_split, min_samples_leaf,␣
  ↪max_features):
```

```python
    params = {
        'rf__n_estimators': int(n_estimators),
        'rf__max_depth': int(max_depth),
        'rf__min_samples_split': int(min_samples_split),
        'rf__min_samples_leaf': int(min_samples_leaf),
        'rf__max_features': max_features,
    }
    pipe.set_params(**params)
    return np.mean(cross_val_score(pipe, X_transformed, y, cv=5,
 ↪scoring='accuracy'))

# Bayesian Optimization
optimizer = BayesianOptimization(
    f=rf_evaluate,
    pbounds={
        'n_estimators': (50, 300),
        'max_depth': (3, 5),
        'min_samples_split': (2, 30),
        'min_samples_leaf': (1, 30),
        'max_features': (0.1, 1)
    },
    random_state=42
)

optimizer.maximize(init_points=5, n_iter=200)

# After optimization, retrieve and adjust best parameters for pipeline
rf_best_params = optimizer.max['params']
rf_best_params_for_pipeline = {'rf__' + key: int(value) if key in ['max_depth',
 ↪'n_estimators', 'min_samples_split', 'min_samples_leaf'] else value for key,
 ↪value in rf_best_params.items()}

# Set the best parameters in the pipeline
pipe.set_params(**rf_best_params_for_pipeline)
pipe.fit(X_transformed, y)

# Print best parameters and performance
print("Best Parameters for Pipeline:", rf_best_params_for_pipeline)
print("Best Performance:", optimizer.max['target'])
```

```python
[ ]: # Further Analysis and Model Fitting

# Load your dataset
X, y = load_data()

# Apply FAMD for dimensionality reduction
famd = FAMD(n_components=13)
```

```python
famd.fit(X)
X_transformed = famd.transform(X)

# Create a pipeline with XGBoost
pipe = Pipeline([
    ('xgb', XGBClassifier(objective='multi:softprob'))  # Using multi:softprob␣
 ↪for multi-class classification
])

# Function to optimize with cross-validation
def xgb_evaluate(n_estimators, max_depth, learning_rate, gamma,␣
 ↪min_child_weight, subsample, colsample_bytree, reg_alpha, reg_lambda):
    params = {
        'xgb__n_estimators': int(n_estimators),
        'xgb__max_depth': int(max_depth),
        'xgb__learning_rate': learning_rate,
        'xgb__gamma': gamma,
        'xgb__min_child_weight': min_child_weight,
        'xgb__subsample': subsample,
        'xgb__colsample_bytree': colsample_bytree,
        'xgb__reg_alpha': reg_alpha,
        'xgb__reg_lambda': reg_lambda
    }
    pipe.set_params(**params)
    return np.mean(cross_val_score(pipe, X_transformed, y, cv=5,␣
 ↪scoring='accuracy'))

# Bayesian Optimization
optimizer = BayesianOptimization(
    f=xgb_evaluate,
    pbounds={
        'n_estimators': (50, 300),
        'max_depth': (3, 10),
        'learning_rate': (0.01, 0.3),
        'gamma': (0, 5),
        'min_child_weight': (1, 10),
        'subsample': (0.5, 1),
        'colsample_bytree': (0.5, 1),
        'reg_alpha': (0, 5),
        'reg_lambda': (0, 5)
    },
    random_state=42
)

optimizer.maximize(init_points=5, n_iter=200)

# After optimization, retrieve and adjust best parameters for pipeline
```

```python
xgb_best_params = optimizer.max['params']
xgb_best_params_for_pipeline = {'xgb__' + key: int(value) if key in␣
 ↪['n_estimators', 'max_depth'] else value for key, value in xgb_best_params.
 ↪items()}


# Set the best parameters in the pipeline
pipe.set_params(**xgb_best_params_for_pipeline)
pipe.fit(X_transformed, y)


# Print best parameters and performance
print("Best Parameters for Pipeline:", xgb_best_params_for_pipeline)
print("Best Performance:", optimizer.max['target'])
```

3 Comparing different models

```python
def categorical(df, col_split):
    """
    Prepares the data for the model by standardizing the continuous features,
    converting the categorical features to strings,
    and converting sparse columns to a dense format for FAMD.
    """
    categorical_cols = df.columns[:col_split]

    # converting the categorical features to strings
    df[categorical_cols] = df[categorical_cols].astype(str)

    return df

def load_data(data_address='beta_dates/beta_data_2_42.csv',␣
 ↪label_address='beta_dates/true_labels.csv'):
    """
    Loads the data and labels from the specified addresses.
    """
    df = pd.read_csv(data_address, index_col=0)
    y = pd.read_csv(label_address, index_col=0).values[:,0]
    # add one to the labels to make them 0, 1, 2 instead of -1, 0, 1 (needed␣
 ↪for XGBClassifier)
    y = y + 1
    X = categorical(df, 4)

    return X, y

# Function to perform cross-validation and return average accuracy and␣
 ↪classification report
def evaluate_model(pipe, X, y, params, n_splits=5):
    """
```

```python
    Performs cross-validation on the given model and returns the average␣
 ↪accuracy and classification report.
    """
    pipe.set_params(**params)
    skf = StratifiedKFold(n_splits=n_splits, shuffle=True)
    accuracy_scores = []
    all_predictions = []
    all_true_values = []

    for train_index, test_index in skf.split(X, y):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]
        pipe.fit(X_train, y_train)
        predictions = pipe.predict(X_test)
        accuracy_scores.append(accuracy_score(y_test, predictions))
        all_predictions.extend(predictions)
        all_true_values.extend(y_test)

    report = classification_report(all_true_values, all_predictions,␣
 ↪zero_division=0)
    return np.mean(accuracy_scores), report


# Load your data
X, y = load_data()

# Apply FAMD for dimensionality reduction
famd = FAMD(n_components=6)
famd.fit(X)
X_transformed = famd.transform(X)

# Evaluate RandomForestClassifier
rf_params = {'rf__max_depth': 4, 'rf__max_features': 0.65,␣
 ↪'rf__min_samples_leaf': 6, 'rf__min_samples_split': 17, 'rf__n_estimators':␣
 ↪200}
avg_rf_accuracy, rf_report = evaluate_model(Pipeline([('rf',␣
 ↪RandomForestClassifier())]), X_transformed, y, rf_params)


# Evaluate XGBClassifier
xgb_params = {'xgb__colsample_bytree': 1.0, 'xgb__gamma': 3.646092407548312,␣
 ↪'xgb__learning_rate': 0.01, 'xgb__max_depth': 6, 'xgb__min_child_weight': 9.
 ↪488817186497869, 'xgb__n_estimators': 81, 'xgb__reg_alpha': 1.
 ↪5988455864986395, 'xgb__reg_lambda': 3.267653422896763, 'xgb__subsample': 1.
 ↪0}
avg_xgb_accuracy, xgb_report = evaluate_model(Pipeline([('xgb',␣
 ↪XGBClassifier(objective='multi:softmax'))]), X_transformed, y, xgb_params)
```

```python
# Evaluate Softmax Logistic Regression
avg_lr_accuracy, lr_report = evaluate_model(Pipeline([('lr',
 ↪LogisticRegression(multi_class='multinomial', solver='lbfgs'))]),
 ↪X_transformed, y, params={})


# Evaluate GDA
avg_gda_accuracy, gda_report = evaluate_model(Pipeline([('gda',
 ↪QuadraticDiscriminantAnalysis())]), X_transformed, y, params={})

print('Average Accuracy Results:')
print('RandomForestClassifier - Average Accuracy:', avg_rf_accuracy)
print('XGBClassifier - Average Accuracy:', avg_xgb_accuracy)
print('Softmax Logistic Regression - Average Accuracy:', avg_lr_accuracy)
print('GDA - Average Accuracy:', avg_gda_accuracy)

print('\nClassification Reports:')
print('RandomForestClassifier - Classification Report:\n', rf_report)
print('XGBClassifier - Classification Report:\n', xgb_report)
print('Softmax Logistic Regression - Classification Report:\n', lr_report)
print('GDA - Classification Report:\n', gda_report)
```

3.0.1 Final Model Training and Evaluation

```python
# Load your data
X, y = load_data()

# Optimized parameters for Random Forest
rf_params = {
    'rf__max_depth': 4,
    'rf__max_features': 0.65,
    'rf__min_samples_leaf': 6,
    'rf__min_samples_split': 17,
    'rf__n_estimators': 200
}

# Optimized parameters for XGBClassifier
xgb_params = {
    'xgb__colsample_bytree': 1.0,
    'xgb__gamma': 3.646092407548312,
    'xgb__learning_rate': 0.01,
    'xgb__max_depth': 6,
    'xgb__min_child_weight': 9.488817186497869,
    'xgb__n_estimators': 81,
    'xgb__reg_alpha': 1.5988455864986395,
```

```python
    'xgb__reg_lambda': 3.267653422896763,
    'xgb__subsample': 1.0}

# split the rf_params and xgb_params on __, and get the split and final index
rf_param_names = {param.split('__')[1]:rf_params[param] for param in rf_params.
 ↪keys()}
xgb_param_names = {param.split('__')[1]:xgb_params[param] for param in␣
 ↪xgb_params.keys()}
split_index, final_index = int(len(X) * 0.5), len(X)

# Initialize lists to store accuracies for each model
rf_predictions = []
xgb_predictions = []
y_actual = []

# Iterate through the data and get the train test split based on the split day
for i in range(split_index, final_index):
    # Get the FAMD transformation
    X_current = X.iloc[:i+1]
    famd = FAMD(n_components=6, engine='sklearn')
    famd.fit(X_current)
    X_famd = famd.transform(X_current)

    # Get the train test split
    X_train, X_test = X_famd[:i], X_famd[i:i+1]
    y_train, y_test = y[:i], y[i:i+1]

    # Create and fit the Random Forest classifier, append the prediction to␣
 ↪rf_guess
    rf_classifier = RandomForestClassifier(warm_start=False)
    rf_classifier.set_params(**rf_param_names)
    rf_classifier.fit(X_train, y_train)
    rf_predictions.append(rf_classifier.predict(X_test))

    # Create and fit the XGBClassifier, append the prediction to xgb_guess
    xgb_classifier = XGBClassifier()
    xgb_classifier.set_params(**xgb_param_names)
    xgb_classifier.fit(X_train, y_train)
    xgb_predictions.append(xgb_classifier.predict(X_test))

    # Append the actual value to y_actual
    y_actual.append(y_test)

# Calculate the accuracy scores for each model
y_actual = np.array(y_actual)
rf_predictions = np.array(rf_predictions)
xgb_predictions = np.array(xgb_predictions)
```

```
rf_accuracy = accuracy_score(y_actual, rf_predictions)
xgb_accuracy = accuracy_score(y_actual, xgb_predictions)

# Print the accuracy scores and classification reports
print('Random Forest Accuracy:', rf_accuracy)
print('XGB Accuracy:', xgb_accuracy)
print()

# Print the classification reports
rf_conf_matrix = classification_report(y_actual, rf_predictions,␣
 ↪zero_division=0)
print('Random Forest Classification Report:\n', rf_conf_matrix)
xgb_conf_matrix = classification_report(y_actual, xgb_predictions,␣
 ↪zero_division=0)
print('XGB Classification Report:\n', xgb_conf_matrix)
```

```
[ ]: def encoder(df, col_split):
         """
         Prepares the data for the model by standardizing the continuous features,
         converting the categorical features to strings,
         and converting sparse columns to a dense format for FAMD.
         """
         categorical_cols = df.columns[:col_split]
         # converting the categorical features to strings
         df[categorical_cols] = df[categorical_cols].astype(str)
         return df

     def load_data(data_address='beta_dates/beta_data_2_42.csv',␣
      ↪label_address='beta_dates/true_labels.csv'):
         df = pd.read_csv(data_address, index_col=0)
         y = pd.read_csv(label_address, index_col=0).values[:,0]
         y = y + 1
         X = encoder(df, 4)
         return X, y

     # Load your data
     X, y = load_data()

     # Parameters for Random Forest
     rf_params = {
         'rf__max_depth': 4,
         'rf__max_features': 0.65,
         'rf__min_samples_leaf': 6,
         'rf__min_samples_split': 17,
         'rf__n_estimators': 200
     }
```

```python
# Parameters for XGBClassifier
xgb_params = {
    'xgb__colsample_bytree': 1.0,
    'xgb__gamma': 3.646092407548312,
    'xgb__learning_rate': 0.01,
    'xgb__max_depth': 6,
    'xgb__min_child_weight': 9.488817186497869,
    'xgb__n_estimators': 81,
    'xgb__reg_alpha': 1.5988455864986395,
    'xgb__reg_lambda': 3.267653422896763,
    'xgb__subsample': 1.0}

# split the rf_params and xgb_params on __, and get the split and final index
rf_param_names = {param.split('__')[1]:rf_params[param] for param in rf_params.
 ↪keys()}
xgb_param_names = {param.split('__')[1]:xgb_params[param] for param in␣
 ↪xgb_params.keys()}
split_index, final_index = int(len(X) * 0.5), len(X)

# Initialize lists to store accuracies for each model
rf_predictions = []
xgb_predictions = []
y_actual = []

# Iterate through the data and get the train test split based on the split day
for i in range(split_index, final_index):
    # Get the FAMD transformation
    X_current = X.iloc[:i+1]
    famd = FAMD(n_components=6, engine='sklearn')
    famd.fit(X_current)
    X_famd = famd.transform(X_current)

    # Get the train test split
    X_train, X_test = X_famd[:i], X_famd[i:i+1]
    y_train, y_test = y[:i], y[i:i+1]

    # Create and fit the Random Forest classifier, append the prediction to␣
 ↪rf_guess
    rf_classifier = RandomForestClassifier(warm_start=False)
    rf_classifier.set_params(**rf_param_names)
    rf_classifier.fit(X_train, y_train)
    rf_predictions.append(rf_classifier.predict(X_test))

    # Create and fit the XGBClassifier, append the prediction to xgb_guess
    xgb_classifier = XGBClassifier()
    xgb_classifier.set_params(**xgb_param_names)
    xgb_classifier.fit(X_train, y_train)
```

```python
    xgb_predictions.append(xgb_classifier.predict(X_test))

    # Append the actual value to y_actual
    y_actual.append(y_test)

# Calculate the accuracy scores for each model
y_actual = np.array(y_actual)
rf_predictions = np.array(rf_predictions)
xgb_predictions = np.array(xgb_predictions)
rf_accuracy = accuracy_score(y_actual, rf_predictions)
xgb_accuracy = accuracy_score(y_actual, xgb_predictions)

# Print the accuracy scores and classification reports
print('Random Forest Accuracy:', rf_accuracy)
print('XGB Accuracy:', xgb_accuracy)
print()

# Print the classification reports
rf_conf_matrix = classification_report(y_actual, rf_predictions,
  ↪zero_division=0)
print('Random Forest Classification Report:\n', rf_conf_matrix)
xgb_conf_matrix = classification_report(y_actual, xgb_predictions,
  ↪zero_division=0)
print('XGB Classification Report:\n', xgb_conf_matrix)
```