University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project
7PAM2002
Department of Physics, Astronomy and Mathematics

Data Science Final Project Report

Project Title:

Investigating Business Confidence: A Comprehensive Analysis of Economic Indicators through Advanced Machine Learning Approaches

Student Name and SRN:

Tanvir Ahmed
20075186

Supervisor: Dr. Vidas Regelskis

Date Submitted: 15/01/2024

Word Count: 7300

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science **in Data Science** at the University of Hertfordshire.

I have read the detailed guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6)

I did not use human participants in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Tanvir Ahmed

Student Name signature: *Tanvir Ahmed*

Student SRN number: 20075186

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Abstract

The Business Confidence Index (BCI) is an indispensible metric for businesses which provides a measure of whether an economy will be conducive or not to facilitate business growth in the short term future. A big influencing factor in economic decision-making, the BCI reflects the sentiments and expectations of businesses within an economy. In this project a comprehensive analysis of the BCI is performed using the US economic data. The application of various machine learning models, including Linear Regression, KNN, SVM Regression, Random Forest, Gradient-Boosted Decision Trees (XGBoost) and Neural Networks are examined and their capacities to explain the complex indicators affecting the BCI is analysed. The analysis has revealed that decision tree based algorithms are better equipped to explain the factors influencing the index. Although, Random Forest has shown a commendable performance, XGBoost model emerged as the most effective method with an R-squared value of 0.895. However, predicting future BCI values with regression models will not necessarily provide a reasonable estimate due to low variability of the index. The qualitative assessment of the selected indicators has revealed that GDP, credit delinquency, consumer loans, real estate prices, and imports are pivotal factors influencing the BCI. The predictive aspect can be better explored with classification models.

This study therefore has aimed at uncovering the intricate relationship between economic indicators and business confidence, offering valuable insights for strategic decision-making.

Keywords: Business Confidence, BCI, Regression, Random Forest, XGBoost, GDP, Business Decision, Business Sentiment

Acknowledgement

I am profoundly grateful for the unwavering support of my family and friends throughout this programme without which this achievement would not have been possible. I cannot thank my brother and my partner enough for their continuous support.

Special appreciation goes to my supervisor, Dr. Vidas Regelskis, whose guidance and insights were indispensable throughout this project. His encouragement to explore the realm of machine learning applications in a relatively uncharted territory was a necessary stimulant for me from time to time.

I extend my sincere thanks to the project module leader, Carolyn Devereux, for her dedicated efforts in ensuring a smooth and enriching learning experience for all students.

I am thankful for the entire experience.

Table of Contents

CHAPTER 1

1. Introduction

The Business Confidence Index (BCI) is a measure of the future prospects of businesses operating within an economy. It quantifies the expectations of business entities in the following 6 to 12 months. In combination with the Consumer Confidence Index (CCI), the BCI serves as a useful metric to gauge the potential for business opportunities for investors and financial policymakers.

The BCI is usually measured through surveys conducted by recognised businesses and financial institutions. In the United States, this is conducted by several prominent organisations that probe the sentiments within businesses about potential growth in the near future. The most notable surveys are conducted by:

1. The Federal Reserve Bank: The Federal Reserve conducts various surveys, including the Beige Book, which provides information on current economic conditions, including business sentiment.
2. The Conference Board: A non-profit research organization that regularly releases the Leading Economic Index (LEI), which incorporates the Business Confidence Index. They also estimate the Consumer Confidence Index (CCI).
3. Institute for Supply Management (ISM): The largest, non-profit supply management association in the world known for its Purchasing Managers' Index (PMI). ISM gathers data on business sentiment and economic conditions.
4. National Federation of Independent Business (NFIB): An association of small businesses, it periodically publishes the Small Business Optimism Index, which reflects the sentiment of small business owners.
5. Moody's Analytics: A division within Moody's Corporation offering perspectives on market risks through credit analysis, economic research, and strategic financial risk insights.

Selected firms or businesses are surveyed, and their non-numeric responses, such as 'positive,' 'negative,' or 'neutral,' are systematically converted into numerical values. This process ensures a consistent and quantitative representation of qualitative sentiments. The resulting numerical values, along with the numeric data on expected production, order books levels, and stock of finished goods, are then averaged. Finally, this averaged value is adjusted to a scale of 100 points according to the OECD Business Survey Handbook 2003. This normalisation provides a standardised scale, facilitating the interpretation of the index. A BCI value above 100 generally indicates optimism, while a value below 100 suggests pessimism. This adjustment allows for a uniform and comparative analysis of the index over different timeframes or entities.

The Organisation for Economic Co-operation and Development (OECD) monthly reports the BCI for its member countries. In this project, BCI data was collected from the OECD database spanning the period from 1950 to 2023.

1.1 Objective of the project

The primary objective of this project was to explore the effectiveness of machine learning models in predicting the BCI with reasonable accuracy and to identify the economic factors influencing the index. Assuming that the mathematical or machine learning models accurately capture the underlying relationships between economic indicators and the BCI, this approach could potentially serve as a valuable alternative to the traditional survey methods, which are often time and resource-intensive.

By leveraging machine learning, businesses can potentially benefit from more timely and cost-effective insights into business confidence levels. Moreover, the ability to assess the confidence level in real-time using existing economic data at any given point allows for quicker strategy development. Additionally, the application of machine learning models opens pathways for a more critical understanding of the complex relationships within economic indicators, potentially uncovering hidden patterns or correlations that might be too obscure to identify through traditional methods.

Furthermore, this project aimed to contribute to the ongoing discourse on the intersection of machine learning and economic analysis, providing insights into the feasibility of utilising advanced analytical tools for forecasting and decision-making in the business environment. If successful, this approach could not only streamline the process of gauging business confidence but also offer a more dynamic and responsive strategy formulation mechanism for businesses navigating today's rapidly changing economic landscape.

CHAPTER 2

2. Literature Review

The forecasting of economic indicators has been a subject of extensive research for decades, with various mathematical models employed to capture and predict the complex dynamics of economies. While traditional econometric models have been widely used, in recent years there have been a surge in interest in leveraging advanced computational techniques, particularly machine learning (ML), to enhance the predictive accuracy.

2.1 Mathematical Models in Economic Analysis

A rich body of literature exists on the application of mathematical models for understanding economic factors. Neumann and Morgenstern, the pioneers of 'Game Theory', introduced fundamental mathematical concepts to analyse decision-making processes in economics in their seminal text 'Theory of Games and Economic Behavior' (1944). This work not only left a lasting impact on economics but also extended its influence into the realms of computer science and political studies.

Tinbergen and Bos (1962) integrated mathematical models to understand the multi-sectoral dynamics and their impact on the overall growth of an economy.

Diebold and Rudebusch (1994) introduced a comprehensive approach to assessing business-cycle dynamics. They proposed a dynamic-factor and regime-switching model that seamlessly integrated theoretical and empirical methods. The model was based on a linear-quadratic equilibrium framework and focused on quarterly economic indicators spanning the period from 1952 to 1993. The authors utilized five Composite Indices, each composed of four coincident indicators, to provide a more realistic understanding of the complex dynamics inherent in business cycles. However, they concluded that their model did not properly account for the no-linearity in the business-cycle dynamics.

These foundational studies established a framework for subsequent research in the early 2000s, shaping the landscape of economic forecasting methodologies.

2.2 Machine Learning and Economic Forecasting

While traditional models have provided valuable insights, the integration of machine learning techniques presents a promising avenue for advancing the accuracy and efficiency of economic predictions.

Schumacher and Breitung (2008) introduced a short-term factor model for forecasting German GDP. Their primary objective was to provide a real-time estimation of GDP using a combination of monthly and quarterly time series data. To address irregularities in data frequency, they utilized an Expectation-Maximization (EM) algorithm along with Principal Component (PC) decomposition. Forecasting was conducted using a standardized VAR time series model, as well as direct and linear multivariate regression models. The dataset covered indicators such as GDP, quarterly demand, gross value added, industrial production, incoming orders by sectors, interest rate, stock price index, exchange rate, business confidence index and others from 1991 to 2005, incorporating a maximum of 52 time series due to different frequencies of values in the dataset. The forecast exhibited modest performance, leading to the conclusion that real-time GDP forecasting posed a challenging task.

Ng et al. (2007) attempted to forecast housing prices by employing Linear Regression Analysis (LRA) and Genetic Algorithm (GA), utilizing a set of 10 economic indicators. Their conclusion indicated that the GA models outperformed the LRA models, with the latter exhibiting low R^2 values.

Stock and Watson (2008) reviewed previous researches on methods predicting inflation and found that the conclusions reached by the authors strongly depended on the time period the sample was from reiterating the complexities regarding forecasting of financial indicators. They proposed an empirical forecasting method based on dynamic factors and compared its performance with alternative models. Quarterly US economic data from 1953:Q1 to 2008:Q1 was selected to assess the performance of six existing prototype models. The prototypes included 4 categories of Single-equation inflation forecasting models which were ARIMA model, Unobserved Components Stochastic Volatility (UC-SV) model (Stock-Watson, 2007), Regression model and Phillips curve forecasts models. Stock and Watson concluded that there is strong evidence of time variation in the inflation process and Phillips curve forecasts, consistent with the literature's varying conclusions based on sample periods. They emphasise the need for further empirical work to formalise these observations.

Research indicated that predicting future values of economic indicators using mathematical and machine learning models holds greater qualitative significance than the specific forecasted values due to the intricate complexities involved. Hendry in his 2011 paper 'Mathematical Models and Economic Forecasting: Some Uses and Mis-Uses of Mathematics in Economics' debated that while some mathematical models have merits in identifying the underlying factors of an economic problem, there are a lot of examples where this is not the case. He argued that the prevalent mathematical approaches deviated at times from realistic economic forecasts due to errors in model selection and expectations.

2.3 Current State of Research on BCI Prediction

Despite the wealth of literature on economic forecasting, a noticeable gap exists in dedicated studies specifically addressing the predictive capabilities of machine learning models for the Business Confidence Index. A comprehensive review revealed limited academic contributions in this area, with only a few attempts to apply ML algorithms to understand or forecast BCI.

Khan and Upadhayaya (2019) examined the potential link between the Business Confidence Index (BCI) and investment growth, utilizing US business confidence survey data from 1955 to 2016. Their key findings suggested that business confidence was a reliable predictor of investment growth. In particular, BCI excelled in forecasting investment downturns and the sign of growth over specific forecast horizons.

In their study on forecasting the Consumer Confidence Index with tree-based MIDAS regressions, Qiu (2020) introduced a tree-based regression model to predict the Consumer Confidence Index (CCI), an equally crucial metric as the BCI. The research incorporated sentiment data from social platforms and applied a mixed data sampling (MIDAS) method, revealing enhanced forecasting efficiency by retaining more sophisticated information from the sentiment data.

Los and Ocheretin (2019) endeavoured to employ regression models for predicting the BCI in five European countries using selected economic indicators, as outlined in Table 1. However, due to the narrow margin of error for BCI, the authors suggested that these models are better suited for qualitative analysis. The limitations of their approach were the use of only 3 to 4 indicators, very small data points and the simplicity of the models. This led to a certain level of ambiguity in the results.

Country	Time Interval	R^2	CI	Economic Indicators
Ukraine	2007 – 2018	0.882	95%	- Producer Prices (index points) - Unemployment rate (% labour force) - GDP (annual growth rate)
Slovenia	2000 -2018	0.701		
Germany	2005 -2018	0.787	95%	- Producer Prices (index) - Unemployment rate (% labour force) - GDP (annual growth rate) - New orders (index points)
Hungary	2001 -2018	0.382	N/A	
Poland	2006 -2018	0.773	95%	

Table 1: Dataset, indicators and results of regression models (Los and Ocheretin, 2019).

2.4 Challenges and Opportunities in BCI Prediction Using ML

The absence of an all-inclusive research in ML applications for BCI underscored the need for further exploration in this domain. Challenges may stem from the unique characteristics of sentiment-based indicators like BCI, requiring tailored approaches for effective modelling. The opportunity lies in the potential of ML to uncover hidden patterns within qualitative sentiments, offering a more subtle understanding that traditional models might overlook.

This literature review therefore indicated that qualitative analysis of economic indicators was more practical, as the actual predicted values from any model might not accurately represent the future BCI value.

CHAPTER 3

3. Dataset

3.1 Dataset Collection and Features

The datasets for this study were obtained from Kaggle, a well-known and reliable data repository for data scientists. The original sources of the collated datasets were cited by the author and verified by myself from the following databases: 'The Federal Reserve Bank', 'US Bureau of Labor Statistics', 'The United States Census Bureau', 'US Bureau of Economic Analysis', 'The World Bank', 'International Monetary Fund (IMF)' and 'Organization for Economic Co-operation and Development (OECD)'.

The initial datasets contained a total of 28 US economic indicators. Some indicators, identified as derivatives of others, were excluded during the data preprocessing step. Table 2 presents the final set of 17 selected indicators which served as the raw dataset for this analysis.

Name	**Recording Frequency**	**Period**	**No. of Data points**
Bank Credit	Weekly	1973 - 2023	2,625
BCI	Monthly	1950 - 2023	886
Commercial Real Estate Prices	Quarterly	2005 – 2022	70
Consumer Loans	Weekly	2000 – 2023	1,191
Unemployment Claims	Weekly	1967 – 2023	2,937
Delinquency Rate	Quarterly	1991 – 2022	128
Federal Funds Effective Rate	Daily	1954 - 2023	25,143
Consumer Price Index	Monthly	1983 – 2023	483
Personal Saving Rate	Monthly	1959 – 2023	771
Commercial Real Estate Loans	Weekly	1987 – 2023	1,871
Real Gross Domestic Product	Quarterly	1947 – 2023	305
SPX500	Daily (Weekend Off)	1979 – 2023	10,934
Sticky Consumer Price Index	Monthly	1967 – 2023	675
Unemployment Rate	Monthly	1948 – 2023	904
US - Exports	Monthly	1987 – 2023	442
US - Imports	Monthly	1987 – 2023	442
US - Inflation	Monthly	1914 - 2023	1,319

Table 2: List of indicators selected for the analysis.

The selection of these indicators was driven by a thorough literature review, notably influenced by previous related works such as Stock and Watson (2003). Additionally, there exists a general consensus among economists, exemplified by Mishkin (1992), highlighting the crucial role of these indicators in evaluating the performance of an economy. These chosen economic factors were considered essential for accurately quantifying various financial metrics and indices, aligning with established research in the field.

The data was numeric in structure, consisted of quantitative variables, which made it easier for mathematical analysis and statistical modelling without requiring much transformation.

As this study exclusively utilised publicly available datasets and did not involve any personal data, there were no ethical concerns regarding data privacy and consent associated with the data processing for this project.

3.2 Indicator definitions

In order to grasp the relationships between the indicators and the BCI within the context of the dataset, it is essential to understand the definitions of the indicators and what they represent. Brief descriptions of the indicators are provided below.

- Bank Credit: This indicator tracks the assets and liabilities of commercial banks in the United States, denominated in billions of US Dollars on a weekly basis, specifically ending each Wednesday. The data is seasonally adjusted, offering a comprehensive measurement.

- BCI: The Business Confidence Index, a monthly metric, provides valuable insights into the forward-looking expectations of businesses. This critical index serves as a key measure for evaluating economic sentiment.

- Commercial Real Estate Prices: Quarterly changes in commercial real estate prices, expressed as a percentage change from the previous year. It serves as a valuable resource for gaining insights into the ever-evolving real estate market.

- Consumer Loans: Consumer loans, including credit cards and revolving plans, from all commercial banks. Measured weekly, the data provides a seasonally adjusted perspective, presented in billions of US Dollars.

- Unemployment Claims: Representing insured unemployment, this weekly metric counts individuals filing continued claims. It serves as a crucial source for understanding prevailing unemployment trends.

- Delinquency Rate: Measuring the delinquency rate on credit card loans from all commercial banks, this quarterly indicator is seasonally adjusted and presented as a percentage.

- Federal Funds Effective Rate: Capturing the daily percentage of interest paid between banks, this metric reflects liquidity needs and is influenced by the Federal Reserve's open market operations.

- Consumer Price Index: The average fluctuation in prices over time for a market basket encompassing consumer goods and services. It is expressed as a percentage change at an annual rate, offering a monthly view that is thoughtfully seasonally adjusted.

- Personal Saving Rate: This is calculated as personal income minus the outlays and taxes. This provides insights into personal saving habits.

- Commercial Real Estate Loans: Focusing on real estate loans from all commercial banks on a weekly basis, this indicator provides a seasonally adjusted perspective, expressed in billions of US Dollars.

- Real Gross Domestic Product: The market value of goods and services produced in the US. Recorded with seasonal adjustment on a quarterly basis in billions of US Dollars.

- SPX500: Standard and Poor's 500 (S&P 500) stock index percentage difference reflecting changes in the stock market.

- Sticky Consumer Price Index: A service-based Consumer Price Index expressed as a percentage change and seasonally adjusted and reported monthly.

- Unemployment Rate: The percentage of people actively looking for work but are currently unemployed, expressed as a percentage of the total labour force. this monthly metric is seasonally adjusted, reflecting the employment landscape.

- US - Exports: The total physical movement of merchandise exported from the United States to foreign countries. It is expressed in millions of US Dollars on a monthly basis.

- US - Imports: The total physical arrivals of merchandise from foreign countries, expressed in millions of US Dollars on a monthly basis.

- US - Inflation: Inflation measured by the consumer price index, expressed as a percentage and seasonally adjusted on a monthly basis.

3.3 Data Preprocessing

The data files for the indicators were initially separate, each containing two columns: date and the corresponding indicator values. These files were consolidated into a single pandas dataframe, and column names were adjusted as needed. A significant challenge was the disparate recording frequencies across datasets. To enable consistent analysis, it was essential to organize the data so that each row represented the respective values of indicators or feature variables on a specific date. Monthly frequency was chosen, necessitating upsampling of quarterly data and downsampling of daily and weekly data. For quarterly data, upsampling involved calculating the mean over the previous three months and equally distributing the values through backfill operation (Figure 1). Daily and weekly values were averaged as well (Figure 2 and 3). The similar method of sampling was employed by in the previous researchers as well (Stock and Watson, 2008). Due to variations in start dates among datapoints, there were a substantial number of missing values in the combined dataset. The missing values were initially imputed using a k nearest neighbour imputer, however this did not improve the model performance or results and therefore were dropped in the successive analysis. The date values were formatted and converted to datetime objects.

The final dataframe comprised 208 rows of data, encompassing 16 feature variables and 1 response variable (BCI) which was used for model building and consequent analysis.

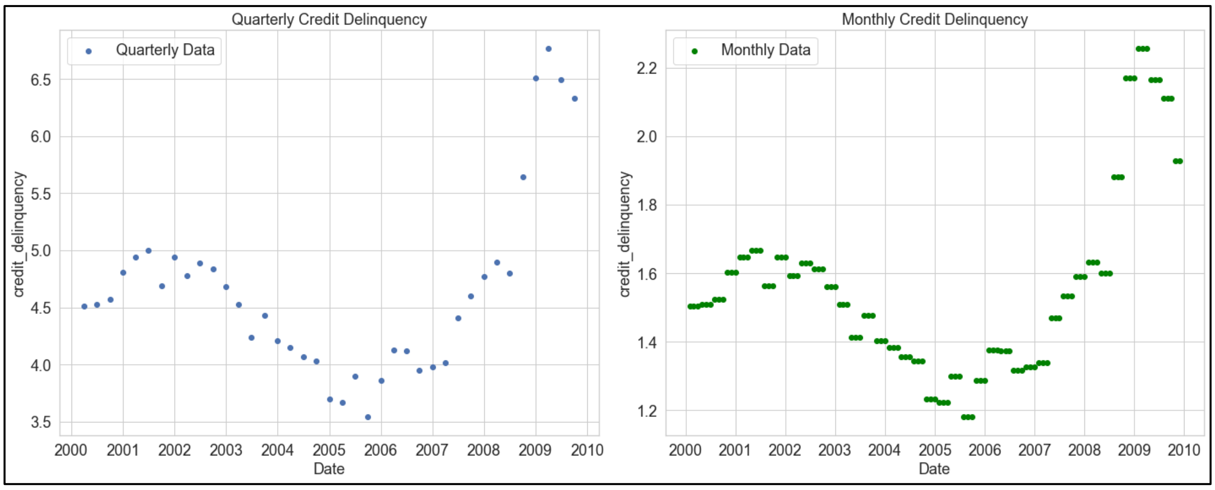

Figure 1: Upsampling of quarterly data.

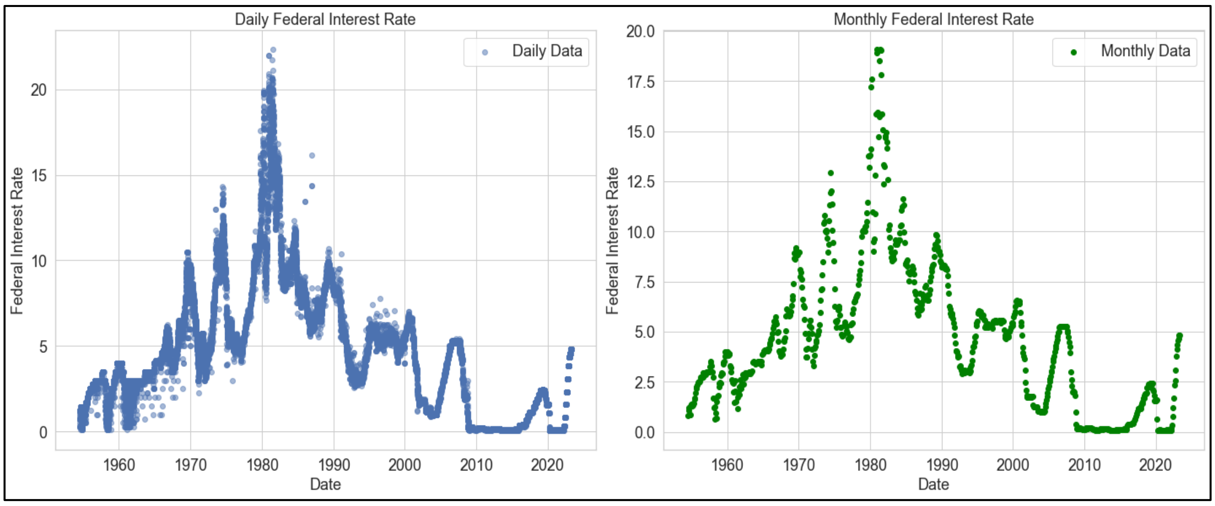

Figure 2: Downsampling of daily data.

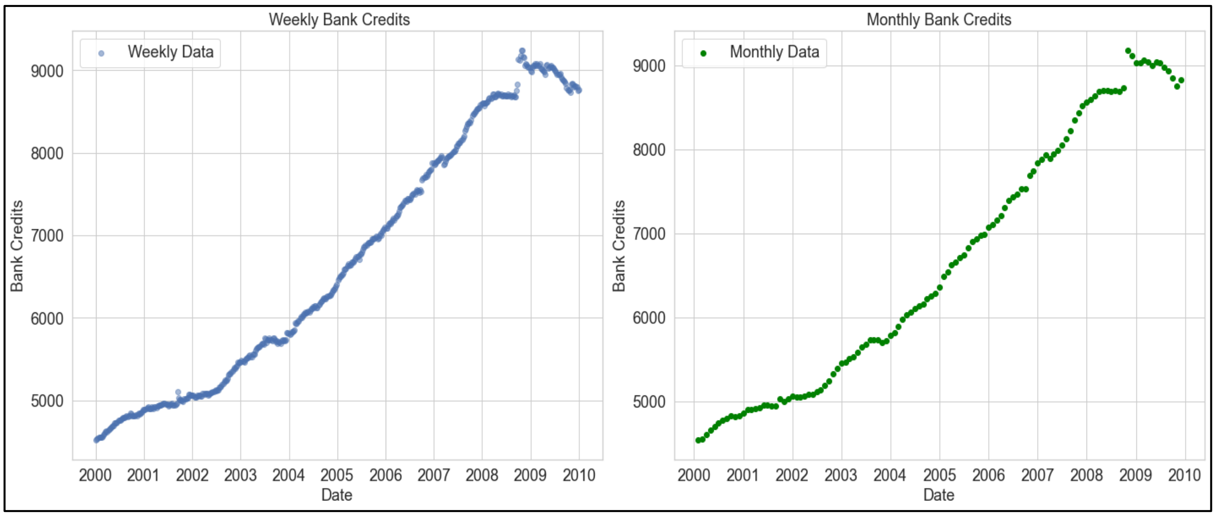

Figure 3: Downsampling of weekly data.

3.4 Exploratory Data Analysis

Scatter plots (time series), histograms, boxplots, heatmaps, and pair plots (Figure 4 to Figure 7) were generated using the feature variables to visually examine the data for irregularities, inconsistencies, correlations or patterns. The analysis revealed that except for exports, import, unemployment rate, unemployment claims there were no specific discernible pattern or correlation between the other features and the dataset demonstrated consistency.

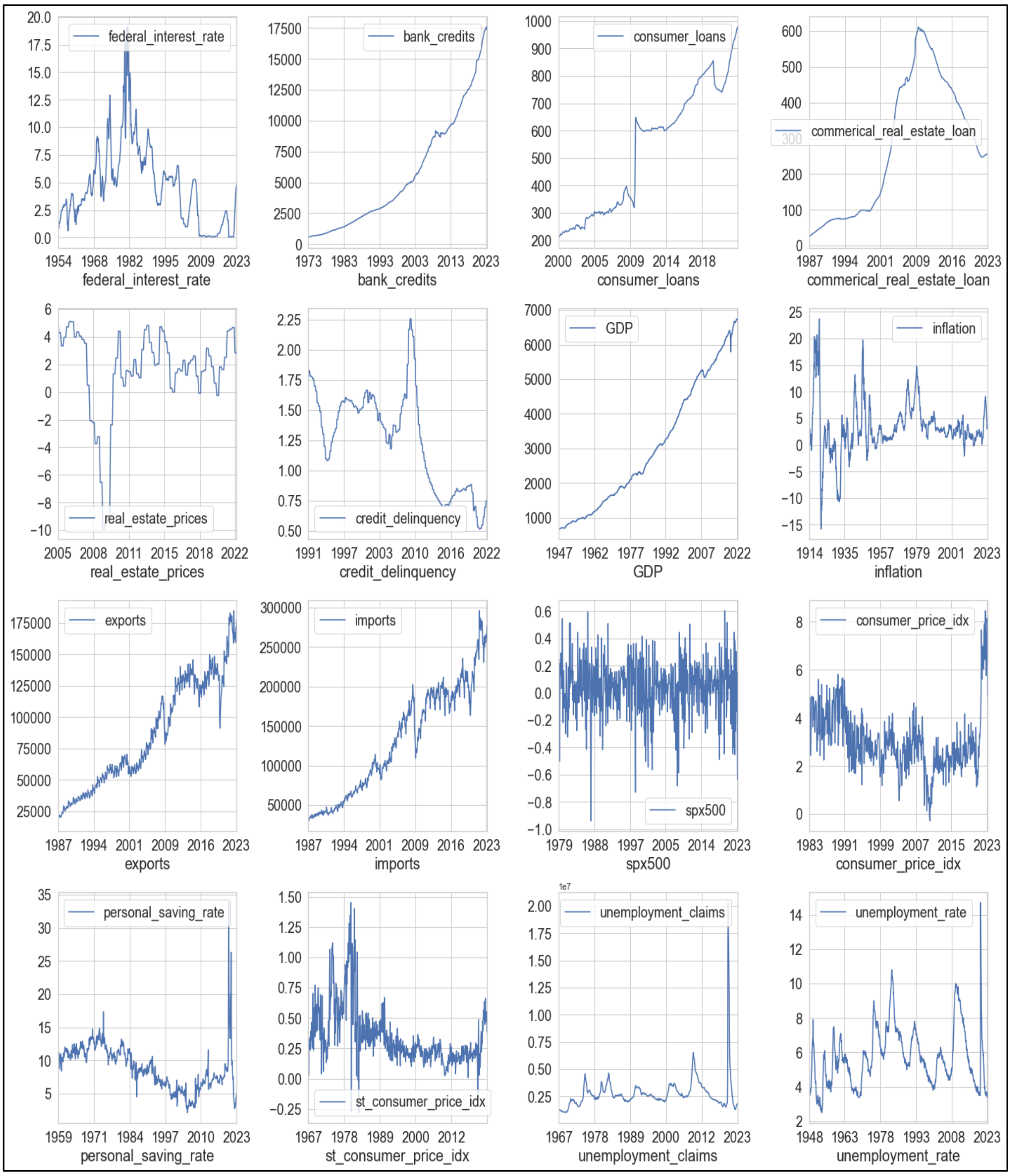

Figure 4: Time series (monthly data) of the features.

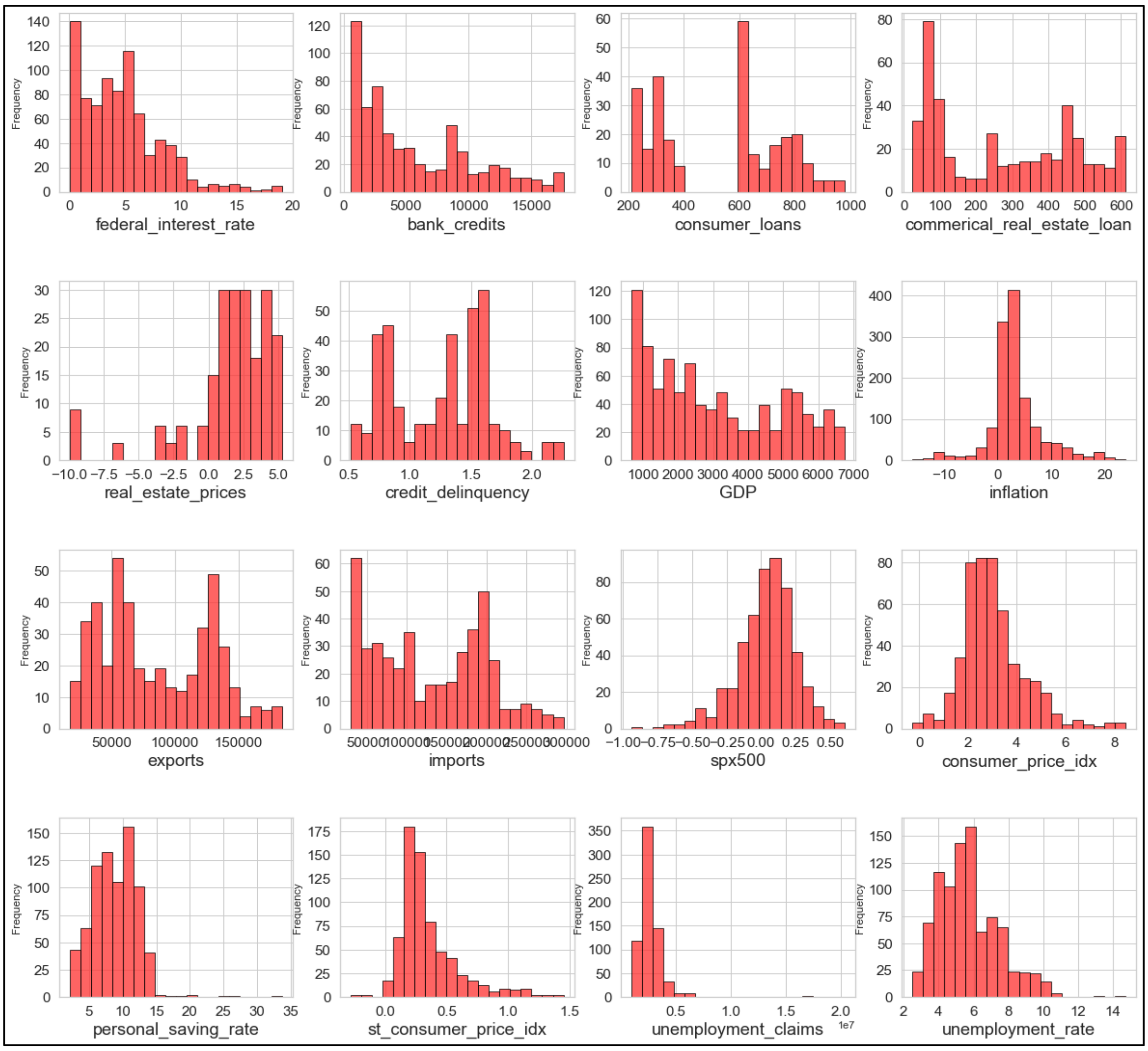

Figure 5: Features histogram.

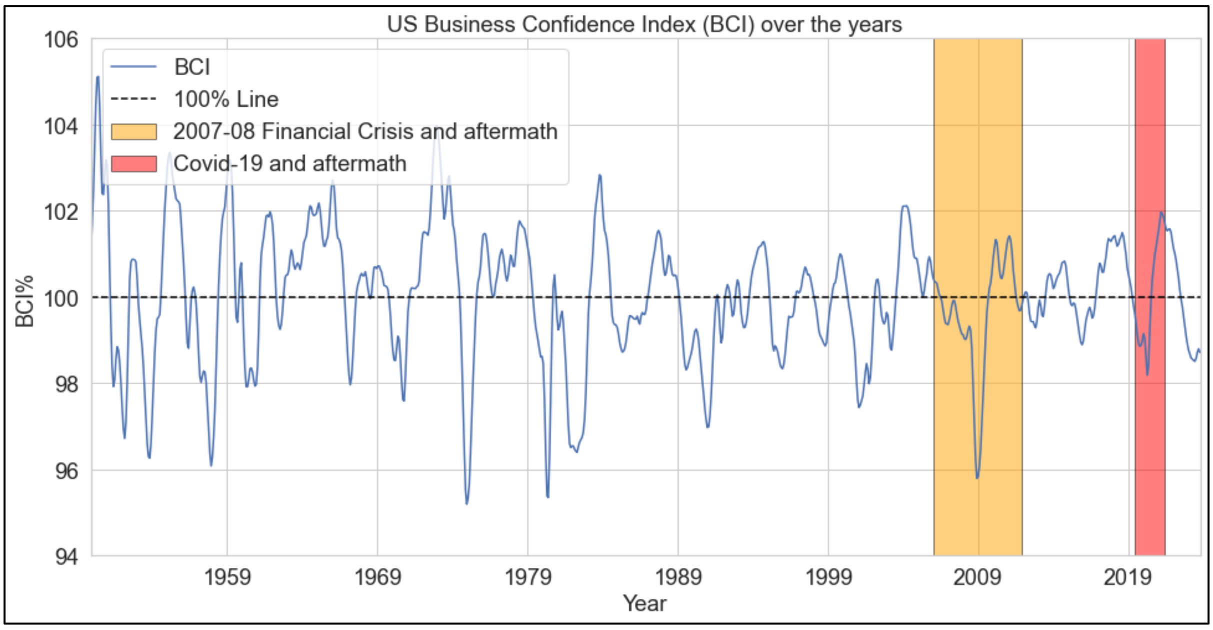

Figure 6: US Business Confidence Index (BCI) over the years.

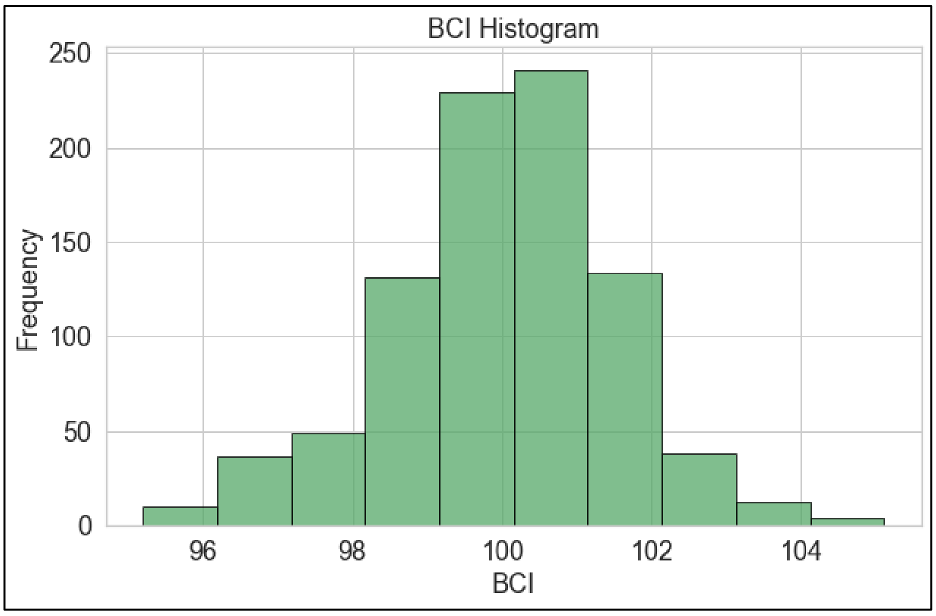

Figure 7: BCI histogram.

In addition to the above plots, summary statistics were calculated for the feature variables to provide a comprehensive overview of their distribution and central tendencies. This thorough analysis aimed to identify potential outliers, trends, or correlations within the dataset. It was observed that four features might have outliers present. However, these potential outliers were addressed during the modelling steps before conducting further analyses.

3.5 Standardisation of the Data

To ensure uniformity and facilitate the modelling process, the final dataframe was standardised using scikit-learn's StandardScaler. Standardisation centres the data around a mean of zero and scales it to have a standard deviation of one. This transformation brought all features to a common scale, preventing any particular variable from dominating the analysis due to its larger magnitude, especially as the dataset had diverse range of feature values. Standardisation, in general, contributes to the stability and convergence of machine learning algorithms during training.

Normalisation was also performed with MinMaxScaler. The key difference between the two transformations is that standardisation preserves the relationship between the data points, whereas normalisation might not be as effective in achieving this. However, normalisation performs slightly better with non-Gaussian distributions. The effect of normalisation did not alter the results of the algorithms. Consequently, only the standardised results are presented in the final analysis.

CHAPTER 4

4. Methodology

This study employed a machine learning approach to predict the Business Confidence Index. The methodology can be divided into several key steps: Data Collection, Data Preprocessing, Exploratory Data Analysis, Data Standardisation, Machine Learning Model Selection, Model Training and Evaluation, Feature Importance Analysis, Results and Conclusion.

Data Collection, Data Preprocessing, Exploratory Data Analysis and Data Standardisation were illustrated in the previous chapter. In this chapter, the model selection and training will be detailed.

4.1 Machine Learning Model Selection

Machine learning model selection is a critical step in the analytical process, and it involves choosing models that best suit the data and the objectives of the analysis. In this section, an array of machine learning models were implemented and assessed for their effectiveness in predicting the target variable. The following models were chosen for this analysis:

4.1.1 Multiple Linear Regression

Multiple Linear Regression (MLR) is a widely used statistical method to fit a model with more than two independent variables (features or predictors) and one dependent variable (response). In this part of the analysis MLR was performed on the standardised data from January 2005 to April 2022 with 208 data points.

The MLR can be represented in matrix form as below (Sheather, 2009):

$$\mathbf{Y} = \mathbf{X}\beta + \mathbf{e}$$

Where: X is the feature matrix, Y is the response variable matrix, β is the regression coefficient matrix and e is the error matrix. The objective is to minimise the errors, e to find the best model.

$$\mathbf{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \mathbf{X} = \begin{pmatrix} 1 & x_{11} \cdots x_{1p} \\ 1 & x_{21} \cdots x_{2p} \\ \vdots & \vdots \\ 1 & x_{n1} \cdots x_{np} \end{pmatrix}, \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}, \mathbf{e} = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix}$$

An initial Multiple Linear Regression (MLR-1) model was established using the statsmodels library in Python, employing the Ordinary Least Squares regression (OLS) method. In this preliminary stage, an effort was made to incorporate all available features. The resultant model exhibited a suboptimal fit with an R^2 value of 0.585. While the residuals vs predictor plots (Figure 8) failed to provide meaningful insights or discernible patterns indicative of the requirement of any transformations, the Studentized residual vs H Leverage plot (Figure 9) highlighted the presence of outliers (based on Cook's distance), significantly influencing the regression model. In a subsequent iteration (MLR-2), these data points were excluded, leading to a notable improvement in the model fit. However, this step revealed that certain features (federal_interest_rate, commercial_real_estate_loan, real_estate_prices, exports, imports, consumer_price_idx, spx500, unemployment_rate) lacked statistical significance in relation to the model output. Consequently, these features were excluded in the subsequent model fitting, leaving 8 indicators for the analysis. Table 3 outlined various parameters for the three distinct model fits.

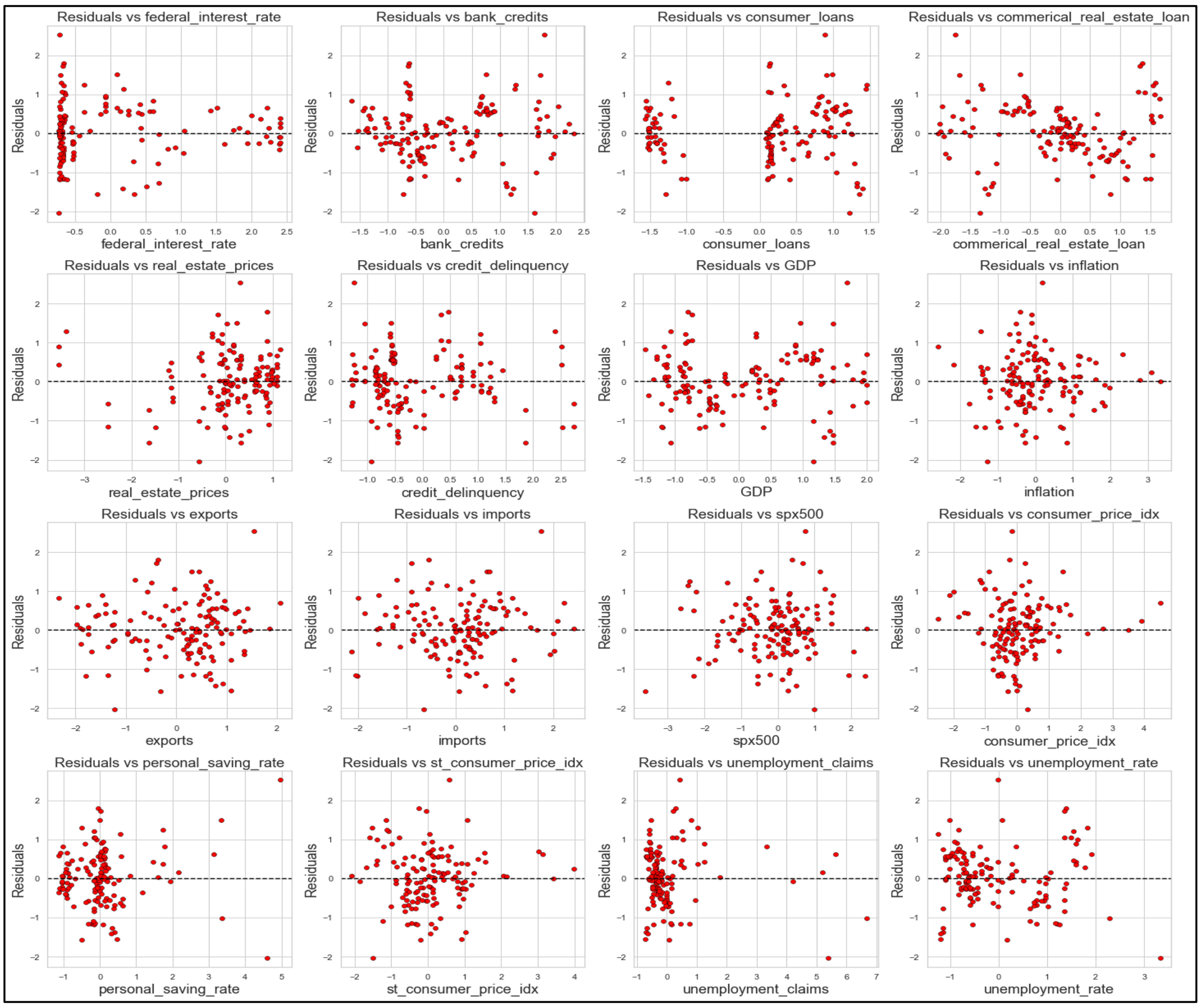

Figure 8: Residuals vs Predictors plot.

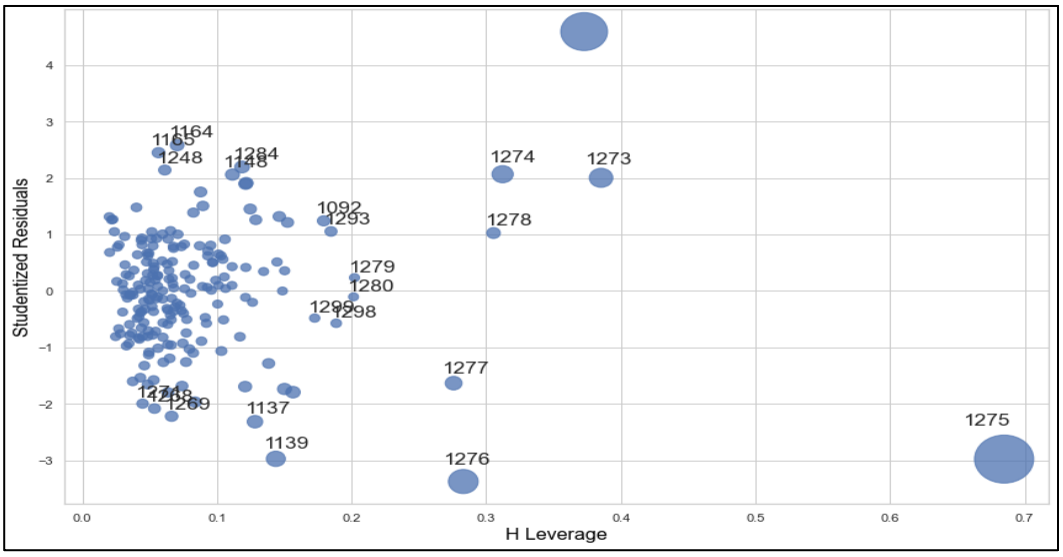

Figure 9: Studentized Residuals vs H Leverage plot shows the outliers and their influence on the regression fit for MLR-1. Absolute residuals value up to 3 and leverage value of 0.2 were considered for the model fit.

	MLR-1	MLR-2	MLR-3	Interpretation
R-squared:	0.585	0.725	0.704	The coefficient of determination. The best model has the value close to 1.0.
MSE:	0.536	0.336	0.345	Mean Squared Errors. The best fit has this value close to 0.
F-statistic:	16.82	27.79	52.58	A test of significance for the model. Ideal value is greater than F-critical value on the F-distribution. In the case of MLR-1, df1 = 16 and df2 = 191. The F-critical = 1.67 for alpha = 0.05.
Prob (Fstatistic):	1.6E-28	4.30E-39	7.39E-43	The p-value of the F-Distribution. Should be close to zero for the model to be valid.
AIC:	476.8	341.4	338.9	The Akaike information criterion (AIC) is an estimator of prediction error, denotes relative quality of statistical models.
Cond. No.:	27.6	50.1	26.1	Indicates the existence of multicollinearity. If the value is above 20 then this should be investigated.
No. of features:	16	16	8	Feature variables used in the model fit.

Table 3: Comparison of MLR models.

The residuals plots were constructed (Figure 10) and it revealed that residual are normally distributed as expected from standard linear models. The R^2 value and MSE for the best model clearly indicated that a linear regression model would not be an appropriate choice to predict BCI values. However, the insights on the predictors in the form of their coefficients can be of importance. This analysis also pointed towards the consideration of dimensionality reduction techniques to address possible multicollinearity issues in the dataset. The coefficients for MLR-3 is listed in Table 4.

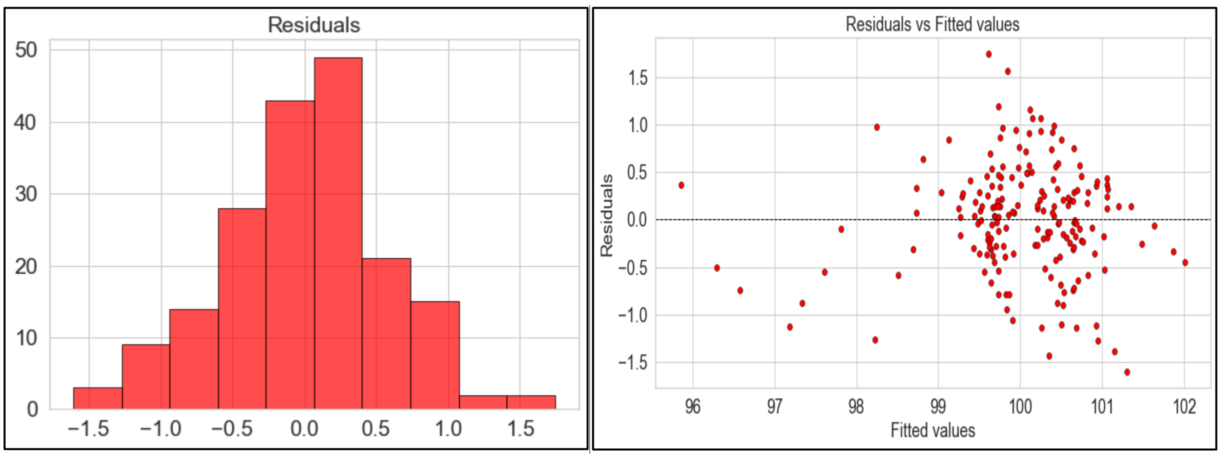

Figure 10: Residuals plots. The histogram had the shape of a normal distribution. Residual vs Fitted Values plot had residuals spread around a mean of zero.

Feature	Coefficient
bank_credits	-3.964204
personal_saving_rate	-1.285306
credit_delinquency	-0.45429
st_consumer_price_idx	-0.263276
consumer_loans	0.331603
inflation	0.504472
unemployment_claims	2.803533
GDP	4.585927
const	100.235796

Table 4: Coefficients for the statistically significant features in the MLR-3 model.

4.1.2 Principal Component Analysis (PCA)

Principal Component Analysis, one of the most popular dimensionality reduction techniques used in multivariate statistics, was implemented to validate the insights from the MLR analysis.

The primary function of PCA is to provide insight on the reduction of the feature variables to an optimum number that can explain the most variations in a dataset by transforming the variables into principal components in an orderly fashion (Jolliffe, 2002). Each principal component is a linear combination of the features. In an N-dimensional space PCA finds the line, planes or hyper-planes (depending on the value of n) that have the largest variance along based on a least squares estimate. Figure 11 illustrates an ideal example of a PCA plot for 2 variables. The PC1 explains the most variance along the line.

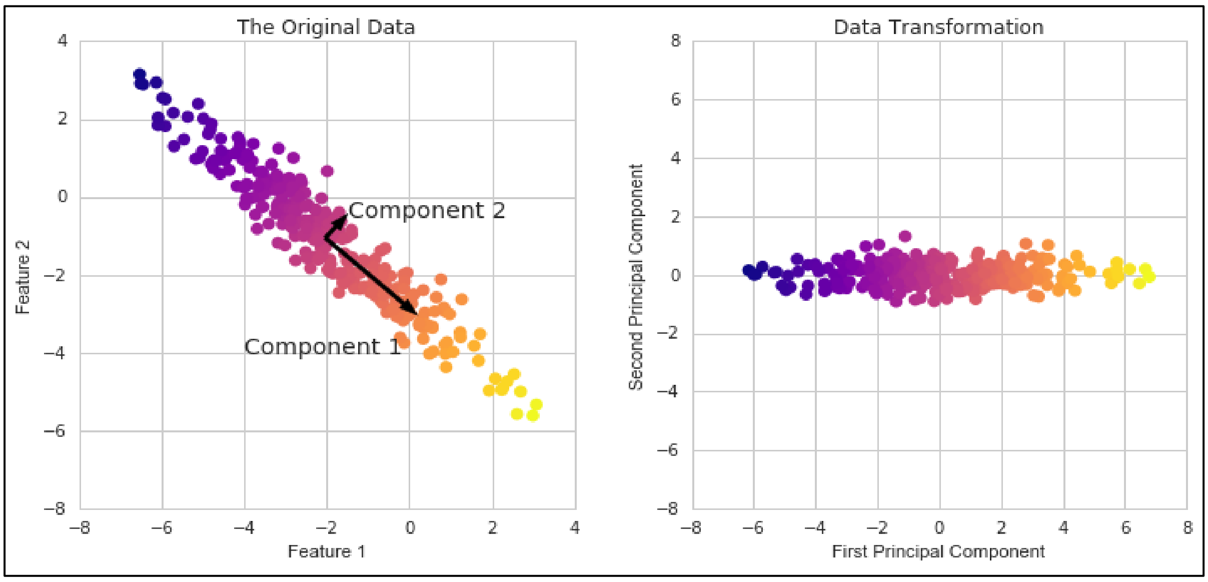

Figure 11: Standard PCA plot.

A PCA model was fit using the standardised dataset with all the features. The PC2 vs PC1 plot (Figure 12) indicated the presence of non-linearity or clusters among the indicators. The explained variance ratio plot shows that 91% of the variations in the dataset can be explained with 6 principal components.

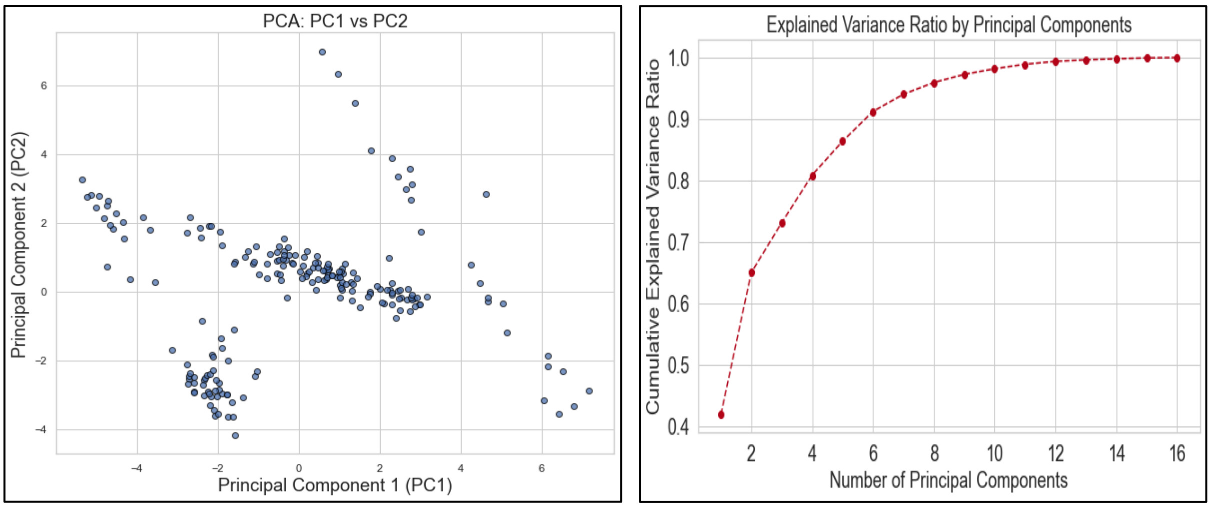

Figure 12: (Left) PCA plot with the first 2 component. (Right) Cumulative explained variance plot.

It was noted that PC1 and PC2 can explain 65% of the variance, these components loadings are listed in the table below.

Loadings for PC1:			Loadings for PC2:	
GDP	0.363651		personal_saving_rate	0.366985
imports	0.352361		unemployment_rate	0.362916
bank_credits	0.342693		unemployment_claims	0.304569
consumer_loans	0.32415		consumer_loans	0.222164
exports	0.321582		bank_credits	0.161481
consumer_price_idx	0.21567		spx500	0.140585
st_consumer_price_idx	0.187717		exports	0.109335
real_estate_prices	0.174719		GDP	0.102857
personal_saving_rate	0.163734		commerical_real_estate_loan	0.087999
inflation	0.134685		imports	0.012683
spx500	0.004905		credit_delinquency	-0.05211
unemployment_claims	-0.05045		real_estate_prices	-0.22266
federal_interest_rate	-0.0998		st_consumer_price_idx	-0.31787
unemployment_rate	-0.18117		consumer_price_idx	-0.31904
commerical_real_estate_loan	-0.3259		inflation	-0.32793
credit_delinquency	-0.32998		federal_interest_rate	-0.39059

Table 5: PC1 and PC2 loading coefficients.

Loading coefficients indicate the strength and direction of the relationship between the original features and the principal components. Feature with a higher absolute loading value denotes stronger influence.

Examination of the top loadings for PC1 revealed that this component captured variance from the indicators that represented the broader economic and financial features. PC2 loadings on the other hand were more related to financial activities by individuals. The emphasis on two different aspects of the economy by the principal components (PCs) suggested that PC1 and PC2 captured the distinct dimensions of variation in the original features.

To summarise, this analysis concurred with the exclusion of features in the multiple linear regression analysis.

In order to tackle potential non-linear relationships among the features as indicated by the above, I explored the applicability of algorithms better equipped to address the challenge.

4.1.3 Support Vector Machine (SVM) Regression

Support Vector Machine Regression or Support Vector Regression (SVR) offers a significant advantage in handling non-linear processes through the introduction of a kernel function, facilitating the projection of original data into a linearly or non-linearly separable high-dimensional space (Smola and Scholkopf, 2003). Consequently, SVR was chosen as the preferred method for constructing regression models, particularly suited for addressing variables with non-stationary or time dependent behaviour. This strategic selection aligned with the inherent capacity of SVR to effectively navigate the complexities associated with non-linear relationships within the dataset.

The non-linear function approximation of SVR expressed as (Huang et al, 2006):

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{G}\mathbf{v}_0 + b$$

Where,

Weighting Vector, $V_o = f(x_i)$

Kernel (Grammian) matrix $G = f(x_i * x_j)$

This is can be expressed as:

$$f(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}_o^T \boldsymbol{\Phi}(\mathbf{x}) + b = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) \boldsymbol{\Phi}^T(\mathbf{x}_i) \boldsymbol{\Phi}(\mathbf{x}) + b$$
$$= \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(\mathbf{x}_i, \mathbf{x}) + b.$$

Here, b is the bias term and has two expressions for upper and lower SV limits

$$b = y_i - \sum_{j=1}^{N free\, upper\, SVs} (\alpha_j - \alpha_j^*) \mathbf{\Phi}^T(\mathbf{x}_j) \mathbf{\Phi}(\mathbf{x}_i) - \varepsilon$$
$$= y_i - \sum_{j=1}^{N free\, upper\, SVs} (\alpha_j - \alpha_j^*) K(\mathbf{x}_i, \mathbf{x}_j) - \varepsilon, \text{ for } 0 < \alpha_i < C$$

$$b = y_i - \sum_{j=1}^{N free\, lower\, SVs} (\alpha_j - \alpha_j^*) \mathbf{\Phi}^T(\mathbf{x}_j) \mathbf{\Phi}(\mathbf{x}_i) + \varepsilon$$
$$= y_i - \sum_{j=1}^{N free\, lower\, SVs} (\alpha_j - \alpha_j^*) K(\mathbf{x}_i, \mathbf{x}_j) + \varepsilon, \text{ for } 0 < \alpha_i^* < C.$$

$$\text{C} = \ regularization\, parameter,$$

$$\in = also\ called\ \varepsilon - insensitivity, loss\ (error)\ function$$

The SVR model was implemented using a Kernel function called Radial Basis Function, (RBF). RBF is commonly used in SVR, and it allowed the SVR model to map input data into a higher-dimensional space to find a non-linear decision boundary. Model was put through an iterative search to optimize hyperparameters C and ε. A higher C value indicated a more accurate separation of training points, while ε was chosen within a specified range, with smaller values being preferable. The optimal values were determined to be C = 50 and ε = 0.05. The resulting model demonstrated an R^2 value of 0.843 and MSE of 0.219.

Feature Name	**Importance**
bank_credits	1.502849
consumer_loans	0.877413
GDP	0.582817
unemployment_rate	0.510278
federal_interest_rate	0.502456
credit_delinquency	0.355209
exports	0.327844
commerical_real_estate_loan	0.327527
real_estate_prices	0.272059
inflation	0.259296
st_consumer_price_idx	0.228587
imports	0.117421
consumer_price_idx	0.08177
personal_saving_rate	0.051516
unemployment_claims	0.036052
spx500	-0.00029

Table 6: Feature importance of SVR model.

4.1.4 Random Forest (RF)

Random Forest is an ensemble machine learning algorithm proposed by Breiman (2001) utilises decision trees to solve both classification and regression problems. An ensemble learning algorithm integrates multiple machine learning models to build a better model. For regression, Random Forest constructs numerous decision trees using bagging and calculates the mean of the aggregated trees. The trees are constrained by hyperparameters at each node, specifically the number of features that can be split. This ensures that all features are represented preventing any specific feature from getting preference or higher influence in the ensemble (Liaw and Wiener, 2002). To prevent overfitting, a random sample is extracted from the initial dataset by each tree while creating its splits, introducing an additional layer of randomness, hence the name Random Forest. The trees are constructed to achieve the maximum information gain.

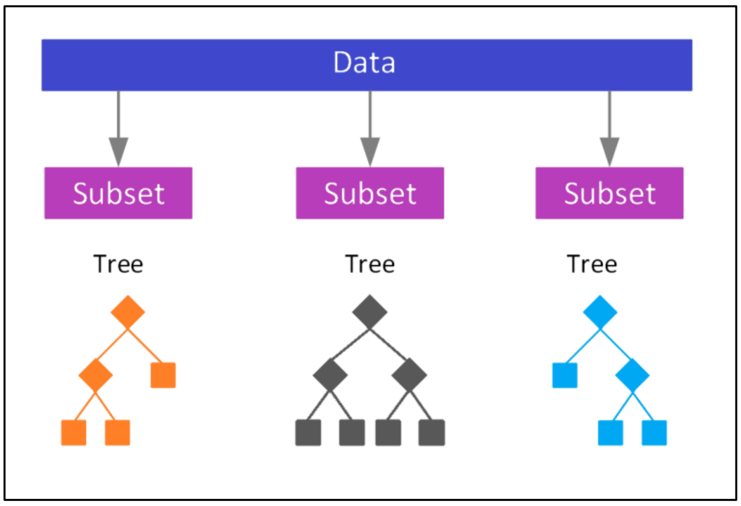

Figure 13: Simplified schematic of a Random Forest ensemble (Nvidia, 2024).

In my RF model, the ensemble aimed to minimize the mean squared error (MSE). The RF model fit had R^2 = 0.829 and MSE = 0.238. Figure 14 and 15 illustrates the quality and feature importance of the model.

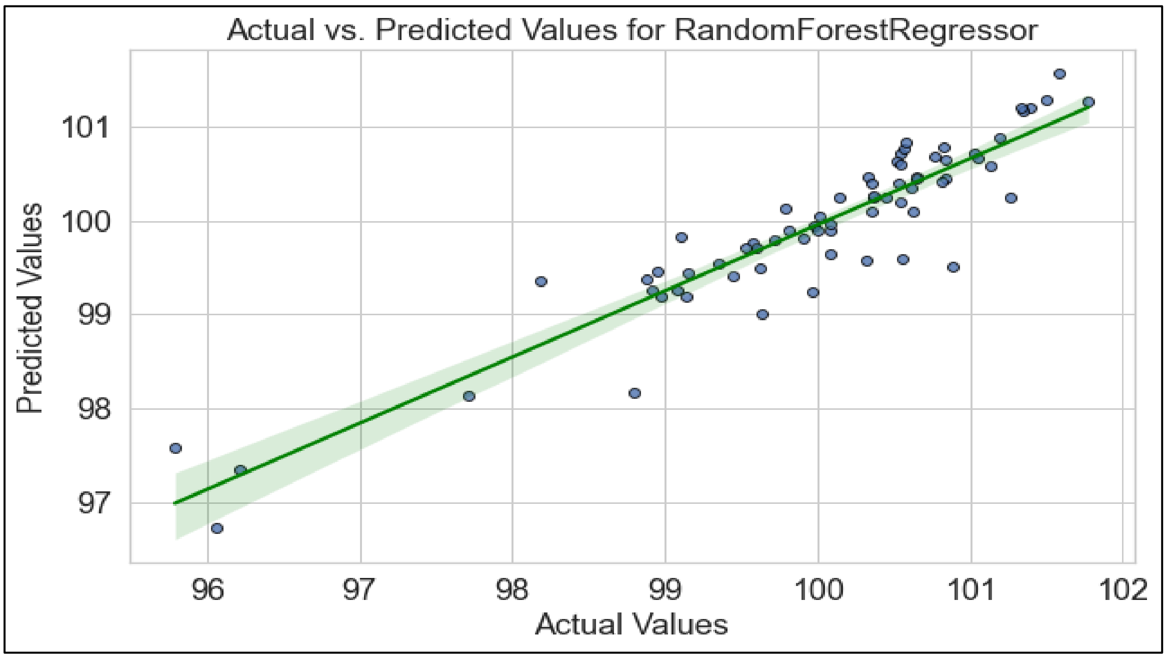

Figure 14: Actual vs predicted values for BCI plot showed a reasonable match for the Random Forest model.

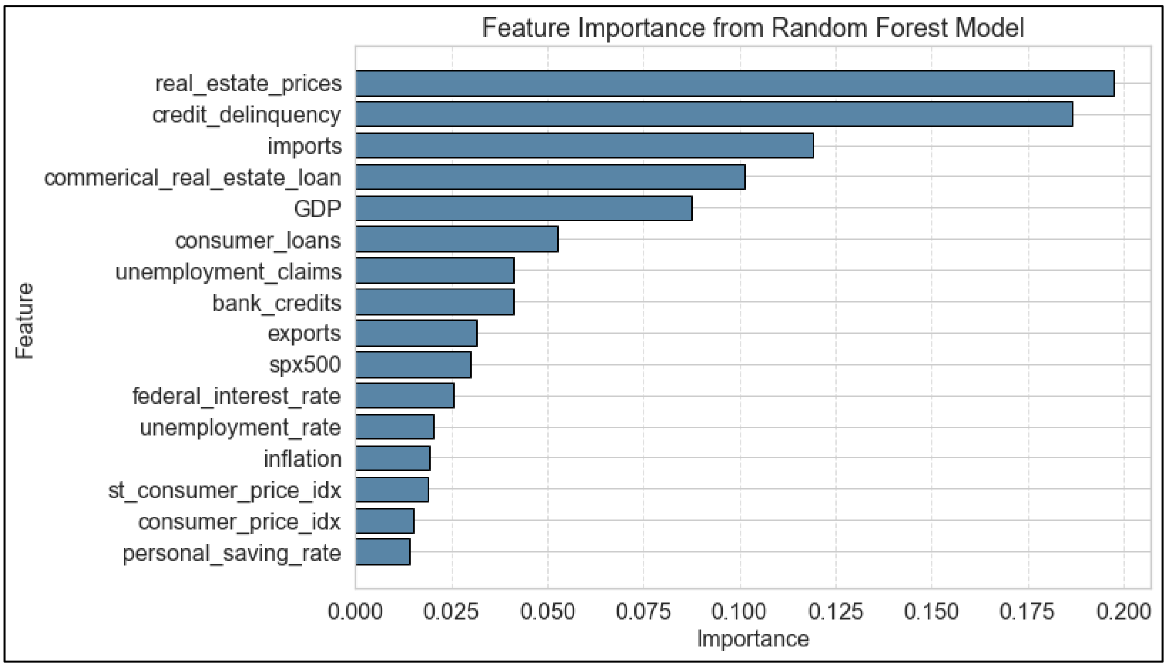

Figure 15: Feature importance plot demonstrated the influence of different features on the model.

4.1.5 Gradient-Boosted Decision Tree (XGBoost)

An advanced ensemble method similar to Random Forests, Gradient-Boosted Decision Trees (Schapire, 1999) involves the iterative construction of decision trees, where each subsequent tree corrects the errors of the previous ones. This boosting technique aims to improve overall model performance by sequentially focusing on the weaknesses of the individual trees, leading to a more accurate and robust predictive model. The iteration for gradient boosting is expressed by the following formula:

$$F_{i+1} = F_i - f_i$$

where:

- F_i is the strong model at step i.
- f_i is the weak model at step i.

The iteration continues until the termination condition is reached.

I used XGBoost (Extreme Gradient Boosting) library to fit the standardised data. An iterative search for the optimum hyperparameters was run. The results are shown in the table below.

Best parameters:	learning_rate = 0.1, max_depth = 5, n_estimators = 500
Mean Squared Error:	0.14643752971996662
R-squared:	0.8948343258494091

Table 7: XGBoost model parameters.

Feature importance plot (Figure 16) for this model showed noticeable differences from Random Forest.

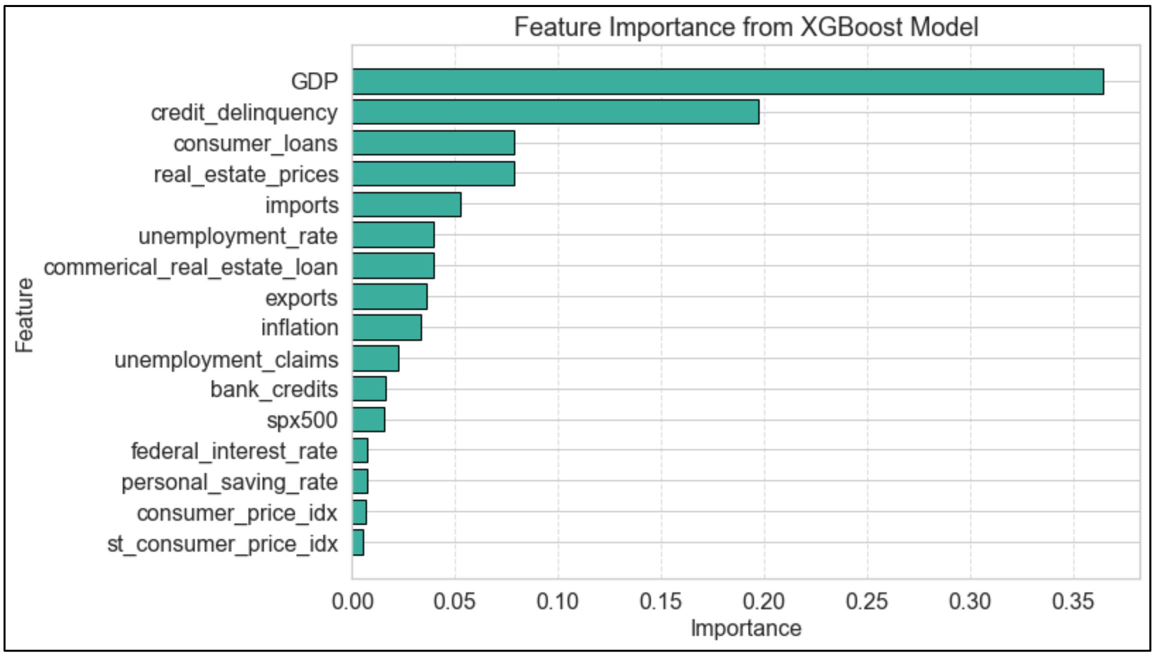

Figure 16: Feature importance plot for XGBoost model.

4.1.6 Artificial Neural Networks

Artificial Neural Networks (ANN) form the foundation of contemporary deep learning applications, operating by mimicking the functioning of the human brain (Müller et al., 1995). At the core of ANN are its layers and their associated weights. Each layer contains nodes which combine the input data with parameters called weights (much like linear regression) during the training phase to predict the output variable. The layers are regulated by selected thresholds called activation functions which incorporate non-linearity to the network, enabling the network to learn and represent complex patterns.

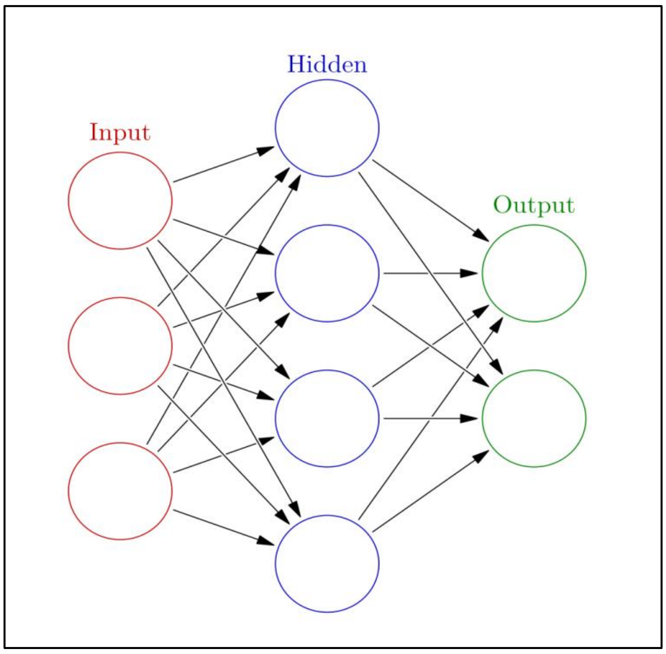

Figure 17: Simplified diagram of a Neural Network (Wikipedia, 2013).

A sequential feed forward neural network was set up 16 dense layers comprising 16 neurons with a Rectified Linear Unit (ReLU) activation function, adept at handling non-linear relationships within the data. Following this, a Dropout layer was added, featuring a dropout rate of 0.5 to act as a regularization technique to mitigate overfitting. The subsequent layer consisted of 8 neurons, ReLU activation, and L2 regularization with a strength of 0.1, aimed at curbing model complexity. The concluding layer was a dense layer with a single neuron with linear activation, specifically tailored for regression tasks. This architecture (Figure 18) was designed to strike a balance between model complexity and generalisation, essential for effective regression predictions.

The ANN model was compiled with Adam optimizer and mean square error as the loss function. It was unable to provide a good response to the dataset when tested with different hyperparameters. The mean squared errors changed erratically on different runs of the model and values ranged between 10 and 1000. The training loss was higher than the validation loss (Figure 19), an unusual result. The 'black box' nature of ANN did not allow any further interpretations on the underlying reasons. However, there is room for exploration in the usages of ANNs.

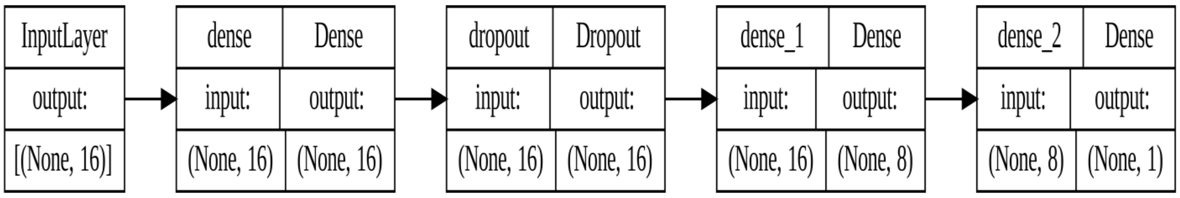

Figure 18: ANN model schematic.

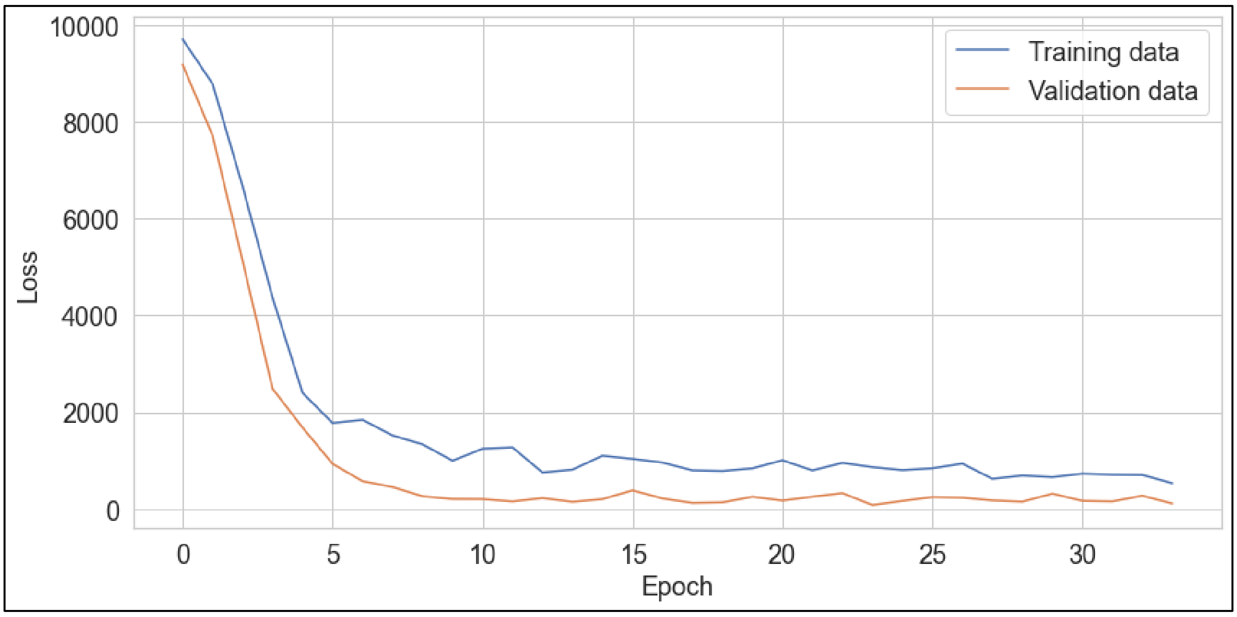

Figure 19: Training and validation losses for the ANN.

CHAPTER 5

5. Results

The metrics commonly used in econometric regression analysis are coefficient of determination, R^2 score and the mean squared error, MSE. R^2 denotes the quality of a model.

$$R^2 = 1 - \frac{RSS}{TSS}$$

Where,	R^2	=	Coefficient of determination
	RSS	=	Sum of squares of residuals
	TSS	=	Total sum of squares

The best model has a R^2 value close to 1 and the MSE close to 0.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

MSE	=	Mean squared error
n	=	Number of data points
Y_i	=	Observed values
$\hat{Y}_i$	=	Predicted values

These 2 metrics were used to assess the performance and quality of the models.

5.1 Multiple Linear Regression (MLR)

MLR-1 was implemented using the statsmodels library with Ordinary Least Squares regression (OLS). All features were considered, resulting in an R^2 value of 0.585. Subsequently, outliers identified in the Studentized residual vs H Leverage plot led to the creation of MLR-2, showing improvement. Features lacking significance were excluded in MLR-3, leaving 8 indicators.

The equation for BCI based on the linear regression coefficients can be expressed as:

BCI = **100.236** + 4.585927 * **GDP** + 2.803533***unemployment_claims** + 0.504472***inflation** + 0.331603***consumer_loans** – (3.964204***bank_credits** + 1.285306***personal_saving_rate** + 0.45429***credit_delinquency** + 0.263276***st_consumer_price_idx**)

5.2 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) was employed to validate MLR insights. PCA reduced feature variables and captured 91% variance with 6 principal components. PC1 and PC2 (Table 5) revealed distinct dimensions of variation, supporting feature exclusion in MLR.

5.3 Support Vector Machine Regression (SVR)

Support Vector Machine Regression (SVR) addressed non-linearities and was able to fit a good model. Optimal hyperparameters (C = 50, ε = 0.05) yielded an R^2 of 0.843 and MSE of 0.219. The top 6 features that influenced the hyperplane were: bank_credits, consumer_loans, GDP, unemployment_rate, federal_interest_rate and credit_delinquency.

5.4 Random Forest (RF)

Random Forest minimized MSE, achieving R^2 = 0.829 and MSE = 0.238. The most important features for the model were: real_estate_prices, credit_delinquency, imports, commerical_real_estate_loan, GDP, and consumer_loans.

5.5 Gradient-Boosted Decision Tree (XGBoost)

XGBoost method showed the best results: R^2 = 0.895 and MSE = 0.146. GDP, credit_delinquency, consumer_loans, real_estate_prices, imports, and unemployment_rate were the vital indicators in the model decision.

5.6 Artificial Neural Networks (ANN)

The Sequential Neural Network struggled with erratic mean squared errors. Therefore, the results lacked reliability..

Comparisons of the implemented models are shown in Table 8 and Table 9.

Table 8: Model Comparison						
Model	**R^2 Score**	**MSE**	**F-statistic**	**AIC**	**Cond. No.**	**No. of features**
MLR-1	0.585	0.536	16.82	476.8	27.6	16
MLR-2	0.725	0.336	27.79	341.4	50.1	16
MLR-3	0.704	0.345	52.58	338.9	26.1	8
SVR	0.843	0.219	-	-	-	16
Random Forest	0.829	0.238	-	-	-	16
XGBoost	0.895	0.146	-	-	-	16
ANN	-	Varied values	-	-	-	16

Table 8: Model Comparison.

	Feature Importance			
Feature Name	**MLR-3**	**SVR**	**RF**	**XGBoost**
bank_credits	High	High	Low	Low
consumer_loans	Moderate	High	Low	Moderate
GDP	High	High	Moderate	High
unemployment_rate	Insignificant	Moderate	Insignificant	Low
federal_interest_rate	Insignificant	Moderate	Insignificant	Insignificant
credit_delinquency	Moderate	Moderate	High	High
real_estate_prices	Insignificant	Insignificant	High	Moderate
imports	Insignificant	Insignificant	Moderate	Moderate
commerical_real_estate_loan	Insignificant	Insignificant	Moderate	Insignificant
inflation	Moderate	Insignificant	Insignificant	Insignificant
unemployment_claims	High	Insignificant	Insignificant	Insignificant
personal_saving_rate	High	Insignificant	Insignificant	Insignificant
st_consumer_price_idx	Low	Insignificant	Insignificant	Insignificant

Table 9: Feature importance comparison.

To summarise, SVR, Random Forest, and XGBoost models outperformed MLR and ANN, suggesting non-linear relationships in BCI prediction. Further exploration into feature importance and model interpretability is presented in the discussion and analysis section.

CHAPTER 6

6. Analysis and Discussion

In this study, I have implemented linear regression, decision tree based models which have explainable parameters as well as Convolutional Neural Network to determine their usability and compare the produced results.

Initially, linear regression was attempted due to its simplicity and interpretability. However, the model exhibited suboptimal performance, prompting an investigation into potential issues such as multicollinearity. However, this simpler model provided valuable insights into the influence of various features on the BCI. Principal Component Analysis was performed to confirm the presence of multicollinearity and explore the underlying structure of the data. PCA revealed that a reduced set of six principal components could capture a substantial 91% of the data's variability. This finding implied a potential non-linear relationship within the dataset, motivating the exploration of more complex models such as SVR.

Support Vector Regression (SVR) has the ability to handle non-linear relationships with fewer critical parameters. However, SVR is known to be sensitive to noise in the data and may underperform with large datasets. Despite this, SVR has been successfully applied in sentiment analysis and proven useful in this case.

Decision tree-based models, including Random Forest (RF) and Gradient Boosted Decision Trees (GBDT), were also investigated. RF, employing 'bagging,' helps combat overfitting, while GBDT, using 'boosting,' minimizes underfitting and eliminates bias in the data. Both methods complemented each other, and features identified as important in both approaches can be considered to have genuine impact.

While decision tree-based models showed promise, the exploration extended to Convolutional Neural Networks (CNNs). Unfortunately, the ANN model did not yield favourable results despite the experimentation with various architectures. Neural Networks, criticized for their 'black-box' nature, hindered further investigation into the root cause of the poor response to the dataset.

Considering model performances, XGBoost demonstrated the best results with an R^2 of 0.895 and MSE of 0.146. However, it is essential to acknowledge the interpretability challenges associated with complex models. Linear regression, despite its multicollinearity issue, provided valuable insights into the impact of features on BCI. The comparison partially aligns with literature expectations, emphasizing the trade-off between model complexity and interpretability.

The results collectively underscored the challenging nature of predicting BCI values, particularly given the threshold of 100. Slight deviations above or below this threshold can lead to false predictions. Quantitative approaches, such as regression or decision trees, may not be ideal for BCI prediction. However, qualitative analysis revealed significant insights into the impact of indicators. Notably, GDP, Credit Delinquency, Consumer Loans, Real Estate Prices, and Imports emerged as key factors influencing the sentiment driving the BCI of the US economy.

7. Conclusion

In this project, a comprehensive analysis on the prediction of Business Confidence Index (BCI) was performed using Linear Regression, Support Vector Regression and Decision Tree based models. Analysis revealed several key findings and considerations on the economic factors affecting the BCI. Unsurprisingly, GDP has been found to be the most important indicator for business predictions. Credit Delinquency, Consumer Loans, Real Estate Prices, and Imports were identified as important influencers as well. Further interpretation of their economic significance would require more economic reasoning which was beyond the scope of this study.

The analysis encountered a number of limitations:

- The challenge of predicting BCI values with high precision, given the threshold nature of the index.
- Multicollinearity impacted linear regression.
- Principal Component Analysis indicated a potential non-linear relationship among the feature variables.
- The 'black-box' nature of neural networks hindered interpretability.

Gradient Boosted XGBoost model demonstrated the best result. However, it is not prudent to use the model for BCI value prediction. Regardless, the insights gained can undoubtedly inform businesses and policymakers about the factors influencing the US BCI sentiment which can provide a guideline for decision-making processes by emphasising on the features with significant impact.

There is room for improvement in the analysis. Future work should consider the following:

- Treating the problem as a classification task rather than regression due to the threshold nature of BCI, allowing for a more practical interpretation of optimistic and pessimistic sentiments.
- Analysis by transforming features to their relative incremental change instead of actual values.
- Investigate advanced techniques for handling multicollinearity in linear regression or explore alternative regression models.
- Conduct a deeper exploration into the interpretability challenges of neural networks or consider alternative architectures (CNN, RNN etc.) that may yield better results.
- Time series analysis with and without time lags.

8. References

Berg, A., & Pattillo, C. (2000). The Challenges of Predicting Economic Crises. International Monetary Fund. July 2000. https://www.imf.org/external/pubs/ft/issues/issues22/

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32.

Diebold, F. X., & Rudebusch, G. D. (1994). Forecasting output with the composite leading index: A real-time analysis. Journal of the American Statistical Association.

Hendry, D. F. (2011). Mathematical Models and Economic Forecasting: Some Uses and Mis-Uses of Mathematics in Economics. Economics Series Working Papers, 549, University of Oxford, Department of Economics.

Huang, T. M., Kecman, V., & Kopriva, I. (2006). Kernel Based Algorithms for Mining Huge Data Sets.

Investopedia. (2023). Recession. Investopedia. Retrieved October 12, 2023, from https://www.investopedia.com/terms/r/recession.asp

Jolliffe, I. (2002). Principal Component Analysis, Second Edition.

Khan, A., & Upadhayaya, K. (2019). Does business confidence matter for investment? Evidence from US business confidence survey data. Journal of Economics and Business.

Kelley, D. (2019). Which Leading Indicators Have Done Better at Signaling Past Recessions? Chicago Fed Letter, No. 425. Retrieved June 27, 2023, from https://www.chicagofed.org/publications/chicago-fed-letter/2019/425#:~:text=Far%20in%20advance%20of%20a,yields)%20is%20the%20best%20predictor%20is%20the%20best%20predictor).

Khan, A., & Upadhayaya, K. (2019). Does business confidence matter for investment? Evidence from US business confidence survey data. Journal of Economics and Business.

Liaw, A., & Wiener, M. (2002). Classification and Regression by randomForest.

Los, C., & Ocheretin, L. (2019). Predicting Business Confidence Index in five European countries using regression models. Economic Computation and Economic Cybernetics Studies and Research.

Los, V., & Ocheretin, D. (2019). Construction of Business Confidence Index Based on a System of Economic Indicators. SHS Web of Conferences, 65(M3E2 2019), 06003. DOI: https://doi.org/10.1051/shsconf/20196506003.

Mishkin, F. S. (1992). Is the Fisher effect for real? Journal of Monetary Economics.

Müller, B., Reinhardt, J., & Strickland, M. T. (1995). Neural Networks: An Introduction.

Neumann, J. von, & Morgenstern, O. (1944). Theory of Games and Economic Behavior. Princeton University Press.

Ng, T. P., Chan, W. Y., & Chan, E. S. (2007). Forecasting commercial properties - A comparison of the predictive accuracy of ARIMA and neural network models. International Real Estate Review.

Nvidia. (2024). Simplified schematic of a Random Forest ensemble.

Qiu, J. (2020). Forecasting the Consumer Confidence Index with tree-based MIDAS regressions. Journal of Forecasting.

Schapire, R. E. (1999). A Brief Introduction to Boosting.

Schumacher, C., & Breitung, J. (2008). Short-term forecasting of German GDP using large monthly datasets. International Journal of Forecasting, 24.

Sheather, S. (2009). A Modern Approach to Regression.

Sheather, S. (2009). Forecasting Inflation.

Smola, A. J., & Scholkopf, B. (2003). A tutorial on support vector regression. RSISE, Australian National University, Canberra, 0200, Australia.

Stock, J. H., & Watson, M. W. (2008). Phillips curve inflation forecasts. Inflation: Causes and Effects.

Tinbergen, J., & Bos, H. (1962). Mathematical Models of Economic Growth. McGraw-Hill.

US Recession and Financial Indicators Dataset. Kaggle. Retrieved October 12, 2023, from https://www.kaggle.com/datasets/mikoajfish99/us-recession-and-financial-indicators/.

Vrontos, S. D., Galakis, J., & Vrontos, I. D. (2020). Modeling and Predicting U.S. Recessions Using Machine Learning Techniques. International Journal of Forecasting, DOI: https://doi.org/10.1016/j.ijforecast.2020.08.005.

Wikipedia. (2013). Simplified diagram of a Feed Forward Neural Network.

9. Appendices

Code for the analysis

```
import os

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

%matplotlib inline

#pip install plotly


# Directory for the datasets


directory = r'C:\Users\Tan\Desktop\Tanvir - 05.05.22\UH -
DSwP\Semester 5 - Final Project\Datasets\Filtered Datasets'

# The 'r' before the path string is to to avoid escape
character interpretation

# without the 'r' have to replace backslash with forward slash

# 'C:/Users/Tan/Desktop/Tanvir - 05.05.22/UH - DSwP/Semester 5
- Final Project/Datasets/Datasets'


# Initializing an empty list to store names of my dataset files
which are in .csv format

csv_files = []


# Listing all files in the directory

for filename in os.listdir(directory):

    if filename.endswith(".csv"):

        csv_files.append(filename)


# Printing the list of CSV files
```

```
for csv_file in csv_files:
    print(csv_file)


# Creating an empty list for a combined dataframe


dataframes = []



for filename in os.listdir(directory):
    if filename.endswith(".csv") and filename not in ['BCI -
US.csv', 'SPX500.csv']:
        file_path = os.path.join(directory, filename)
        df = pd.read_csv(file_path, parse_dates=['DATE'],
dayfirst=True)
        dataframes.append(df)


spx_file_path = os.path.join(directory, 'SPX500.csv')
spx_df = pd.read_csv(spx_file_path, parse_dates=['DATE'])


spx_df


# Convert 'S_&_P' to numeric values
spx_df['S&P_change %'] =
pd.to_numeric(spx_df['S&P_change %'].str.rstrip('%'),
errors='coerce')
spx_df


# Creating a combined DataFrame


combined_df = pd.concat(dataframes, ignore_index=True)


combined_df = pd.concat([combined_df, spx_df],
ignore_index=True)
```

```
combined_df.columns = ['date', 'bank_credits', 
'real_estate_prices', 'consumer_loans',
                       'unemployment_claims', 
'credit_delinquency', 'federal_interest_rate',
                       'consumer_price_idx', 
'personal_saving_rate', 'commerical_real_estate_loan',
                       'GDP', 'st_consumer_price_idx', 
'unemployment_rate',
                       'exports', 'imports', 'inflation', 
'spx500']


combined_df.head()


# Reading the BCI data file


file_path_bci = os.path.join(directory, 'BCI - US.csv')
df_bci = pd.read_csv(file_path_bci, parse_dates=['DATE'], 
dayfirst=True)


df_bci.columns = ['date', 'BCI']


combined_df = pd.concat([combined_df, df_bci], 
ignore_index=True)


df = combined_df


df.info()



# Iterating through columns starting from the second column 
(index 1)
columns_to_plot = df.columns[1:]
num_rows = 6
num_cols = 3
```

```
# Create a 5x5 grid of subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 20))


# Flatten the axes array to simplify indexing
axes = axes.flatten()


# Iterate through columns and plot in subplots
for i, col in enumerate(columns_to_plot):
    # Create a new DataFrame with only the 'Date' and the
current column and set 'Date' as the index
    df_col = df[['date', col]].dropna().set_index('date')


    # Convert the entire column to float (if it contains
numeric data in text format, e.g.: houseowner_equity_RE column)
    df_col[col] = pd.to_numeric(df_col[col], errors='coerce')


    # Plot histograms for the data on the corresponding subplot
    df_col.plot(ax=axes[i], linestyle='dotted')
    axes[i].set_title(col, fontsize=18)




# Adjust layout
plt.tight_layout()


# Show the plots
plt.show()
```

Data pre-processing

Downsampling the feature recorded at frequency = daily

```
df[['date', 'federal_interest_rate']].dropna()
```

```
# Create a copy of the DataFrame

df_copy = df.copy()


# Set 'date' as the index in the copy

df_copy.set_index('date', inplace=True)


# Resample to monthly and take the first value of each month

#df_monthly_fed_ir =
df_copy['federal_interest_rate'].resample('MS').first().reset_i
ndex()


# Resample to monthly and take the mean value of each month

df_monthly_fed_ir =
df_copy['federal_interest_rate'].dropna().resample('MS').mean()
.reset_index()



# Now, df_monthly_fed_ir contains the first day of each month
for federal_interest_rate without changing the original df


df_monthly_fed_ir


axes[0].legend(fontsize=18)


# Create a copy of the DataFrame

df_copy = df.copy()


# Converting 'date' to datetime

df_copy['date'] = pd.to_datetime(df_copy['date'], format='%Y-
%m-%d')


# Set 'date' as the index in the copy

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

```
# Specify the column for mean change
change_column = 'spx500'

# Resample to monthly and calculate the mean change
df_monthly_mean_change =
df_copy[change_column].dropna().resample('M').mean()

# Shift the resampled values to the last day of the previous
month
df_monthly_mean_change.index = df_monthly_mean_change.index -
pd.offsets.MonthBegin(1)

# Create a new DataFrame with the resampled and shifted values
df_spx = pd.DataFrame({'date': df_monthly_mean_change.index,
change_column: df_monthly_mean_change.values})

df_spx
```

<u>Downsampling the feature recorded at frequency = weekly</u>

```
weekly_columns = ['bank_credits', 'consumer_loans',
'unemployment_claims', 'commerical_real_estate_loan']

df_copy[weekly_columns]

# Create a copy of the DataFrame
df_copy = df.copy()

# Converting 'date' to datetime
df_copy['date'] = pd.to_datetime(df_copy['date'], format='%Y-
%m-%d')
```

```
# Set 'date' as the index in the copy
df_copy.set_index('date', inplace=True)



# Create individual DataFrames for each column
for col in weekly_columns:
    # Resample to monthly and calculate the mean
    df_monthly_mean =
df_copy[col].dropna().resample('M').mean()

    # Shift the resampled values to the first day of the month
    df_monthly_mean = df_monthly_mean.shift(1, freq='D')

    # Create a new DataFrame with the resampled and shifted
values
    globals()[f'df_{col}'] = pd.DataFrame({'date':
df_monthly_mean.index, col: df_monthly_mean.values})

# Now, I have individual DataFrames df_bank_credits,
df_consumer_loans, etc., each with the mean shifted to the
first day of the month

# Plotting
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 8))

### selecting specific years to plot
d = df[['date', 'bank_credits']][(df['date'] > '2000-01-01') &
(df['date'] < '2010-01-01')].dropna()

# Scatter plot for daily data
axes[0].scatter(d['date'], d['bank_credits'], label='Weekly
Data', alpha=0.5)
axes[0].set_title(' Weekly Bank Credits', fontsize=18)
```

```
axes[0].set_xlabel('Date', fontsize=18)
axes[0].set_ylabel('Bank Credits', fontsize=18)

axes[0].tick_params(axis='x', labelsize=18)
axes[0].tick_params(axis='y', labelsize=18)



### selecting specific years to plot
dm = df_bank_credits[(df_bank_credits['date'] > '2000-01-01') &
                     (df_bank_credits['date'] < '2010-01-01')]


# Scatter plot for resampled monthly data
axes[1].scatter(dm['date'], dm['bank_credits'], label='Monthly
Data', color='green', marker='o')

axes[1].set_title('Monthly Bank Credits', fontsize=18)
axes[1].set_xlabel('Date', fontsize=18)
axes[1].set_ylabel('Bank Credits', fontsize=18)


plt.xticks(fontsize=18)
plt.yticks(fontsize=18)

# Adding legend
axes[0].legend(fontsize=18)
axes[1].legend(fontsize=18)

# Adjust layout to prevent clipping of titles
plt.tight_layout()
plt.show()
```

Upsampling the feature recorded at frequency = quarterly

```
# Create a copy of the DataFrame
df_copy = df.copy()

# Converting 'date' to datetime
df_copy['date'] = pd.to_datetime(df_copy['date'], format='%Y-
%m-%d')

# Set 'date' as the index in the copy
df_copy.set_index('date', inplace=True)

# Specify the quarterly columns
quarterly_columns = ['real_estate_prices',
'credit_delinquency', 'GDP']

# Create individual DataFrames for each quarterly column
for col in quarterly_columns:

    # Convert the entire column to float (if it contains
numeric data in text format, e.g.: houseowner_equity_RE column)
    df_copy[col] = pd.to_numeric(df_copy[col], errors='coerce')

    # Resample to monthly and distribute the quarterly values
evenly across the three previous months
    df_monthly_distributed =
df_copy[col].dropna().resample('M').mean().transform(lambda x:
x / 3).bfill()
```

```
    # Shift the resampled and distributed values to the first
day of the month

    df_monthly_distributed = df_monthly_distributed.shift(-1,
freq='D')


    # Explicitly set the date to the first day of the month as
a string

    df_monthly_distributed.index =
df_monthly_distributed.index.to_period('M').astype(str) + '-01'


    # Convert the index to datetime format

    df_monthly_distributed.index =
pd.to_datetime(df_monthly_distributed.index, format='%Y-%m-%d')


    # Create a new DataFrame with the resampled and distributed
values

    globals()[f'df_{col}'] = pd.DataFrame({'date':
df_monthly_distributed.index, col:
df_monthly_distributed.values})


# Now, I have individual DataFrames df_real_estate_prices,
df_credit_delinquency, etc., each with the quarterly values
distributed to the previous three months


# Plotting

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 8))


### selecting specific years to plot

cd = df[['date', 'credit_delinquency']][(df['date'] > '2000-01-
01') & (df['date'] < '2010-01-01')].dropna()


# Scatter plot for daily data

axes[0].scatter(cd['date'], cd['credit_delinquency'],
label='Quarterly Data')

axes[0].set_title('Quarterly Credit Delinquency', fontsize=18)
```

```
axes[0].set_xlabel('Date', fontsize=18)

axes[0].set_ylabel('credit_delinquency', fontsize=18)


axes[0].tick_params(axis='x', labelsize=18)

axes[0].tick_params(axis='y', labelsize=18)




### selecting specific years to plot

cdm = df_credit_delinquency[['date',
'credit_delinquency']][(df_credit_delinquency['date'] > '2000-
01-01') &

(df_credit_delinquency['date'] < '2010-01-01')].dropna()



# Scatter plot for resampled monthly data

axes[1].scatter(cdm['date'], cdm['credit_delinquency'],
label='Monthly Data', color='green', marker='o')


axes[1].set_title('Monthly Credit Delinquency', fontsize=18)

axes[1].set_xlabel('Date', fontsize=18)

axes[1].set_ylabel('credit_delinquency', fontsize=18)



plt.xticks(fontsize=18)

plt.yticks(fontsize=18)


# Adding legend

axes[0].legend(fontsize=18)

axes[1].legend(fontsize=18)
```

```
# Adjust layout to prevent clipping of titles
plt.tight_layout()

plt.show()


monthly_columns =
df.columns.drop(weekly_columns).drop(quarterly_columns).drop(['
date', 'federal_interest_rate',

'spx500'])

monthly_columns

# Loop through each column and create a DataFrame

for col in monthly_columns:
    globals()[f'df_{col}'] = pd.DataFrame({'date': df['date'],
col: df[col]}).dropna()

# List of all the individual dataframes created

dfs = [df_monthly_fed_ir, df_bank_credits, df_consumer_loans,
       df_commerical_real_estate_loan, df_real_estate_prices,
df_credit_delinquency,
       df_GDP, df_inflation, df_exports, df_imports, df_spx,
df_consumer_price_idx,
       df_personal_saving_rate, df_st_consumer_price_idx,
df_unemployment_claims,
       df_unemployment_rate, df_BCI]

# Merge DataFrames one by one using reduce
from functools import reduce
```

```
# Define a function to merge two DataFrames
def merge_dataframes(left, right):
    return pd.merge(left, right, on='date', how='outer')

# Use reduce to apply the merging function to all DataFrames in
the list
df_final = reduce(merge_dataframes, dfs)

# Sort the DataFrame by the 'date' column
df_final.sort_values('date', inplace=True)

# Optionally, reset the index if needed
df_final.reset_index(drop=True, inplace=True)

df_final.info()

df_final.describe()

# Iterating through columns starting from the second column
(index 1)
columns_to_plot = df_final.columns[1:-1]
num_rows = 4
num_cols = 4

# Create a 4x4 grid of subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 20))

# Flatten the axes array to simplify indexing
axes = axes.flatten()

# Iterate through columns and plot in subplots
```

```
for i, col in enumerate(columns_to_plot):

    # Create a new DataFrame with only the 'date' and the
current column and set 'date' as the index

    df_col = df_final[['date', col]].dropna().set_index('date')


    # Convert the entire column to float (if it contains
numeric data in text format, e.g.: houseowner_equity_RE column)

    df_col[col] = pd.to_numeric(df_col[col], errors='coerce')


    # Plot the data on the corresponding subplot

    #df_col.plot(ax=axes[i], linestyle='dotted', fontsize=18)

    df_col.plot(ax=axes[i], fontsize=18)


    # Set xticks and labels to display 6 or 7 ticks

    x_ticks = df_col.index[::len(df_col.index)//5]

    x_tick_labels = [date.strftime('%Y') for date in x_ticks]
# Use '%Y' for year-only format

    axes[i].set_xticks(x_ticks)

    axes[i].set_xticklabels(x_tick_labels)


    axes[i].set_xlabel(col, fontsize=20)

    axes[i].legend(fontsize=18)


# Adjust layout

plt.tight_layout()


# Show the plots

plt.show()



# Iterating through columns starting from the second column
(index 1)
```

```
columns_to_plot = df_final.columns[1:-1]
num_rows = 4
num_cols = 4

# Create a 4x4 grid of subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 20))

# Adjust vertical space between rows
plt.subplots_adjust(hspace=0.5)

# Flatten the axes array to simplify indexing
axes = axes.flatten()

# Iterate through columns and plot in subplots
for i, col in enumerate(columns_to_plot):
    # Create a new DataFrame with only the 'date' and the
current column and set 'date' as the index
    df_col = df_final[['date', col]].dropna().set_index('date')

    # Convert the entire column to float (if it contains
numeric data in text format, e.g.: houseowner_equity_RE column)
    df_col[col] = pd.to_numeric(df_col[col], errors='coerce')

    # Plot the data on the corresponding subplot
    df_col[col].plot(kind='hist', bins=20, ax=axes[i],
color='#FE302F',
                     alpha=0.75, edgecolor='black',
fontsize=15)

    axes[i].set_xlabel(col, fontsize=18)

# Adjust layout
```

```
#plt.tight_layout()

# Show the plots
plt.show()

# Iterating through columns starting from the second column
(index 1)
columns_to_plot = df_final.columns[1:-1]
num_rows = 4
num_cols = 4

# Create a 4x4 grid of subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 20))

# Adjust vertical space between rows
plt.subplots_adjust(hspace=0.5)

# Flatten the axes array to simplify indexing
axes = axes.flatten()

# Define colors for the boxplots
colors = ['#FE302F', '#3BAF9D']

# Iterate through columns and plot in subplots
for i, col in enumerate(columns_to_plot):
    # Create a new DataFrame with only the 'date' and the
current column
    df_col = df_final[['date', col]].dropna()

    # Convert the entire column to float (if it contains
numeric data in text format, e.g.: houseowner_equity_RE column)
    df_col[col] = pd.to_numeric(df_col[col], errors='coerce')
    # Plot the data using a vertical boxplot with two colors
for second and third quartiles
    bp = df_col.boxplot(column=col, ax=axes[i], fontsize=15,
vert=True, patch_artist=True,
                        boxprops=dict(facecolor=colors[0]),
                        medianprops=dict(color='red'))  # Set
color of the median line

    # Set different color for the third quartile
    for patch in bp.artists:
        patch.set_facecolor(colors[1])

    axes[i].set_xlabel(col, fontsize=18)

# Adjust layout
plt.tight_layout()

# Show the plots
plt.show()

plt.figure(figsize=(10, 6))
df_final['BCI'].plot(kind='hist', alpha=0.75, color='g',
edgecolor='black', fontsize=18)

plt.title('BCI Histogram', fontsize=18)
plt.xlabel('BCI', fontsize=18)
plt.ylabel('Frequency', fontsize=18)

plt.show()

df_bci['date'] = pd.to_datetime(df_bci['date'])  # Make sure
the 'Date' column is in datetime format
```

```
# Get the y-coordinate for the middle of the plot

#middle_line_y =
df_bci.dropna().set_index('Date').mean().values[0]

middle_line_y = 100.0



# Plot the data

df_bci.dropna().set_index('date').plot(figsize=(16, 8))


# Draw a horizontal line at the middle

plt.axhline(y=middle_line_y, color='black', linestyle='--',
label='100% Line')


# Set y-axis limits

plt.ylim(94, 106)


# Highlight a rectangular area (2007 to 2011) with yellow color

rect_start = '2006-01-01'

rect_end = '2011-12-31'


# Convert date strings to datetime objects

rect_start_date = pd.to_datetime(rect_start)

rect_end_date = pd.to_datetime(rect_end)


# Use axvspan to highlight the region

plt.axvspan(rect_start_date, rect_end_date, facecolor='orange',
edgecolor = 'black',

            alpha=0.5, label='2007-08 Financial Crisis and
aftermath')


rect_2_start = '2019-06-01'
```

```
rect_2_end = '2021-06-01'


# Convert date strings to datetime objects

rect_2_start_date = pd.to_datetime(rect_2_start)

rect_2_end_date = pd.to_datetime(rect_2_end)


# Use axvspan to highlight the region

plt.axvspan(rect_2_start_date, rect_2_end_date,
facecolor='red', edgecolor = 'black',

            alpha=0.5, label='Covid-19 and aftermath')


# Show the legend

plt.title('US Business Confidence Index (BCI) over the years',
fontsize=18)

plt.xlabel('Year', fontsize=18)

plt.ylabel('BCI%', fontsize=18)

plt.xticks(fontsize=18)

plt.yticks(fontsize=18)

plt.legend(loc='upper left', fontsize=18)


# Show the plot

plt.show()
```

Standardizing the features

```
from sklearn.preprocessing import StandardScaler


features = df_final.iloc[:, 1:-1].dropna()


# Standardizing the features

scaler = StandardScaler()

features_standardised = scaler.fit_transform(features)
```

```
# Creating a DataFrame with the normalized values, original
column names, and index
standardised_features = pd.DataFrame(features_standardised,
columns=features.columns, index=features.index)

df_standardised_features = standardised_features

# Adding the 'BCI' column to the normalised_features DataFrame
df_standardised_features['BCI'] = df_final.dropna()['BCI']

# Iterating through columns starting from the second column
(index 1)
columns_to_plot = df_standardised_features.columns[:-1]
num_rows = 4
num_cols = 4

# Create a 5x3 grid of subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 20))

# Adjust vertical space between rows
plt.subplots_adjust(hspace=0.5)

# Flatten the axes array to simplify indexing
axes = axes.flatten()

# Iterate through columns and plot in subplots
for i, col in enumerate(columns_to_plot):
    # Plot the data on the corresponding subplot
    axes[i].scatter(x = df_standardised_features[col].dropna(),
                    y =
df_standardised_features['BCI'].dropna(),
                    color = '#009CDF')
    axes[i].set_xlabel(col, fontsize=18)

    axes[i].set_ylabel('BCI', fontsize=18)

# Adjust layout
plt.tight_layout()

# Show the plots
plt.show()
plt.scatter
```

Model development

Linear Regression
Using StatsModel

```
import statsmodels.api as sm

from sklearn.model_selection import train_test_split

# X and y are the features and target variable respectively
X = df_standardised_features.drop('BCI', axis=1)
y = df_standardised_features['BCI']

# Add a constant term to the features
X = sm.add_constant(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, 
test_size=0.33, random_state=10)

# Fit the model
```

```
#lr_sm_model = sm.OLS(y_train, X_train).fit()

lr_sm_model = sm.OLS(y, X).fit()


# Print the summary

print(lr_sm_model.summary())


lr_sm_model.summary()


coeff = pd.DataFrame({'Coefficient': lr_sm_model.params},
X.columns, columns=['Coefficient'])


#coeff

coeff.sort_values(by='Coefficient')



# Plotting datapoints having more influence on the regression


fig, ax = plt.subplots(figsize=(16, 12))

influence_plot = sm.graphics.influence_plot(lr_sm_model,
criterion="cooks", ax=ax)

fig.tight_layout(pad=1.0)

plt.show()


df_standardised_features.loc[1270:1290]


df_final.loc[1270:1290]


plt.figure(figsize=(12, 6))

sns.scatterplot(x=lr_sm_model.fittedvalues,
y=lr_sm_model.resid, color='red', edgecolor='black')

plt.axhline(y=0, color='black', linestyle='--', linewidth=1)  #
Adding a horizontal line at y=0
```

```
plt.title('Residuals Plot', fontsize=18)
plt.xlabel('Fitted values', fontsize=18)
plt.ylabel('Residuals', fontsize=18)
plt.title("Residuals vs Fitted values")
plt.show()



plt.figure(figsize=(8,6))
plt.hist(lr_sm_model.resid, color='red', alpha=
0.7 ,edgecolor='black')
plt.title("Residuals")

lr_sm_model.mse_resid

lr_sm_model.rsquared

# Iterating through columns starting from the second column
(index 1)
columns_to_plot = X_train.columns[1:]     # removing the first
column name as it is a constant column added for OLS Regression
num_rows = 4
num_cols = 4

# Create a 4x4 grid of subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 20))

# Adjust vertical space between rows
plt.subplots_adjust(hspace=0.6)

# Flatten the axes array to simplify indexing
axes = axes.flatten()
```

```
# Iterate through columns and plot in subplots
for i, col in enumerate(columns_to_plot):
    # Plot the data on the corresponding subplot
    sns.scatterplot(x=X_train[col], y=lr_sm_model.resid, 
ax=axes[i], color='red', edgecolor='black')
    axes[i].axhline(y=0, color='black', linestyle='--', 
linewidth=1.5)
    axes[i].set_xlabel(col, fontsize=18)
    axes[i].set_ylabel('Residuals', fontsize=18)
    axes[i].set_title(f'Residuals vs {col}', fontsize=18)

# Adjust layout
plt.tight_layout()

# Show the plots
plt.show()
```

<u>Runnning model prediction</u>

```
y_pred = lr_sm_model.predict(X_test)

# Calculating residuals
residuals = y_test - y_pred

# Plotting residuals
plt.figure(figsize=(12, 6))
sns.scatterplot(x=y_pred, y=residuals, color='red', 
edgecolor='black')
plt.axhline(y=0, color='black', linestyle='--', linewidth=1)  # 
Adding a horizontal line at y=0
plt.title('Residuals Plot', fontsize=18)
plt.xlabel('Predicted BCI Values', fontsize=18)
plt.ylabel('Residuals', fontsize=18)
plt.show()


# Plotting residuals

plt.figure(figsize=(12, 6))


sns.scatterplot(x=y_pred, y=residuals, color='red',
edgecolor='black', label= 'Predicted BCI Value')


sns.scatterplot(x=y_test, y=residuals, color='green',
edgecolor='black', label= 'Test BCI Value')


plt.axhline(y=0, color='black', linestyle='--', linewidth=1)  #
Adding a horizontal line at y=0

plt.title('Residuals Plot')

plt.xlabel('BCI Values')

plt.ylabel('Residuals')

plt.legend()

plt.show()



plt.figure(figsize=(8,6))

plt.hist(residuals, alpha=0.7, color='red', edgecolor='black')

#sns.histplot(residuals, kde=True, color='red',
edgecolor='black', alpha=0.7)

plt.show()


# Iterating through columns starting from the second column
(index 1)

columns_to_plot = X_test.columns[1:]     # removing the first
column name as it is a constant column added for OLS Regression

num_rows = 4

num_cols = 4
```

```
# Create a 4x4 grid of subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 20))

# Adjust vertical space between rows
plt.subplots_adjust(hspace=0.5)

# Flatten the axes array to simplify indexing
axes = axes.flatten()

# Iterate through columns and plot in subplots
for i, col in enumerate(columns_to_plot):
    # Plot the data on the corresponding subplot
    sns.scatterplot(x=X_test[col], y=residuals, ax=axes[i],
color='red', edgecolor='black')
    axes[i].axhline(y=0, color='black', linestyle='--',
linewidth=1.5)
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Residuals')
    axes[i].set_title(f'Residuals vs {col}')

# Adjust layout
plt.tight_layout()

# Show the plots
plt.show()

exclude_rows_idx = np.arange(1270, 1291)

exclude_rows_idx = [1154] + list(exclude_rows_idx)

# X and y are the features and target variable respectively
```

```
X = df_standardised_features.drop(index=exclude_rows_idx,
axis=0).drop('BCI', axis=1)

y = df_standardised_features.drop(index=exclude_rows_idx,
axis=0)['BCI']


# Add a constant term to the features

X = sm.add_constant(X)


# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=10)


# Fit the model

#lr_sm_model = sm.OLS(y_train, X_train).fit()

lr_sm_model = sm.OLS(y, X).fit()


# Print the summary

print(lr_sm_model.summary())


fig, ax = plt.subplots(figsize=(12, 8))

influence_plot = sm.graphics.influence_plot(lr_sm_model,
criterion="cooks", ax=ax)

fig.tight_layout(pad=1.0)

plt.show()




# X and y are the features and target variable respectively

X = df_standardised_features.drop(index=exclude_rows_idx,
axis=0).drop(['federal_interest_rate',


'commerical_real_estate_loan', 'real_estate_prices',
'exports','imports', 'consumer_price_idx',

'spx500', 'unemployment_rate', 'BCI'], axis=1)
y = df_standardised_features.drop(index=exclude_rows_idx,
axis=0)['BCI']


# Add a constant term to the features
X = sm.add_constant(X)


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=10)


# Fit the model
#lr_sm_model = sm.OLS(y_train, X_train).fit()
lr_sm_model = sm.OLS(y, X).fit()


# Print the summary
print(lr_sm_model.summary())


print(lr_sm_model.mse_resid)


coeff = pd.DataFrame({'Coefficient': lr_sm_model.params},
X.columns, columns=['Coefficient'])


#coeff
coeff.sort_values(by='Coefficient')


plt.figure(figsize=(12, 6))
sns.scatterplot(x=lr_sm_model.fittedvalues,
y=lr_sm_model.resid, color='red', edgecolor='black')
```

```
plt.axhline(y=0, color='black', linestyle='--', linewidth=1)  #
Adding a horizontal line at y=0

plt.title('Residuals Plot', fontsize=18)

plt.xlabel('Fitted values', fontsize=18)

plt.ylabel('Residuals', fontsize=18)


plt.xticks(fontsize=18)

plt.yticks(fontsize=18)


plt.title("Residuals vs Fitted values", fontsize=18)

plt.show()


plt.figure(figsize=(8,6))

plt.hist(lr_sm_model.resid, color='red', alpha=
0.7 ,edgecolor='black')

plt.xticks(fontsize=18)

plt.yticks(fontsize=18)

plt.title("Residuals", fontsize=18)
```

PCA Analysis

```
from sklearn.decomposition import PCA


# X is the feature matrix

X = df_standardised_features.iloc[:, :-1]


# Apply PCA

pca = PCA()

X_pca = pca.fit_transform(X)


# Create a DataFrame with the principal components

columns = [f'PC{i+1}' for i in range(X_pca.shape[1])]
```

```
df_pca = pd.DataFrame(data=X_pca, columns=columns)


plt.figure(figsize=(10, 8))

plt.scatter(df_pca['PC1'], df_pca['PC2'], alpha=0.75,
edgecolor='black')

plt.title('PCA: PC1 vs PC2', fontsize=18)

plt.xlabel('Principal Component 1 (PC1)', fontsize=18)

plt.ylabel('Principal Component 2 (PC2)', fontsize=18)

plt.grid(True)

plt.show()


# Plotting interactive plot with plotly


fig = px.scatter(df_pca, x='PC1', y='PC2', title='PCA: PC1 vs
PC2',

                 labels={'PC1': 'Principal Component 1 (PC1)',
'PC2': 'Principal Component 2 (PC2)'},

                 hover_data={'index': df_pca.index},
opacity=0.75)


# Add grid lines

fig.update_xaxes(showgrid=True, gridwidth=1,
gridcolor='Lightblue')

fig.update_yaxes(showgrid=True, gridwidth=1,
gridcolor='LightGray')


# Show the interactive plot

fig.show()


from sklearn.cluster import KMeans


X = df_pca[['PC1', 'PC2']]
```

```
# Choose the number of clusters (you can adjust this)
num_clusters = 3

# Fit the KMeans model
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
df_pca['cluster'] = kmeans.fit_predict(X)

# Plot the clustered data using Plotly
fig = px.scatter(df_pca, x='PC1', y='PC2', color='cluster',
                 title='Cluster Analysis: PC1 vs PC2',
                 labels={'PC1': 'Principal Component 1 (PC1)',
'PC2': 'Principal Component 2 (PC2)'},
                 hover_data={'index': df_pca.index},
opacity=0.75)

# Add grid lines
fig.update_xaxes(showgrid=True, gridwidth=1,
gridcolor='LightBlue')
fig.update_yaxes(showgrid=True, gridwidth=1,
gridcolor='LightGray')

# Show the interactive plot
fig.show()



# Explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_

# Print the explained variance ratio for each principal
component
for i, ratio in enumerate(explained_variance_ratio, 1):
    print(f'Explained Variance Ratio PC{i}: {ratio:.4f}')
```

```
# Visualize the explained variance ratio
plt.figure(figsize=(10,6))
plt.plot(range(1, len(explained_variance_ratio) + 1),
explained_variance_ratio.cumsum(),
         color = '#B20317', marker='o', linestyle='--')
plt.xlabel('Number of Principal Components', fontsize=18)
plt.xticks(fontsize=18)
plt.ylabel('Cumulative Explained Variance Ratio', fontsize=18)
plt.yticks(fontsize=18)
plt.title('Explained Variance Ratio by Principal Components',
fontsize=18)
plt.show()
```

This shows that 6 feature variables can explain 91% of the variations in the data and 8 can 96%. Therefore 8 features can explain BCI without losing much information from keeping 16 features.

```
# X_pca is the result of PCA transformation
# Columns is the list of original feature names
feat_columns = df_standardised_features.columns[:-1]

loadings = pca.components_

# Create a DataFrame with loadings
loadings_df = pd.DataFrame(loadings, columns=feat_columns,
index=columns)

# Display the loadings for the first few principal components
num_components_to_display = 8
for i in range(num_components_to_display):
    print(f"\nLoadings for PC{i+1}:")
    print(loadings_df.iloc[i].sort_values(ascending=False))
```

```
loadings = pca.components_.T  # Transpose to align with
original features


# Create a DataFrame with loadings

loadings_df = pd.DataFrame(loadings, columns=columns,
index=df_standardised_features.columns[:-1])


# Identify the most important features for PC1 and PC2

top_features_pc1 =
loadings_df['PC1'].abs().sort_values(ascending=False).index

top_features_pc2 =
loadings_df['PC2'].abs().sort_values(ascending=False).index


# Print or use the top features

print("Top features for PC1:", top_features_pc1)

print("Top features for PC2:", top_features_pc2)


# Generating a correlation circle


pcs = pca.components_.T


# Creating a scatter plot for the first two principal
components

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

plt.scatter(pcs[:, 0], pcs[:, 1], alpha=0.6)


# Adding labels to each point representing the feature names
with larger font size

for i, (x, y) in enumerate(zip(pcs[:, 0], pcs[:, 1])):

    # Define an offset to adjust label position

    offset = 0.01
    # Display the label with an offset to avoid overlap

    plt.text(x + offset, y + offset, feat_columns[i],
fontsize=14, ha='right', va='bottom')


# Adding arrows to show the magnitude

for i, feature in enumerate(feat_columns):

    plt.arrow(0, 0, pcs[i, 0], pcs[i, 1], color='black',
alpha=0.5, width=0.001)


# Adding unit circle

circle = plt.Circle((0, 0), 1, edgecolor='black',
facecolor='none', linewidth=1.5, linestyle='--')

plt.gca().add_patch(circle)


# Setting axis labels

plt.xlabel('Principal Component 1', fontsize=18)

plt.ylabel('Principal Component 2', fontsize=18)


# Setting plot title

plt.title('Correlation Circle', fontsize=18)


# Showing the plot

plt.show()
```

Support Vector Machines (SVM) Regression

SVR (Support Vector Regression)

```
from sklearn.model_selection import GridSearchCV


from sklearn.svm import SVR
```

```
# X and y are the features and target variable respectively

X = df_standardised_features.iloc[:, :-1]

y = df_standardised_features['BCI']


# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=10)



# Define a parameter grid

param_grid = {'C': [0.1, 1, 10, 50], 'epsilon': [0.02, 0.05,
0.1, 0.2]}


# Initialize SVR

svr_model = SVR()


# Perform grid search

grid_search = GridSearchCV(estimator=svr_model,
param_grid=param_grid, scoring='neg_mean_squared_error', cv=5)

grid_search.fit(X_train, y_train)


# Get the best parameters

best_params_svr = grid_search.best_params_


# Train the final SVR model with the best parameters

final_svr_model = SVR(C=best_params_svr['C'],
epsilon=best_params_svr['epsilon'])

final_svr_model.fit(X_train, y_train)


# Predict on the test set

y_pred_svr = final_svr_model.predict(X_test)
```

```
# Evaluate the SVR model
mse_svr = mean_squared_error(y_test, y_pred_svr)
print(f'Mean Squared Error (SVR): {mse_svr}')

r2_svr = r2_score(y_test, y_pred_svr)
print('R-squared (SVR):', r2_svr)

print('Best Parameters', best_params_svr)

from sklearn.inspection import permutation_importance

feature_names = X.columns

# Feature coefficients (importance)
perm_importance = permutation_importance(final_svr_model,
X_test, y_test, n_repeats=30, random_state=42)

# Retrieving the mean importance scores and feature names
mean_importance = perm_importance.importances_mean
feature_importance_dict = dict(zip(feature_names,
mean_importance))

feature_importance_df =
pd.DataFrame(list(feature_importance_dict.items()),
columns=['Feature', 'Importance'])

feature_importance_df =
feature_importance_df.sort_values(by='Importance',
ascending=False)

feature_importance_df
```

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor


X = df_final.iloc[:, 1:-1].dropna()

y = df_final.dropna()['BCI']


# Split data into predictors X and output Y

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=10)


# Create a RandomForestRegressor

rf_model = RandomForestRegressor(n_estimators=500,
random_state=50)#bootstrap=False,


# Fit the model to the training data

rf_model.fit(X_train, y_train)


# Make predictions on the test set

y_pred = rf_model.predict(X_test)


# Evaluate the model

mse = mean_squared_error(y_test, y_pred)

print(f'Mean Squared Error: {mse}')


r2 = r2_score(y_test, y_pred)

print('R-squared:', r2)


feature_importances = rf_model.feature_importances_


feature_names = X.columns
```

```
# Create a DataFrame with feature names and importances

feature_importance_df = pd.DataFrame({'Feature': feature_names,
'Importance': feature_importances})


# Sort the DataFrame by importance in descending order

feature_importance_df =
feature_importance_df.sort_values(by='Importance',
ascending=False)


# Display the DataFrame

print("Feature Importances:")

feature_importance_df


# Sorting by descending order to have higher importance at the
top

feature_importance_df =
feature_importance_df.sort_values(by='Importance',
ascending=True)



# Plotting feature importances

plt.figure(figsize=(10, 7))

plt.barh(feature_importance_df['Feature'],
feature_importance_df['Importance'], color='#5985A6',
edgecolor='black')

plt.xlabel('Importance', fontsize=16)

plt.xticks(fontsize=16)

plt.ylabel('Feature', fontsize=16)

plt.yticks(fontsize=16)

plt.title('Feature Importance from Random Forest Model',
fontsize=18)

plt.grid(axis='x', linestyle='--', alpha=0.6)

plt.show()
```

```
# Plot actual vs. predicted values

plt.figure(figsize=(10, 6))

sns.regplot(x=y_test, y=y_pred, ci=95, scatter_kws =
{'edgecolor':'black'}, line_kws={'color': 'green'})

plt.xlabel('Actual Values', fontsize=18)

plt.xticks(fontsize=18)

plt.ylabel('Predicted Values', fontsize=18)

plt.yticks(fontsize=18)

plt.title('Actual vs. Predicted Values for
RandomForestRegressor', fontsize=18)

plt.show()
```

<u>XGBoost (optimized distributed gradient boosting)</u>

```
#pip install xgboost


from xgboost import XGBRegressor


# X and y are the features and target variable respectively

X = df_standardised_features.drop('BCI', axis=1)

y = df_standardised_features['BCI']


# Splitting data into predictors X and output Y

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=10)

# Define parameter grid for hyperparameter tuning (adjust
parameters and values)

param_grid = {

    'n_estimators': [100, 200, 500, 1000],

    'learning_rate': [0.01, 0.1, 0.3],

    'max_depth': [3, 5, 8, 10]

}
```

```
# Create and fit XGBoost model with GridSearchCV for
hyperparameter tuning

xgb_model = GridSearchCV(XGBRegressor(), param_grid=param_grid,
cv=5, scoring='neg_mean_squared_error')

xgb_model.fit(X_train, y_train)


# Print best parameters and score

print("Best parameters:", xgb_model.best_params_)

print("Best score(negative mean sq error):",
xgb_model.best_score_)


# Get trained model

best_model = xgb_model.best_estimator_


# Make predictions on test set

y_pred = best_model.predict(X_test)


# Evaluate model performance with chosen metric (e.g., MSE)

mse = mean_squared_error(y_test, y_pred)

print("Mean Squared Error:", mse)


# R Squared

r2 = r2_score(y_test, y_pred)

print('R-squared:', r2)


column_names = X.columns


# DataFrame to store feature importances with corresponding
column names

feature_importance_df = pd.DataFrame({'Feature': column_names,
'Importance': best_model.feature_importances_})
```

```
# Sorting the DataFrame by importance in descending order
feature_importance_df =
feature_importance_df.sort_values(by='Importance',
ascending=False)



feature_importance_df


# Sorting by descending order to have higher importance at the
top
feature_importance_df =
feature_importance_df.sort_values(by='Importance',
ascending=True)



# Plotting feature importances
plt.figure(figsize=(10, 7))
plt.barh(feature_importance_df['Feature'],
feature_importance_df['Importance'], color='#3BAF9D',
edgecolor='black')
plt.xlabel('Importance', fontsize=16)
plt.xticks(fontsize=16)
plt.ylabel('Feature', fontsize=16)
plt.yticks(fontsize=16)
plt.title('Feature Importance from XGBoost Model', fontsize=18)
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.show()
```

Neural Network

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

```
# X and y are the features and target variable respectively

X = df_standardised_features.iloc[:, :-1]

y = df_standardised_features['BCI']


#X = df_final_interval.drop(['date', 'BCI'], axis=1).dropna()

#y = df_final_interval.dropna()['BCI']


# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


# Neural network model with regularization and dropout

model = Sequential()

model.add(Dense(16, activation='relu',
input_shape=(X_train.shape[1],)))

model.add(Dropout(0.5))  # Dropout layer for regularization

model.add(Dense(8, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.1)))

model.add(Dense(1, activation='linear'))


# Compile the model

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=
1e-2), loss='mean_squared_error')


model.summary()


# Early stopping callback

early_stopping =
tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=10, restore_best_weights=True)


# Train the model
```

```
history = model.fit(X_train, y_train, epochs=100, batch_size=8,
                    validation_split=0.33, verbose=2,
shuffle=True,
                   callbacks=[early_stopping])

# Evaluate the model on the test set
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

print(f'Mean Squared Error: {mse}')



plt.figure(figsize=(12,6))

plt.plot(history.history['loss'], label= 'Training data')
plt.plot(history.history['val_loss'], label= 'Validation data')
plt.xlabel('Epoch', fontsize=18)
plt.xticks(fontsize=18)
plt.ylabel('Loss', fontsize=18)
plt.yticks(fontsize=18)

plt.legend(fontsize=18)

plt.tight_layout()
plt.show()
----------------------------------------------
```