University of Essex

School of Mathematics, Statistics and Actuarial Science

MA981 DISSERTATION

An Analysis of Enhancing Unemployment Rate Forecasting with Machine Learning

Max Sharman

Supervisor: **Professor Fytas**

September 14, 2024
Colchester

Chapter 1

Abstract

This study examines the effectiveness of hybrid forecasting models in predicting time series data, focusing specifically on UK unemployment rates. The main goal is to create and assess different hybrid models using accuracy metrics like Root-mean-square-error (RSME), by combining econometric techniques with machine learning methods. The study combines econometric models such as Exponential Smoothing (ES) and Vector Autoregression (VAR) with machine learning approaches including Random Forest (RF) and Long Short-Term Memory (LSTM). These models were trained and tested using historical UK unemployment data from 2007 to 2024. The findings showed that hybrid models incorporating RFR demonstrated higher accuracy, while those using LSTM showed less reliable performance. The results varied significantly depending on the combination of econometric and machine learning methods used. This study underscores the complexities involved in selecting suitable hybrid models for unemployment forecasting and emphasises the importance of choosing models based on specific data characteristics.

Keywords: Hybrid forecasting, machine learning, econometrics, unemployment prediction, time series analysis.

Contents

1 Abstract **2**

2 Introduction **5**

3 Literature Review **8**

3.1 Unemployment Overview 8

3.2 Overview of Economic Forecasting 9

3.3 Studies Forecasting Unemployment Rates 11

3.4 Comparative Studies 14

3.5 Studies Using Hybrid Models 16

3.6 Research Discussion 19

4 Dataset **20**

4.1 Source of Dataset 20

4.2 Dimension of Datasets 20

5 Methodology **22**

5.1 Dataset Processing 22

5.2 Data Exploration 23

5.2.1 Time Series Line Graphs 23

5.2.2 Boxplots 26

5.2.3 Correlation Matrix 27
5.2.4 Seasonal Decomposition 28
5.2.5 Autocorrelation Plot 29
5.2.6 Statistical Analysis on Unemployment Data 30
5.3 Methodological approach 31
5.3.1 Development and Training of Models 31
5.3.2 Traditional Econometric Methods Used 33
5.3.3 Machine Learning Models Used 39

6 Results & Discussion 48

6.1 Results 48
6.1.1 ARIMA - LSTM Hybrid Model 48
6.1.2 ARIMA - GRU Hybrid Model 50
6.1.3 ES - SVR Hybrid Model 53
6.1.4 ES - RFR Hybrid Model 55
6.1.5 VAR - LSTM Hybrid Model 57
6.1.6 VAR - GRU Hybrid Model 60
6.1.7 GARCH - RFR Hybrid Model 62
6.1.8 GARCH - SVR Hybrid Model 65
6.1.9 ARIMAX-LSTM Hybrid Model 67
6.2 Discussion and Recommendations 70

7 Conclusions 73

8 Appendix 75

CHAPTER 2

Introduction

Time series forecasting plays a crucial role in economic analysis [1], especially for predicting key indicators such as unemployment rates [2]. Traditional econometric models like exponential smoothing, developed in the 1950s [3], and vector autoregression, developed in the 1980s [4], have been used for a long time for this purpose. However, the integration of machine learning techniques has led to the development of hybrid models, which offer improved accuracy and adaptability, as demonstrated in previous studies discussed in the literature review.

While hybrid forecasting models are increasingly used in economic studies, there is limited research on their application to UK unemployment data. Current studies compare these models across different countries but do not focus on forecasting challenges within the UK. This study aims to address this gap by applying and evaluating hybrid models specifically to UK unemployment data published by the Statista Research Department [5]. This targeted analysis will consider the economic environment of the UK. In addition to unemployment data, other datasets, such as GDP and inflation, will be used for multivariate methods like ARIMAX and VAR, which require additional features.

The main goals of this research are to compare different hybrid forecasting models that combine machine learning techniques with traditional econometric methods and to determine the most accurate and reliable models for forecasting UK unemployment rates. This study will involve developing, testing, and evaluating these hybrid models using Python packages such as sci-kit, Keras, and statsmodels. The models will be applied to economic datasets, and their accuracy and reliability will be assessed using performance metrics.

Before delving into more complex topics, such as traditional econometric methods and machine learning algorithms for forecasting, it is important first to understand the concept of time series data and time series analysis.

A time series is a sequence of data points occurring one after the other over a period of time. Time series analysis can analyse these sequences of data points collected and recorded over a set period-for example, the number of mobile phones purchased each month over a year. For time series analysis to be reliable, it is important to have sufficient data points, account for seasonal variance, and ensure no outliers are present in any discovered trends or patterns [6].

Time series analysis has existed since the 1920s and 1930s [7], with G. U. Yule and J. Walker making the first applications to data with autoregressive models [8]. In the current data era, businesses are collecting time series data exponentially [7]. This kind of data provides valuable insights to businesses, helping them make well-informed decisions and strategic plans based on historical trends [7]. Furthermore, the introduction of the big data movement is transforming industries, allowing them to leverage all data from diverse sources and optimise business performance [9].

When working with time series data, it's crucial to consider critical characteristics like trend, seasonality, and outliers. Trend analysis helps determine whether the data sequences increase or decrease over time. Seasonality analysis checks for regular repeating patterns in data, such as those related to seasons, quarters, months, or days of the week. Outlier analysis checks for any data points significantly distant from the rest of the data [10].

Non-stationary data, which constantly fluctuates over time or is affected by time, such as financial stock data, is commonly used for time series analysis and forecasting. Time series analysis is applied in various industries, including finance, retail, and economics [6]. However, this project specifically focuses on time series analysis for economic forecasting.

Considering the significance of economic forecasting, the expected outcome of this research is to provide valuable insights into the most effective hybrid forecasting models for UK economic data. By highlighting the strengths and weaknesses of these combined approaches, the study aims to offer guidance on optimising model performance to improve the accuracy of economic forecasts.

The significance of this research lies in its potential to help economists, data scientists, and policymakers make informed decisions based on unemployment predictions. It aims to understand how machine learning and traditional econometric models perform in forecasting economic time series data. This research will contribute to the field by providing practical recommendations for economists on the most effective methods and approaches for economic data analysis and forecasting.

The dissertation is organised as follows:

Chapter 3 presents a thorough literature review, delving into the significance of unemployment rates, economic forecasting, and various studies comparing individual methods and hybrid models. This chapter highlights the gaps in the literature that this study aims to address.

In Chapter 4, the discussion will focus on the datasets used in this study. This will include information on where the datasets can be found and how to access them. I will also explore the dimensions of each dataset and how they have been merged to create a unified dataset suitable for use with each model. Additionally, a brief overview of the key unemployment dataset will be provided to give extra background information.

In Chapter 5, the methodology of the study is covered, including data processing methods and an exploratory data analysis. This is to gain a better understanding of the data before delving into more complex areas. Additionally, this section outlines the methods used and how they have been structured and applied using Python.

Chapter 6 presents the study results, including a detailed analysis of the performance of different hybrid models applied to UK unemployment data. Additionally, accuracy visualisations have been included to provide more context for the performance metrics.

Chapter 7 concludes the dissertation, summarises the key findings and includes areas for future research.

CHAPTER 3

Literature Review

3.1 Unemployment Overview

The unemployment rate is one of the most used indicators when understanding the conditions of the labour market. Economists use the unemployment rate to help them describe the supply and demand for labour by businesses and organisations.

Additionally, unemployment rates also provide economists with insights into how the economy is performing, which is why it is important when discussing monetary policy.

Unemployment happens when a person is ready and able to work but has yet to have a paid job. The unemployment rate represents the percentage of unemployed people in the labour force. Determining if someone is unemployed involves making practical judgments, such as deciding how much-paid work qualifies as having a job and counting the number of employed and unemployed individuals [11].

The Office for National Statistics (ONS) faces a challenge when determining the employment status of certain individuals. For example, should someone over the age of 65 who is claiming a pension be considered unemployed? Similarly, should someone who is long-term sick and unable to work be classified as unemployed? To address this issue, the ONS adopts the internationally agreed definition from the International

Labour Organization (ILO). According to this definition, individuals aged 16 and above fall into one of three categories: employed, unemployed, or economically inactive. Those who engage in at least one hour of paid work or are temporarily away from a job are classified as employed. Individuals who are available for work and actively seeking employment are considered unemployed. The economically inactive category includes individuals who are full-time students, ill or disabled, retired, or caring for family members, among other reasons [12].

Unemployment is a critical issue in modern economics as it affects economic growth, employment, and social stability. Predicting unemployment is crucial for economists, forecasters, and policymakers, particularly during economic downturns. Attaining high economic growth with low, stable unemployment is a primary macroeconomic policy objective globally because low unemployment helps sustain economic and social systems.

Indicators like unemployment rates, employment-to-population ratios, labour force participation rates, and the employment intensity of growth give us insights into an economy's ability to create enough jobs. The decrease in the job creation aspect of economic growth is a cause for concern. By integrating employment and decent work into economic growth, we can maximise the benefits and ensure that growth is sustainable and inclusive, ultimately helping to reduce poverty[2].

3.2 Overview of Economic Forecasting

Forecasting is a data science task that is central to many activities within an organisation. Producing high quality forecasts is not an easy problem for either machines or for most analysts [13].

In today's environment, we are constantly flooded with forecasts and predictions about the future. These range from long-term weather forecasts and economic projections by central banks to health organisations predicting the spread of diseases and scientists forecasting climate change impacts. The media is full of pundits predicting either doom or utopia for companies, political parties, or countries based on current decisions. Many of these predictions turn out to be wrong, often due to the conflation of forecast and prediction and the inherent complexity of the world [14].

Time-series forecasts are crucial in various economic activities, such as setting fiscal and monetary policies, budgeting at the state and local levels, financial management, and financial engineering [15].

Unemployment is an important factor in forecasting successful employment in the future [2]. Therefore, it is necessary to discuss the methods used in economic forecasting, types of forecasts, the challenges faced, and the importance of accurate forecasting.

Economic forecasting employs a variety of methods such as expert judgement, extrapolation, leading indicators, surveys, econometric systems [16]. However, during this study, time-series models will be the key focus. Time-series models are a popular type of forecasting method that describes the historical patterns of data. They have been found to be competitive relative to econometric systems of equations, particularly in their multivariate forms. These models are considered the workhorses of the forecasting industry and focus on measurable uncertainty [16].

Moreover, economic forecasting is a challenging process that requires a balanced use of various models, ad hoc indicators, and a significant amount of high-quality data to achieve accurate results [17]. The primary challenge with economic forecasting is the uncertainty of the future, which stems from the probabilities involved and unknown factors that may exist [18].

Certain critical factors must be considered when dealing with economic forecasting, such as selecting the most appropriate forecasting models to address the problem, assessing the uncertainty associated with the forecast, and ensuring that the model remains stable over time [15].

In economic forecasting, several challenges need to be addressed. Economies evolve over time and can face unexpected shocks, such as technological advancements resulting from scientific discoveries. Significant events like changes in legislation or the elimination of exchange controls can create structural breaks in the economy, disrupting the stable relationships between economic variables and leading for forecast errors. Additionally, the models used for economic forecasting are imperfect representations of reality and may struggle to anticipate changes, making it challenging to evaluate the impact of new developments. Finally, economic models involve complex interactions between deterministic terms, observed stochastic variables, and unobserved errors. Errors in formulating the models, inaccurate estimation, or unexpected changes in these relationships can lead to poor forecast performance.

Accurate forecasting is vital for informed decision-making in economic activities. Forecasts can vary widely in their foundation and accuracy. The uncertainty of the future, stemming from probabilities and unknown factors, poses a primary challenge. Even well-tested forecasting methods may not always inspire confidence due to the inherent uncertainty.

A forecast can be judged as successful if it is close to the actual outcome, but the definition of "close" can vary. Despite the challenges, achieving accuracy in forecasts is critical for economic stability and growth [16].

3.3 Studies Forecasting Unemployment Rates

Having discussed the importance of unemployment rates and the challenges of economic forecasting, it is now worth exploring past studies on forecasting the unemployment rate.

In a 2005 paper, Christos Floros suggests that several research papers employed time series models to forecast macroeconomic variables. Furthermore, various techniques, from the simple OLS method to the GARCH models, have also been used to explain the forecasting performance of US and UK unemployment rates.

Christos Floros conducts testing and reporting on the competition between various models and forecasting periods using UK unemployment data. In their study, they compare the performance of twenty-three models for UK unemployment using data from January 1971 to December 2002. They employ RMSE, MAE, and MAPE criteria to compare the performance of the different models.

Christos Floros estimates various ARMA and (G)ARCH models for comparison purposes. They use their selected models to produce dynamic and static forecasts, utilising approximately an 80-20 split on parameter estimation to forecast evaluation.

It has been discovered that simple models are the most suitable for forecasting. The findings indicate that autoregressive (AR) and moving average (MA) models perform well in terms of forecasting, unlike other research papers. The researchers believe that these models excel because traditional, simple time-series models effectively capture the dynamic structure that generates unemployment levels. However, it's important to note that forecasting results may vary depending on the forecasting periods and the selection of in-sample and forecast data.

Finally, Christos Floros recommends further investigation of more complex forecasting methods to predict European and Asian unemployment rates [19].

Forecasters and policymakers often use Okun's law or basic time-series models to predict the unemployment rate. However, in a paper published in 2012, Regis Barnichon and Christopher J. Nekarda introduced a nonlinear model for forecasting unemployment. This model incorporates information on labour force flows based on economic theory and is not typically utilised in other forecasts.

The model performed better than basic time-series models and the Federal Reserve Board's Greenbook forecast in short time horizons. Its success can be attributed to two main factors: first, the unemployment rate converges to its expected stable level within 3 to 5 months, and second, the unique time-series characteristics of unemployment inflows and outflows.

Based on empirical evidence, the two-state model has a root-mean-squared forecast error that is 30% smaller than the next-best forecast for the current quarter and 10% smaller for the next quarter. This model demonstrates excellence in predicting business cycle turning points and large recessions. When combined with the SPF forecast, this model improves current-quarter forecast accuracy by 35% and next-quarter forecast accuracy by 25%.

The two proposed models each have their advantages and disadvantages. The two-state model is simpler and has less noisy historical data, but it relies on inferred hazard rates that become less reliable after 2009. On the other hand, the three-state model better represents the labour market and provides consistent forecasts for the unemployment rate, labour force participation rate, and employment-population ratio. Although it is less accurate for longer time horizons, the three-state model has outperformed the two-state model and SPF since 2007, mainly due to accounting for the significant decline in labour force participation during this period [20].

In a 2015 paper, Barinichon and Nekarda showcased the value of unemployment flows in predicting the unemployment rate. They introduced a new approach that utilises data on unemployment flows for predicting the unemployment rate.

They utilised a basic vector autoregression (VAR) model for unemployment flows to predict the real-time unemployment rate and leading indicators like initial claims for unemployment insurance and job vacancies. The forecasts based on this model outperformed the survey of professional forecasters (SPF), the Federal Reserve Board's Greenbook forecast, and basic univariate time-series models for near-term forecast horizons in their sample.

On the other hand, Brent Meyer and Murat Tasci have explored an alternative model for forecasting performance. They utilised unemployment flow data along with several univariate time series models. Their findings indicate that this approach performs similarly to Barinichon and Nekarda's forecasting models. Although the difference in forecasting performance is somewhat small overall, Tasci's trends-based approach from 2012 is well-structured and does not necessitate additional contemporary information on leading indicators [21].

In a paper published in 2021, Michal Gostkowski and Tomasz Rokicki aimed to compare the most significant predictive methods for modelling the unemployment rate. They compared the performance of methods such as the naïve method, regression model, ARIMA, Holt model, and Winters model using data collected by the Central Statistical Office.

The study aimed to develop a forecast of the unemployment rate in Poland and test its accuracy by comparing it to historical data. The data used covered the period from January 1, 2008, to December 31, 2018. The data was split into training and testing sets, with 80% of the observations used for training and 20% for testing. A total of 108 observations were used to build the models and forecasts, and 24 observations were used to assess the accuracy of the forecasts.

They discovered that for the 24-month time horizon, the most effective models were the multiplicative Winters model and the model with a quadratic trend. They acknowledge that forecasting the unemployment rate is highly challenging and resource-intensive, but it can also serve as an effective tool to support the planning process. However, their study only partially addressed the issue because obtaining a forecast that accurately reflects the actual value is difficult. The authors suggest that it would be interesting to compare modern predictive models like regression trees, neural networks, or deep-learning models, since the study only used popular time series models. They believe that analysing these models would further the continuation of their paper [22].

3.4 Comparative Studies

María E. Pérez-Pons and her colleagues conducted a study comparing traditional econometric models with machine learning (ML) algorithms in economic applications. They found that ML algorithms, particularly artificial neural networks (ANN) and random forests (RF), often outperform traditional models like ARIMA and various regression models.

The rapid expansion of ML applications, driven by real-time solution demands, has led to numerous models for economic problems, especially in the stock market. ML methods generally enhance prediction accuracy. Testing with diverse datasets from sectors such as energy, tourism, and agriculture, the authors concluded that while both ML and econometric models are effective, their combined use provides the best predictive performance due to their complementary nature [23].

Arthur Charpentier and colleagues noted that while econometrics and ML share the goal of building predictive models, they have developed distinct cultures. Econometrics focuses on creating probabilistic models to describe economic phenomena, whereas ML employs algorithms that learn from their mistakes, primarily for classification purposes.

They argue that ML methods often outperform traditional econometric methods, highlighting the need for econometricians to understand both cultures. ML tools can enhance econometric models by detecting nonlinear effects or overlooked cross effects, thus improving predictive power [24].

Sendhil Mullainathan and Jann Spiess highlight that the success of ML in tasks like face recognition, voice understanding, and language translation is due to its ability to adapt flexible models to data and generalise new data well. This differs from econometrics, where models estimate specific parameters reliably.

They argue that econometricians should understand the strengths and limitations of ML to use it effectively. ML can enhance econometric models by identifying nonlinear effects or overlooked interactions, making it particularly useful for handling large and complex datasets.

Mullainathan and Spiess have categorised machine learning applications in economics into three main areas. The first area involves using new data types for traditional questions, such as measuring economic activity with satellite images or classifying industries with text analysis. The second area focuses on improving parameter estimation, where machine learning is utilised in linear instrumental variables regression or in estimating different treatment effects. The third area pertains to direct policy applications, such as predicting the added value of hiring a particular teacher.

The authors recommend using machine learning tools thoughtfully to avoid naïve application or misinterpretation of results. When applying machine learning in econometrics, important choices must be made, including function selection and regularisation, encoding and transforming variables, the role of economic theory, and tuning procedures. These choices are vital and should be guided by both machine learning literature and economic theories [25].

The traditional econometric models have pros and cons. One main limitation is their potential to overfit data, leading to forecast errors. While these models work well with small datasets, they may be less effective with Big Data, where ML models offer better predictive and analytical capabilities. The authors suggest improving prediction accuracy by combining econometric models, feature engineering, and machine learning models in six phases: Data selection, Data refinement, Application of econometric models, Feature engineering, Application of machine learning models, and Selection of the best ML model.

Among several ML models, Random Forest, particularly as an Ensemble method, provides highly accurate predictions with limited data. Deep learning techniques are more effective with larger datasets, especially when parameters were finely tuned for high accuracy. Future directions include hybrid machine learning models, stochastic time series methods, and creating new deep learning models will likely lead to even higher accuracy with economic data [26].

In conclusion, combining ML and econometric models enhances predictive performance. While ML methods often outperform traditional econometric models in predictive tasks, their integration provides the most accurate and reliable forecasts due to the complementary strengths of each approach. Understanding and applying ML techniques thoughtfully in econometric contexts can significantly improve economic modelling and forecasting.

3.5 Studies Using Hybrid Models

Mustafa Yurtsever's study aims to investigate how their proposed hybrid method developed using LSTM and GRU can accurately predict unemployment rates. They utilised data from several countries being the USA, Great Britian, France, and Italy. This data was from the year 1983 to 2022. The data from 1983 to 2010 were used as the training set and the data used for the test set was between 2010 and 2022.

The proposed hybrid model integrated LSTM and GRU models. Data for the years 1983-2010 were used as the training set. While the test set used data between 2010 and 2022.The training and test data sets are divided into 70% and 30%, and normalised between 0 and 1 using a min-max scaler.

The model architecture consists of three layers: an LSTM layer with 128 hidden neurons, a GRU layer with 64 hidden neurons, and a dense layer with one neuron. The LSTM layer processes the input data first, generating weighted values that are then passed to the GRU layer. The GRU layer further processes this data, producing a final output which is then fed into the dense layer for the final prediction. The cost function is calculated by comparing this prediction with actual values, and the weights are adjusted accordingly to minimise the cost function. These optimised weights are stored for future use. This model is designed to provide accurate forecasts by leveraging the strengths of both LSTM and GRU networks in handling sequential data. They used RMSE to measure the accuracy of predictions, for their UK forecast the LSTM at 500 epochs the LSTM scored 0.47, GRU scored 0.30, and the hybrid model scored 0.28.The best accuracy was obtained with the hybrid model for all countries expect Italy where GRU scored an accuracy only slightly better than the hybrid model.

They suggested that one of the limitations of this study could be the hyperparameters of the model which may need to be further optimised to achieve better results. And also, the performance of the model may vary for different countries due to economic and social factors affecting the unemployment rate [27].

Mero, K.; Salgado, N.; Meza, J.; Pacheco-Delgado, J. and, Ventura, S aim to predict unemployment in Ecuador, using a genetic algorithm and LSTM method (GA-LSTM), Their research uses LSTM neural network to overcome complexities and nonlinearities in unemployment predictions and complementing them with genetic algorithms to optimise the parameters. Their main objective was to evaluate the accuracy of the hybrid GA-LSTM system and the deep learning neural networks BiLSTM and GRU to predict the unemployment rate in Ecuador.

Unfortunately, there is limited availability of data for Ecuador compared to other countries presenting a significant challenge for adapting the unemployment prediction models. The features used for their models were inflation, minimum wage, GDP and Gross fixed capital formation.

They aim to evaluate how well their model predicts outcomes and how the genetic algorithm impacts prediction accuracy. The genetic algorithm identified a 3-month time window as best for predicting unemployment rates. Additionally, they found that using 12 LSTM units in the hidden layer produced optimal results. They suggested that it is challenging to find the best values in machine learning because algorithm hyperparameters control various aspects of data training. Since the differences between actual and predicted values were small, a t-test was conducted to make an informed decision.

The one-layer GA-LSTM model accurately predicts unemployment rate scores, coming close to the actual values with a mean squared error (MSE) of 0.052. The predictions were generated from a sample spanning 22 months using BiLSTM and GRU neural network models, and 19 months using the GA-LSTM hybrid model. They performed a significance level test at 1% and found no evidence to suggest a significant difference between the models. The t-test was used to compare the following pairs:

GA-LSTM (one layer) and GRU (two layers), as well as BiLSTM (one layer) and BiLSTM (two layers).

The paired t-test revealed significant differences in most model comparisons, except for two specific cases where the models performed similarly. The GA-LSTM (one layer) model's predictions closely matched the actual unemployment rate scores, demonstrating its effectiveness. In contrast, other models did not demonstrate the same level of accuracy in predicting the actual scores. The author also suggests investigating the application of GA-LSTM modelling in comparison to other hybrid models for predicting economic indicators in various datasets from different countries [28].

Chakraborty, T; Kumar Chakraborty, A; Biswas, M; Banerjee, S and, Bhattacharya, S, proposed a hybrid model based on ARIMA and ARNN model to forecast the unemployment rates for seven countries being: Canada, Germany, Japan, Netherlands, Sweden, Switzerland, and New Zealand.

The proposed model filters out linearity using the ARIMA model and predicts nonlinearities present in the error residuals with an ARNN model. They evaluate the performances of individual models such as ARIMA, ANN, and SVM and two hybrid models ARIMA-SVM and ARIMA-ANN in comparison with the proposed hybrid model for the seven countries. Performance metrics such as RMSE, MAE, and MAPE were used for comparisons, with each model also performing a 1-Year ahead forecast and a 3-Year ahead forecast.

The best performing model was the hybrid ARIMA-ARNN model and were superior as compared to all the individual models tested. Comparing their three hybrid models often the ARIMA-ANN model would come close to the RMSE score of the ARIMA-ARNN, however, Japan was the only case where the ARIMA-ANN outperformed the proposed method with a score of 0.191 RMSE while the proposed method scored 0.22 RSME. For example, the Netherlands, saw the ARIMA-ANN scoring 0.143 RSME and the proposed method scoring 0.140 RSME, this close score was a common occurrence for all countries tested whilst performing the 1-Year forecast and the 3-Year forecast. It is evident that from this study that the same model can achieve different accuracy levels for different countries [29].

3.6 Research Discussion

This literature has revealed some interesting insights on the research topic of this study. It has been found that, in general, machine learning models outperform traditional models when predicting unemployment rates. As a result, research on hybrid models combining both methods was conducted. However, not all studies using hybrid models that were researched necessarily incorporated both methods; some opted for hybrid models using only machine learning methods. Nevertheless, these studies still provide an understanding of how hybrid models can predict and improve accuracy when forecasting unemployment rates. It was discovered that models perform differently when used on data from different countries. Additionally, it was found that the United Kingdom has not been a focal point in any of the studies examined.

As traditional econometric methods often outperform machine learning methods in accuracy, it was decided not to use them as an accuracy comparison, as there is substantial evidence indicating that they do not perform as well. Therefore, moving forward with this study, traditional models will not be used to compare with the developed hybrid models. Hybrid models will be compared against each other using performance metrics.

The literature review has identified gaps within the current research, providing areas to explore in this study. For instance, although hybrid models have been applied to UK data, they have not been a focus of these studies, as they typically compare similar models across different countries. This study will focus on the UK and the performance of multiple hybrid models to determine which performs best in this context.

CHAPTER 4

Dataset

4.1 Source of Dataset

Three datasets have been taken from the Office of National Statistics (ONS) and Statista. These datasets are open access and are as follows:

1. 'GDP Monthly Estimate, UK: April 2024' Published on 12th June 2024, by ONS. Access Reference: [30].

2. 'Unemployment Rate in the United Kingdom from March 1971 to April 2024' Published on 4th July 2024, by Statista Research Department. Access Reference: [31]

3. 'Inflation Rate for the Consumer Price Index (CPI) in the United Kingdom from January 1989 to May 2024' Published on 3rd July 2024, by D. Clark, published on Statista Access Reference: [32]

4.2 Dimension of Datasets

The previous subsection outlined the datasets used, all three datasets were originally downloaded as .XLS files containing two sheets per file, the data alongside with an overview/information about the data. To allow for analysis with this data in python an adjustment to the file type was needed. The datasets were converted to .csv files and the overview sheet was removed from each dataset, so as a result this left the actual dataset to work with in python. For this project the file conversion was done outside of code and within the Microsoft Excel application.

The key dataset for this study being examined is the unemployment rate dataset which includes two columns and 641 rows. This dataset includes unemployment rate and the date from March 1971 to April 2024. An important aspect about this dataset is that it only includes those persons of 16 years of ages and older. The next dataset is a GDP dataset with 215 rows and 2 columns specifying the date and the Monthly GDP from January 2007 to April 2024. Lastly, the inflation dataset with 428 rows and 3 columns which are the date, inflation rate, and a column to specify that the inflation rate is in a percent format. The variety in observations in each datasets suggests that there will be an imbalance in the data when using it for forecasting. Therefore, data processing was utilised to create a merged dataset which includes all three datasets with reducing the number of observations from the Unemployment and Inflation datasets so the merged dataset only includes data from 2007 to 2024.

Overview of the data:

The main dataset examined in this study is the unemployment dataset, which will be used for forecasting prediction. Therefore, providing a brief overview of important data aspects is necessary. In March 2024, the unemployment rate in the UK was 4.4 percent, a 0.1 percent increase from the previous month. Before the COVID-19 pandemic, the UK had low levels of unemployment compared to the mid-1970s. Within this dataset, the highest unemployment level was in the Spring of 1984, when unemployment hit 11.9 percent, with peaks occurring in 1993 and late 2011 [31].

CHAPTER 5

Methodology

5.1 Dataset Processing

Before merging the three datasets, data processing was necessary to ensure the datasets were clean and accurate to allow model implementation. Although each dataset was from reliable sources, missing values and duplicates were checked. Fortunately, no duplicate rows were found. However, there were missing values in the datasets. These missing values found did not affect the integrity of the data as they were found at the top of the datasets and had been placed as white space to split the title of the data from the actual data. The corresponding rows with missing values were dropped from the dataframes, removing 100% of the missing values.

Unnecessary columns were also dropped from both the Inflation and Unemployment datasets. Both had an extra column displaying that the previous column is in a percent format, for example, "in %." These columns were not necessary for the analysis, so they were removed. The datasets did not include headings for the columns, so it was necessary to add names to these columns. The heading "Date" was added to each dataset to name the date and ensure consistency across all datasets. The second column of each dataset was unique, so they were named differently: "Monthly GDP," "Inflation Rate," and "Unemployment Rate."

Finally, the datasets had to be resized to ensure they were the same size for merging. The GDP dataset, which had the lowest number of observations, was used as a benchmark for resizing the other datasets. The unemployment dataset had rows 0 to 430 removed, and the inflation dataset had rows 0 to 216 removed. As a result, both datasets now start from January 2007 and end at April 2024. This allowed for the merge on the Date column, which they all now have in common after some data manipulation.

5.2 Data Exploration

Data exploration is necessary for understanding the data distribution using data visualisations. Exploratory data analysis (EDA) will uncover hidden insights and meanings within the data that are not visible at first glance.

5.2.1 Time Series Line Graphs

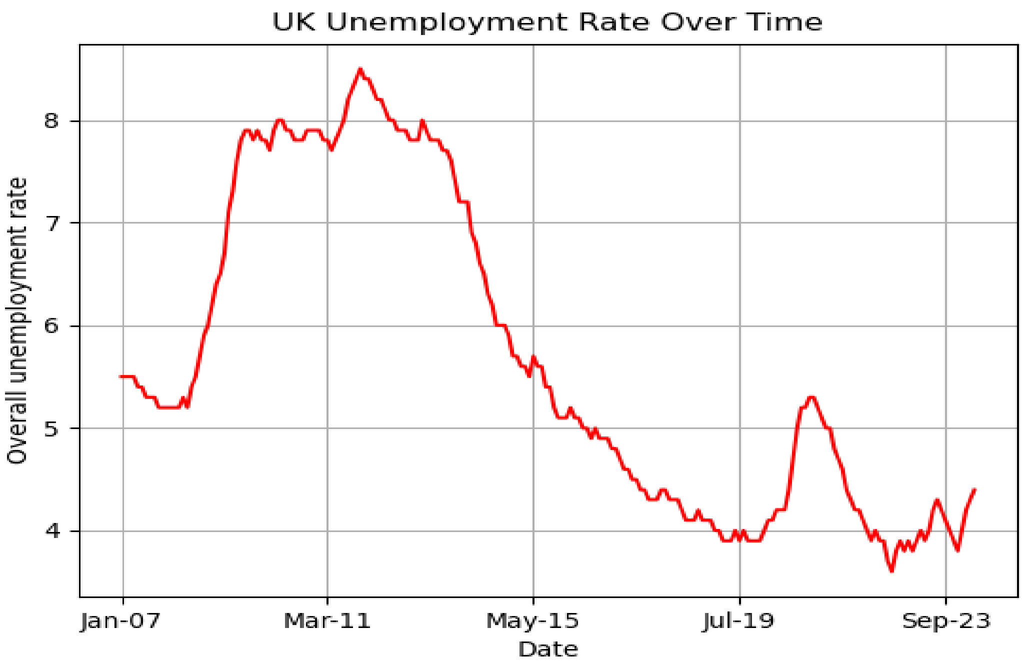

Figure 1: UK Unemployment Rate 2007-2024

Figure 1 displays the line graph 'UK Unemployment Rate Over Time' shows the changes in the overall unemployment rate from January 2007 to April 2024. There was a major increase around 2011, followed by a decrease and another peak in 2020.

It is not unexpected to see a disruption in 2020, as this coincided with the peak of the COVID-19 pandemic.

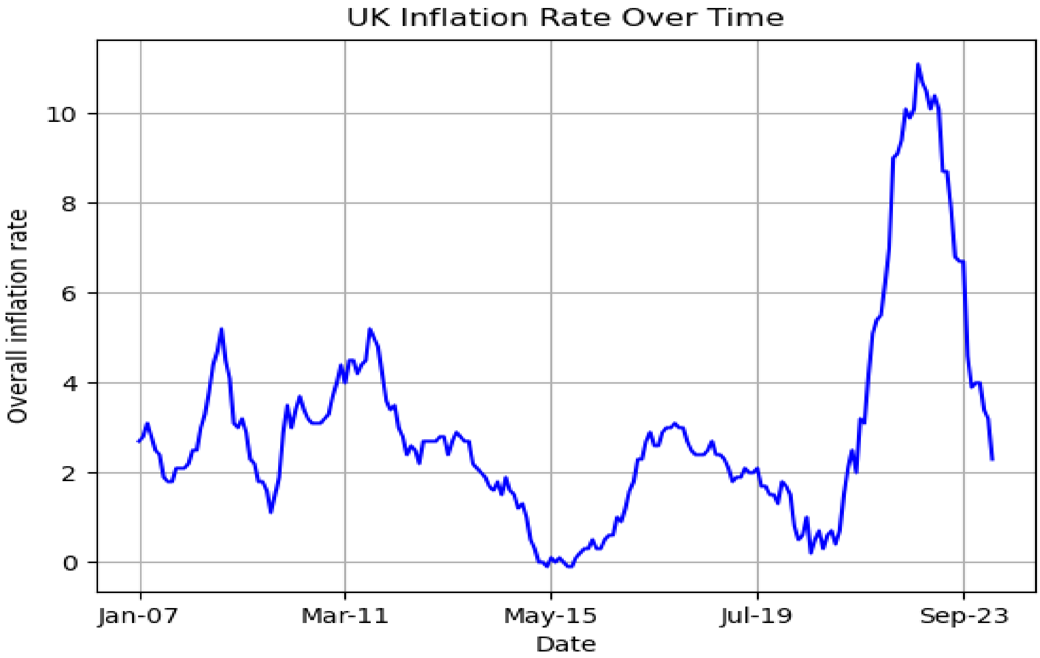

Figure 2: UK Inflation Rate 2007-2024

Figure 2 displays the line graph titled "UK Inflation Rate Over Time" shows the changes in the inflation rate from January 2007 to April 2024. There is a significant decrease around May 2015, a very steep increase towards the end of the timeline in 2020, and a sharp decrease in 2023.

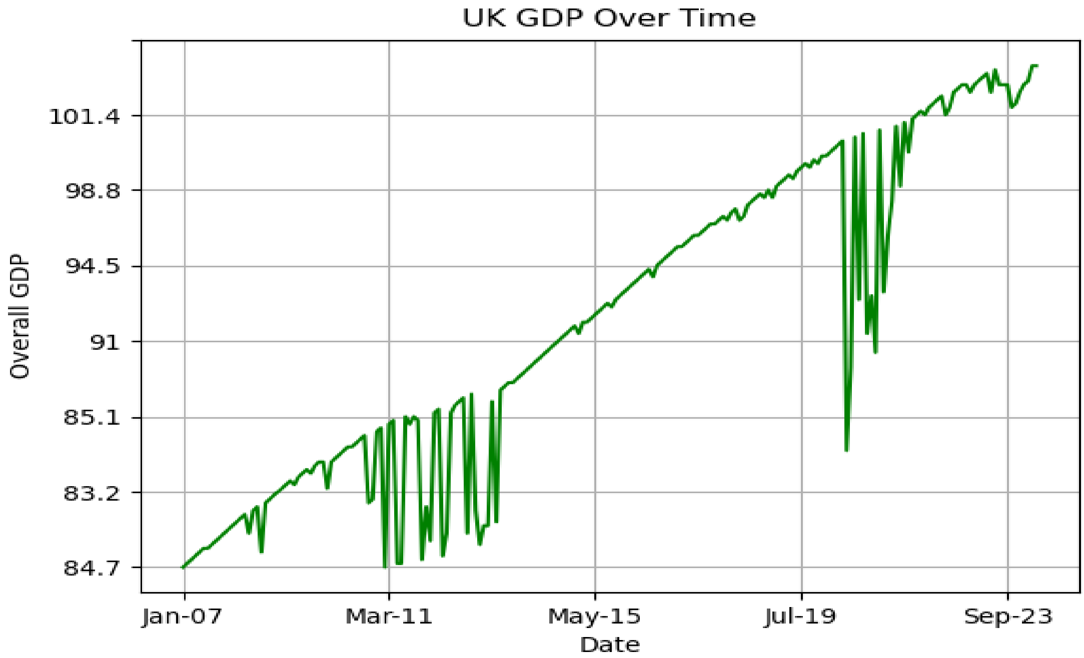

Figure 3: UK GDP 2007-2024

Figure 3 displays the line graph labelled 'UK GDP Over Time' illustrates the GDP of the United Kingdom from January 2007 to April 2024. The graph depicts an upward trend in the UK's GDP during this period. There are significant declines and subsequent recoveries in two periods: March 2011 and July 2019. This increase in GDP reflects economic growth in the UK.

Overall, these graphs indicate several relationships. For example, GDP dropped during periods of increased unemployment. Inflation rates also rose when GDP decreased, and unemployment rates increased.

5.2.2 Boxplots

Figure 4 shows three boxplots representing the three economic indicators: GDP, inflation, and unemployment.

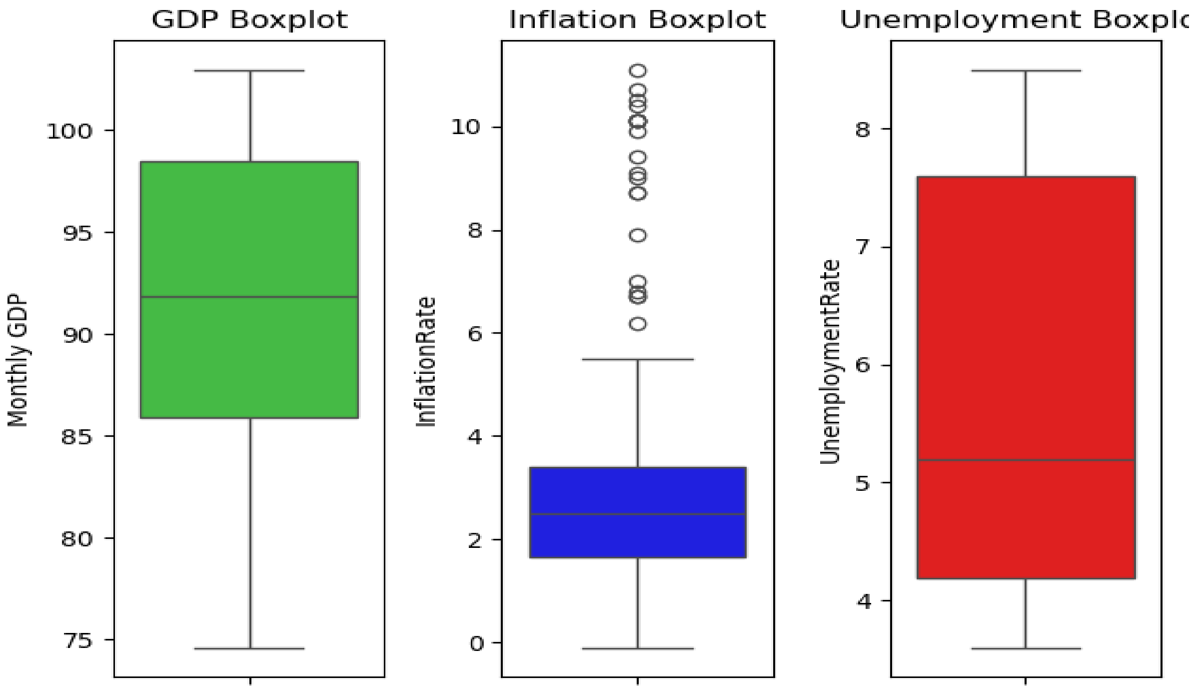

Figure 4: Boxplots

The first green boxplot represents the GDP. The range of values is approximately between 85 and 100. The central median line is closer to the upper quartile, suggesting that the GDP is higher around Q3.

The second blue boxplot represents inflation with a narrower range, between 2 and 4. There are many outliers present above the maximum value, and the box is closer to the first quartile.

The third boxplot displays unemployment rates ranging from approximately 4 to 8. The red box in the centre of the graph indicates variability in unemployment rates.

5.2.3 Correlation Matrix

Figure 5 displays a heatmap for the correlation between the variables: Date, Monthly GDP, Unemployment, and Inflation Rate. The colours in the heatmap indicate the strength and direction of the correlation: dark red indicates a strong positive correlation (closer to 1), dark blue represents a strong negative correlation (closer to -1), and white represents no correlation (closer to 0).

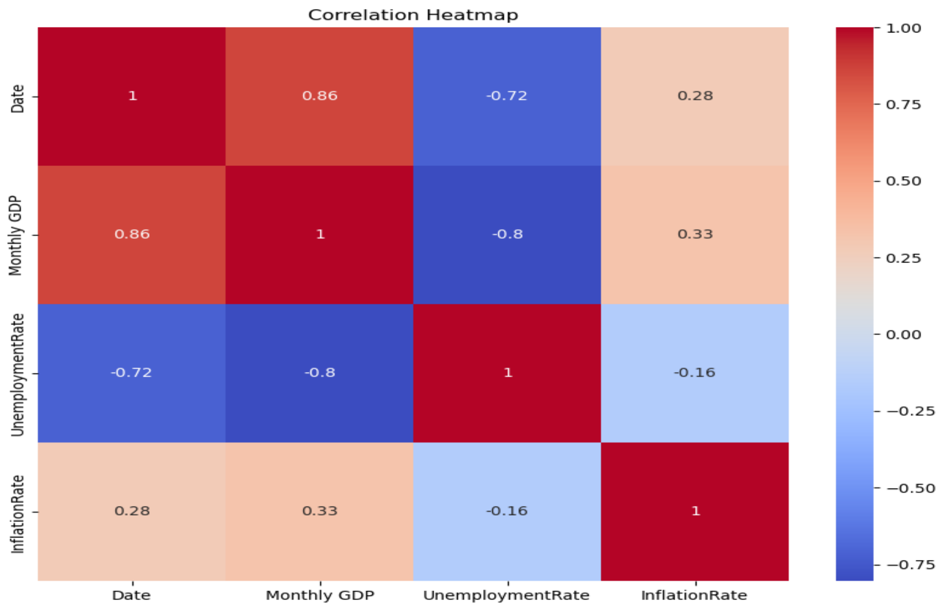

Figure 5: Correlation matrix

The correlation between Date and Monthly GDP is strongly positive at 0.86, indicating that GDP tends to increase over time. Additionally, Monthly GDP and Unemployment Rates show a strong negative correlation of -0.8, suggesting that when GDP rises, unemployment tends to decrease.

5.2.4 Seasonal Decomposition

The study focuses on unemployment rates. Therefore, the remaining subsections will focus on unemployment rates. Figure 6 below shows the original time series, trend, seasonal component, and residual component in four separate graphs.

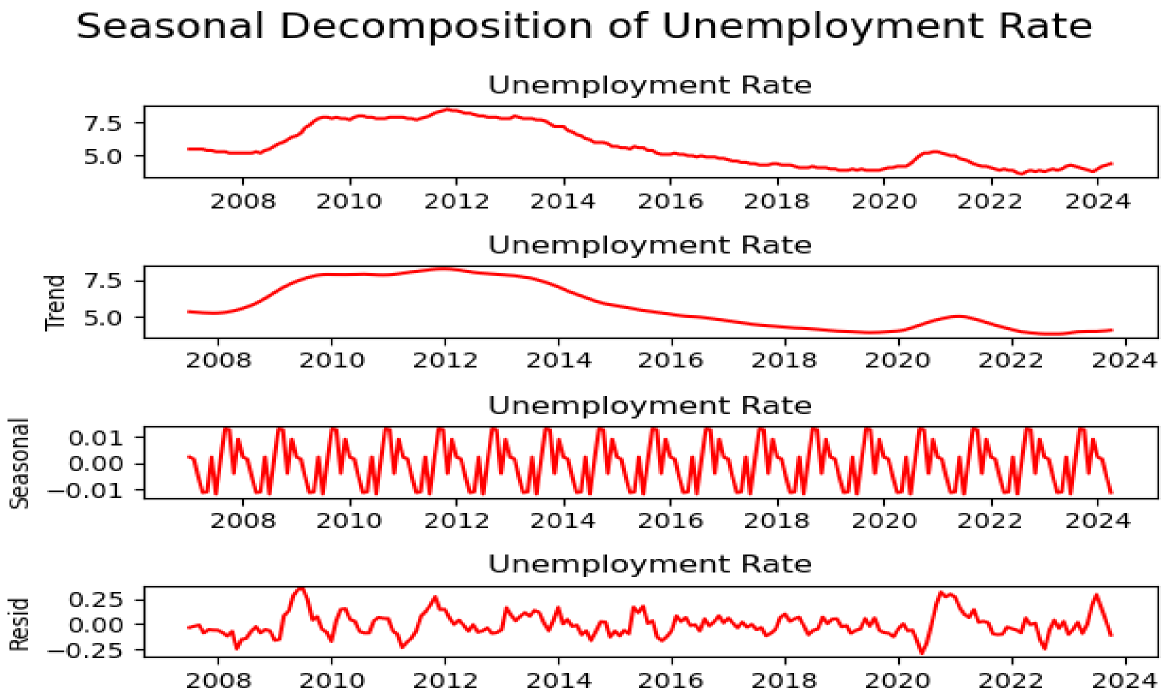

Figure 6: Seasonal Decomposition

The first graph displays the original time series mentioned earlier. The second graph, representing the trend component, captures the long-term movement and overall trend in unemployment rates. Over the years, the trend rises and falls, indicating periods of economic growth and recession.

The next graph depicts the seasonal component, showing regular fluctuations in unemployment rates within each year. The bottom graph represents the residual component, which shows no clear pattern or regularity. This represents irregular factors that could affect unemployment rates, such as unexpected events.

5.2.5 Autocorrelation Plot

Figure 7 illustrates the Autocorrelation Plot of Unemployment Rate. Each bar in the plot represents the autocorrelation coefficient for a specific lag, with the y-axis showing values from -1.00 to 1.00 and the lag observations ranging from 0 to 40. All the points on the plot are positive, indicating a positive correlation between unemployment rates and past values. The blue-shaded area around the horizontal line at 0 autocorrelations represents confidence intervals. Statistically, half of the points fall within this shaded area, highlighting their significance.

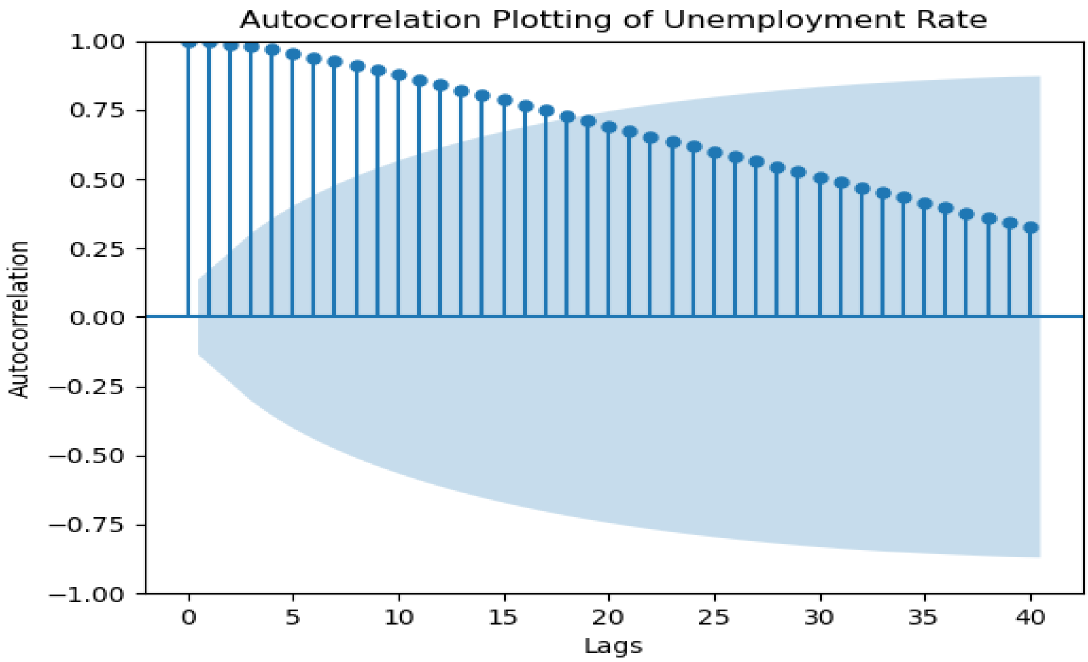

Figure 7: Unemployment Rate Autocorrelation Plot

5.2.6 Statistical Analysis on Unemployment Data

Statistical Measures	**Result**
Mean	5.65
Median	5.2
Maximum	8.5
Minimum	3.6
Std. Dev.	1.54
Skewness	0.48
Observations	208
ADF-level	-1.037
P-value	0.73
ADF - 1st diff.	-5.5
P-value	1.9966e-06
Kwiatkowski-Phillips-Schmidt-Shin	1.396
P-value	0.010

Table 5.1: Unemployment descriptive statistics.

Table 5.1 displays a number of statistics. The statistics reveal that the average unemployment rate over the observed period is 5.65 percent. The median value is 5.2 percent, indicating that half of the observations are above this value and half are below. This is only slightly lower than the mean. The unemployment rate ranges from a minimum of 3.6 percent to a maximum of 8.5 percent, demonstrating the variability in the data. The skewness of 0.48 suggests a slight positive skew.

As previously discussed, it is important to test for stationarity. If the data is stationary, we cannot derive insights from it. Therefore, an Augmented Dickey-Fuller test was performed to test for stationarity. The ADF statistic was approximately -1.037, with a p-value of 0.73. Thus, we cannot reject the null hypothesis at this level since the p-value is much greater than 0.05. This provides evidence that the unemployment data is non-stationary. After differencing once, the p-value was very small at 1.9966e-06, which rejects the null hypothesis. Therefore, the unemployment rate data is stationary after differencing.

After conducting another test for stationarity using the Kwiatkowski-Phillips-Schmidt-Shin test, the p-value was found to be 0.01, which is higher than the significance level of 0.05. This provides further evidence that the data is non-stationary.

Furthermore, knowing whether to use non-stationary or stationary data depends on the type of model used, which will be further explored in the next section where the Models used in this study will be discussed.

5.3 Methodological approach

5.3.1 Development and Training of Models

Train-test split of dataset: The dataset has been split for model training. Each model has been used twice, once for a 4-year horizon and once for a 2-year horizon, to evaluate its performance over different time-frames. For the 2-year horizon, the split is approximately 88% for training and 12% for testing, while for the 4-year horizon, the split is approximately 77% for training and 23% for testing.

Selection of models: Eight hybrid models have been developed for this study. These include ARIMA-LSTM, ARIMA-GRU, ES-SVR, ES-RFR, VAR-LSTM, VAR-GRU, GARCH-RFR, GARCH-SVR and ARIMAX-LSTM. All nine models are forms of supervised learning, as they require historical data for training and prediction.

Evaluation of Models:

It is important to analyse the performance of each forecast prediction model to determine its accuracy. All hybrid models will be evaluated based on their accuracy in predicting actual values. Metrics such as Root Mean Squared Error (RMSE), Mean Squared Error (MSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE) will be used to provide accuracy scores for each model, allowing for comparison.

Structure of Hybrid Models:

This section will briefly explore how each model works and how they were employed using Python. In this study, combining different models serves the purpose of testing various methods and assessing their accuracy. The literature review in this study proved that Machine Learning models often outperform econometric methods. By combining both types of methods, predictive performance can be improved, surpassing the individual methods.

Four well-known econometric methods were selected for this study, which were then combined with four machine learning algorithms suitable for forecasting tasks. To make the study more manageable, these models have not all been applied to each other, instead paired in ways that they may complement each other.

Each model was programmed in a Jupyter Notebook using Python. Note that the code in this study was inspired by an open-access article that can be accessed through the following reference [33]. This article used Prophet with LSTM, although this model was not employed in this study. Inspiration was taken specifically from the way the author combined the forecasts by adding both predictions together to create a hybrid forecast.

All the developed hybrid models follow a similar structure in terms of the way they were programmed. All models apart from the VAR and ARIMAX models are univariate and only use past values from the target variable to create predictions. After creating each econometric method forecast the residuals were then calculated. These residuals were then scaled between the range [0, 1] and also reshaped, the reshaping was needed so that it would be possible to feed the residuals through the for loop created to create new arrays for the machine learning models to train on and to capture patterns from After this, the results were plotted in line plots with two lines, one representing the actual unemployment percentage and the other representing the hybrid model forecast, and the evaluation metrics were also calculated. So that was a basic outline for the structure of each hybrid model, now the implementation of the models will be discussed.

5.3.2 Traditional Econometric Methods Used

The Traditional econometric methods included in this study are exponential smoothing (ES), Autoregressive Integrated Moving Average (ARIMA), Vector Autoregression (VAR), and Generalized ARCH (GARCH) models, each with unique features and applications.

(a). Exponential Smoothing

Exponential smoothing, developed in the 1950s, by Brown, Holt, and Winters, is a forecasting method for univariate time series data [3]. It uses a weighted moving average which is effective in quickly forecasting, particularly useful for short-term predictions.

This method requires historical observations, the latest data observations, and a smoothing coefficient (α) between 0 and 1, determining the degree of smoothing. A lower α results in higher smoothing, while a higher α makes the model more responsive to recent changes [34].

The simple exponential smoothing formula is:

$$-F_t = X_t \cdot \alpha + (1 - \alpha) + F_{t-1}$$

Where F_{t-1} represents the previous forecast, X_t represents the current observation and α represents the smoothing coefficient. This method assumes that data is stationary, without trends or seasonal patterns [3]. For data with cycles or trends, more advanced techniques are needed [34].

Exponential smoothing is efficient for inventory optimisation in retail by adjusting forecastsin demand patterns [35]. However, its simplicity can be a drawback, as it may overlook significant data points and rely heavily on initialisation and optimisation procedures [36].

Exponential Smoothing Model Structure:

The exponential smoothing technique utilised the training data to forecast the target variable. It also took into account additive trends and seasonal components, with a specified seasonal cycle of 12 periods, indicating yearly seasonality. Residuals were calculated from the exponential smoothing forecast and then scaled and reshaped to ensure a two-dimensional shape for the scaler.

The 'Statsmodels' documentation was utilised to assist in constructing this model. This documentation is open-source and can be accessed through the following reference [37].

(b). ARIMA

Box and Jenkins introduced the ARIMA (AutoRegressive Integrated Moving Average) model, which is a comprehensive approach to time series forecasting. The methodology involves three stages: identification, estimation, and diagnostic checking of models [38].

ARIMA models are well-suited for non-stationary data because they include mechanisms to transform non-stationary data into stationary data by differencing. Differencing removes or reduces trend and seasonality by stabilising the mean of the time series and eliminating changes [3].

ARIMA models utilise differencing as well as autoregressive (AR) and moving average (MA) terms. The autoregressive component incorporates past values into the model, while the moving average component addresses past forecast errors. This combination enables ARIMA models to effectively forecast future values based on historical data, making them a robust tool for time series forecasting [3].

The ARIMA(p, d, q) model uses:

- p: order of autoregression
- d: degree of differencing
- q: order of moving average [3].

When an ARIMA model includes other time series as input variables, the model is sometimes referred to as an ARIMAX model. Pankratz refers to the ARIMAX model as dynamic regression [39].

These models predict values based on past data, useful in economic applications such as forecasting stock prices and economic indicators [40]. However, ARIMA models have limitations, including sensitivity to parameter selection and difficulty handling nonlinearities and external factors [41].

ARIMA Model structure:

While constructing the ARIMA models, inspiration was drawn from an open-source article available via reference [42]. Furthermore, [43] aided in forecasting the test data.

To develop the ARIMA model, it was necessary to determine the best orders for p, d, and q. A PACF plot was used to determine the value of p.

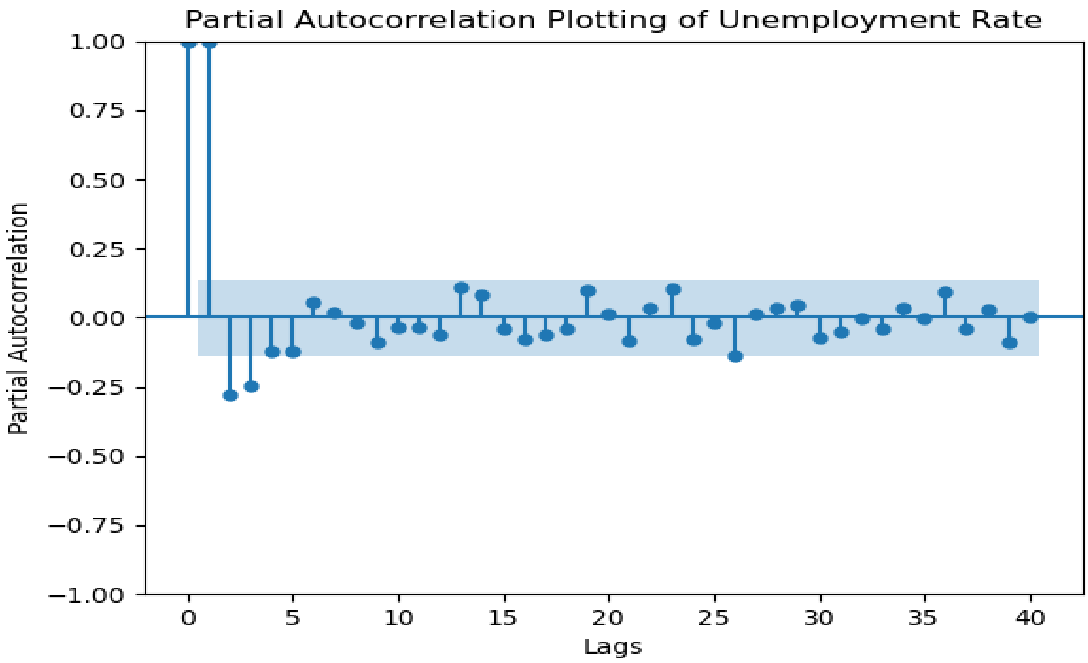

Figure 8: PACF Plot

Figure 8 displays a PACF plot, there are two noticeable spikes at lags one and two, indicating that the autoregressive component p should be either 1 or 2. After considering this, it was determined that component p would equal 2. The descriptive statistics section was used to figure out the value for d, where the ADF test was conducted to check for non-stationarity. It was found that first-order differencing would make the data stationary for the ARIMA model. Therefore, it was decided that component d

would equal 1.

In order to determine the value for component p, the ACF was used from the EDA section. This plot showed a gradual decline with no sharp cut off, leading us to choose a value of 0 for this component. The final order of the ARIMA model is: (2, 1, 0). This was then applied the ARIMA model to the training set using the target feature for prediction, and generated the forecast by calculating the residuals, which involved subtracting the test target feature from the ARIMA forecast. It's important to note that the ARIMA model automatically performs differencing on the unemployment data.

(c). ARCH/GARCH

ARCH and GARCH models, introduced by Engle (1982) and Bollerslev (1986) are crucial for modelling and forecasting financial volatility [44] [45]. The ARCH model allows the variability of a time series to change over time, capturing this conditional heteroskedasticity. The ARCH model captures changing variability over time with:

$$\epsilon_t = z_t h_t^{\frac{1}{2}}$$
$$h_t = \alpha_0 + \sum_{j=1}^{q} \alpha_j \epsilon_{t-j}^2$$

Where ϵ_t has a conditional variance h_t based on past errors. Despite its initial application to economic data, the ARCH model was quickly found to be useful in financial contexts, particularly in modelling and forecasting the volatility of assets such as interest rates, exchange rates, and stock returns.

The GARCH(p,q) model extends the ARCH by including lagged conditional variance terms:

$$h_t = \alpha_0 + \sum_{j=1}^{q} \alpha_j \epsilon_{t-j}^2 + \sum_{i=1}^{q} \beta_i h_{t-i}$$

This model provides a more parsimonious representation and better captures the long-memory properties of financial time series. The GARCH model can be viewed as an infinite-order ARCH model, accommodating long-memory properties and offering a more comprehensive framework for analysing time series with changing volatility.

This versatility makes GARCH a preferred choice in financial econometrics for volatility modelling.

Despite their strengths, ARCH/GARCH models require parameter constraints to ensure positivity and may need variance targeting for simplification [45].

GARCH Model Structure:

To build the GARCH model, the arch_model function was used with the volatility parameter set to 'GARCH'. The training set for the unemployment rate was included in the function. The order for the GARCH model was determined by analysing PACF and ACF plots, which suggested setting p to 2 and q to 1, the same order as for the ARIMA model. The model was set to forecast future values over the length of the test dataset, and the forecasted mean values were extracted for that period. These mean values were then used to find the residuals by subtracting them from the test data. The residuals were then scaled and transformed into a 2D array as required by the scaler. While constructing the GARCH Model, inspiration was taken from the code which can be accessed via the reference [46].

(d). VAR

VAR model is one of the multivariate econometric models used in this study. In 1980 Christopher Sims introduced the Vector AutoRegression (VAR) model, challenging the traditional simultaneous equations models. Sims' findings highlighted the limitations of pre-imposed theoretical restrictions and advocated for a model where all variables are treated as endogenous.

By using a VAR framework, Sims demonstrated that it is possible to capture the dynamic interrelationships among multiple time series without relying on arbitrary exogenous classifications. His empirical results showed that VAR models could effectively describe the joint behaviour of macroeconomic variables, such as GDP, interest rates, and inflation [4].

This VAR models joint dynamics and causal relations among a set of macroeconomic variables dominating time series econometrics modelling [4]. Each variable is a linear function of past values of itself and other variables. A (VAR(1)) model for three variables $x_{t,1},\ x_{t,2},$ and $x_{t,3}$ can be represented as:

$$x_{t,1} = \alpha_1 + \phi_{11}x_{t-1,1} + \phi_{12}x_{t-1,2} + \phi_{13}x_{t-1,3} + w_{t,1}$$
$$x_{t,2} = \alpha_2 + \phi_{21}x_{t-1,1} + \phi_{22}x_{t-1,2} + \phi_{23}x_{t-1,3} + w_{t,2}$$
$$x_{t,3} = \alpha_3 + \phi_{31}x_{t-1,1} + \phi_{32}x_{t-1,2} + \phi_{33}x_{t-1,3} + w_{t,3}$$

Each variable in the set is expressed as a linear function of the lag 1 values of all the other variables [47].

Despite their utility, VARs are criticised for being atheoretical; they are not based on economic theory that imposes a theoretical structure on the equations. Each variable in the equations is assumed to influence every other variable, making a direct interpretation of the estimated coefficients challenging [48].

VAR Structure:

As the VAR model is multivariate, additional features were needed to accompany the forecast. The features used included lagged versions of GDP, inflation, and unemployment.

VAR was also used in combination with LSTM. Like the ARIMA model, VAR also requires the data to be stationary. Therefore, the target feature for this model was the differenced unemployment rate. However, unlike the ARIMA model, the VAR model does not automatically difference the data. Therefore, the output in this model still involves differenced data, making the unemployment rate appear lower than it is.

The VAR model utilised the training dataset and its features for making predictions. In this model, it was determined that setting the maximum number of lags to 10 resulted in a lower AIC value, indicating a better fit for the model. After fitting the model, forecasts were generated using VAR, and the residuals were subsequently calculated. Whilst coding the VAR model inspiration was taken from the 'statsmodels' package documentation which can be accessed via the reference [49].

5.3.3 Machine Learning Models Used

(a). Long Short-Term Memory:

Long short-term memory (LSTM) networks are a type of recurrent neural network (RNN) designed to handle sequential data effectively. Introduced in 1997, LSTMs have internal memory and multiplicative gates, making them valuable for tasks such as predictions, pattern classification, and sequence generation [50].

LSTM is one of the most advanced models to process temporal sequences. For example, this model can be applied to financial market predictions. Financial data poses some huge forecasting challenges such as non-linearity, nonstationary, and sequence correlation [51].

LTSMs contain three gates and a cell: an input gate, an output gate, and a forget gate. The input gate controls what information gets added, the output gate decides the degree of forgetting current states, and the forget gate decides the degree of forgetting previous states [50]. Lastly, the cell which remembers values over arbitrary time intervals, the three gates help the cell by regulating the flow of information paired with the cell [29].

The architecture of this model involves a set of recurrently connected sub-networks, known as memory blocks. These memory blocks maintain its state over time and also regulate the information flow through non-linear gating units [29].

The hidden state of LSTMs is computed as follows:

Forget Gate:

$$f_t = \sigma\left(w_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Input Gate:

$$i_t = \sigma\left(w_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\tilde{C}_t = \tanh\left(w_c \cdot [h_{t-1}, x_t] + b_c\right)$$

Update of Cell State:

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$$

Output Gate:

$$O_t = \sigma\left(w_o \cdot [h_{t-1}, x_t] + b_o\right)$$

$$h_t = O_t \times \tanh(C_t)$$

where, σ represents the sigmoid function, w and b are the weight matrices and biases for each gate, respectively, x_t is the input at time t, and h_t is the output [52].

LSTMs excel at handling noisy data, distributed representations, and continuous values without predefined finite states. They also generalise well with separated and inconsistent input sequences. However, LSTMs require a significant number of weights due to additional units for each memory cell block [53].

(b). Gated Recurrent Unit:

Gated recurrent unit (GRU) is a variant of the RNN that simplifies the LSTM by reducing the number of gates from three to two [54]: the update gate (z_t) and the reset gate (r_t). [55].

The success of GRU models in RNN is attributed to their gating network signals, which control the influence of current inputs and previous memory in updating the current activation and produce the current state. These gating networks come with their own set of weights, which are adaptively updated in the learning phase.

The weights associated with the gates are updated using backpropagation through time and stochastic gradient descent, aiming to minimise a loss or cost function. Each parameter update involves information about the overall state of the network. Thus, the latest state variable reflects all details regarding the current input and previous hidden states. There is a redundancy in the signals influencing the gating signals, the primary driving signal should be the networks internal state. Furthermore, the adaptive parameter updates involve components of the systems internal state [56].

The update gate decides how much the unit updates its activation and content, while the reset gate allows the unit to forget the previously computed state [55].

The GRU equations are:

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

$$\tilde{h}_t = g\left(W_h X_t + U_h(r_t \odot h_{t-1}) + b_h\right)$$

The gates are calculated using the equations:

$$z_t = \sigma\left(W_z X_t + U_z h_{t-1} + b_z\right)$$

$$r_t = \sigma\left(W_r X_t + U_r h_{t-1} + b_r\right)$$

Equations are from [56].

These models can enhance the learning capabilities of RNNs, and also increase the parameterisation through their gate networks. Various studies have explained that GRU RNN competes in performance with LSTM and outperforms it in most cases [56]. However, despite GRU being simpler and having fewer parameters than LSTM, GRU struggles with learning long sequences and does not explore the importance of each element in the sequences [54].

LSTM/GRU Structure:

The architecture for both LSTM and GRU was the same, so they will be discussed together. An online article was used as inspiration for building the LSTM model. Adjustments were made to the model to fit the unemployment rate data, such as choosing suitable parameters. The article can be accessed via reference [57]. Another article suggested incorporating sequences and specifying the window size for prediction. This method was applied to each model, this article can be accessed via the reference [58]. This article was also used to aid with the GRU model which can be accessed via the reference [58].

Both hybrid models were designed with the same architecture to ensure fair testing. This study aims to explore how econometric methods complement machine learning models. Using different ML architectures could potentially bias the results and lead to an inaccurate comparison.

Both models were provided with sequences to train them to predict the next data point. As the unemployment data is monthly, a timestep of 12 was chosen. This means that the input sequence would consist of 12 data points, enabling the model to learn data patterns over this window size.

The LSTM and GRU architecture included four layers. Layer 1 used ten neurons. This decision was made because only three features were used to predict unemployment, and the unemployment data was not complex. Return sequences were also utilised for the model to return the full sequence of outputs, allowing the next layer to receive all sequence outputs. The input shape was defined in layer 1, which used 12 time steps and 3 features at each step. The second layer adds another ten neurons with return sequences set to false, as this is the last layer before the dense layers that expect a single input vector. Finally, both layers 3 and 4 are dense layers.

The models were trained using the Adam optimiser and MSE as the loss function. Each model went through 50 epochs, which means the training data was fed through the algorithm 50 times. A batch size of 30 was used, indicating the number of samples processed at each stage of training. This batch size was chosen because the dataset only contains 208 observations, making higher numbers of epochs unnecessary.

The combination of ARIMA with LSTM was decided because they have complementary strengths. ARIMA is effective for capturing linear relationships and seasonality, while LSTM is capable of learning complex, non-linear patterns and long-term dependencies. By combining these two models, hopefully this will be able capture a broader range of patterns in the data, which should potentially improve forecast accuracy.

Additionally, it was decided to combine VAR with LSTM. The VAR model filters the linear interdependencies among the time series, and the LSTM model captures linear trends in the residuals computed from the VAR model [59].

The decision to combine ARIMAX with LSTM was made because we had previously decided to combine both ARIMA and LSTM models. However, the main reason behind using the ARIMAX-LSTM model is to compare the performance with previous multivariate models.

(c). Support Vector Regression:

Support Vector Regression (SVR) is a supervised machine-learning technique for regression problems. Unlike support vector machine (SVM) classification, which produces binary output, SVR estimates a real-valued function, making it suitable for predicting continuous outcomes [60]. SVR is designed for regression and prediction problems and became an important research topic due to its success in classification and regression tasks, particularly in time series prediction and financial applications.

SVR generalises SVM by introducing an ϵ-insentive region, called the ϵ-tube, around the function. The optimisation problem aims to find the flattest tube that approximates the continuous-valued function, balancing model complexity and prediction error. SVR uses convex ϵ-insensitive loss function, and the foal is to minimise this function while ensuring training instances are within the tube.

The solution is obtained through convex optimisation, yielding a unique result. Support vectors, which are training samples outside the tube, significantly influence the tubes shape. As in SVM, training and test data are assumed to be independent and identically distributed, drawn from the same unknown probability distribution [61].

The SVR problem formulation is often best derived from a geometrical perspective, using a one-dimensional example. The continuous-values function being approximated is written as:

$$y = f(x) = \langle w, x \rangle + b = \sum_{j=l}^{M} (w_j x_j) + b, \quad y, b \in \mathbb{R},\ x, w \in \mathbb{R}^M$$

For multidimensional data, x is augmented by one and b is included in the w vector, resulting in the multivariate regression:

$$f(x) = \begin{bmatrix} w \\ b \end{bmatrix}^T \begin{bmatrix} x \\ 1 \end{bmatrix} = w^T x + b, \quad x, w \in \mathbb{R}^{M+1}$$

SVR formulates this function approximation as an optimisation problem to find the narrowest tube centred around the function while minimising prediction error.

The objective function for this optimisation is:

$$\min_{w} \frac{1}{2}\|w\|^2$$

Here, $\|w\|$ is the magnitude of the normal vector to the surface being approximated. This magnitude can be interpreted as a measure of flatness. For example, in the polynomial function $f(x) = w_i x_i$, increasing $\|w\|$ results in more nonzero w_i, leading to higher-order solutions [61].

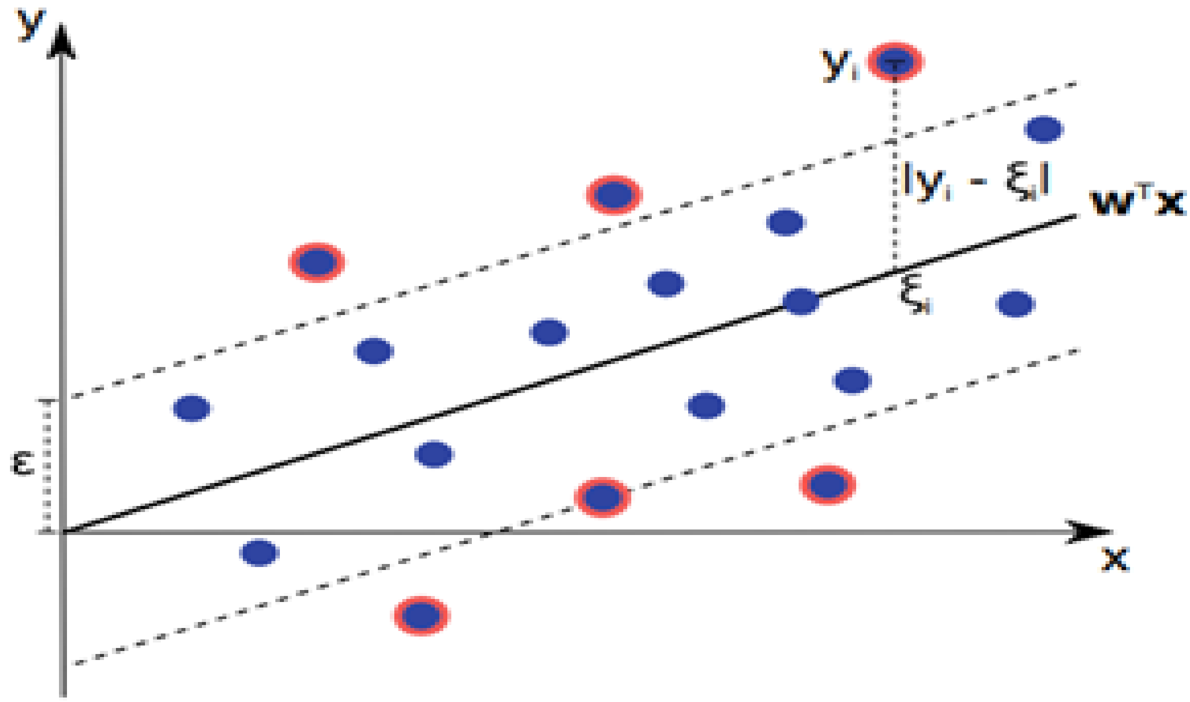

Figure 9: SVR diagram

In Figure 9, there is a SVR regression function depicted by $w^T x$. The ϵ-insensitive tube around the function is shown as a gray tube. $\xi_i = w^T x_i$ represents the predicted target value of ξ and y_i represents the actual target value. Additionally, the support vectors are marked by a red border [62].

Additionally, SVR can handle nonlinearity using kernels and does not require a predefined model, leading to more accurate modelling of complex relationships. It also calculates intervals for variable importance, enhancing the understanding of input-output relationships [60].

SVR Model Structure:

The data was pre-processed using sequences to enable the SVR model to learn from historical values and make predictions for future values. Two arrays were then formed: one to store the input sequences for the SVR model, and another to store the targeted residuals from the previous model used in the hybrid models. At each time point, the preceding 12 residuals are used to predict the current residual.

The SVR model is first set up with a radial basis function kernel to handle non-linear relationships. Next, the model is trained using the input sequences and the target residuals. Once trained, the model predicts the residuals based on the input sequences. These predicted residuals are then scaled back to their original scale using the inverse transform of the scaler, ensuring that they are on the same scale as the original unemployment rates.

For the creation of the SVR model inspiration was taken from an online article which can be accessed through the following reference [63].

The decision to combine both ES and SVR was based on the simplicity of both models, which have the potential to complement each other. SVR does not require a predefined model, making it simple to use, while the ES model's simplicity enables it to make quick short-term forecasts. However, the simplicity of the models could be perceived as a drawback when dealing with more complex data.

The combination of both GARCH and SVR was chosen because both have strengths in financial applications. Although financial data was not used in this study, the characteristics of financial data could be similar to that of the unemployment data used, as both could potentially be volatile. Therefore, it is expected that this model could perform well in this study.

(d). Random Forest

Random forests, introduced by Breiman in 2001, are popular for their high performance on high-dimensional data, fast computation, and east of tuning. However, traditional random forests may not effectively capture time-dependent structures, as they assume observations are independent [64].

In his example, Goehry, B, provides a regression function for random forests. He suggests using a random sequence, denoted as $(x_t, y_t)_t \in \mathbb{Z} \times \mathbb{X} \times \mathbb{Y}$ such that $Y_t = f(x_t)$ and an error term denoted as ϵ_t is such that $\mathbb{E}[\epsilon_t \mid X_t] = 0$. Goehry, B, proposes that the primary aim of random forests is to estimate the regression function based solely on observing a training sample $D_n = \{(X_t, Y_t), \ldots, (X_n, Y_n)\}$, the regression function is as follows:

$$\forall x \in X, \; f(x) = \mathbb{E}[Y_t \mid X_t = x]$$

[65].

As the baseline model is unsuitable for time series analysis, this study will use the Random Forest Regressor (RFR) as we are using the random forest for a regression task. RFR is a popular ensemble model in regression machine learning. It employs bootstrap aggregation, where creating multiple trees, and averaging the predictions made by all estimators. During the process of selecting split points, the algorithm uses a random and limited sample to choose the best feature predictor.

The RFR model offers various parameters that influence its structure and learning process. These parameters include the number of estimators, the maximum depth of each tree, the maximum number of nodes, and those controlling the learning process, such as the splitting criterion, the minimum weight fraction of each leaf, and the method for determining the number of features to consider.

Despite the models bagging procedure, RFR can still be prone to overfitting and is computationally expensive. Nevertheless, there are several advantages, for instance, its robustness and flexibility. It also performs well even in the presence of outlier or missing values, and feature scaling is not required [66].

Random Forest Model Structure:

The sequences were initially created for the model using input sequences of 12 steps. These prepared sequences were then used to train a random forest model. The random forest employed 100 decision trees and a random state to ensure reproducible results.

The model was trained using the input sequences and the target residuals. Following training, the random forest was used to predict residuals based on the input sequences. These residuals were then transformed back to their original scale using the inverse transform scaler to enable plotting with the actual unemployment rates. Sci-kit learn documentation was utilised to aid in constructing this model. The documentation is available online via the following reference [67].

The decision to combine GARCH and RFR was made due to their ability to handle volatility and non-linear relationships. The GARCH model's strength lies in financial data, so it is expected to perform well with the volatility of the UK unemployment data.

The decision to combine ES and RFR was made because the simplicity of the ES model and its usefulness in short-term predictions could complement the RFR, which is more computationally expensive. Additionally, the flexibility and robustness of RFR could support ES in areas where it falls short in longer-term forecasts.

Chapter 6

Results & Discussion

6.1 Results

6.1.1 ARIMA - LSTM Hybrid Model

The ARIMA model used for the 3-year forecast resulted in an AIC of -307.220, indicating a good fit as this is a low number. For the hybrid model, the RMSE was 0.225, suggesting that the model performs well on average but may have deviations in some periods. The MAE was 0.193, which was similar to the RMSE, indicating consistent errors. The MSE was 0.051, and the MAPE was 0.047. These performance metrics reveal that the model has overlooked some important trends in the data. While the model can capture the general trend in unemployment rates as shown in figure 10, it fails to accurately predict significant peaks, such as the one from May 2023 to September 2023. Therefore, its reliability could be improved. These results suggest that the model may handle long-term trends well, however, it may not capture short-term fluctuations or sudden changes.

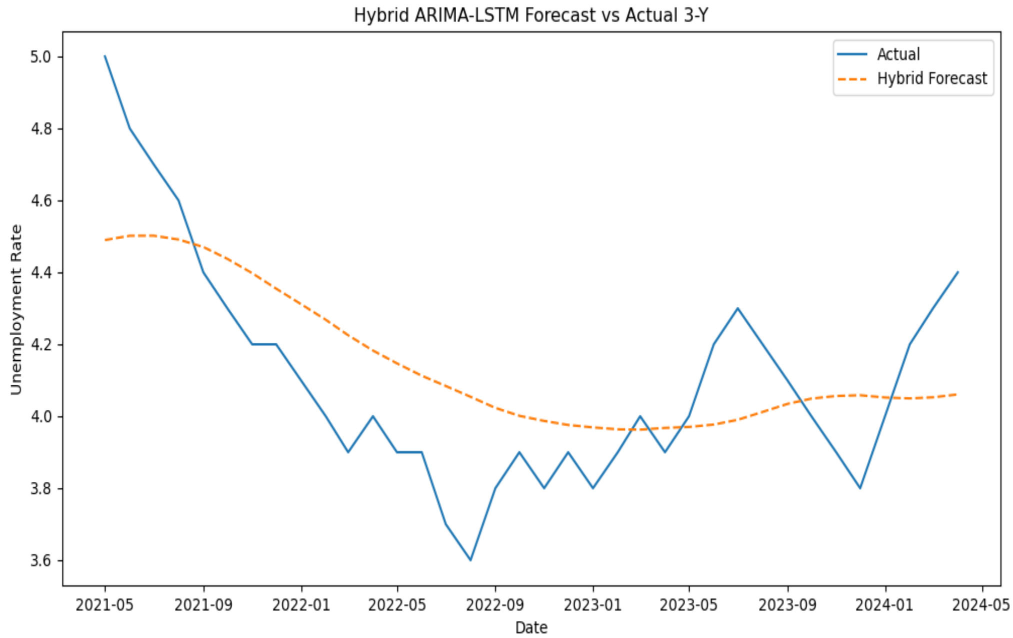

Figure 10: ARIMA-LSTM-3Y Forecast

The ARIMA model's AIC for the 1-year forecast was -344.560, which indicates an improved fit for short-term predictions. This means that the model performs better in the short term. The RMSE was 0.182, smaller than the 3-year forecast, indicating better accuracy in shorter-term forecasts. This suggests that the model performs more reliably over shorter periods. The MAE also supports the model's improved performance with a lower score of 0.157, showing fewer deviations from actual values. The lower MSE also indicates a more stable and accurate prediction. The lower MAPE of 0.0038 confirms better accuracy for the short term, suggesting that the model's prediction is closer to the actual values in percentage terms. Despite the better metrics, figure 11 shows that the model misses a significant dip from November 2023 to January 2024. This suggests that while the ARIMA-LSTM model is effective in capturing trends, it still struggles with sudden deviations in the unemployment rate data.

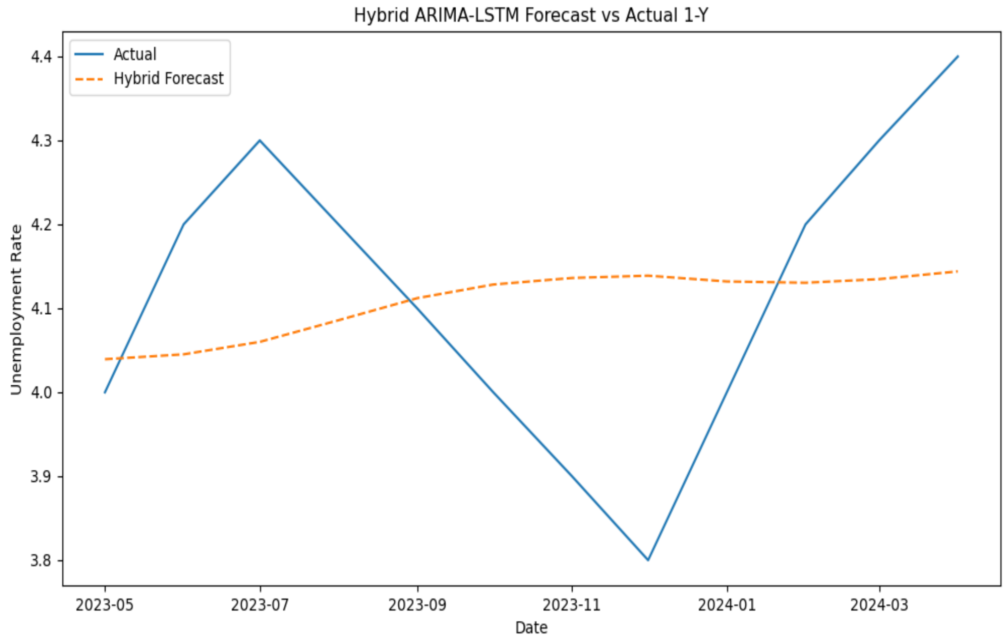

Figure 11: ARIMA-LSTM-1Y Forecast

The ARIMA-LSTM model shows better performance for short-term forecasts than for long-term forecasts, as evidenced by the improved AIC, RMSE, MAE, MSE, and MAPE. This suggests that the model is more accurate at capturing immediate patterns but struggles with longer timeframes. Both periods fail to predict specific peaks and dips, indicating areas for improvement in this model. This might be due to the limitations of ARIMA, such as its inability to handle nonlinearities effectively, as previously discussed. Both figures 12 and 13 miss the same dip in the data, indicating a common issue at that specific point in time. This dip is also a significant decrease in the unemployment rate, suggesting that both models are unreliable when considering the visualisations.

6.1.2 ARIMA - GRU Hybrid Model

For the 3-year forecast, the ARIMA model had an AIC of -307.220. This AIC is consistent with the earlier ARIMA model, suggesting that the ARIMA component in the hybrid model holds a similar level of fit. The RMSE came to 0.17, which seems like a reasonable prediction error. It is lower than the ARIMA-LSTM hybrid model, suggesting that the GRU has enhanced the model's ability to capture underlying patterns. The low MAE score of 0.139 indicates that the model makes close predictions across the time series. A lower MSE of 0.029 also indicates fewer large errors. The MAPE resulted in 0.034. This low value suggests that the model is accurate in terms of the percentage deviation from the actual unemployment rate, making it more reliable.

In Figure 12, it is clear that the hybrid forecast line captures the general trend and volatility of the data slightly better than the ARIMA-LSTM model at the 3-year horizon. The GRU model appears to capture more of the non-linear behaviours in the unemployment data and closely follows the actual unemployment rates. The forecast is most accurate at the beginning, with the prediction gradually diverging as time progresses. This indicates that while the GRU model improves the prediction using the ARIMA model, uncertainty increases over time.

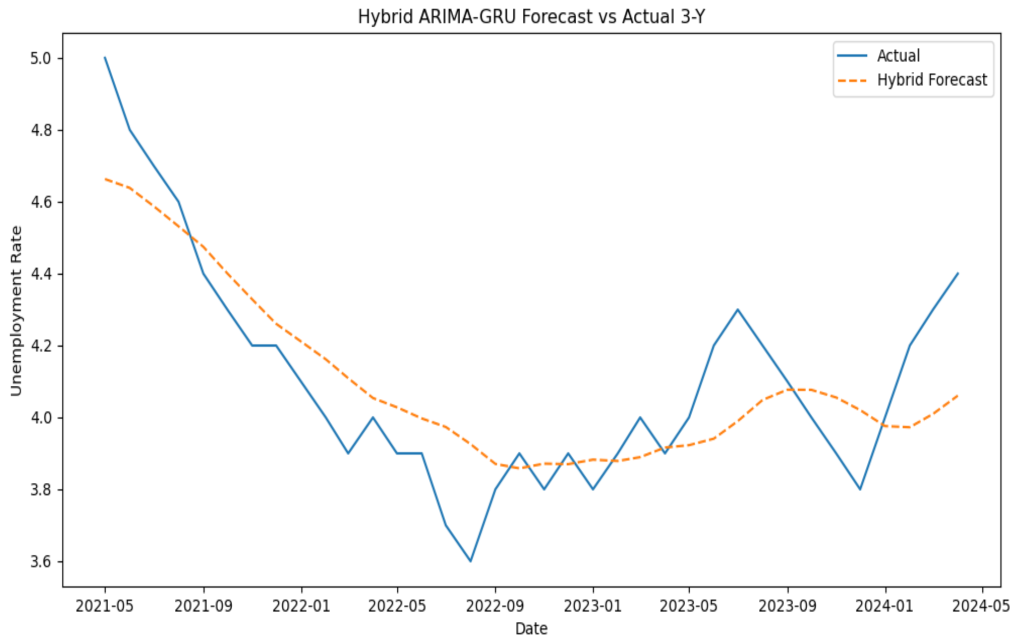

Figure 12: ARIMA-GRU-3Y

For the 1-year forecast, the AIC remains consistent with the previous ARIMA-LSTM model. This shows that the ARIMA component's fit remains stable when combined with GRU in the short-term forecast. Despite the consistent fit, the accuracy in the short-term forecast presents challenges. The higher RMSE of 0.184 compared to the 3-year forecast reflects an increased error in the short-term prediction. This suggests that the model is less effective at capturing sudden changes over shorter periods. The higher MAE of 0.161 indicates larger average errors in the short-term forecast, highlighting the model's struggle with the fluctuations present in the 1-year horizon. Although the MSE resulted in 0.034, proving that the GRU component helps reduce large errors, the model's short-term accuracy still suffers. The MAPE of 0.039 reflects a higher percentage error in the short-term, indicating that the prediction accuracy of this model decreases when forecasting shorter time periods.

Figure 13 demonstrates that the model is unable to accurately represent the significant drop and instead predicts a gradual increase in the unemployment rate followed by a slow decrease. This graph further supports the idea that while the GRU model is effective for long-term forecasting, it may not be well-suited for short-term forecasting and sudden changes in unemployment rate data. Despite missing the main drop, the model correctly identifies a slight increasing trend toward the end of the forecast, indicating that it still captures the overall direction of the unemployment rates. However, this prediction still deviates significantly from the actual unemployment rate.

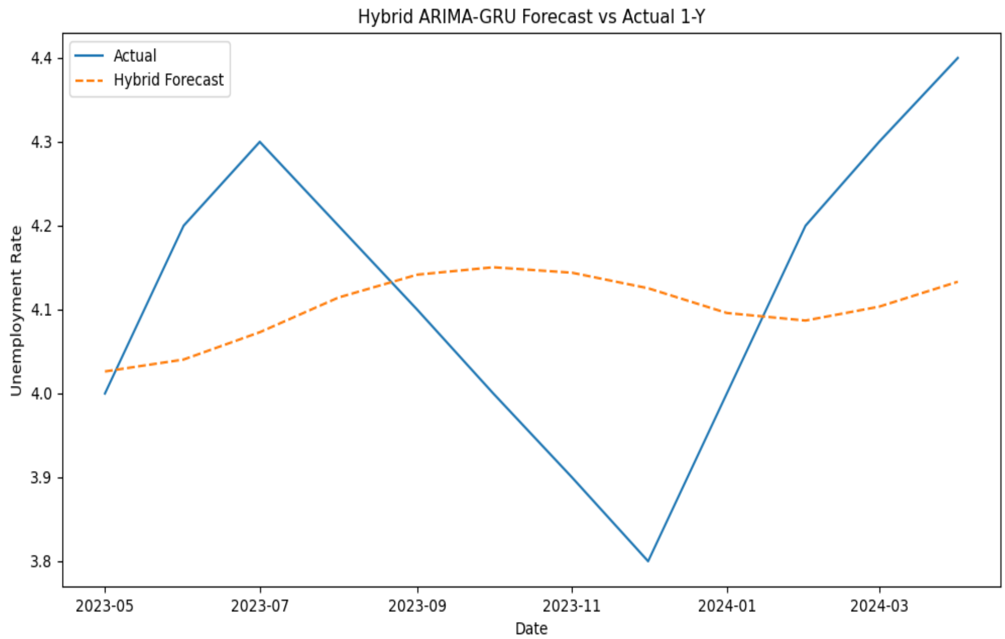

Figure 13: ARIMA-GRU-1Y

The ARIMA-GRU hybrid model performs better than the ARIMA-LSTM model in long-term forecasting. This is evident from the lower RMSE, and the overall trend captured in the 3-year forecast shown in figure 12. It suggests that the GRU component may be better suited to handling long-term dependencies and trends compared to the LSTM for the unemployment rate data. Both ARIMA-LSTM and ARIMA-GRU struggled with short-term predictions, especially in capturing the main dip in the data, indicating a common challenge in these models' ability to handle sudden changes. This could be due to the ARIMA's difficulties with non-linear data. Improvements could be made to the ARIMA model to more accurately predict short-term trends, such as reviewing the parameters and fine-tuning the lags or degree of differencing to make the model more responsive to the shorter time period.

6.1.3 ES - SVR Hybrid Model

The ES-SVR model shows a high level of accuracy in forecasting the 3-year unemployment rate, as indicated by the performance metrics. The RMSE is 0.132, which is a low prediction error over the long term, suggesting that the model effectively captures the overall trend of the unemployment rates. The low MAE of 0.103 highlights that the average error between the forecast and the actual unemployment rates is small. The low MSE of 0.017 shows that large deviations from the actual data are uncommon, reinforcing the model's reliability for longer-term trends. The low MAPE of 0.025 suggests that the percentage deviation from actual values is small, indicating that the model is highly accurate at predicting the unemployment rates over this period.

Based on Figure 14, the model effectively captures the general trend in unemployment rates with minimal errors. However, it struggles to predict the peak between May 2023 and September 2023. This challenge is consistent with the ARIMA-GRU model, suggesting that the ES-SVR model is strong in trend prediction, but encounters difficulties in forecasting specific volatile periods.

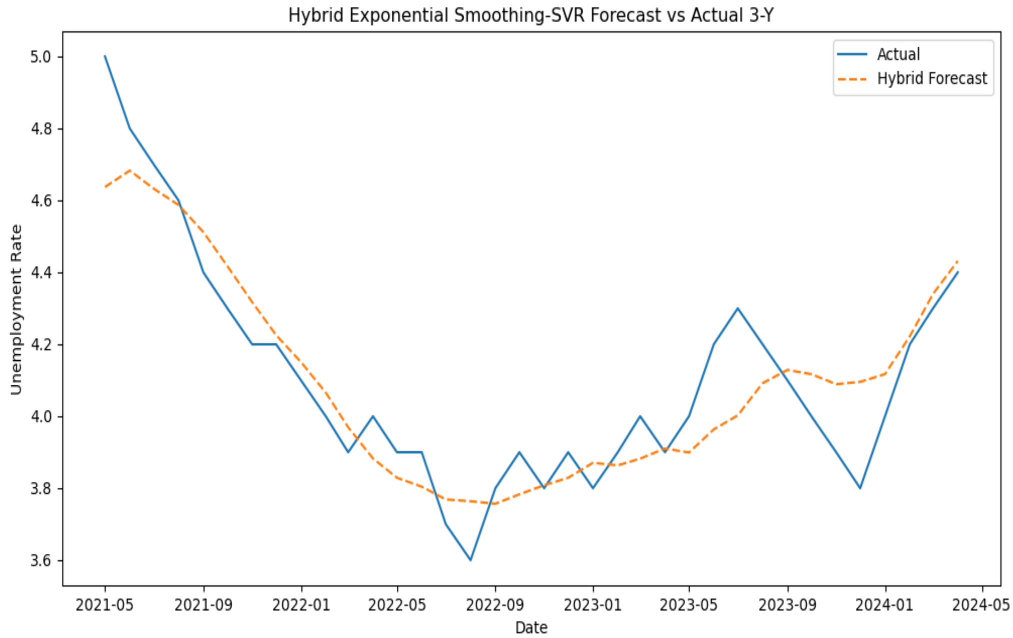

Figure 14: ES-SVR-3Y

The ES-SVR model performed exceptionally well in the 1-year forecast, demonstrating higher accuracy than the 3-year forecast.

The RMSE decreased significantly to 0.054, indicating a closer prediction to the actual unemployment rates over the shorter time period and highlighting stronger precision. The lower MAE of 0.043 suggests that, on average, the model's predictions are very close to the actual values, proving its high accuracy in short-term forecasting. The very low MSE of 0.003 shows minimal large errors in the model, reinforcing its high precision. The low MAPE of 0.011 further confirms the slight percentage deviation from the actual values, suggesting that it can effectively capture short-term fluctuations.

In Figure 15, it is evident that the hybrid model forecast closely tracks the trend of the unemployment rate. The model consistently follows this trend with only minor inaccuracies, such as a slight dip from November 2023 to January 2024, which previous models had difficulty predicting. However, the ES-SVR model comes even closer to the actual unemployment rate, showcasing its superior short-term forecasting capabilities.

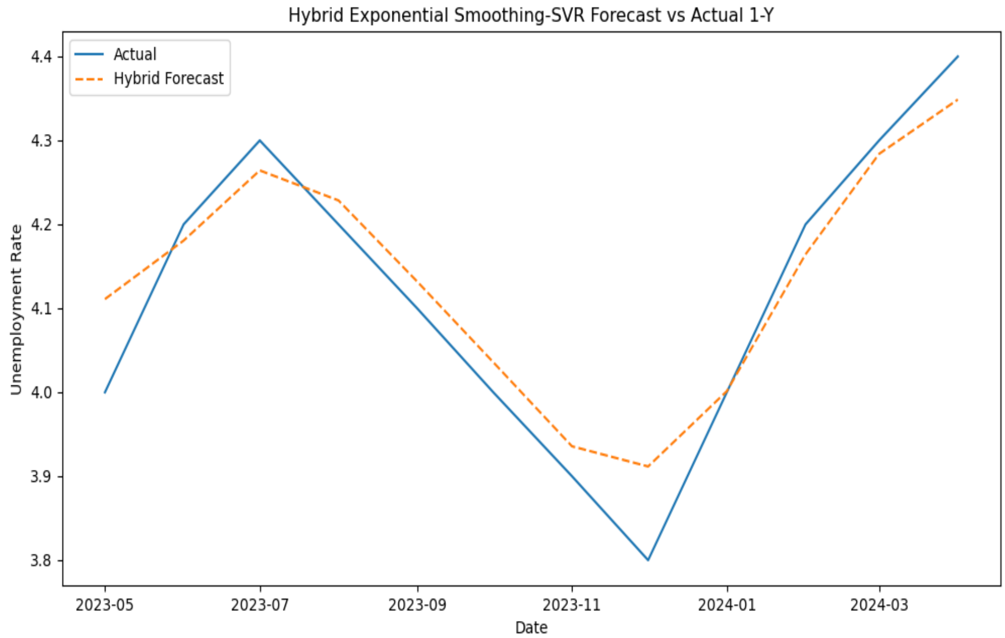

Figure 15: ES-SVR-1Y

The difference between the 3-year and 1-year forecasts is significant. This difference may not be evident when analysing the graphs. Therefore, it's essential to use metrics to assess the accuracy of both time periods. The RMSE for the 1-year forecast is more than half as small as that for the 3-year forecast. This suggests that the ES-SVR hybrid model is particularly more capable of handling short-term variations in unemployment rates.

6.1.4 ES - RFR Hybrid Model

The ES-RFR model demonstrates a high level of accuracy for both 3-year and 1-year forecasts, as evidenced by the performance metrics. For the 3-year forecast, the RMSE was 0.065, which is lower than the previous ES-SVR model. This suggests that the ES-R model is better at capturing long-term trends for unemployment rates. The MAE also yielded a low score of 0.043, demonstrating that, on average, the forecasted values are close to the actual values, indicating consistent model accuracy throughout the forecast. The low MSE of 0.004 also indicates that forecast errors are small. The very low MAPE of 0.01 suggests that the model's percentage error is minimal, indicating highly accurate forecasts by the ES-RFR model.

In figure 16, the 3-year forecast displays a highly accurate forecast line, closely aligning with the actual values. With very minimal error, this ES-RFR forecast appears to be the most accurate so far.

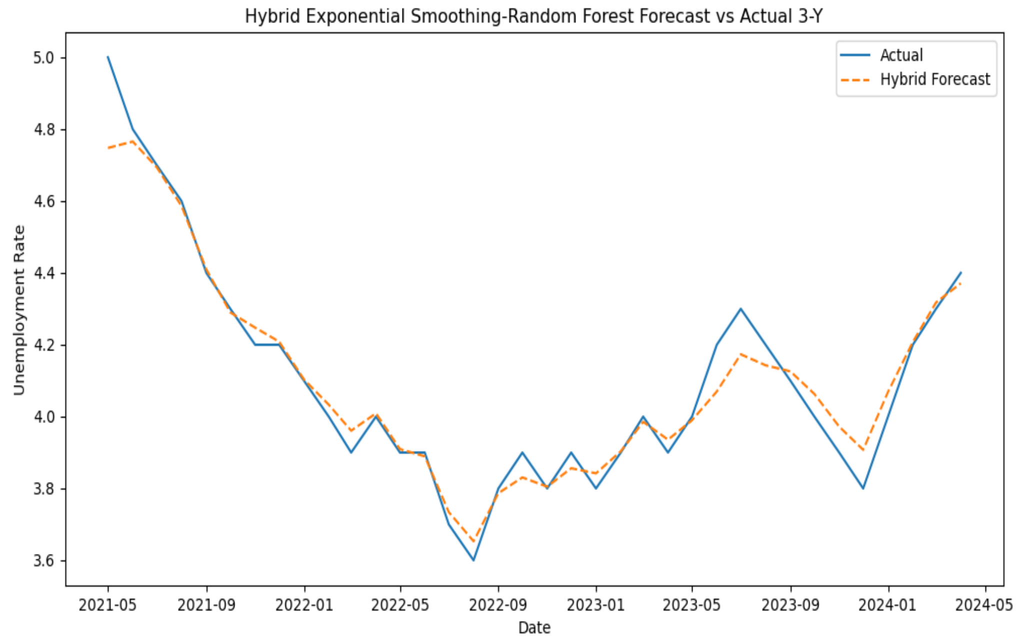

Figure 16: ES-RFR-3Y

For the 1-year model, the RMSE was 0.056, which is slightly higher than the 3-year forecast and ES-SVR 1-year forecast. However, this result is still relatively low and indicates a strong performance in short-term predictions. The MAE is also slightly higher than the 3-year forecast, resulting in 0.046. This remains a low value, indicating that predictions are close to the actual unemployment rates, although there is a small increase in error for the 1-year forecast. The MSE was 0.003, suggesting minimal prediction errors and proving that forecast accuracy is high in the short term. The MAPE for the 1-year forecast was slightly higher than the 3-year forecast at 0.011 but still indicates a low percentage error, confirming the model's reliable performance in capturing short-term trends.

In Figure 17, the forecast line for the 1-year ES-RFR model is similar to that of the ES-SVR model, indicating high accuracy. It appears that the RFR component did not have a significant impact on the forecast accuracy for the 1-year forecast, suggesting that the ES-RFR model's performance is similar to the ES-SVR in this case.

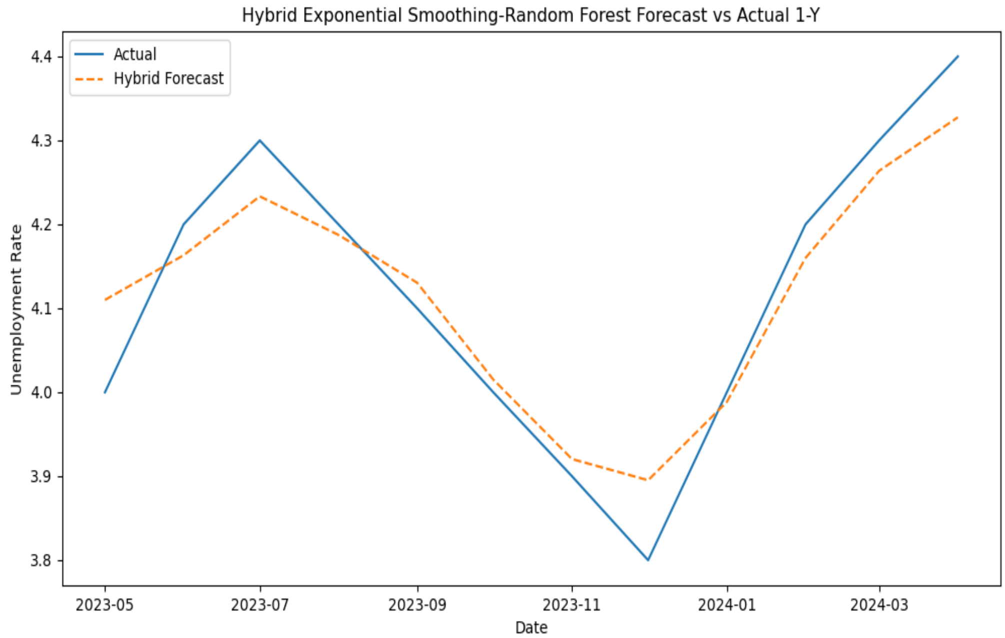

Figure 17: ES-RFR-1Y

The ES-RFR model performs very well in both 3-year and 1-year forecasts, with very similar RMSE and MAE values for both periods. This suggests more consistent forecasting capabilities. However, the 3-year forecast performs slightly better than the 1-year forecast, indicating that the ES-RFR model may be slightly more effective for longer-term predictions using the unemployment data. However, this difference is minimal. These results indicate that the ES-RFR model is reliable across different forecast horizons and appears to be a versatile tool for predicting the unemployment rates.

6.1.5 VAR - LSTM Hybrid Model

The VAR-LSTM model performed reasonably well when considering the performance metrics in both the 3-year and 1-year forecasts. For the 3-year forecast, the RMSE was 0.138, indicating a moderate level of prediction compared to previous models. This suggests that the model struggled to accurately capture the actual unemployment rates over the 3-year period. The MAE further supports this, with a score of 0.116 indicating that the predictions deviated significantly from the actual unemployment rates on average, showing inconsistency in the model's predictions. With an MSE of 0.019, it's clear that there were larger errors in the forecast, highlighting the model's struggles in predicting certain areas of the data. The AIC for the VAR component of the hybrid model was -7.19, significantly higher than that of the previous ARIMA models. This suggests that the AIC model may not be capturing underlying patterns in the data, affecting the model's performance. Since differencing was required before using the VAR model, MAPE could not be calculated.

In Figure 18, the 3-year forecast line shows a general increasing trend, which is in line with the actual unemployment rates. However, the model tends to predict significant events, such as peaks and dips, too early. For example, between May 2022 and September 2022, the model anticipated a peak in unemployment before it actually occurred, resulting in an unreliable forecast. This suggests a lack of alignment in the model's predictions, leading to decreased accuracy.

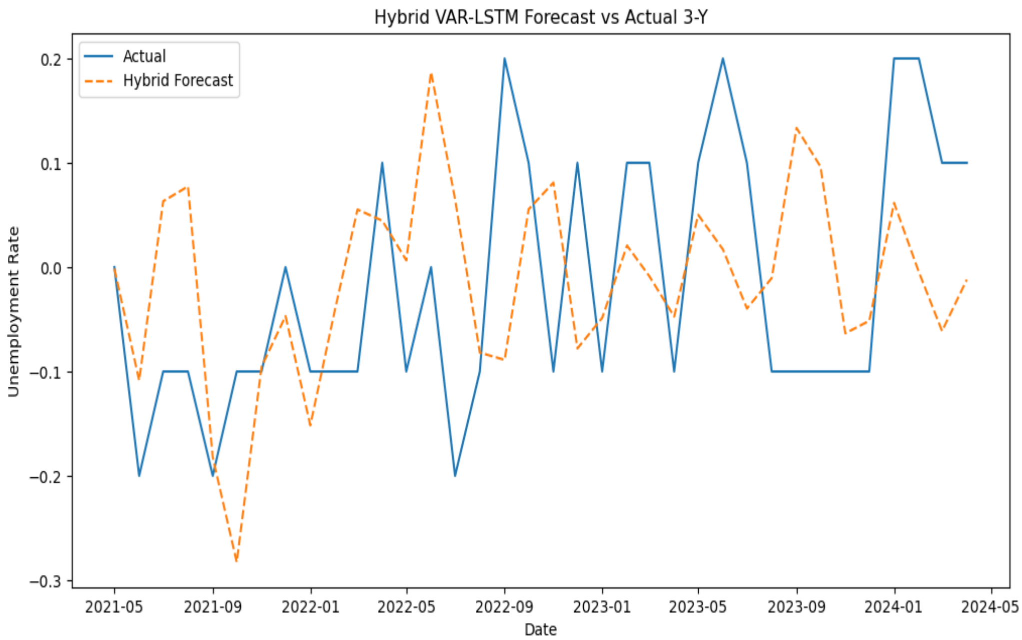

Figure 18: VAR-LSTM-3Y

The RMSE for the 1-year model is slightly lower than the 3-year forecast, at 0.134, suggesting minimal improvement in accuracy for the shorter period. This indicates ongoing challenges for the VAR-LSTM model. The MAE is similar to the 3-year forecast, at 0.018, implying that the model is slightly better at avoiding larger errors, although this may be due to the shorter forecast.

The AIC for the 1-year forecast is slightly higher than the 3-year forecast, at -5.67, showing that the VAR component for this model does not fit the unemployment data very well.

Figure 19 displays the 1-year forecast for the VAR-LSTM model. It is evident that there are several significant inaccuracies. The forecast depicts a downward trend, with unemployment rates sharply increasing after November 2023. This suggests that the VAR-LSTM 1-year model has misinterpreted the actual unemployment rates, highlighting its limitations in capturing short-term trends, particularly in response to sudden changes seen in the actual unemployment rates.

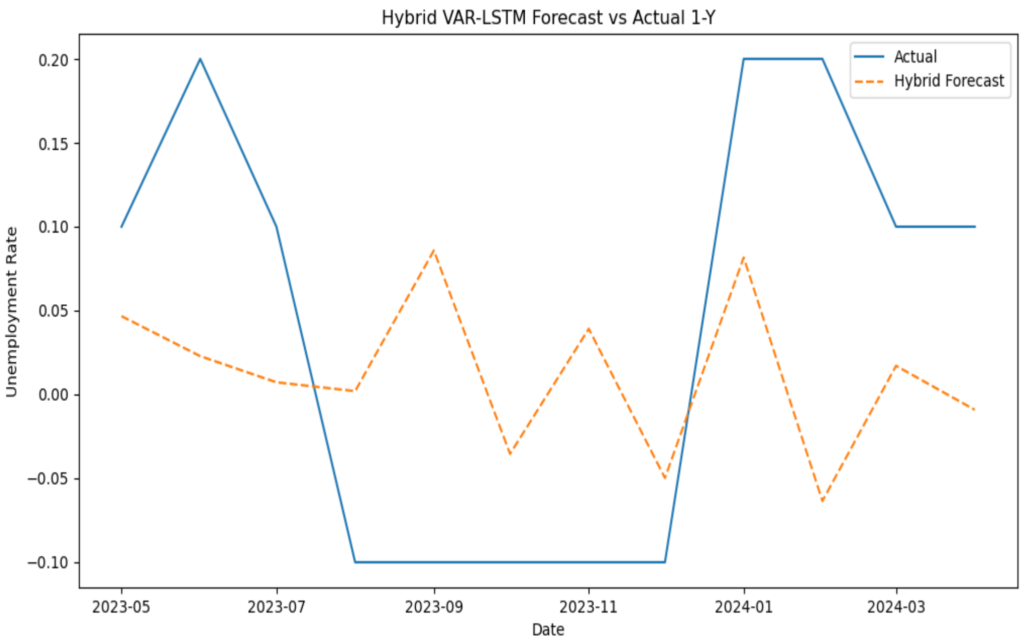

Figure 19: VAR-LSTM-1Y

When comparing the VAR-LSTM model at both horizons, the 1-year forecast shows slight improvements over the 3-year forecast in terms of RMSE and MAE. However, the difference is minimal, indicating that the model faces consistent challenges across both time horizons. Both Figures 20 and 21 indicate that the model does not perform well. The 3-year forecast shows instances of over and underpredictions early on, with key events predicted too early. The data being so volatile after differencing could make it more difficult for the model to produce accurate predictions. The 1-year model forecast demonstrates a even bigger failure, with the forecast line failing to capture the overall trend, leading to a prediction completely disconnected from the actual unemployment rates. This model shows significant limitations, making it clear that further refinement should be made before utilising the VAR-LSTM to improve its accuracy or even staying away from it when using it for unemployment rate data.

6.1.6 VAR - GRU Hybrid Model

The VAR-GRU model did not produce very impressive results. For the 3-year forecast, the AIC was -7.19, which is consistent with the AIC from the previous VAR-LSTM model. This indicates that the model remained stable, with no improvements upon switching to the GRU component. The VAR-GRU model resulted in an RMSE of 0.141, slightly higher than the VAR-LSTM model for the same 3-year period, indicating greater overall error in predictions with the VAR-GRU model. This suggests that the GRU component did not provide any enhancements in longer-term forecasting accuracy. The MAE was 0.115, similar to the VAR-LSTM model, indicating that the average deviation from the actual unemployment rate is almost identical in average forecast accuracy. The MSE also showed little difference from the VAR-LSTM model, reinforcing that the GRU component did not improve the forecast quality.

The forecast line for the 3-year horizon shown in Figure 20 resembles the pattern of the VAR-LSTM model. While the forecast reflects the overall trend of unemployment rates, it doesn't significantly improve the accuracy or timing of predictions. The similarity between the forecast lines of the VAR-GRU and VAR-LSTM models further supports the observation that the machine learning component of this hybrid forecast did not influence the model's performance.

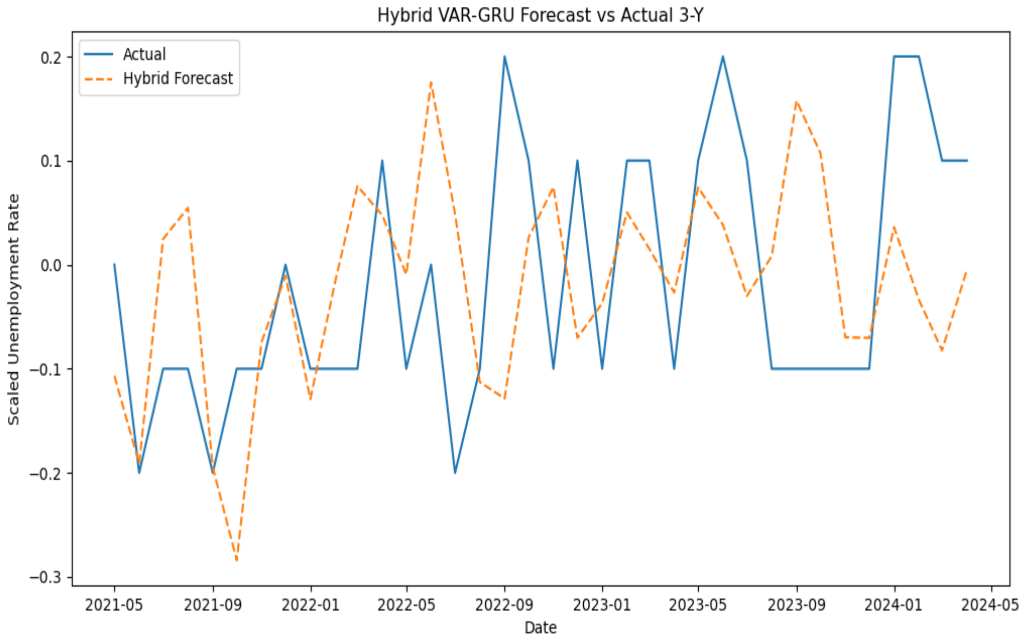

Figure 20: VAR-GRU-3Y

For the 1-year VAR-GRU forecast, the AIC for the VAR component was found to be -5.67. This result is consistent with the VAR-LSTM 1-year forecast model, indicating that the model still does not seem to fit the unemployment data well. The RMSE for the VAR-LSTM 1-year forecast was 0.138, slightly lower than the 3-year forecast but still similar to the VAR-LSTM 1-year forecast. Consequently, the VAR-GRU model does not offer an advantage over the VAR-LSTM in minimising prediction errors for the 1-year forecast. The MAE was 0.122, which is close to the 3-year forecast, suggesting that the average prediction error is higher than the VAR-LSTM model, although not significantly so.

In Figure 21, the forecast line for the 1-year forecast closely resembles that of the VAR-LSTM 1-year forecast line, displaying a similar overall downward trend. Like the VAR-LSTM model, the VAR-GRU model fails to capture the sharp rise after November 2023. The lack of significant variation between the forecasts indicates that switching from LSTM to GRU did not noticeably improve short-term forecast accuracy.

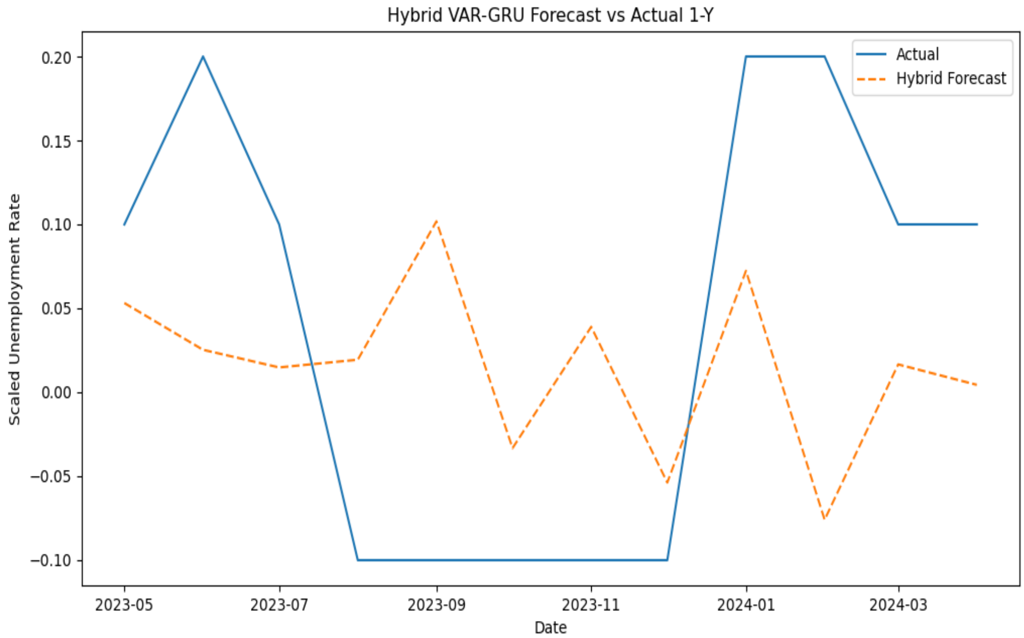

Figure 21: VAR-GRU-1Y

The change from using LSTM to using GRU components in the hybrid model did not result in a significant difference in forecast accuracy. This is evident from the nearly identical performance metrics for both models. It suggests that the LSTM and GRU models performed similarly when combined with VAR for forecasting unemployment rates. The graphs also showed similar trends, with neither model providing a clear advantage in capturing unemployment rate trends. Like the VAR-LSTM model, the VAR-GRU model is not reliable and struggles to capture the volatile trends in the differenced unemployment rate data.

6.1.7 GARCH - RFR Hybrid Model

The GARCH-RFR model performed exceptionally well for both 3-year and 1-year forecasts. For the 3-year forecast, the GARCH component was optimised and terminated after 17 iterations, achieving a final function value of 211, which is a significant improvement from the initial value of 1197.9. This substantial reduction indicates that the model effectively captured the underlying trends and volatility of unemployment rates. The RMSE for the 3-year forecast is the lowest yet at 0.049, indicating a very accurate prediction with minimal error across the entire forecast. The MAE of 0.039 is also very low, suggesting that the average deviation of the forecasted values from the actual unemployment rate is small, emphasising the model's accuracy and reliability in predicting unemployment rates. The MSE is extremely low at 0.002, reinforcing the fact that the model successfully minimized larger errors and highlights a smooth and accurate 3-year forecast.

The 3-year forecast line, as shown in Figure 22, closely matches the actual unemployment rates with minimal deviations from the data. The performance metrics confirm that the GARCH-RFR hybrid model produces highly accurate 3-year forecasts and is suitable for long-term predictions.

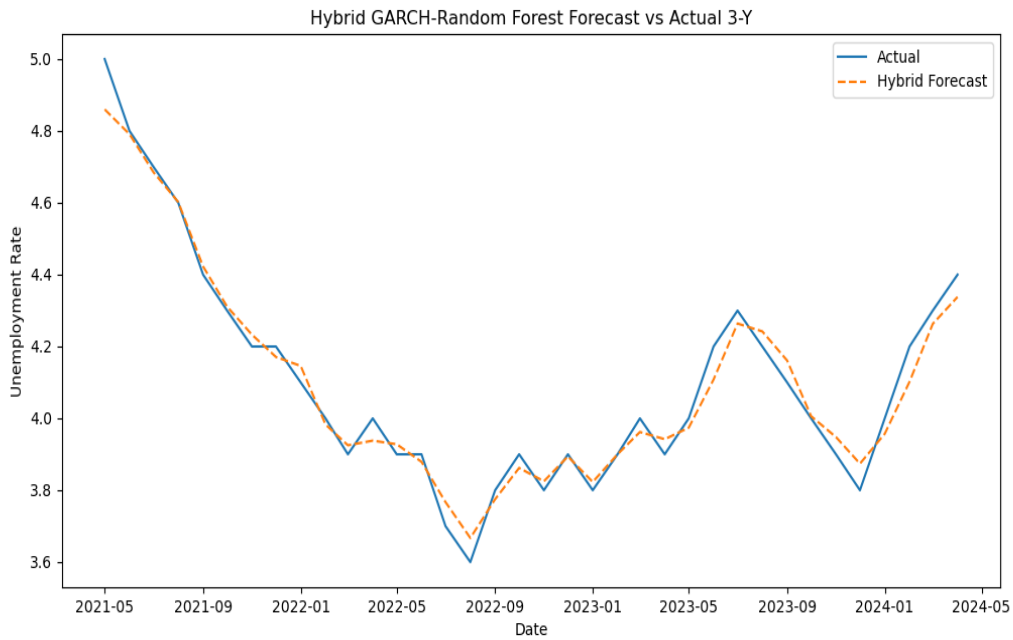

Figure 22: GARCH-RFR-3Y

For the 1-year forecast, the GARCH model was optimised over 21 iterations, resulting in a final function value of 226.63, a significant improvement from the initial value, which was extremely high. This substantial reduction in function value emphasizes the GARCH component's ability to refine its fits for a shorter forecast. The RMSE for the 1-year model is slightly higher than the 3-year forecasts, at a value of 0.057, but it still remains low, indicating the model's continued accurate predictions with minimal errors. The MAE for this model was 0.045, slightly higher than the 3-year forecast, suggesting an increase in the average prediction error. However, this is still a low MAE and indicates that the model offers a very accurate forecast for the 1-year forecast. The MSE resulted in 0.003, remaining low, albeit slightly higher than the 3-year forecast. This suggests that the model maintains a high level of accuracy in predicting percentage changes in unemployment rates.

The 1-year forecast line, shown in figure 23, demonstrates high accuracy and closely aligns with the unemployment rates. However, there is a slight deviation from the dip in December 2024, which appears to be a common issue that most models are experiencing. The graph also shows minimal errors and confirms that the GARCH-RFR model is very effective in short-term forecasting. Its prediction line successfully captures the fluctuations in the unemployment data.

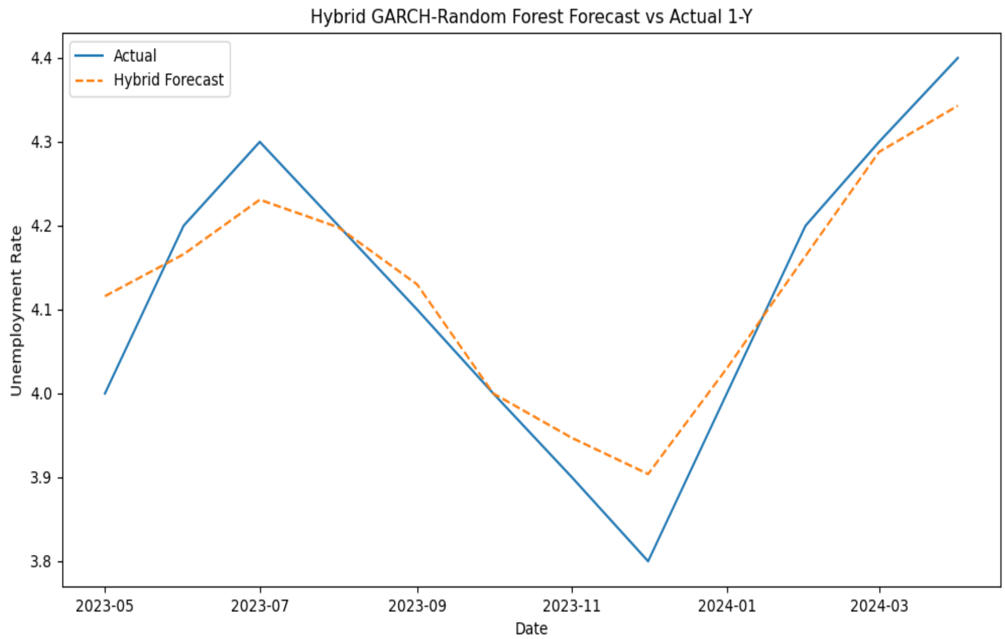

Figure 23: GARCH-RFR-1Y

The GARCH-RFR hybrid model has demonstrated its effectiveness in long-term forecasting, as reflected in the consistently low performance metrics for the 3-year forecast. This model's capability to capture volatility and the non-linear relationships present in the data makes it highly suitable for long-term predictions. Although the 1-year forecast displays a slight improvement in performance metrics compared to the 3-year forecast, the variances are minimal. The model continues to deliver strong performance, yielding accurate forecasts with minimal errors. Furthermore, the substantial reduction in function values observed in the GARCH components during the optimisation process underscores the model's robustness and its ability to optimise for various forecasting periods. Consequently, this model is a reliable option for both long-term and short-term forecasting.

6.1.8 GARCH - SVR Hybrid Model

The GARCH-SVR model produced impressive results for both the 3-year and 1-year forecasts. For the 3-year forecast, the GARCH component of the model was optimised after 17 iterations, achieving a function value of 211.8. This result aligns with findings from other hybrid models like GARCH-RF, indicating that the GARCH model fits the data well and accurately captures unemployment rate trends and volatility. The RMSE for the 3-year forecast was 0.084, suggesting that the GARCH-SVR model provides an accurate forecast. This value is slightly higher than other high-performing models such as GARCH-RFR and ES-RFR, indicating that the SVR component does not significantly impact the accuracy of the 3-year forecast. The MAE was 0.065, indicating a low average forecasting error and overall good performance for this horizon. The MSE was 0.007, indicating few large deviations from the actual unemployment rates. The MAPE was 0.016, showing that the model's predictions are close to the actual unemployment rates, although this error is slightly higher compared to some other developed hybrid models.

The forecast line in Figure 24 represents the 3-year forecast for GARCH-SVR. This forecast line is accurate, but it deviates slightly more from the actual unemployment rates compared to the ES-RFR and GARCH-RFR 3-year forecasts. This indicates that the model is effective, but it may not capture long-term trends as precisely as the other top-performing models.

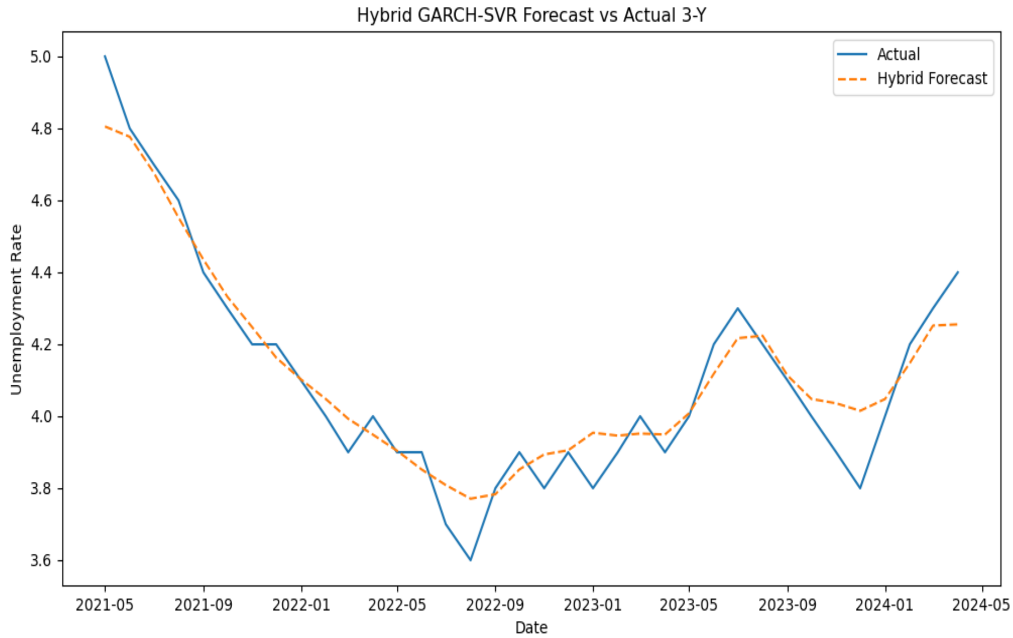

Figure 24: GARCH-SVR-3Y

For the 1-year forecast, the GARCH component was optimised with a function value of 226.63 after 21 iterations, which is consistent with the previous GARCH-RFR models for the 1-year forecast. The RMSE for the GARCH-SVR 1-year forecast was 0.038, indicating very low error and suggesting that the GARCH-SVR model performs very well over the 1-year period. This result also indicates that the GARCH-RFR model is more accurate during the 1-year forecast compared to the 3-year forecast. The MAE is also low, showing the model's ability to predict closely to the unemployment rate data. The MSE resulted in 0.001, showing minimal larger errors, making the model highly reliable for short-term unemployment predictions. The MAPE resulted in 0.007, which reinforces the exceptional accuracy of the GARCH-SVR model for short-term forecasting.

Figure 25 shows the GARCH-SVR 1-year forecast line. The forecast closely matches the actual unemployment rates with minimal room for error, indicating that the GARCH-SVR excels in short-term forecasting. The model accurately captures trends and fluctuations, with only a small error where the dip occurs, which has been a common issue with other models.

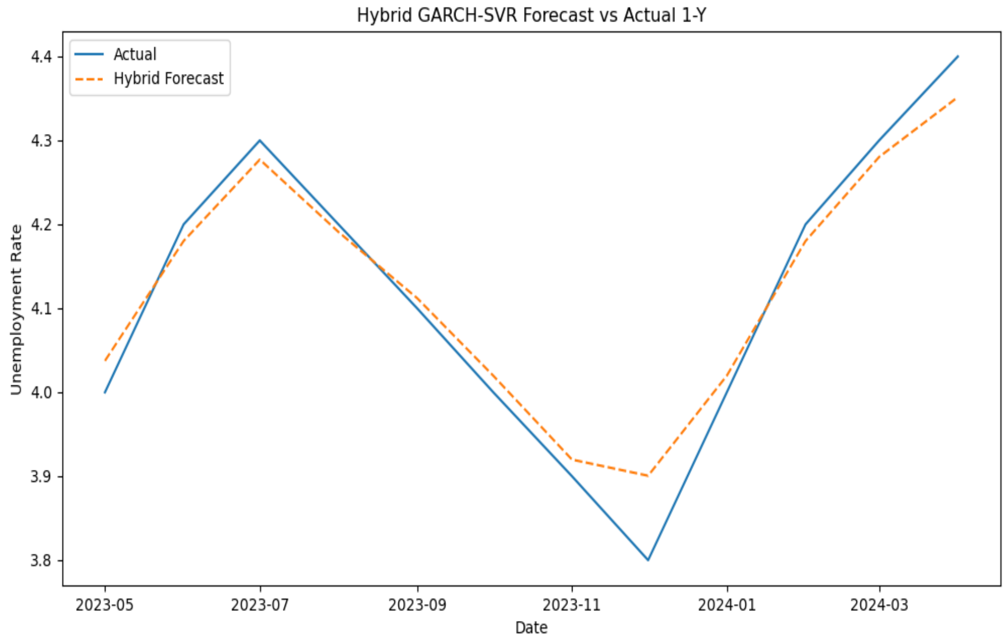

Figure 25: GARCH-SVR-1Y

The GARCH-SVR hybrid model is very accurate over the 3-year forecast, but has slightly more errors compared to previously high-performing models such as ES-RFR and GARCH-RF. This suggests that the SVR component is powerful for making predictions; however, it may not provide a significant advantage over longer-term predictions when paired with GARCH. The GARCH-SVR model demonstrated high performance over the 1-year forecast with low errors across all metrics. This model proved its suitability for short-term trends by effectively capturing trends and variations in the 1-year forecast. It is evident that the GARCH-SVR model is reliable, mainly in short-term forecasts. However, its slight underperformance in the 3-year forecast compared to other top performers suggests that the SVR component could be more suitable for short-term forecasting.

6.1.9 ARIMAX-LSTM Hybrid Model

The ARIMAX-LSTM 3-year forecast model produced an AIC of -301.627, which is close to that of the AIC from the previous ARIMA-LSTM model. The similarity of the ARIMA fits suggests that the ARIMAX-LSTM will perform similarly to the ARIMA-LSTM model. However, the LSTM component plays a crucial role in determining forecast accuracy.

The RMSE for the ARIMAX-LSTM 3-year model resulted in 0.213, which is higher than previous forecasts. This indicates larger deviations between the forecast and the actual unemployment rates, suggesting that the model struggles with the complexity of the unemployment data over this longer period. The MSE resulted in 0.175, supporting the observation that the predictions deviate significantly from the actual unemployment rates. Additionally, the MSE resulted in 0.046, highlighting the presence of larger errors in this forecast. The MAPE resulted in 0.042, indicating the model's accuracy compared to the other models.

In Figure 26, the ARIMAX-LSTM model is displayed for the 3-year forecast. The forecast line is similar to the ARIMA-LSTM model; however, it appears to be less smooth. This suggests that the model may be struggling with the volatility of the data, leading to less reliable predictions.

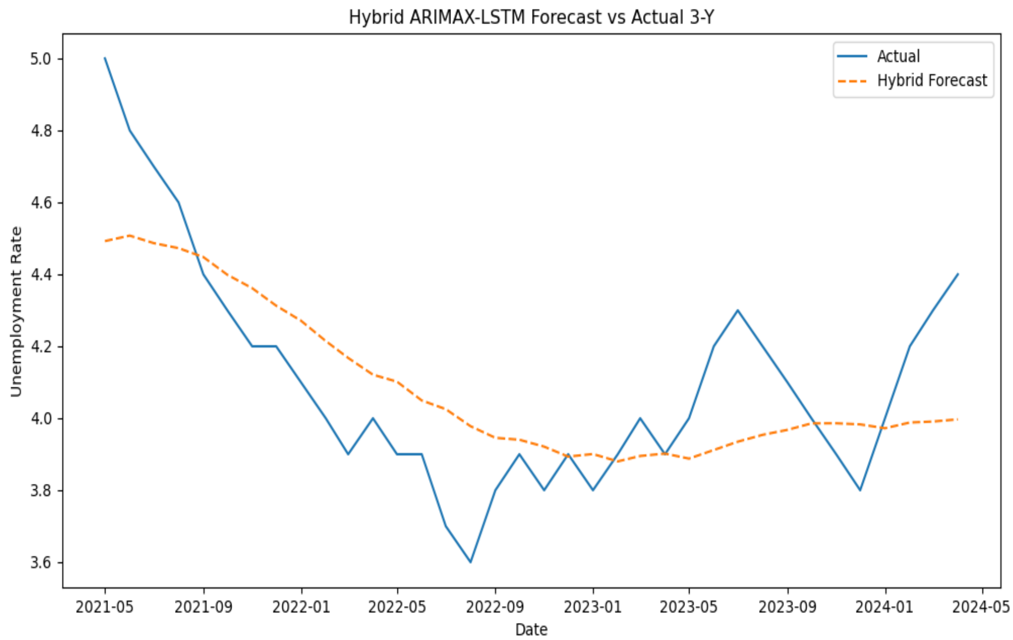

Figure 26: ARIMAX-LSTM-3Y

For the 1-year forecast, the ARIMAX component produced an AIC of -339.217, indicating a better fit than that of the 3-year forecast. This suggests that this model could be better suited for the 1-year forecast than the 3-year forecast. The RMSE for the 1-year forecast resulted in 0.180, indicating improved accuracy for this prediction.

The MAE also resulted in a lower value of 0.155, showing that, on average, the predictions were closer to the actual unemployment rates. The MSE resulted in 0.032, further supporting the reduced errors for the 1-year forecast. The MAPE resulted in 0.038, which is slightly better than the 3-year forecast, confirming the improved accuracy for the 1-year forecast.

Figure 27 displays the 1-year forecast line for ARIMAX-LSTM. The forecasted line indicates a gradual linear upward trend. Upon visual inspection, the forecast line may appear less accurate than the 3-year forecast; however, based on the performance metrics, this is not the case.

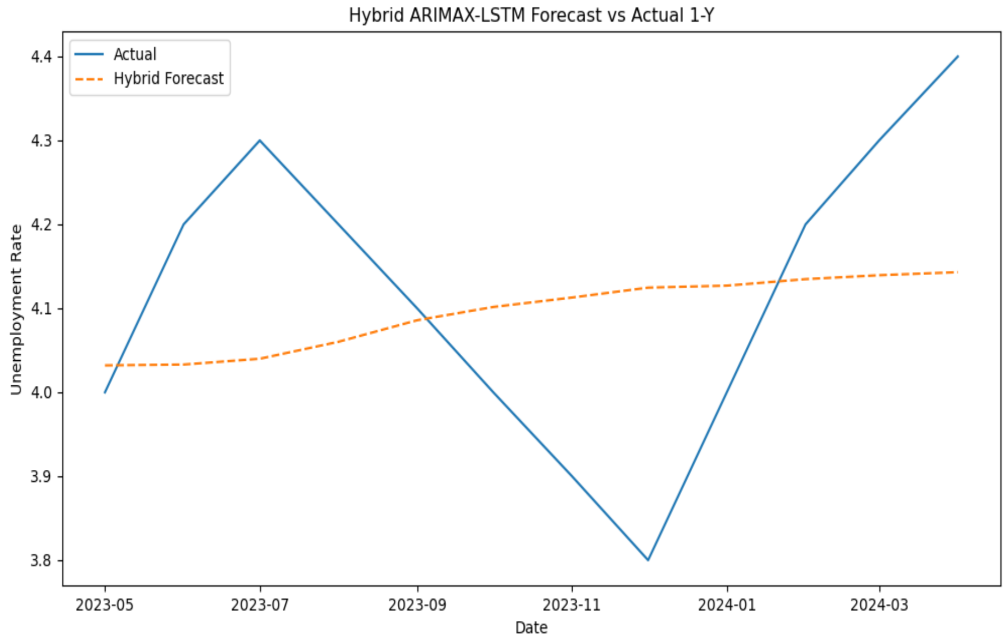

Figure 27: ARIMAX-LSTM-1Y

The 3-year ARIMAX-LSTM model performs noticeably worse than the 1-year forecast, as evidenced by the performance metrics. This suggests that the model's ability to capture longer-term forecasts for the unemployment rate is limited, possibly due to the multivariate nature of the ARIMAX model.

When compared to other multivariate models like the VAR-based hybrid models, the ARIMAX-LSTM model does not perform well. The higher performance metrics and less smooth 3-year forecast line suggest that the ARIMA-LSTM model struggles to match the accuracy and reliability of other models.

6.2 Discussion and Recommendations

3-Y MODELS	RMSE	MAE	MSE	MAPE
ARIMA-LSTM	0.225	0.193	0.051	0.047
ARIMA-GRU	0.17	0.139	0.029	0.034
ES-SVR	0.132	0.103	0.017	0.025
ES-RFR	**0.065**	**0.043**	**0.004**	**0.01**
VAR-LSTM	0.138	0.116	0.019	....
VAR-GRU	0.141	0.115	0.022	....
GARCH-RFR	**0.049**	**0.039**	**0.002**	**0.01**
GARCH-SVR	0.084	0.065	0.007	0.016
ARIMAX-LSTM	0.213	0.175	0.046	0.042

Table 6.1: Performance metrics for 3-Y models

1-Y MODELS	RMSE	MAE	MSE	MAPE
ARIMA-LSTM	0.182	0.157	0.033	0.038
ARIMA-GRU	0.184	0.161	0.034	0.039
ES-SVR	**0.054**	**0.043**	**0.003**	**0.011**
ES-RFR	**0.056**	**0.046**	**0.003**	**0.011**
VAR-LSTM	0.134	0.12	0.018	....
VAR-GRU	0.138	0.122	0.019	....
GARCH-RFR	**0.057**	**0.045**	**0.003**	**0.011**
GARCH-SVR	**0.038**	**0.029**	**0.001**	**0.007**
ARIMAX-LSTM	0.18	0.155	0.032	0.038

Table 6.2: Performance metrics for 1-Y models

Tables 6.1 and 6.2 display the results of testing Hybrid models at 3-year and 1-year horizons respectively. Conditional formatting was used in an Excel spreadsheet to highlight lower values, which indicate better fitting models. Darker red indicates a better-fitting model, while lighter red or white indicates a less well-fitting model.

For the 3-year horizon, it is clear that the ES-RFR and GARCH-RFR perform very well compared to other models with impressive performance metrics and visualisations.

Among these two models, the GARCH-RFR performed the best overall during this horizon, with the worst-performing model being the ARIMA-LSTM. The ARIMA-LSTM model performed considerably low.

In the 1-year forecast, four models stood out for their exceptional performance: ES-SVR, ES-RFR, GARCH-RFR, and GARCH-SVR. Among these models, GARCH-SVR performed the best, while the ARIMA-GRU model performed the worst overall. It is worth noting that hybrid models incorporating SVR showed significantly better performance in the 1-year forecast. ES-RFR performed with little variation, and GARCH-RFR showed slightly worse performance in the 1-year forecast.

The consistent performance of ES-RFR across both forecast horizons suggests that it is the most reliable model developed in this study, as other models showed varying performance. For example, GARCH-SVR performed substantially worse in the 3-year forecast compared to the 1-year forecast performance.

During the 3-year forecast, the VAR-LSTM model performed the best out of the two multivariate models, while the ARIMAX-LSTM model performed the worst. However, for the 1-year forecast, the VAR-GRU model was the most successful. Therefore, there is not necessarily a more favourable multivariate method. The results show that the choice of model should depend on the length of the time period.

These results demonstrate that certain models perform slightly better than others depending on the time horizon. This information is valuable for economists and data scientists when choosing which hybrid model to use. Since there are numerous combinations of hybrid models, there may not be a single best one; the choice depends on the specific purpose.

After conducting a thorough evaluation of various hybrid models, the GARCH-RFR model is recommended for use due to its exceptional performance and its alignment with the study's objective of identifying suitable hybrid models for predicting UK unemployment rates. The GARCH-RFR model exhibited the highest accuracy for the 3-year forecast, with a RMSE of 0.049, indicating its strong predictive ability.

However, for the 1-year forecast, the GARCH-RFR model's performance was similar to that of the high achieving ES-RFR model further proving the GARCH-RFR models stability. It is also apparent from the 3-year forecast line in Figure 22 that the GARCH-

RFR model was able to closely predict the dip after September 2023, a challenge that many alternative models struggled with.

By using the GARCH-RFR model to predict future UK unemployment rates, it is expected that users will experience improved forecasting compared to using individual econometric methods, resulting in more accurate and reliable predictions.

Furthermore, a Kolmogorov-Smirnov (K-S) test was performed on the forecasted unemployment rates compared to the actual values. However, due to the limited range of the UK unemployment data (from 5.0 to 3.6), the test consistently yielded low p-values, irrespective of the model's performance. For example, the poorly performing ARIMA-LSTM 3-year forecast resulted in a P-value of 0.03. On the other hand, the 3-year GARCH-RFR model, which performed extremely well, resulted in a P-value of 7.83E-08. This indicates that if a 5% significance level was used, both models would be considered highly accurate, which is not true. As a result, this test did not provide any meaningful insights into the goodness of fit for each model.

Chapter 7

Conclusions

This study set out to explore the accuracy of hybrid forecasting models in predicting time series data, focusing on UK unemployment rates. The objective was to develop various hybrid models by combining well-known econometric methods with suitable machine-learning techniques for regression tasks.

The results show that there is a wide variation in the accuracy of the hybrid models that were developed. It is important to note that the models that used the Random Forest Regressor tended to be more accurate, while those that used LSTM networks were less reliable and could only capture the general trends of the unemployment data.

Interestingly, the choice of machine learning model sometimes significantly influenced the results, while at other times, it had little effect. Similarly, the impact of econometric method varied, complicating the process of identifying the most effective model combinations. For example, there was a noticeable difference in performance between the ES-RFR and ES-SVR models, whereas the VAR-LSTM and VAR-GRU models showed similar accuracy levels. These results highlight the challenges of choosing the most accurate models, particularly when dealing with unemployment data. Previous studies have shown that the accuracy of hybrid models can significantly differ across datasets from various countries.

The study faced limitations, notably the reduction of the unemployment dataset to match the size of the GDP dataset for the use of econometric multivariate models.

This constraint reduced the amount of data available for the training models. It is believed that with more data, the accuracy of LSTM models could improve, given that these more complex models require larger datasets for effective training due to their numerous parameters. The narrow range of the unemployment data made it difficult to conduct an effective goodness of fit test. Therefore, the only way to evaluate the model's performance was through performance metrics. To address this, a wider range of data with a higher range could be used. Had the data not been reduced, the results from the K-S test might have provided valuable insights.

Further research could build on this study by investigating different combinations of methods and developing a list of the best-performing models for various forecasting purposes. Such as a resource would be valuable for data scientists and economists, given the varying accuracy of models depending on the forecasting context. Additionally, further exploration of the GARCH-RFR model, which was found to be the most accurate, could involve testing its performance on unemployment data from different countries.

Moreover, hyperparameter tuning of more complex models like LSTM could be pursued to improve their predictive accuracy, as these models showed the lowest performance in this study. To address the main limitation identified in this research, future studies might consider dropping multivariate methods in favour of using a longer-term unemployment dataset. This approach could potentially improve the accuracy of LSTM and GRU models, which may benefit from the increased amount of training data.

In conclusion, this dissertation highlights the complexities involved in selecting the correct models for predicting unemployment rates. The variability in model performance, influenced by both econometric and machine learning methods, underscores the need for further research to better understand and optimise hybrid forecasting models for time series data.

CHAPTER 8

Appendix

```
1.      Loading Software Packages
# Basic Packages
import numpy as np
import pandas as pd
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast\_node\_interactivity = 'all'

# Machine learning
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, GRU
from scikeras.wrappers import KerasClassifier, KerasRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error,
    mean_absolute_percentage_error
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from arch import arch_model

# Statistics
```

```
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import adfuller
from scipy.stats import chisquare
from scipy import stats
from statsmodels.graphics.tsaplots import plot_acf
from pandas.plotting import lag_plot
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import kpss
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_pacf

# Visualisation
from matplotlib import pyplot
import matplotlib.pyplot as plt
import seaborn as sns

2.      Loading Datasets
# Reading in CSV files
gdp = pd.read_csv("C:/Users/maxsh/OneDrive/Desktop/University of
    Essex/Msc applied data science/Final
    Project/data/Figure_1__UK_GDP_is_estimated_to_have_shown_no_growth_in_April_2
unemployment =
    pd.read_csv("C:/Users/maxsh/OneDrive/Desktop/University of
    Essex/Msc applied data science/Final
    Project/data/statistic_id279898_unemployment-rate-of-the-uk-1971-2024.csv")
inflation = pd.read_csv("C:/Users/maxsh/OneDrive/Desktop/University
    of Essex/Msc applied data science/Final
    Project/data/statistic_id306648_inflation-rate-in-the-uk-1989-2024.csv")

3.      Data Processing
# Dropping unnecessary rows from dataframes
gdp = gdp.drop([0, 1, 2, 3, 4, 5])
```

```
unemployment = unemployment.drop([0, 1])
inflation = inflation.drop([0, 1])

# Drop unnecessary columns
unemployment = unemployment.drop('Unnamed: 2', axis = 1)
inflation = inflation.drop('Unnamed: 2', axis = 1)

# Renaming columns
gdp = gdp.rename(columns={'Figure 1: UK GDP is estimated to have
    shown no growth in April 2024, but grew by 0.7% in the three
    months to April 2024 ': 'Date'})
gdp = gdp.rename(columns={'Unnamed: 1': 'Monthly GDP'})
unemployment = unemployment.rename(columns={'Unemployment rate of
    the UK 1971-2024': 'Date'})
unemployment = unemployment.rename(columns={'Unnamed: 1':
    'UnemploymentRate'})
inflation = inflation.rename(columns={'Inflation rate in the UK
    1989-2024': 'Date'})
inflation = inflation.rename(columns={'Unnamed: 1': 'InflationRate'})

#Fitting the Unemployment and inflation data with the GDP data by
    reducing their size
unemployment = unemployment.drop(unemployment.index[0:430])
inflation = inflation.drop(inflation.index[0:216])
inflation = inflation.drop(426)

# Merging Data on the Date column which I previously made sure
    matched up within each dataset.
indicators_merged = pd.merge(gdp, unemployment, on='Date')
indicators_merged = pd.merge(indicators_merged, inflation, on='Date')
print(indicators_merged)

4.      Descriptive Statistics
# Descriptive statistics
```

```
mean = unemployment['UnemploymentRate'].mean() # Calculating Mean UnemploymentRate
median = unemployment['UnemploymentRate'].median() # Calculating Median UnemploymentRate
maximum = unemployment['UnemploymentRate'].max() # Calculating The Maximum UnemploymentRate
minimum = unemployment['UnemploymentRate'].min() # Calculating The Minimum UnemploymentRate
std_dev = unemployment['UnemploymentRate'].std() # Calculating The Standard Deviation For UnemploymentRate
skewness = unemployment['UnemploymentRate'].skew() # Calculating The UnemploymentRate Skewness

# Performing the Standard ADF Test//Checking for Stationarity
adf_level_result = adfuller(unemployment['UnemploymentRate'])
adf_level_stat = adf_level_result[0]
adf_level_p_value = adf_level_result[1]

# Performing the ADF Test At The First Order Difference.
# To see if the result changes from the previous ADF Test.
adf_diff_result = adfuller(unemployment['UnemploymentRate'].diff().dropna())
adf_diff_stat = adf_diff_result[0]
adf_diff_p_value = adf_diff_result[1]

# KPSS test for further stationarity evidence
stat, p, lags, crit = kpss(unemployment['UnemploymentRate'])

# Printing the Results to Create a Table for the Write Up.
print("Descriptive Statistics")
print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Maximum: {maximum}")
print(f"Minimum: {minimum}")
```

```
print(f"Std. Dev.: {std_dev}")
print(f"Skewness: {skewness}")

print("\nADF Test - Level")
print(f"ADF Statistic: {adf_level_stat}")
print(f"P-Value: {adf_level_p_value}")

print("\nADF Test - First Difference")
print(f"ADF Statistic: {adf_diff_stat}")
print(f"P-value: {adf_diff_p_value}")

print("\nKPSS Test")
print(f"KPSS Statistic: {stat}")
print(f"P-value: {p}")

5.      Data Exploration

# Creating 3 Separate line graphs to show the original time series data.

# Original UnemploymentRate Time Series
plt.plot('Date', 'UnemploymentRate', data = unemployment, color = 'red')
plt.title('UK Unemployment Rate Over Time')
plt.xlabel('Date')
plt.ylabel('Overall unemployment rate')
plt.xticks(ticks=range(0, 208, 50)) # Reducing the number of figures present on the X-Axis
plt.grid()
plt.savefig('Original_UnemploymentRate.png') # Saving visualisation as a png file
plt.show();

# Original InflationRate Time Series
```

```
plt.plot('Date', 'InflationRate', data = inflation, color = 'blue')
plt.title('UK Inflation Rate Over Time')
plt.xlabel('Date')
plt.ylabel('Overall inflation rate')
plt.xticks(ticks=range(0, 208, 50)) # Reducing the number of figures
    present on the X-Axis
plt.grid()
plt.savefig('Original_InflationRate.png') # Saving visualisation as
    a png file
plt.show();


# Original Monthly GDP Time Series
plt.plot('Date', 'Monthly GDP', data = gdp, color = 'green')
plt.title('UK GDP Over Time')
plt.xlabel('Date')
plt.ylabel('Overall GDP')
plt.xticks(ticks=range(0, 208, 50)) # Reducing the number of figures
    present on the X-Axis
plt.yticks(ticks=range(0, 150, 20)) # Reducing the number of figures
    present on the Y-Axis
plt.grid()
plt.savefig('Original_GDP.png') # Saving visualisation as a png file
plt.show();

# Economic Indicator Boxplots
# Changing the GDP data from float to numeric to avoid the Y-axis
    being full of figures.
gdp_numeric = pd.to_numeric(gdp['Monthly GDP'])

# GDP Boxplot
plt.subplot(1, 3, 1)
sns.boxplot(y=gdp_numeric, color = 'limegreen')
plt.title('GDP Boxplot')
```

```

# Inflation Boxplot
plt.subplot(1, 3, 2)
sns.boxplot(y=inflation['InflationRate'], color = 'blue')
plt.title('Inflation Boxplot')


# Unemployment Boxplot
plt.subplot(1, 3, 3)
sns.boxplot(y=unemployment['UnemploymentRate'], color = 'red')
plt.title('Unemployment Boxplot')

# Display the plots
plt.tight_layout()
plt.savefig('Boxplots.png') # Sacing the Boxplots as a png file.
plt.show();

# Converting Dates into datetime format.
# Changing dates into number format to allow for model fit.
# And to allow for the rest of the EDA plots to function.
unemployment['Date'] = pd.to_datetime(unemployment['Date'],
    format='%b-%y')
inflation['Date'] = pd.to_datetime(inflation['Date'], format='%b-%y')
gdp['Date'] = pd.to_datetime(gdp['Date'], format='%b-%y')
indicators_merged['Date'] =
    pd.to_datetime(indicators_merged['Date'], format='%b-%y')

## Correlation Heatmap.
sns.heatmap(indicators_merged.corr(), annot=True, cmap='coolwarm') #
    Specifying the colour, and box annotations.
plt.title('Correlation Heatmap')
plt.savefig('Heatmap.png') # Saving the correlation heatmap as a png
    file.
plt.show();
```

```
# Plotting Seasonal Decomposition of Unemployment
unemployment.set_index('Date', inplace=True) # Setting the Date to Index

# Using seasonal_decompose to decompose into the three components
decompose = seasonal_decompose(unemployment['UnemploymentRate'])

# Original time series
plt.subplot(411) # Adding to the main plot
plt.plot(unemployment['UnemploymentRate'], color = 'red')
plt.title('Unemployment Rate')

# Trend Component
plt.subplot(412) # Adding to the main plot
plt.plot(decompose.trend, color = 'red') # Specifying for the trend component
plt.title('Unemployment Rate')
plt.ylabel('Trend')

# Seasonal Component
plt.subplot(413) # Adding to the main plot
plt.plot(decompose.seasonal, color = 'red') # Specifying for the seasonal component
plt.title('Unemployment Rate')
plt.ylabel('Seasonal')

# Residual Component
plt.subplot(414) # Adding to the main plot
plt.plot(decompose.resid, color = 'red') # Specifying for the residual component
plt.title('Unemployment Rate')
plt.ylabel('Resid')
```

```
plt.suptitle('Seasonal Decomposition of Unemployment Rate', fontsize=16) # Adding Plotting Title.

plt.tight_layout() # Making sure the plot titles fit the graph appropriately
plt.savefig('Seasonal_Decomposition.png') # Saving as a png file
plt.show();

# Creating a ACF plot for the UnemploymentRate
# This will be used to determine the order for both ARIMA and GARCH models
plot_acf(unemployment['UnemploymentRate'], lags=40) # Specifying the number of lags.
plt.xlabel('Lags')
plt.ylabel('Autocorrelation')
plt.title('Autocorrelation Plotting of Unemployment Rate')
plt.savefig('ACF_plot.png'); # Saving the plot as a png file.

# Creating a PACF plot for the UnemploymentRate
# This will be used to determine the order for both ARIMA and GARCH models
plot_pacf(indicators_merged['UnemploymentRate'], lags=40) #Specifying the number of lags.
plt.title('Partial Autocorrelation Plotting of Unemployment Rate')
plt.xlabel('Lags')
plt.ylabel('Partial Autocorrelation')
plt.savefig('PACF_plot.png'); # Saving the plot as a png file.

6.      Hybrid Models
# Creating a differenced unemployment variable for the VAR model.
# Adding the Differenced data to the merged dataset.
indicators_merged['Unemployment_Diff'] = indicators_merged['UnemploymentRate'].diff()
indicators_merged['Unemployment_Diff'].fillna(0, inplace=True)
```

```
# Creating lagged features to use for the multivariate models such
    as ARIMAX and VAR.
indicators_merged['gdp_lag'] = indicators_merged['Monthly
    GDP'].shift(1).fillna(0)
indicators_merged['inflation_lag'] =
    indicators_merged['InflationRate'].shift(1).fillna(0)
indicators_merged['unemployment_lag'] =
    indicators_merged['Unemployment_Diff'].shift(1).fillna(0)

# Setting the Date into the index.
indicators_merged.set_index('Date', inplace=True)

# Changing the frequency of the time series data so that the models
    can expect the data to occur monthly.
unemployment = unemployment.asfreq('MS')
indicators_merged = indicators_merged.asfreq('MS')

LSTM + ARIMA 3-Year Forecast
# Split data into train and test sets 3-Year Forecast
forecast_period = 4 * 12  # Due to the use of a timestep of 12
    months, the forecast period is adjusted.

# Here, the forecast period is subtracted from the length of the
    dataset to create the training size.
train_size = len(unemployment) - forecast_period
train, test = unemployment[:train_size], unemployment[train_size:] #
    training and test data is then split.

target = 'UnemploymentRate' # Creating a target object to hold the
    UnemploymentRate.

# Fitting the ARIMA model
```

```
arima_order = (2, 1, 0) # Setting the ARIMA order, refer to methodology section.
arima_model = ARIMA(train[target], order=arima_order) # Adding the UnemploymentRate to the ARIMA model. Along with the models order.
arima_fit = arima_model.fit() # Fitting the ARIMA model.
print(arima_fit.summary()) # Showing summary results. Such as AIC.

# Forecast and calculate residuals
arima_forecast = arima_fit.forecast(steps=len(test)) # Creates the ARIMA forecast, and specifying the length of the forecast.
residuals = test[target] - arima_forecast # Calculating ARIMA residuals for the LSTM model.
# Normalize residuals
scaler_residuals = MinMaxScaler(feature_range=(0, 1)) # Creating a scaler using a scale from 0 to 1.
# Fits the scaler and transforms the data into a 2D array (The LSTM expects a 2D array).
scaled_residuals = scaler_residuals.fit_transform(residuals.values.reshape(-1, 1))

# Prepare the data for LSTM
time_step = 12 # Setting the time step to 12, so the LSTM uses 12 months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to create new lists suitable for the LSTM model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate over scaled residuals from the time step 12.
    X.append(scaled_residuals[i-time_step:i, 0]) # For each postion i, extract a sequence of 12 time steps to use.
    y.append(scaled_residuals[i, 0]) # Appends value of i to the list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed into NumPy arrays which is needed for the LSTM model.
```

```

# The forecast dates were not displaying correctly when plotting the results.
# Therefore, the dates were extracted separately and then added to the plot.
test_dates = unemployment.index[train_size:]

# Defining the LSTM model with 4 layers.
model = Sequential() # Sequential model
model.add(LSTM(10, return_sequences=True, input_shape=(time_step, 1))) # Add 10 units, returns full sequence, and specifies shape of input data.
model.add(LSTM(10, return_sequences=False)) # Adds another 10 units, only returns the output of the last step.
model.add(Dense(5)) # Adding a dense layer of 5 units, reducing the output to 5 units.
model.add(Dense(1)) # Adding a dense layer with 1 unit, produces a single value as output.
model.compile(optimizer='adam', loss='mean_squared_error') # Compiles the model with the adam optimiser, and MSE as the loss function.

print(model.summary())

# Trainingging the model. Fitting the data to the model.
model.fit(X, y, epochs=50, batch_size=32, verbose=1)

# Clearing session to start fresh.
tf.keras.backend.clear_session()

# Predicting residuals using LSTM model
# Making sure prediction and training shapes are the same
predicted_residuals = model.predict(X)
```

```python
LSTM_predicted_residuals = scaler_residuals.inverse_transform(predicted_residuals)

# Aligninging the lengths of the ARIMA forecast and predicted residuals
arima_forecast = arima_forecast[time_step:]

# Combining the ARIMA forecast and LSTM predicted residuals
actual_unemployment = test[target][time_step:] # Specifying the actual unemployment rate for the plot.
ARIMA_LSTM_3Y_forecast = arima_forecast.values + LSTM_predicted_residuals.flatten() # Combining both forescasts

# Plotting results
plt.figure(figsize=(12, 6)) # Setting the figure size
plt.plot(test_dates[time_step:], actual_unemployment, label='Actual') # Plotting thr actual unemployment rate line
plt.plot(test_dates[time_step:], ARIMA_LSTM_3Y_forecast, label='Hybrid Forecast', linestyle='--') # plotting the hybrid forecast line
plt.xlabel('Date') # Adding Date label to X-axis
plt.ylabel('Unemployment Rate') # Adding Unemployment rate label to y-axis
plt.title('Hybrid ARIMA-LSTM Forecast vs Actual 3-Y') # Adding main title
plt.legend() # Adding legend
plt.savefig('ARIMA-LSTM-3Y.png') # Saving the plot as a png file.
plt.show(); # Displaying plot

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment, ARIMA_LSTM_3Y_forecast))
print(f'RMSE: {rmse}')
```

```
# Calculating MAE
mae = mean_absolute_error(actual_unemployment,
    ARIMA_LSTM_3Y_forecast)
print(f'MAE: {mae}')

# Calculating MSE
mse = mean_squared_error(actual_unemployment, ARIMA_LSTM_3Y_forecast)
print(f'MSE: {mse}')

# Calculating MAPE
mape = mean_absolute_percentage_error(actual_unemployment,
    ARIMA_LSTM_3Y_forecast)
print(f'MAPE: {mape}')

# Performing the K-S test
# Perform paired t-test
ks_stat, p_value = stats.kstest(ARIMA_LSTM_3Y_forecast,
    actual_unemployment)
print(f'K-S Test Statistic: {ks_stat}')
print(f'Paired t-Test p-value: {p_value}')

ARIMA + LSTM 1-Year Forecast
# Split data into train and test sets 1-Year Forecast
forecast_period = 2 * 12 # Due to the use of a time step of 12
    months, the forecast period is adjusted to 2-years.

# The forecast period is subtracted from the length of the dataset
    to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:] # train and test is split

target = 'UnemploymentRate' # Creating the target

```

```
# Fitting the ARIMA model
arima_order = (2, 1, 0) # Order of ARIMA determined from plots
arima_model = ARIMA(train[target], order=arima_order) # Adding the
    unemployment rate and ARIMA order to the model.
arima_fit = arima_model.fit() # Fitting the model.
print(arima_fit.summary()) # Printing the ARIMA model summary.

# Forecast and calculate residuals
arima_forecast = arima_fit.forecast(steps=len(test)) # Forecasting
    with the ARIMA fit.
residuals = test[target] - arima_forecast # Calculating the ARIMA
    models residuals
# Normalising residuals
scaler_residuals = MinMaxScaler(feature_range=(0, 1)) # Scaling from
    0 to 1
# Fitting scaler and transforming the data into a 2D array for the
    LSTM model
scaled_residuals =
    scaler_residuals.fit_transform(residuals.values.reshape(-1, 1))

# Prepare the data for LSTM
time_step = 12 # Setting the time step to 12, so the LSTM uses 12
    months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to
    create new lists suitable for the LSTM model.
for i in range(time_step, len(scaled_residuals)):  # Loop to iterate
    over scaled residuals from the time step 12.
     X.append(scaled_residuals[i-time_step:i, 0]) # For each postion
         i, extract a sequence of 12 time steps to use.
     y.append(scaled_residuals[i, 0]) # Appends value of i to the
         list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed
    into NumPy arrays which is needed for the LSTM model.
```

```
# The forecast dates were not displaying correctly when plotting the
    results.
# Therefore, the dates were extracted separately and then added to
    the plot.
test_dates = indicators_merged.index[train_size:]

# Defining the LSTM model with 4 layers.
model = Sequential() # Sequential model
model.add(LSTM(10, return_sequences=True, input_shape=(time_step,
    1))) # Add 10 units, returns full sequence, and specifies shape
    of input data.
model.add(LSTM(10, return_sequences=False)) # Adds another 10 units,
    only returns the output of the last step.
model.add(Dense(5)) # Adding a dense layer of 5 units, reducing the
    output to 5 units.
model.add(Dense(1)) # Adding a dense layer with 1 unit, produces a
    single value as output.
model.compile(optimizer='adam', loss='mean_squared_error') #
    Compiles the model with the adam optimiser, and MSE as the loss
    function.

print(model.summary())

# Trainingging the model. Fitting the data to the model.
model.fit(X, y, epochs=50, batch_size=32, verbose=1)

# Clearing session to start fresh.
tf.keras.backend.clear_session()

# Predicting residuals using LSTM model
# Making sure prediction and training shapes are the same
predicted_residuals = model.predict(X)
```

```
predicted_residuals = scaler_residuals.inverse_transform(predicted_residuals)

# Aligninging the lengths of the ARIMA forecast and predicted residuals
arima_forecast = arima_forecast[time_step:]

# Combining the ARIMA forecast and LSTM predicted residuals
actual_unemployment = test[target][time_step:] # Specifying the actual unemployment rate for the plot.
ARIMA_LSTM_1Y_forecast = arima_forecast.values + predicted_residuals.flatten() # Combining both forescasts

# Plotting results
plt.figure(figsize=(12, 6)) # Setting the figure size
plt.plot(test_dates[time_step:], actual_unemployment, label='Actual') # Plotting the actual unemployment rate line
plt.plot(test_dates[time_step:], ARIMA_LSTM_1Y_forecast, label='Hybrid Forecast', linestyle='--') # plotting the hybrid forecast line
plt.xlabel('Date') # Adding Date label to X-axis
plt.ylabel('Unemployment Rate') # Adding Unemployment rate label to y-axis
plt.title('Hybrid ARIMA-LSTM Forecast vs Actual 1-Y') # Adding main title
plt.legend() # Adding legend
plt.savefig('ARIMA-LSTM-1Y.png') # Saving the plot as a png file.
plt.show(); # Displaying plot

# Calculating the RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment, ARIMA_LSTM_1Y_forecast))
print(f'MSE: {rmse}')
# Calculating the MAE
```

```
mae = mean_absolute_error(actual_unemployment,
    ARIMA_LSTM_1Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, ARIMA_LSTM_1Y_forecast)
print(f'MSE: {mse}')
# Calculating the MAPE
mape = mean_absolute_percentage_error(actual_unemployment,
    ARIMA_LSTM_1Y_forecast)
print(f'MAPE: {mape}')

ks_stat, p_value = stats.kstest(ARIMA_LSTM_1Y_forecast,
    actual_unemployment)
print(f'K-S Test Statistic: {ks_stat}')
print(f'Paired t-Test p-value: {p_value}')

ARIMA + GRU 3-Year Forecast
# Split data into train and test sets 4-Year Forecast
forecast_period = 4 * 12 # Due to the use of a timestep of 12
    months, the forecast period is adjusted.
# The forecast period is subtracted from the length of the dataset
    to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:] # training and test data is then
    split.

target = 'UnemploymentRate' # Setting the target variable
features = ['gdp_lag', 'inflation_lag', 'unemployment_lag'] #
    Setting the additional features for GRU

# Fit ARIMA model
arima_order = (2, 1, 0) # Setting the ARIMA order.
```

```
arima_model = ARIMA(train[target], order=arima_order)  # Adding the UnemploymentRate and ARIMA order to the model
arima_fit = arima_model.fit() # Fitting the model
print(arima_fit.summary()) # Displaying the ARIMA summary

# Forecast and calculate residuals
arima_forecast = arima_fit.forecast(steps=len(test)) # Creating the ARIMA forecast, and specifying the length of the forecast.
residuals = test[target] - arima_forecast # Calculating ARIMA residuals for the GRU model.
# Normalising residuals
scaler_residuals = MinMaxScaler(feature_range=(0, 1)) # scaler from 0 to 1.
# Scales the residules and transforms the residuals into a 2D array for the GRU.
scaled_residuals = scaler_residuals.fit_transform(residuals.values.reshape(-1, 1))

# Preparing the data for GRU
time_step = 12 # Setting the time step to 12, so the GRU uses 12 months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to create new lists suitable for the GRU model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate over scaled residuals from the time step 12.
    X.append(scaled_residuals[i-time_step:i, 0]) # For each postion i, extract a sequence of 12 time steps to use.
    y.append(scaled_residuals[i, 0]) # Appends value of i to the list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed into NumPy arrays which is needed for the GRU model.

```

```
# The forecast dates were not displaying correctly when plotting the results.
# Therefore, the dates were extracted separately and then added to the plot.
test_dates = unemployment.index[train_size:]

# Defining the GRU model with 4 layers.
model = Sequential() # Sequential model
model.add(GRU(10, return_sequences=True, input_shape=(time_step, 1))) # Add 10 units, returns full sequence, and specifies shape of input data.
model.add(GRU(10, return_sequences=False))  # Adds another 10 units, only returns the output of the last step.
model.add(Dense(5))  # Adding a dense layer of 5 units, reducing the output to 5 units.
model.add(Dense(1)) # Adding a dense layer with 1 unit, produces a single value as output.
model.compile(optimizer='adam', loss='mean_squared_error')  # Compiles the model with the adam optimiser, and MSE as the loss function.

print(model.summary())

# Trainingging the model. Fitting the data to the model.
model.fit(X, y, epochs=50, batch_size=32, verbose=1)

# Clearing session to start fresh.
tf.keras.backend.clear_session()

# Predicting residuals using GRU model
predicted_residuals = model.predict(X)
predicted_residuals = scaler_residuals.inverse_transform(predicted_residuals)

```

```
# Aligning the lengths of the ARIMA forecast and predicted residuals
arima_forecast = arima_forecast[time_step:]

# Combine ARIMA forecast and GRU predicted residuals
# Making sure prediction and training shapes are the same
actual_unemployment = test[target].values[time_step:]
ARIMA_GRU_3Y_forecast = arima_forecast.values +
    predicted_residuals.flatten()

# Plotting results
plt.figure(figsize=(12, 6)) # Setting the figure size
plt.plot(test_dates[time_step:], actual_unemployment,
    label='Actual') # Plotting the actual unemployment rate line
plt.plot(test_dates[time_step:], ARIMA_GRU_3Y_forecast,
    label='Hybrid Forecast', linestyle='--') # plotting the hybrid
    forecast line
plt.xlabel('Date') # Adding Date label to X-axis
plt.ylabel('Unemployment Rate') # Adding Unemployment rate label to
    y-axis
plt.title('Hybrid ARIMA-GRU Forecast vs Actual 3-Y') # Adding main
    title
plt.legend() # Adding legend
plt.savefig('ARIMA-GRU-3Y.png') # Saving the plot as a png file.
plt.show(); # Displaying plot

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment ,
    ARIMA_GRU_3Y_forecast))
print(f'RMSE: {rmse}')
# Calculating MAE
mae = mean_absolute_error(actual_unemployment ,
    ARIMA_GRU_3Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
```

```
mse = mean_squared_error(actual_unemployment , ARIMA_GRU_3Y_forecast)
print(f'MSE: {mse}')
# Calculating MAPE
mape = mean_absolute_percentage_error(actual_unemployment ,
    ARIMA_GRU_3Y_forecast)
print(f'MAPE: {mape}')


ks_stat, p_value = stats.kstest(ARIMA_GRU_3Y_forecast,
    actual_unemployment)
print(f'K-S Test Statistic: {ks_stat}')
print(f'Paired t-Test p-value: {p_value}')

ARIMA - GRU 1-Year Forecast
# Split data into train and test sets 1-Year Forecast
forecast_period = 2 * 12 # Due to the use of a time step of 12
    months, the forecast period is adjusted to 2-years.
# The forecast period is subtracted from the length of the dataset
    to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:] # Trainingging and test data is
    then split.

target = 'UnemploymentRate' # Setting the target variable
features = ['gdp_lag', 'inflation_lag', 'unemployment_lag'] #
    Setting the additional features for GRU

# Fitting the ARIMA model
arima_order = (2, 1, 0) # Setting the ARIMA order - p,d,q.
arima_model = ARIMA(train[target], order=arima_order) # Adding the
    UnemploymentRate and ARIMA order to the model
arima_fit = arima_model.fit() # Fitting the model
print(arima_fit.summary()) # Displaying the ARIMA summary
```

```

# Forecast and calculate residuals
arima_forecast = arima_fit.forecast(steps=len(test)) # Creating the ARIMA forecast, and specifying the length of the forecast.
residuals = test[target] - arima_forecast   # Calculating ARIMA residuals for the GRU model.
# Normalising residuals
scaler_residuals = MinMaxScaler(feature_range=(0, 1)) # scaler from 0 to 1.
# Scales the residules and transforms the residuals into a 2D array for the GRU.
scaled_residuals = scaler_residuals.fit_transform(residuals.values.reshape(-1, 1))


# Preparing the data for GRU
time_step = 12 # Setting the time step to 12, so the GRU uses 12 months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to create new lists suitable for the GRU model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate over scaled residuals from the time step 12.
    X.append(scaled_residuals[i-time_step:i, 0]) # For each postion i, extract a sequence of 12 time steps to use.
    y.append(scaled_residuals[i, 0]) # Appends value of i to the list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed into NumPy arrays which is needed for the GRU model.

# The forecast dates were not displaying correctly when plotting the results.
# Therefore, the dates were extracted separately and then added to the plot.
```

```
test_dates = unemployment.index[train_size:]

# Defining the GRU model with 4 layers.
model = Sequential() # Sequential model
model.add(GRU(10, return_sequences=True, input_shape=(time_step,
    1))) # Add 10 units, returns full sequence, and specifies shape
    of input data.
model.add(GRU(10, return_sequences=False))  # Adds another 10 units,
    only returns the output of the last step.
model.add(Dense(5))  # Adding a dense layer of 5 units, reducing the
    output to 5 units.
model.add(Dense(1)) # Adding a dense layer with 1 unit, produces a
    single value as output.
model.compile(optimizer='adam', loss='mean_squared_error')  #
    Compiles the model with the adam optimiser, and MSE as the loss
    function.

print(model.summary())

# Traininging the model. Fitting the data to the model.
model.fit(X, y, epochs=50, batch_size=32, verbose=1)

# Clearing session to start fresh.
tf.keras.backend.clear_session()

# Predicting residuals using GRU model
predicted_residuals = model.predict(X)
predicted_residuals =
    scaler_residuals.inverse_transform(predicted_residuals)

# Aligning the lengths of the ARIMA forecast and predicted residuals
arima_forecast = arima_forecast[time_step:]

# Combine ARIMA forecast and GRU predicted residuals
```

```
# Making sure prediction and training shapes are the same
actual_unemployment = test[target].values[time_step:]
ARIMA_GRU_1Y_forecast = arima_forecast.values + predicted_residuals.flatten()

# Plotting results
plt.figure(figsize=(12, 6)) # Setting the figure size
plt.plot(test_dates[time_step:], actual_unemployment, label='Actual') # Plotting the actual unemployment rate line
plt.plot(test_dates[time_step:], ARIMA_GRU_1Y_forecast, label='Hybrid Forecast', linestyle='--') # plotting the hybrid forecast line
plt.xlabel('Date') # Adding Date label to X-axis
plt.ylabel('Unemployment Rate') # Adding Unemployment rate label to y-axis
plt.title('Hybrid ARIMA-GRU Forecast vs Actual 1-Y') # Adding main title
plt.legend() # Adding legend
plt.savefig('ARIMA-GRU-1Y.png') # Saving the plot as a png file.
plt.show(); # Displaying plot

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment , ARIMA_GRU_1Y_forecast))
print(f'RMSE: {rmse}')
# Calculating MAE
mae = mean_absolute_error(actual_unemployment , ARIMA_GRU_1Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment , ARIMA_GRU_1Y_forecast)
print(f'MSE: {mse}')
# Calculating MAPE
```

```
mape = mean_absolute_percentage_error(actual_unemployment ,
    ARIMA_GRU_1Y_forecast)
print(f'MAPE: {mape}')

ks_stat, p_value = stats.kstest(ARIMA_GRU_1Y_forecast,
    actual_unemployment)
print(f'K-S Test Statistic: {ks_stat}')
print(f'Paired t-Test p-value: {p_value}')


ES + SVR 3-Year Forecast
# Split data into train and test sets
forecast_period = 4 * 12  # Due to the use of a timestep of 12
    months, the forecast period is adjusted.
# The forecast period is subtracted from the length of the dataset
    to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:] # training and test data is then
    split.

target = 'UnemploymentRate' # Setting the target variable

# Fit Exponential Smoothing model
# Adding the training data to the model, additive trend component,
    seasonal component, and indicating the length of the seasonal
    cycle.
es_model = ExponentialSmoothing(train[target], trend='add',
    seasonal='add', seasonal_periods=12)
es_fit = es_model.fit()  # Fits the model to the training data
es_forecast = es_fit.forecast(steps=len(test)) # Forecast future
    values using the fitted model

residuals = test[target] - es_forecast # Calculating the residuals
```

```

# Scaling residuals
scaler = StandardScaler() # scaler from 0 to 1.
# Scales the residules and transforms the residuals into a 2D array
    for the SVR.
scaled_residuals = scaler.fit_transform(residuals.values.reshape(-1,
    1))

# Create sequences for SVR
time_step = 12 # Setting the time step to 12, so the SVR uses 12
    months of data to predict the next prediction.
X_svr, y_svr = [], [] # Creating empty lists for both X and y in
    order to create new lists suitable for the SVR model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate
    over scaled residuals from the time step 12.
    X_svr.append(scaled_residuals[i-time_step:i, 0]) # For each
        postion i, extract a sequence of 12 time steps to use.
    y_svr.append(scaled_residuals[i, 0]) # Appends value of i to the
        list y. This is the target.

X_svr, y_svr = np.array(X_svr), np.array(y_svr) # These lists are
    then transformed into NumPy arrays which is needed for the SVR
    model.

# The forecast dates were not displaying correctly when plotting the
    results.
# Therefore, the dates were extracted separately and then added to
    the plot.
test_dates = unemployment.index[train_size:]


# Training SVR model
svr_model = SVR(kernel='rbf') # Specifying kernal fpr the algorithm:
    Radial Basis Function.
```

```
svr_model.fit(X_svr, y_svr) # Fit the SVR model to the training data

# Predicting residuals using SVR model
predicted_residuals = svr_model.predict(X_svr)
predicted_residuals =
    scaler.inverse_transform(predicted_residuals.reshape(-1,
    1)).flatten()

# Aligning the lengths of the ES forecast and predicted residuals
es_forecast = es_forecast[time_step:]

# Combine ES forecast and SVR predicted residuals
actual_unemployment = test[target].values[time_step:]
ES_SVR_3Y_forecast = es_forecast.values + predicted_residuals

# Plotting results
plt.figure(figsize=(12, 6)) # Setting the figure size
plt.plot(test_dates[time_step:], actual_unemployment,
    label='Actual') # Plotting the actual unemployment rate line
plt.plot(test_dates[time_step:], ES_SVR_3Y_forecast, label='Hybrid
    Forecast', linestyle = '--') # plotting the hybrid forecast line
plt.legend() # Adding legend
plt.title('Hybrid Exponential Smoothing-SVR Forecast vs Actual 3-Y')
    # Adding main title
plt.xlabel('Date') # Adding Date label to X-axis
plt.ylabel('Unemployment Rate') # Adding Unemployment rate label to
    y-axis
plt.savefig('ES-SVR-3Y.png') # Saving the plot as a png file.
plt.show(); # Displaying plot

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment,
    ES_SVR_3Y_forecast))
print(f'RMSE: {rmse}')
```

```
# Calculating MAE
mae = mean_absolute_error(actual_unemployment, ES_SVR_3Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, ES_SVR_3Y_forecast)
print(f'MSE: {mse}')
# Calculating MAPE
mape = mean_absolute_percentage_error(actual_unemployment,
    ES_SVR_3Y_forecast)
print(f'MAPE: {mape}')

ks_stat, p_value = stats.kstest(ES_SVR_3Y_forecast,
    actual_unemployment)


print(f'K-S Test Statistic: {ks_stat}')
print(f'P-Value: {p_value}')

ES - SVR 1-Year Forecast
# Split data into train and test sets
forecast_period = 2 * 12  # Due to the use of a time step of 12
    months, the forecast period is adjusted to 2-years.
# The forecast period is subtracted from the length of the dataset
    to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:] # training and test data is then
    split.

target = 'UnemploymentRate' # Setting the target variable

# Fit Exponential Smoothing model
# Adding the training data to the model, additive trend component,
    seasonal component, and indicating the length of the seasonal
```

```
cycle.
es_model = ExponentialSmoothing(train[target], trend='add', seasonal='add')
es_fit = es_model.fit() # Fits the model to the training data
es_forecast = es_fit.forecast(steps=len(test)) # Forecast future values using the fitted model

residuals = test[target] - es_forecast # Calculating the residuals

# Scaling residuals
scaler = StandardScaler() # scaler from 0 to 1.
# Scales the residules and transforms the residuals into a 2D array for the SVR.
scaled_residuals = scaler.fit_transform(residuals.values.reshape(-1, 1))

# Create sequences for SVR
time_step = 12 # Setting the time step to 12, so the SVR uses 12 months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to create new lists suitable for the SVR model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate over scaled residuals from the time step 12.
    X.append(scaled_residuals[i-time_step:i, 0]) # For each postion i, extract a sequence of 12 time steps to use.
    y.append(scaled_residuals[i, 0]) # Appends value of i to the list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed into NumPy arrays which is needed for the SVR model.

# The forecast dates were not displaying correctly when plotting the results.
```

```
# Therefore, the dates were extracted separately and then added to the plot.
test_dates = unemployment.index[train_size:]

# Training SVR model
svr_model = SVR(kernel='rbf') # Specifying kernal fpr the algorithm: Radial Basis Function.
svr_model.fit(X, y) # Fit the SVR model to the training data

# Predicting residuals using SVR model
predicted_residuals = svr_model.predict(X)
predicted_residuals = scaler.inverse_transform(predicted_residuals.reshape(-1, 1)).flatten()

# Aligning the lengths of the ES forecast and predicted residuals
es_forecast = es_forecast[time_step:]

# Combine ES forecast and SVR predicted residuals
actual_unemployment = test[target].values[time_step:]
ES_SVR_1Y_forecast = es_forecast.values + predicted_residuals

# Plotting results
plt.figure(figsize=(12, 6)) # Setting the figure size
plt.plot(test_dates[time_step:], actual_unemployment, label='Actual') # Plotting the actual unemployment rate line
plt.plot(test_dates[time_step:], ES_SVR_1Y_forecast, label='Hybrid Forecast', linestyle = '--') # plotting the hybrid forecast line
plt.legend() # Adding legend
plt.title('Hybrid Exponential Smoothing-SVR Forecast vs Actual 1-Y') # Adding main title
plt.xlabel('Date') # Adding Date label to X-axis
plt.ylabel('Unemployment Rate') # Adding Unemployment rate label to y-axis
```

```
plt.savefig('ES-SVR-1Y.png') # Saving the plot as a png file.
plt.show(); # Displaying plot


# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment,
    ES_SVR_1Y_forecast))
print(f'RMSE: {rmse}')
# Calculating MAE
mae = mean_absolute_error(actual_unemployment, ES_SVR_1Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, ES_SVR_1Y_forecast)
print(f'MSE: {mse}')
# Calculating MAPE
mape = mean_absolute_percentage_error(actual_unemployment,
    ES_SVR_1Y_forecast)
print(f'MAPE: {mape}')

ks_stat, p_value = stats.kstest(ES_SVR_1Y_forecast,
    actual_unemployment)

print(f'K-S Test Statistic: {ks_stat}')
print(f'P-Value: {p_value}')

ES - RFR 3-Year Forecast
# Split data into train and test sets
forecast_period = 4 * 12  # Due to the use of a timestep of 12
    months, the forecast period is adjusted.
# The forecast period is subtracted from the length of the dataset
    to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:] # training and test data is then
```

```
split.

target = 'UnemploymentRate' # Setting the target variable

# Fit Exponential Smoothing model
# Adding the training data to the model, additive trend component, seasonal component, and indicating the length of the seasonal cycle.
es_model = ExponentialSmoothing(train[target], trend='add', seasonal='add', seasonal_periods=12)
es_fit = es_model.fit() # Fits the model to the training data
es_forecast = es_fit.forecast(steps=len(test)) # Forecast future values using the fitted model

residuals = test[target] - es_forecast # Calculating the residuals

# Scaling residuals
scaler = StandardScaler() # scaler from 0 to 1.
# Scales the residules and transforms the residuals into a 2D array for the RFR.
scaled_residuals = scaler.fit_transform(residuals.values.reshape(-1, 1))

# Creating sequences for Random Forest
time_step = 12 # Setting the time step to 12, so the RFR uses 12 months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to create new lists suitable for the RFR model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate over scaled residuals from the time step 12.
    X.append(scaled_residuals[i-time_step:i, 0]) # For each postion i, extract a sequence of 12 time steps to use.
    y.append(scaled_residuals[i, 0]) # Appends value of i to the list y. This is the target.
```

```

X, y = np.array(X), np.array(y) # These lists are then transformed into NumPy arrays which is needed for the RFR model.

# The forecast dates were not displaying correctly when plotting the results.
# Therefore, the dates were extracted separately and then added to the plot.
test_dates = unemployment.index[train_size:]

# Training Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=1)
rf_model.fit(X, y)

# Predicting residuals using Random Forest model
predicted_residuals = rf_model.predict(X)
predicted_residuals =
    scaler.inverse_transform(predicted_residuals.reshape(-1,
    1)).flatten()

# Aligning the lengths of the ES forecast and predicted residuals
es_forecast_aligned = es_forecast[time_step:]

# Combine ES forecast and Random Forest predicted residuals
actual_unemployment = test[target].values[time_step:]
ES_RF_3Y_forecast = es_forecast_aligned.values + predicted_residuals

# Plotting results
plt.figure(figsize=(12, 6)) # Setting the figure size
plt.plot(test_dates[time_step:], actual_unemployment,
    label='Actual') # Plotting the actual unemployment rate line
plt.plot(test_dates[time_step:], ES_RF_3Y_forecast, label='Hybrid
    Forecast', linestyle = '--') # plotting the hybrid forecast line
plt.legend() # Adding legend
```

```
plt.title('Hybrid Exponential Smoothing-Random Forest Forecast vs Actual 3-Y') # Adding main title
plt.xlabel('Date') # Adding Date label to X-axis
plt.ylabel('Unemployment Rate') # Adding Unemployment rate label to y-axis
plt.savefig('ES-RF-3Y.png') # Saving the plot as a png file.
plt.show(); # Displaying plot

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment, ES_RF_3Y_forecast))
print(f'RMSE: {rmse}')
# Calculating MAE
mae = mean_absolute_error(actual_unemployment, ES_RF_3Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, ES_RF_3Y_forecast)
print(f'MSE: {mse}')
# Calculating MAPE
mape = mean_absolute_percentage_error(actual_unemployment, ES_RF_3Y_forecast)
print(f'MAPE: {mape}')

# Performing the K-S test
ks_stat, p_value = stats.kstest(ES_RF_3Y_forecast, actual_unemployment)
print(f'K-S Test Statistic: {ks_stat}')
print(f'Paired t-Test p-value: {p_value}')

ES - RFR 1-Year Forecast
# Split data into train and test sets
forecast_period = 2 * 12  # Due to the use of a time step of 12 months, the forecast period is adjusted to 2-years.
```

```python
# The forecast period is subtracted from the length of the dataset to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size], indicators_merged[train_size:] # training and test data is then split.

target = 'UnemploymentRate' # Setting the target variable

# Fit Exponential Smoothing model
# Adding the training data to the model, additive trend component, seasonal component, and indicating the length of the seasonal cycle.
es_model = ExponentialSmoothing(train[target], trend='add', seasonal='add', seasonal_periods=12)
es_fit = es_model.fit() # Fits the model to the training data
es_forecast = es_fit.forecast(steps=len(test)) # Forecast future values using the fitted model

residuals = test[target] - es_forecast # Calculating the residuals

# Scaling residuals
scaler = StandardScaler() # scaler from 0 to 1.
# Scales the residules and transforms the residuals into a 2D array for the RF.
scaled_residuals = scaler.fit_transform(residuals.values.reshape(-1, 1))

# Create sequences for Random Forest
time_step = 12 # Setting the time step to 12, so the RFR uses 12 months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to create new lists suitable for the RFR model.
```

```
for i in range(time_step, len(scaled_residuals)): # Loop to iterate
    over scaled residuals from the time step 12.
     X.append(scaled_residuals[i-time_step:i, 0]) # For each postion
         i, extract a sequence of 12 time steps to use.
     y.append(scaled_residuals[i, 0]) # Appends value of i to the
         list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed
    into NumPy arrays which is needed for the RFR model.

# The forecast dates were not displaying correctly when plotting the
    results.
# Therefore, the dates were extracted separately and then added to
    the plot.
test_dates = unemployment.index[train_size:]

# Training Random Forest model, setting the number of trees to 100,
    and setting a random state for reproducible results
rf_model = RandomForestRegressor(n_estimators=100, random_state=1)
rf_model.fit(X, y) # Fit the model to the training data

# Predicting residuals using Random Forest model
predicted_residuals = rf_model.predict(X)
predicted_residuals =
    scaler.inverse_transform(predicted_residuals.reshape(-1,
    1)).flatten()

# Aligning the lengths of the ES forecast and predicted residuals
actual_unemployment = test[target].values[time_step:]
es_forecast_aligned = es_forecast[time_step:]

# Combine ES forecast and Random Forest predicted residuals
ES_RF_1Y_forecast = es_forecast_aligned.values + predicted_residuals

```

```
# Plotting results
plt.figure(figsize=(12, 6))
plt.plot(test_dates[time_step:], test[target][time_step:],
    label='Actual')
plt.plot(test_dates[time_step:], ES_RF_1Y_forecast, label='Hybrid
    Forecast', linestyle = '--')
plt.legend()
plt.title('Hybrid Exponential Smoothing-Random Forest Forecast vs
    Actual 1-Y')
plt.xlabel('Date')
plt.ylabel('Unemployment Rate')
plt.savefig('ES-RF-1Y.png')
plt.show();

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment,
    ES_RF_1Y_forecast))
print(f'RMSE: {rmse}')
# Calculating MAE
mae = mean_absolute_error(actual_unemployment, ES_RF_1Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, ES_RF_1Y_forecast)
print(f'MSE: {mse}')
# Calculating MAPE
mape = mean_absolute_percentage_error(actual_unemployment,
    ES_RF_1Y_forecast)
print(f'MAPE: {mape}')

# Performing the K-S test
ks_stat, p_value = stats.kstest(actual_unemployment,
    ES_RF_1Y_forecast)

print(f'K-S Test Statistic: {ks_stat}')
```

```
print(f'P-Value: {p_value}')

VAR + LSTM 3-Year Forecast
# Split data into train and test sets for a 3-Year Forecast
forecast_period = 4 * 12  # Due to the use of a timestep of 12
    months, the forecast period is adjusted.
# The forecast period is subtracted from the length of the dataset
    to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:] # training and test data is then
    split.

target = 'Unemployment_Diff' # Setting the target variable
features = ['gdp_lag', 'inflation_lag', 'unemployment_lag'] #
    Setting additional features for the VAR

# Converting data into numeric.
train = train.apply(pd.to_numeric)
test = test.apply(pd.to_numeric)

# Fit VAR model with the additional features
var_model = VAR(train[features])
var_result = var_model.fit(maxlags=15) # Fitting the model.
    Specifies that the model should consider up to 15 lag periods.
print(var_result.summary()) # Print summary of the fitted model

# Make predictions using VAR model
# Selects the last 'k_ar' observations from the training data.
    'k_ar' is the number of lags chosen during the model fitting.
# Also, specifying the number of time steps
var_forecast =
    var_result.forecast(train[features].values[-var_result.k_ar:],
    steps=len(test))
```

```
var_forecast_df = pd.DataFrame(var_forecast, index=test.index, columns=features)

# Calculating residuals
residuals = test[features].values - var_forecast

# Normalising residuals
scaler_residuals = MinMaxScaler(feature_range=(0, 1))
# Scales the residules and transforms the residuals into a 2D array
scaled_residuals = scaler_residuals.fit_transform(residuals[:, -1].reshape(-1, 1))

# Prepare the data for LSTM
time_step = 12 # Setting the time step to 12, so the LSTM uses 12 months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to create new lists suitable for the LSTM model.
for i in range(time_step, len(scaled_residuals)):  # Loop to iterate over scaled residuals from the time step 12.
    X.append(scaled_residuals[i-time_step:i, 0]) # For each postion i, extract a sequence of 12 time steps to use.
    y.append(scaled_residuals[i, 0]) # Appends value of i to the list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed into NumPy arrays which is needed for the LSTM model.

# The forecast dates were not displaying correctly when plotting the results.
# Therefore, the dates were extracted separately and then added to the plot.
test_dates = indicators_merged.index[train_size:]

# Defining the LSTM model with 4 layers.
```

```
model = Sequential() # Sequential model
model.add(LSTM(10, return_sequences=True, input_shape=(time_step, 1))) # Add 10 units, returns full sequence, and specifies shape of input data.
model.add(LSTM(10, return_sequences=False)) # Adds another 10 units, only returns the output of the last step.
model.add(Dense(5)) # Adding a dense layer of 5 units, reducing the output to 5 units.
model.add(Dense(1)) # Adding a dense layer with 1 unit, produces a single value as output.
model.compile(optimizer='adam', loss='mean_squared_error') # Compiles the model with the adam optimiser, and MSE as the loss function.

print(model.summary())

# Trainingging the model. Fitting the data to the model.
model.fit(X, y, epochs=50, batch_size=32, verbose=1)

# Clearing session to start fresh.
tf.keras.backend.clear_session()

# Predicting residuals using LSTM model
# Making sure prediction and training shapes are the same
predicted_residuals = model.predict(X)
predicted_residuals = scaler_residuals.inverse_transform(predicted_residuals)

# Aligninging the lengths of the VAR forecast and predicted residuals
var_forecast_aligned = var_forecast_df[features[-1]].values[time_step:]

# Combine VAR forecast and LSTM predicted residuals
```

```
actual_unemployment = test[target].values[time_step:] # Specifying
    the actual unemployment rate for the plot.
VAR_LSTM_3Y_forecast = var_forecast_aligned +
    predicted_residuals.flatten() # Combining both forescasts

# Plotting results
plt.figure(figsize=(12, 6)) # Setting the figure size
plt.plot(test_dates[time_step:], actual_unemployment,
    label='Actual') # Plotting the actual unemployment rate line
plt.plot(test_dates[time_step:], VAR_LSTM_3Y_forecast, label='Hybrid
    Forecast', linestyle='--') # plotting the hybrid forecast line
plt.xlabel('Date') # Adding Date label to X-axis
plt.ylabel('Unemployment Rate') # Adding Unemployment rate label to
    y-axis
plt.title('Hybrid VAR-LSTM Forecast vs Actual 3-Y') # Adding main
    title
plt.legend() # Adding legend
plt.savefig('VAR-LSTM-3Y.png') # Saving the plot as a png file.
plt.show(); # Displaying plot

# Calculating performance metrics
# Calculating the RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment,
    VAR_LSTM_3Y_forecast))
print(f'RMSE: {rmse}')
# Calculating the MAE
mae = mean_absolute_error(actual_unemployment, VAR_LSTM_3Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, VAR_LSTM_3Y_forecast)
print(f'MSE: {mse}')

# Performing the K-S test
```

```
ks_stat, p_value = stats.kstestp(actual_unemployment,
    VAR_LSTM_3Y_forecast)

print(f'K-S Test Statistic: {ks_stat}')
print(f'P-Value: {p_value}')

VAR + LSTM 1-Year Forecast
# Split data into train and test sets for a 1-Year Forecast
forecast_period = 2 * 12 # Due to the use of a time step of 12
    months, the forecast period is adjusted to 2-years.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:]

target = 'Unemployment_Diff' # Setting the target variable
features = ['gdp_lag', 'inflation_lag', 'unemployment_lag'] #
    Setting additional features for the VAR

# Converting data into numeric. Necessary with the differenced data.
train = train.apply(pd.to_numeric)
test = test.apply(pd.to_numeric)

# Fit VAR model with the additional features
var_model = VAR(train[features])
var_result = var_model.fit(maxlags=15) # Fitting the model.
    Specifies that the model should consider up to 15 lag periods.
print(var_result.summary()) # Print summary of the fitted model

# Make predictions using VAR model
# Selects the last 'k_ar' observations from the training data.
    'k_ar' is the number of lags chosen during the model fitting.
# Also, specifying the number of time steps
var_forecast =
    var_result.forecast(train[features].values[-var_result.k_ar:],
```

```
    steps=len(test))
var_forecast_df = pd.DataFrame(var_forecast, index=test.index,
    columns=features)

# Calculating residuals
residuals = test[features].values - var_forecast

# Normalize residuals
scaler_residuals = MinMaxScaler(feature_range=(0, 1))
# Scales the residules and transforms the residuals into a 2D array
scaled_residuals = scaler_residuals.fit_transform(residuals[:,
    -1].reshape(-1, 1))

# Prepare the data for LSTM
time_step = 12 # Setting the time step to 12, so the LSTM uses 12
    months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to
    create new lists suitable for the LSTM model.
for i in range(time_step, len(scaled_residuals)):  # Loop to iterate
    over scaled residuals from the time step 12.
     X.append(scaled_residuals[i-time_step:i, 0]) # For each postion
        i, extract a sequence of 12 time steps to use.
     y.append(scaled_residuals[i, 0]) # Appends value of i to the
        list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed
    into NumPy arrays which is needed for the LSTM model.

# The forecast dates were not displaying correctly when plotting the
    results.
# Therefore, the dates were extracted separately and then added to
    the plot.
test_dates = indicators_merged.index[train_size:]

```

```
# Defining the LSTM model with 4 layers.
model = Sequential() # Sequential model
model.add(LSTM(10, return_sequences=True, input_shape=(time_step,
    1))) # Add 10 units, returns full sequence, and specifies shape
    of input data.
model.add(LSTM(10, return_sequences=False)) # Adds another 10 units,
    only returns the output of the last step.
model.add(Dense(5)) # Adding a dense layer of 5 units, reducing the
    output to 5 units.
model.add(Dense(1)) # Adding a dense layer with 1 unit, produces a
    single value as output.
model.compile(optimizer='adam', loss='mean_squared_error') #
    Compiles the model with the adam optimiser, and MSE as the loss
    function.

print(model.summary())

# Trainingng the model. Fitting the data to the model.
model.fit(X, y, epochs=50, batch_size=32, verbose=1)

# Clearing session to start fresh.
tf.keras.backend.clear_session()

# Predicting residuals using LSTM model
# Making sure prediction and training shapes are the same
predicted_residuals = model.predict(X)
predicted_residuals =
    scaler_residuals.inverse_transform(predicted_residuals)

# Aligninging the lengths of the VAR forecast and predicted residuals
var_forecast_aligned =
    var_forecast_df[features[-1]].values[time_step:]

# Combine VAR forecast and LSTM predicted residuals
```

```
actual_unemployment = test[target].values[time_step:]
VAR_LSTM_1Y_forecast = var_forecast_aligned + predicted_residuals.flatten()

# Plotting results
plt.figure(figsize=(12, 6)) # Setting the figure size
plt.plot(test_dates[time_step:], actual_unemployment, label='Actual') # Plotting the actual unemployment rate line
plt.plot(test_dates[time_step:], VAR_LSTM_1Y_forecast, label='Hybrid Forecast', linestyle='--') # plotting the hybrid forecast line
plt.xlabel('Date') # Adding Date label to X-axis
plt.ylabel('Unemployment Rate') # Adding Unemployment rate label to y-axis
plt.title('Hybrid VAR-LSTM Forecast vs Actual 1-Y') # Adding main title
plt.legend() # Adding legend
plt.savefig('VAR-LSTM-1Y.png') # Saving the plot as a png file.
plt.show(); # Displaying plot

# Calculating performance metrics
# Calculating the RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment, VAR_LSTM_1Y_forecast))
print(f'RMSE: {rmse}')
# Calculating the MAE
mae = mean_absolute_error(actual_unemployment, VAR_LSTM_1Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, VAR_LSTM_1Y_forecast)
print(f'MSE: {mse}')

# Performing the K-S test
ks_stat, p_value = stats.kstest(VAR_LSTM_1Y_forecast, actual_unemployment)
```

```

print(f'K-S Test Statistic: {ks_stat}')
print(f'P-Value: {p_value}')

VAR - GRU 3-Year Forecast
# Split data into train and test sets for a 4-Year Forecast
forecast_period = 4 * 12  # Due to the use of a timestep of 12 months, the forecast period is adjusted.
# The forecast period is subtracted from the length of the dataset to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size], indicators_merged[train_size:] # training and test data is then split.

target = 'Unemployment_Diff' # Setting the target variable
features = ['gdp_lag', 'inflation_lag', 'unemployment_lag'] # Setting additional features for the VAR

# Converting data into numeric. Necessary with the differenced data.
train = train.apply(pd.to_numeric)
test = test.apply(pd.to_numeric)

# Fit VAR model with the additional features
var_model = VAR(train[features])
var_result = var_model.fit(maxlags=15) # Fitting the model. Specifies that the model should consider up to 15 lag periods.
print(var_result.summary()) # Print summary of the fitted model

# Make predictions using VAR model
# Selects the last 'k_ar' observations from the training data. 'k_ar' is the number of lags chosen during the model fitting.
# Also, specifying the number of time steps
```

```
var_forecast =
    var_result.forecast(train[features].values[-var_result.k_ar:],
    steps=len(test))
var_forecast_df = pd.DataFrame(var_forecast, index=test.index,
    columns=features)

# Calculating residuals
residuals = test[features].values - var_forecast

# Normalising residuals
scaler_residuals = MinMaxScaler(feature_range=(0, 1))
# Scales the residules and transforms the residuals into a 2D array
scaled_residuals = scaler_residuals.fit_transform(residuals[:,
    -1].reshape(-1, 1))


# Preparing the data for GRU
time_step = 12 # Setting the time step to 12, so the GRU uses 12
    months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to
    create new lists suitable for the GRU model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate
    over scaled residuals from the time step 12.
     X.append(scaled_residuals[i-time_step:i, 0]) # For each postion
        i, extract a sequence of 12 time steps to use.
     y.append(scaled_residuals[i, 0]) # Appends value of i to the
        list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed
    into NumPy arrays which is needed for the GRU model.

# The forecast dates were not displaying correctly when plotting the
    results.
```

```
# Therefore, the dates were extracted separately and then added to the plot.
test_dates = unemployment.index[train_size:]

# Defining the GRU model with 4 layers.
model = Sequential() # Sequential model
model.add(GRU(10, return_sequences=True, input_shape=(time_step, 1))) # Add 10 units, returns full sequence, and specifies shape of input data.
model.add(GRU(10, return_sequences=False))  # Adds another 10 units, only returns the output of the last step.
model.add(Dense(5))  # Adding a dense layer of 5 units, reducing the output to 5 units.
model.add(Dense(1)) # Adding a dense layer with 1 unit, produces a single value as output.
model.compile(optimizer='adam', loss='mean_squared_error')  # Compiles the model with the adam optimiser, and MSE as the loss function.

print(model.summary())

# Traininging the model. Fitting the data to the model.
model.fit(X, y, epochs=50, batch_size=32, verbose=1)

# Clearing session to start fresh.
tf.keras.backend.clear_session()

# Predicting residuals using GRU model
predicted_residuals = model.predict(X)
predicted_residuals = scaler_residuals.inverse_transform(predicted_residuals)


# Aligning the lengths of the VAR forecast and predicted residuals
```

```
var_forecast_aligned = var_forecast_df[features[-1]].values[time_step:]

# Combine VAR forecast and GRU predicted residuals
# Making sure prediction and training shapes are the same
actual_unemployment = test[target].values[time_step:]
VAR_GRU_3Y_forecast = var_forecast_aligned + predicted_residuals.flatten()

# Plotting results
plt.figure(figsize=(12, 6)) # Setting the figure size
plt.plot(test_dates[time_step:], actual_unemployment, label='Actual') # Plotting the actual unemployment rate line
plt.plot(test_dates[time_step:], VAR_GRU_3Y_forecast, label='Hybrid Forecast', linestyle='--') # plotting the hybrid forecast line
plt.xlabel('Date') # Adding Date label to X-axis
plt.ylabel('Scaled Unemployment Rate') # Adding Unemployment rate label to y-axis
plt.title('Hybrid VAR-GRU Forecast vs Actual 3-Y') # Adding main title
plt.legend() # Adding legend
plt.savefig('VAR-GRU-3Y.png')  # Saving the plot as a png file.
plt.show(); # Displaying plot

# Calculating performance metrics
# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment, VAR_GRU_3Y_forecast))
print(f'Hybrid Model RMSE: {rmse}')
# Calculating MAE
mae = mean_absolute_error(actual_unemployment, VAR_GRU_3Y_forecast)
print(f'Hybrid Model MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, VAR_GRU_3Y_forecast)
```

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

# Performing the K-S test
ks_stat, p_value = stats.kstest(VAR_GRU_3Y_forecast,
    actual_unemployment)

print(f'K-S Test Statistic: {ks_stat}')
print(f'P-Value: {p_value}')


VAR-GRU 1-Year Forecast
# Split data into train and test sets for a 1-Year Forecast
forecast_period = 2 * 12 # Due to the use of a time step of 12
    months, the forecast period is adjusted to 2-years.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:]

target = 'Unemployment_Diff' # Setting the target variable
features = ['gdp_lag', 'inflation_lag', 'unemployment_lag'] #
    Setting additional features for the VAR

# Converting data into numeric. Necessary with the differenced data.
train = train.apply(pd.to_numeric)
test = test.apply(pd.to_numeric)

# Fit VAR model with the additional features
var_model = VAR(train[features])
var_result = var_model.fit(maxlags=15) # Fitting the model.
    Specifies that the model should consider up to 15 lag periods.
print(var_result.summary()) # Print summary of the fitted model

# Make predictions using VAR model
```

```
# Selects the last 'k_ar' observations from the training data. 'k_ar' is the number of lags chosen during the model fitting.
# Also, specifying the number of time steps
var_forecast = var_result.forecast(train[features].values[-var_result.k_ar:], steps=len(test))
var_forecast_df = pd.DataFrame(var_forecast, index=test.index, columns=features)

# Calculating residuals
residuals = test[features].values - var_forecast

# Normalising residuals
scaler_residuals = MinMaxScaler(feature_range=(0, 1))
# Scales the residules and transforms the residuals into a 2D array
scaled_residuals = scaler_residuals.fit_transform(residuals[:, -1].reshape(-1, 1))


# Preparing the data for GRU
time_step = 12 # Setting the time step to 12, so the GRU uses 12 months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to create new lists suitable for the GRU model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate over scaled residuals from the time step 12.
    X.append(scaled_residuals[i-time_step:i, 0]) # For each postion i, extract a sequence of 12 time steps to use.
    y.append(scaled_residuals[i, 0]) # Appends value of i to the list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed into NumPy arrays which is needed for the GRU model.

```

```
# The forecast dates were not displaying correctly when plotting the
    results.
# Therefore, the dates were extracted separately and then added to
    the plot.
test_dates = unemployment.index[train_size:]

# Defining the GRU model with 4 layers.
model = Sequential() # Sequential model
model.add(GRU(10, return_sequences=True, input_shape=(time_step,
    1))) # Add 10 units, returns full sequence, and specifies shape
    of input data.
model.add(GRU(10, return_sequences=False))  # Adds another 10 units,
    only returns the output of the last step.
model.add(Dense(5))  # Adding a dense layer of 5 units, reducing the
    output to 5 units.
model.add(Dense(1)) # Adding a dense layer with 1 unit, produces a
    single value as output.
model.compile(optimizer='adam', loss='mean_squared_error')  #
    Compiles the model with the adam optimiser, and MSE as the loss
    function.

print(model.summary())

# Trainingging the model. Fitting the data to the model.
model.fit(X, y, epochs=50, batch_size=32, verbose=1)

# Clearing session to start fresh.
tf.keras.backend.clear_session()

# Predicting residuals using GRU model
predicted_residuals = model.predict(X)
predicted_residuals =
    scaler_residuals.inverse_transform(predicted_residuals)

```

```python

# Aligninging the lengths of the VAR forecast and predicted residuals
var_forecast_aligned =
    var_forecast_df[features[-1]].values[time_step:]

# Combine VAR forecast and GRU predicted residuals
# Making sure prediction and training shapes are the same
actual_unemployment = test[target].values[time_step:]
VAR_LSTM_1Y_forecast = var_forecast_aligned +
    predicted_residuals.flatten()

# Plotting results
plt.figure(figsize=(12, 6))
plt.plot(test_dates[time_step:], actual_unemployment, label='Actual')
plt.plot(test_dates[time_step:], VAR_LSTM_1Y_forecast, label='Hybrid
    Forecast', linestyle='--')
plt.xlabel('Date')
plt.ylabel('Scaled Unemployment Rate')
plt.title('Hybrid VAR-GRU Forecast vs Actual 1-Y')
plt.legend()
plt.savefig('VAR-GRU-1Y.png')
plt.show();

# Calculating performance metrics
# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment,
    VAR_GRU_1Y_forecast))
print(f'RMSE: {rmse}')
# Calculating MAE
mae = mean_absolute_error(actual_unemployment, VAR_GRU_1Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, VAR_GRU_1Y_forecast)
print(f'MSE: {mse}')
```

```


# Performing the K-S test
ks_stat, p_value = stats.kstest(VAR_GRU_1Y_forecast,
    actual_unemployment)

print(f'K-S Test Statistic: {ks_stat}')
print(f'P-Value: {p_value}')

GARCH - RFR 3-Year Forecast
# Split data into train and test sets
forecast_period = 4 * 12  # Due to the use of a timestep of 12
    months, the forecast period is adjusted.
# The forecast period is subtracted from the length of the dataset
    to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:] # training and test data is then
    split.

target = 'UnemploymentRate' # Setting the target variable

# GARCH model fitting
garch_model = arch_model(train[target], vol='Garch', p=1, q=1) #
    Adding the GARCH model, and the order of the GARCH p and q.
garch_fit = garch_model.fit() # Fits the GARCH model to the training
    data
garch_forecast = garch_fit.forecast(horizon=len(test)) # Forecasts
    future values
garch_forecast_mean = garch_forecast.mean.iloc[-1].values # Extracts
    forecasted mean values

# Calculating residuals
residuals = test[target].values - garch_forecast_mean
```

```python

# Scaling residuals
scaler = StandardScaler() # scaler from 0 to 1.
# Scales the residules and transforms the residuals into a 2D array
    for the RF.
scaled_residuals = scaler.fit_transform(residuals.reshape(-1, 1))

# Creating sequences for Random Forest
time_step = 12 # Setting the time step to 12, so the RFR uses 12
    months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to
    create new lists suitable for the RFR model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate
    over scaled residuals from the time step 12.
    X.append(scaled_residuals[i-time_step:i, 0]) # For each postion
        i, extract a sequence of 12 time steps to use.
    y.append(scaled_residuals[i, 0]) # Appends value of i to the
        list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed
    into NumPy arrays which is needed for the RFR model.

# The forecast dates were not displaying correctly when plotting the
    results.
# Therefore, the dates were extracted separately and then added to
    the plot.
test_dates = unemployment.index[train_size:]

# Training Random Forest model, setting the number of trees to 100,
    and setting a random state for reproducible resultsl
rf_model = RandomForestRegressor(n_estimators=100, random_state=1)
rf_model.fit(X, y)

# Predicting residuals using Random Forest model
```

```
predicted_residuals = rf_model.predict(X)
# Making sure prediction and training shapes are the same
predicted_residuals =
    scaler.inverse_transform(predicted_residuals.reshape(-1,
    1)).flatten()

# Aligning the lengths of the GARCH forecast and predicted residuals
garch_forecast_aligned = garch_forecast_mean[time_step:]

# Combine GARCH forecast and Random Forest predicted residuals
actual_unemployment = test[target][time_step:]
GARCH_RF_3Y_forecast = garch_forecast_aligned + predicted_residuals

# Plotting results
plt.figure(figsize=(12, 6))
plt.plot(test_dates[time_step:], actual_unemployment, label='Actual')
plt.plot(test_dates[time_step:], GARCH_RF_3Y_forecast, label='Hybrid
    Forecast', linestyle='--')
plt.legend()
plt.title('Hybrid GARCH-Random Forest Forecast vs Actual 3-Y')
plt.xlabel('Date')
plt.ylabel('Unemployment Rate')
plt.savefig('GARCH-RF-3Y.png')
plt.show();

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment,
    GARCH_RF_3Y_forecast))
print(f'RMSE: {rmse}')
# Calculating MAE
mae = mean_absolute_error(actual_unemployment, GARCH_RF_3Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, GARCH_RF_3Y_forecast)
```

```
print(f'MSE: {mse}')
# Calculating MAPE
mape = mean_absolute_percentage_error(actual_unemployment,
    GARCH_RF_3Y_forecast)
print(f'MAPE: {mape}')

# Performing the K-S test
ks_stat, p_value = stats.kstest(GARCH_RF_3Y_forecast,
    actual_unemployment)

print(f'K-S Test Statistic: {ks_stat}')
print(f'P-Value: {p_value}')

GARCH + RFR 1-Year Forecast
# Splitting the data
forecast_period = 2 * 12  # Due to the use of a time step of 12
    months, the forecast period is adjusted to 2-years.
# The forecast period is subtracted from the length of the dataset
    to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:] # training and test data is then
    split.

target = 'UnemploymentRate' # Setting the target variable

# GARCH model fitting
garch_model = arch_model(train[target], vol='Garch', p=1, q=1) #
    Adding the GARCH model, and the order of the GARCH p and q.
garch_fit = garch_model.fit() # Fits the GARCH model to the training
    data
garch_forecast = garch_fit.forecast(horizon=len(test)) # Forecasts
    future values
```

```
garch_forecast_mean = garch_forecast.mean.iloc[-1].values # Extracts
    forecasted mean values

# Calculating residuals
residuals = test[target].values - garch_forecast_mean

# Scaling residuals
scaler = StandardScaler() # scaler from 0 to 1.
# Scales the residules and transforms the residuals into a 2D array
    for the RF.
scaled_residuals = scaler.fit_transform(residuals.reshape(-1, 1))

# Creating sequences for Random Forest
time_step = 12 # Setting the time step to 12, so the RFR uses 12
    months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to
    create new lists suitable for the RFR model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate
    over scaled residuals from the time step 12.
    X.append(scaled_residuals[i-time_step:i, 0]) # For each postion
        i, extract a sequence of 12 time steps to use.
    y.append(scaled_residuals[i, 0]) # Appends value of i to the
        list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed
    into NumPy arrays which is needed for the RFR model.

# The forecast dates were not displaying correctly when plotting the
    results.
# Therefore, the dates were extracted separately and then added to
    the plot.
test_dates = unemployment.index[train_size:]

```

```
# Training Random Forest model, setting the number of trees to 100,
    and setting a random state for reproducible resultsl
rf_model = RandomForestRegressor(n_estimators=100, random_state=1)
rf_model.fit(X, y)

# Predicting residuals using Random Forest model
predicted_residuals = rf_model.predict(X)
# Making sure prediction and training shapes are the same
predicted_residuals =
    scaler.inverse_transform(predicted_residuals.reshape(-1,
    1)).flatten()

# Aligning the lengths of the GARCH forecast and predicted residuals
garch_forecast_aligned = garch_forecast_mean[time_step:]

# Combine GARCH forecast and Random Forest predicted residuals
actual_unemployment = test[target][time_step:]
GARCH_RF_1Y_forecast = garch_forecast_aligned + predicted_residuals

# Plotting results
plt.figure(figsize=(12, 6))
plt.plot(test.index[time_step:], actual_unemployment, label='Actual')
plt.plot(test.index[time_step:], GARCH_RF_1Y_forecast, label='Hybrid
    Forecast', linestyle='--')
plt.legend()
plt.title('Hybrid GARCH-Random Forest Forecast vs Actual 1-Y')
plt.xlabel('Date')
plt.ylabel('Unemployment Rate')
plt.savefig('GARCH-RF-1Y.png')
plt.show();

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment,
    GARCH_RF_1Y_forecast))
```

```
print(f'RMSE: {rmse}')
# Calculating MAE
mae = mean_absolute_error(actual_unemployment, GARCH_RF_1Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, GARCH_RF_1Y_forecast)
print(f'MSE: {mse}')
# Calculating MAPE
mape = mean_absolute_percentage_error(actual_unemployment,
    GARCH_RF_1Y_forecast)
print(f'MAPE: {mape}')

# Performing the K-S test
ks_stat, p_value = stats.kstest(GARCH_RF_1Y_forecast,
    actual_unemployment)

print(f'K-S Test Statistic: {ks_stat}')
print(f'P-Value: {p_value}')

GARCH - SVR 3-Year Forecast
# Split data into train and test sets
forecast_period = 4 * 12  # Due to the use of a timestep of 12
    months, the forecast period is adjusted.
# The forecast period is subtracted from the length of the dataset
    to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:] # training and test data is then
    split.

target = 'UnemploymentRate' # Setting the target variable

# GARCH model fitting
```

```
garch_model = arch_model(train[target], vol='Garch', p=1, q=1) # Adding the GARCH model, and the order of the GARCH p and q.
garch_fit = garch_model.fit() # Fits the GARCH model to the training data
garch_forecast = garch_fit.forecast(horizon=len(test)) # Forecasts future values
garch_forecast_mean = garch_forecast.mean.iloc[-1].values # Extracts forecasted mean values

# Calculating residuals
residuals = test[target].values - garch_forecast_mean

# Scaling residuals
scaler = StandardScaler() # scaler from 0 to 1.
# Scales the residules and transforms the residuals into a 2D array for the SVR.
scaled_residuals = scaler.fit_transform(residuals.reshape(-1, 1))

# Creating sequences for SVR
time_step = 12 # Setting the time step to 12, so the SVR uses 12 months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to create new lists suitable for the SVR model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate over scaled residuals from the time step 12.
    X.append(scaled_residuals[i-time_step:i, 0]) # For each postion i, extract a sequence of 12 time steps to use.
    y.append(scaled_residuals[i, 0]) # Appends value of i to the list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed into NumPy arrays which is needed for the SVR model.

```

```
# The forecast dates were not displaying correctly when plotting the
    results.
# Therefore, the dates were extracted separately and then added to
    the plot.
test_dates = unemployment.index[train_size:]

# Training SVR model
svr_model = SVR(kernel='rbf') # Specifying kernal fpr the algorithm:
    Radial Basis Function.
svr_model.fit(X, y) # Fit the SVR model to the training data

# Predicting residuals using SVR model
predicted_residuals = svr_model.predict(X)
# Making sure prediction and training shapes are the same
predicted_residuals =
    scaler.inverse_transform(predicted_residuals.reshape(-1,
    1)).flatten()

# Aligning the lengths of the GARCH forecast and predicted residuals
garch_forecast_aligned = garch_forecast_mean[time_step:]

# Combine GARCH forecast and Random Forest predicted residuals
GARCH_SVR_3Y_forecast = garch_forecast_aligned + predicted_residuals
actual_unemployment = test[target].values[time_step:]

# Plotting results
plt.figure(figsize=(12, 6))
plt.plot(test_dates[time_step:], actual_unemployment, label='Actual')
plt.plot(test_dates[time_step:], GARCH_SVR_3Y_forecast,
    label='Hybrid Forecast', linestyle = '--')
plt.legend()
plt.title('Hybrid GARCH-SVR Forecast vs Actual 3-Y')
plt.xlabel('Date')
plt.ylabel('Unemployment Rate')
```

```
plt.savefig('GARCH-SVR-3Y.png')
plt.show();

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment,
    GARCH_SVR_3Y_forecast))
print(f'RMSE: {rmse}')
# Calculating MAE
mae = mean_absolute_error(actual_unemployment, GARCH_SVR_3Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, GARCH_SVR_3Y_forecast)
print(f'MSE: {mse}')
# Calculating MAPE
mape = mean_absolute_percentage_error(actual_unemployment,
    GARCH_SVR_3Y_forecast)
print(f'MAPE: {mape}')

# Performing the K-S test
ks_stat, p_value = stats.kstest(GARCH_SVR_3Y_forecast,
    actual_unemployment)

print(f'K-S Test Statistic: {ks_stat}')
print(f'P-Value: {p_value}')

GARCH - SVR 1-Year Forecast
# Split data into train and test sets
forecast_period = 2 * 12  # Due to the use of a timestep of 12
    months, the forecast period is adjusted.
# The forecast period is subtracted from the length of the dataset
    to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:] # training and test data is then
```

```
    split.

target = 'UnemploymentRate' # Setting the target variable

# GARCH model fitting
garch_model = arch_model(train[target], vol='Garch', p=1, q=1) #
    Adding the GARCH model, and the order of the GARCH p and q.
garch_fit = garch_model.fit() # Fits the GARCH model to the training
    data
garch_forecast = garch_fit.forecast(horizon=len(test)) # Forecasts
    future values
garch_forecast_mean = garch_forecast.mean.iloc[-1].values # Extracts
    forecasted mean values

# Calculating residuals
residuals = test[target].values - garch_forecast_mean

# Scaling residuals
scaler = StandardScaler() # scaler from 0 to 1.
# Scales the residules and transforms the residuals into a 2D array
    for the SVR.
scaled_residuals = scaler.fit_transform(residuals.reshape(-1, 1))

# Creating sequences for SVR
time_step = 12 # Setting the time step to 12, so the SVR uses 12
    months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to
    create new lists suitable for the SVR model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate
    over scaled residuals from the time step 12.
     X.append(scaled_residuals[i-time_step:i, 0]) # For each postion
        i, extract a sequence of 12 time steps to use.
     y.append(scaled_residuals[i, 0]) # Appends value of i to the
        list y. This is the target.
```

```

X, y = np.array(X), np.array(y) # These lists are then transformed into NumPy arrays which is needed for the SVR model.

# The forecast dates were not displaying correctly when plotting the results.
# Therefore, the dates were extracted separately and then added to the plot.
test_dates = unemployment.index[train_size:]

# Training SVR model
svr_model = SVR(kernel='rbf') # Specifying kernal fpr the algorithm: Radial Basis Function.
svr_model.fit(X, y) # Fit the SVR model to the training data

# Predicting residuals using SVR model
predicted_residuals = svr_model.predict(X)
# Making sure prediction and training shapes are the same
predicted_residuals = scaler.inverse_transform(predicted_residuals.reshape(-1, 1)).flatten()

# Aligning the lengths of the GARCH forecast and predicted residuals
garch_forecast_aligned = garch_forecast_mean[time_step:]

# Combining GARCH forecast and Random Forest predicted residuals
GARCH_SVR_1Y_forecast = garch_forecast_aligned + predicted_residuals
actual_unemployment = test[target].values[time_step:]

# Plotting results
plt.figure(figsize=(12, 6))
plt.plot(test.index[time_step:], test[target][time_step:], label='Actual')
```

```
plt.plot(test.index[time_step:], GARCH_SVR_1Y_forecast,
    label='Hybrid Forecast', linestyle = '--')
plt.legend()
plt.title('Hybrid GARCH-SVR Forecast vs Actual 1-Y')
plt.xlabel('Date')
plt.ylabel('Unemployment Rate')
plt.savefig('GARCH-SVR-1Y.png')
plt.show();

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment,
    GARCH_SVR_1Y_forecast))
print(f'RMSE: {rmse}')
# Calculating MAE
mae = mean_absolute_error(actual_unemployment, GARCH_SVR_1Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, GARCH_SVR_1Y_forecast)
print(f'MSE: {mse}')
# Calculating MAPE
mape = mean_absolute_percentage_error(actual_unemployment,
    GARCH_SVR_1Y_forecast)
print(f'MAPE: {mape}')

# Performing the K-S test
ks_stat, p_value = stats.kstest(GARCH_SVR_1Y_forecast,
    actual_unemployment)

print(f'K-S Test Statistic: {ks_stat}')
print(f'P-Value: {p_value}')

ARIMAX - LSTM 3-Year Forecast
# Split data into train and test sets
```

```python
forecast_period = 4 * 12 # Due to the use of a timestep of 12 months, the forecast period is adjusted to 4-years.
# Here, the forecast period is subtracted from the length of the dataset to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size], indicators_merged[train_size:] # training and test data is then split.

train = train.apply(pd.to_numeric)
test = test.apply(pd.to_numeric)

target = 'UnemploymentRate' # Creating a target object to hold the UnemploymentRate.
features = ['gdp_lag', 'inflation_lag', 'unemployment_lag'] # Adding additional features for the multivariate model

# Fit ARIMAX model
arima_order = (2, 1, 0) # Setting the ARIMA order, refer to methodology section.
# Adding the UnemploymentRate to the ARIMA model. Along with the models order
arimax_model = ARIMA(train[target], order=arima_order, exog=train[features]) # Adding the features for the ARIMAX model
arimax_fit = arimax_model.fit() # Fitting the ARIMAX model.
print(arimax_fit.summary()) # Showing summary

# Forecast and calculate residuals
arimax_forecast = arimax_fit.forecast(steps=len(test), exog=test[features]) # Creates the ARIMAX forecast, and specifying the length of the forecast.
residuals = test[target] - arimax_forecast # Calculating ARIMAX residuals
# Normalising residuals
```

```
scaler_residuals = MinMaxScaler(feature_range=(0, 1)) # Creating a
    scaler using a scale from 0 to 1.
# Fits the scaler and transforms the data into a 2D array
scaled_residuals =
    scaler_residuals.fit_transform(residuals.values.reshape(-1, 1))

# Prepare the data for LSTM
time_step = 12 # Setting the time step to 12, so the LSTM uses 12
    months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to
    create new lists suitable for the LSTM model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate
    over scaled residuals from the time step 12.
     X.append(scaled_residuals[i-time_step:i, 0]) # For each postion
        i, extract a sequence of 12 time steps to use.
     y.append(scaled_residuals[i, 0]) # Appends value of i to the
        list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed
    into NumPy arrays which is needed for the LSTM model.

# The forecast dates were not displaying correctly when plotting the
    results.
# Therefore, the dates were extracted separately and then added to
    the plot.
test_dates = unemployment.index[train_size:]

# Defining the LSTM model with 4 layers.
model = Sequential() # Sequential model
model.add(LSTM(10, return_sequences=True, input_shape=(time_step,
    1))) # Add 10 units, returns full sequence, and specifies shape
    of input data.
model.add(LSTM(10, return_sequences=False)) # Adds another 10 units,
    only returns the output of the last step.
```

```
model.add(Dense(5)) # Adding a dense layer of 5 units, reducing the output to 5 units.
model.add(Dense(1)) # Adding a dense layer with 1 unit, produces a single value as output.
model.compile(optimizer='adam', loss='mean_squared_error') # Compiles the model with the adam optimiser, and MSE as the loss function.


# Trainingingthe model. Fitting the data to the model.
model.fit(X, y, epochs=50, batch_size=32, verbose=1)

# Clearing session to start fresh.
tf.keras.backend.clear_session()

# Predicting residuals using LSTM model
# Make sure to use the same shape for prediction as used for training
predicted_residuals = model.predict(X)
predicted_residuals = scaler_residuals.inverse_transform(predicted_residuals)

# Aligning the lengths of the ARIMAX forecast and predicted residuals
arimax_forecast = arimax_forecast[time_step:]

# Combine ARIMA forecast and LSTM predicted residuals
actual_unemployment = test[target][time_step:] # Specifying the actual unemployment rate for the plot.
ARIMAX_LSTM_3Y_forecast = arimax_forecast.values + predicted_residuals.flatten() # Combining both forecasts

# Plotting results
plt.figure(figsize=(12, 6))
plt.plot(test_dates[time_step:], actual_unemployment, label='Actual')
```

```
plt.plot(test_dates[time_step:], ARIMAX_LSTM_3Y_forecast, label='Hybrid Forecast', linestyle='--')
plt.xlabel('Date')
plt.ylabel('Unemployment Rate')
plt.title('Hybrid ARIMAX-LSTM Forecast vs Actual 3-Y')
plt.legend()
plt.savefig('ARIMAX-LSTM-3Y.png')
plt.show();

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment, ARIMAX_LSTM_3Y_forecast))
print(f'RMSE: {rmse}')
# Calculating MAE
mae = mean_absolute_error(actual_unemployment, ARIMAX_LSTM_3Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment, ARIMAX_LSTM_3Y_forecast)
print(f'MSE: {mse}')
# Calculating MAPE
mape = mean_absolute_percentage_error(actual_unemployment, ARIMAX_LSTM_3Y_forecast)
print(f'MAPE: {mape}')

# Performing the K-S test
ks_stat, p_value = stats.kstest(ARIMAX_LSTM_3Y_forecast, actual_unemployment)

print(f'K-S Test Statistic: {ks_stat}')
print(f'P-Value: {p_value}')

ARIMA -LSTM 1 -Year Forecast
```

```
# Split data into train and test sets
forecast_period = 2 * 12 # Due to the use of a time step of 12
    months, the forecast period is adjusted to 2-years.
# Here, the forecast period is subtracted from the length of the
    dataset to create the training size.
train_size = len(indicators_merged) - forecast_period
train, test = indicators_merged[:train_size],
    indicators_merged[train_size:] # training and test data is then
    split.

train = train.apply(pd.to_numeric)
test = test.apply(pd.to_numeric)

target = 'UnemploymentRate' # Creating a target object to hold the
    UnemploymentRate.
features = ['gdp_lag', 'inflation_lag', 'unemployment_lag'] # Adding
    additional features for the multivariate model

# Fit ARIMAX model
arima_order = (2, 1, 0) # Setting the ARIMA order, refer to
    methodology section.
# Adding the UnemploymentRate to the ARIMA model. Along with the
    models order
arimax_model = ARIMA(train[target], order=arima_order,
    exog=train[features]) # Adding the features for the ARIMAX model
arimax_fit = arimax_model.fit() # Fitting the ARIMAX model.
print(arimax_fit.summary()) # Showing summary

# Forecast and calculate residuals
arimax_forecast = arimax_fit.forecast(steps=len(test),
    exog=test[features]) # Creates the ARIMAX forecast, and
    specifying the length of the forecast.
residuals = test[target] - arimax_forecast # Calculating ARIMAX
    residuals
```

```
# Normalising residuals
scaler_residuals = MinMaxScaler(feature_range=(0, 1)) # Creating a scaler using a scale from 0 to 1.
# Fits the scaler and transforms the data into a 2D array
scaled_residuals = scaler_residuals.fit_transform(residuals.values.reshape(-1, 1))

# Prepare the data for LSTM
time_step = 12 # Setting the time step to 12, so the LSTM uses 12 months of data to predict the next prediction.
X, y = [], [] # Creating empty lists for both X and y in order to create new lists suitable for the LSTM model.
for i in range(time_step, len(scaled_residuals)): # Loop to iterate over scaled residuals from the time step 12.
    X.append(scaled_residuals[i-time_step:i, 0]) # For each postion i, extract a sequence of 12 time steps to use.
    y.append(scaled_residuals[i, 0]) # Appends value of i to the list y. This is the target.

X, y = np.array(X), np.array(y) # These lists are then transformed into NumPy arrays which is needed for the LSTM model.

# The forecast dates were not displaying correctly when plotting the results.
# Therefore, the dates were extracted separately and then added to the plot.
test_dates = unemployment.index[train_size:]

# Defining the LSTM model with 4 layers.
model = Sequential() # Sequential model
model.add(LSTM(10, return_sequences=True, input_shape=(time_step, 1))) # Add 10 units, returns full sequence, and specifies shape of input data.
```

```
model.add(LSTM(10, return_sequences=False)) # Adds another 10 units,
    only returns the output of the last step.
model.add(Dense(5)) # Adding a dense layer of 5 units, reducing the
    output to 5 units.
model.add(Dense(1)) # Adding a dense layer with 1 unit, produces a
    single value as output.
model.compile(optimizer='adam', loss='mean_squared_error') #
    Compiles the model with the adam optimiser, and MSE as the loss
    function.


# Trainging the model. Fitting the data to the model.
model.fit(X, y, epochs=50, batch_size=32, verbose=1)

# Clearing session to start fresh.
tf.keras.backend.clear_session()

# Predicting residuals using LSTM model
# Make sure to use the same shape for prediction as used for training
predicted_residuals = model.predict(X)
predicted_residuals =
    scaler_residuals.inverse_transform(predicted_residuals)

# Aligning the lengths of the ARIMAX forecast and predicted residuals
arimax_forecast = arimax_forecast[time_step:]

# Combine ARIMA forecast and LSTM predicted residuals
actual_unemployment = test[target][time_step:]
ARIMAX_LSTM_1Y_forecast = arima_forecast.values +
    predicted_residuals.flatten()

# Plotting results
plt.figure(figsize=(12, 6))
plt.plot(test_dates[time_step:], actual_unemployment, label='Actual')
```

```
plt.plot(test_dates[time_step:], ARIMAX_LSTM_1Y_forecast,
    label='Hybrid Forecast', linestyle='--')
ARIMAX_LSTM_1Y_forecast
plt.xlabel('Date')
plt.ylabel('Unemployment Rate')
plt.title('Hybrid ARIMAX-LSTM Forecast vs Actual 1-Y')
plt.legend()
plt.savefig('ARIMAX-LSTM-1Y.png')
plt.show();

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(actual_unemployment,
    ARIMAX_LSTM_1Y_forecast))
print(f'RMSE: {rmse}')
# Calculating MAE
mae = mean_absolute_error(actual_unemployment,
    ARIMAX_LSTM_1Y_forecast)
print(f'MAE: {mae}')
# Calculating MSE
mse = mean_squared_error(actual_unemployment,
    ARIMAX_LSTM_1Y_forecast)
print(f'MSE: {mse}')
# Calculating MAPE
mape = mean_absolute_percentage_error(actual_unemployment,
    ARIMAX_LSTM_1Y_forecast)
print(f'MAPE: {mape}')

# Performing the K-S test
ks_stat, p_value = stats.kstest(ARIMAX_LSTM_1Y_forecast,
    actual_unemployment)

print(f'K-S Test Statistic: {ks_stat}')
print(f'P-Value: {p_value}')
```

Bibliography

[1] Chris Chatfield. Time-series forecasting: Chris chatfield: Taylor francis ebooks, r, Oct 2000.

[2] Chulsu Jo, Doo Hwan Kim, and Jae Woo Lee. Forecasting unemployment and employment: A system dynamics approach. *Technological Forecasting and Social Change*, 194:122715, 2023.

[3] otexts. Forecasting: Principles and practice (2nd ed), 2018.

[4] Christopher Sims. Macroeconomics and reality, Jan 1980.

[5] Statista Research Department. Uk unemployment rate 2024, Aug 2024.

[6] Rifaat Abdalla, Mohammed El-Diasty, Andrey Kostogryzov, and Nikolay Makhutov. *Time Series Analysis*. IntechOpen, Rijeka, Jan 2023.

[7] Manu Joseph. *Modern Time Series Forecasting with Python*. Packt Publishing, Birmingham, 1 edition, November 2022.

[8] Stockholm University. A brief history of time series analysis, May 2022.

[9] Bill Schmarzo. Big data: Understanding how data powers big business. In *BigData*, 2013.

[10] PennState. Overview of time series characteristics, 2024.

[11] Reserve Bank of Australia. Unemployment: Its measurement and types: Explainer: Education, May 2023.

[12] Office For National Statistics. Unemployment and the claimant count.

[13] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1), 2018.

[14] David Stuart. *Predictions and Forecasts*. Facet, 2020.

[15] M. W. Watson. Time series: Economic forecasting, 2001.

[16] Michael Clements and David Hendry. An overview of economic forecasting. *A Companion to Economic Forecasting. Oxford: Blackwell*, pages 1–18, 2002.

[17] P Andersen and TH Rasmussen. *Economic forecasting: Models, indicators and data needs*. Office for Official Publications of the European Communities, Luxembourg, 2003.

[18] DF Hendry and NR Ericsson. *Understanding Economic Forecasts*. The MIT Press, The MIT Press, August 2003.

[19] Christos Floros. Forecasting the uk unemployment rate: Model comparisons. *International Journal of Applied Econometrics and Quantitative Studies*, 2:57–72, 02 2005.

[20] Regis Barnichon and Christopher J. Nekarda. The ins and outs of forecasting unemployment, 2012.

[21] Brent Meyer and Murat Tasci. Lessons for forecasting unemployment in the United States: use flow rates, mind the trend. FRB Atlanta Working Paper 2015-1, Federal Reserve Bank of Atlanta, February 2015.

[22] Michal Gostkowski and Tomasz Rokicki. Forecasting the Unemployment Rate: Application of Selected Prediction Methods. *European Research Studies Journal*, 0(3):985–1000, 2021.

[23] María E. Pérez-Pons, Javier Parra-Dominguez, Sigeru Omatu, Enrique Herrera-Viedma, and Juan Manuel Corchado. Machine learning and traditional econometric models: A systematic mapping study. *Journal of Artificial Intelligence and Soft Computing Research*, 12(2):79–100, 2021.

[24] Arthur Charpentier, Emmanuel Flachaire, and Antoine Ly. Econometrics and machine learning. *Economie Et Statistique*, 505(1), April 2019.

[25] Sendhil Mullainathan and Jann Spiess. Machine learning: An applied econometric approach. *The Journal of Economic Perspectives*, 31(2):87–106, 2017.

[26] G. Shobana and K. Umamaheswari. Forecasting by machine learning techniques and econometrics: A review. In *2021 6th International Conference on Inventive Computation Technologies (ICICT)*, pages 1010–1016, 2021.

[27] Mustafa Yurtsever. Unemployment rate forecasting: Lstm-gru hybrid approach. *Journal for Labour Market Research*, 57(1):1–9, 2023.

[28] Kevin Mero, Nelson Salgado, Jaime Meza, Janeth Pacheco-Delgado, and Sebastián Ventura. Unemployment rate prediction using a hybrid model of recurrent neural networks and genetic algorithms. *Applied Sciences*, 14(8), 2024.

[29] Tanujit Chakraborty, Ashis Kumar Chakraborty, Munmun Biswas, Sayak Banerjee, and Shramana Bhattacharya. Unemployment rate forecasting: A hybrid approach. *Computational Economics*, 57(1):183–201, Aug 2020.

[30] Office for National Statistics. Gdp monthly estimate, uk: April 2024, Jun 2024.

[31] Statista. Uk unemployment rate 2024, Jun 2024.

[32] Statista Research Department. Uk inflation rate 2024, Aug 2024.

[33] Subash Palvel. Time series forecasting with prophet and lstm hybrid mode, Sep 2023.

[34] Sue Nugus. Financial planning using excel : forecasting planning and budgeting techniques. In *Financial planning using Excel*, 2009.

[35] Jonhariono Sihotang. Optimization of inventory ordering decision in retail business using exponential smoothing approach and decision support system, August 2023.

[36] Guckan Yapar, Idil Yavuz, and Hanife Taylan Selamlar. Why and how does exponential smoothing fail? an in depth comparison of ata-simple and simple exponential smoothing., August 2017.

[37] Statsmodels Statistics In Python. Statsmodels.tsa.holtwinters.exponentialsmoothing.

[38] Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. Wiley, 2015.

[39] Peter Äurka and Silvia Pastoreková. Arima vs. arimax - which approach is better to analyze and forecast macroeconomic time series?, 2012.

[40] Richard Evans. Understanding arima models for econometrics, 2024.

[41] Jin Liu. Navigating the financial landscape: The power and limitations of the arima model, 2024.

[42] Jason Brownlee. How to create an arima model for time series forecasting in python, Nov 2023.

[43] Divyesh Bhatt. Time series analysis and forecasting with arima in python, Nov 2023.

[44] Robert Engle. Garch 101: The use of arch/garch models in applied econometrics, 2001.

[45] Timo Terasvirta. An introduction to univariate garch models, 2006.

[46] Jason Brownlee. How to model volatility with arch and garch for time series forecasting in python, Aug 2019.

[47] Pennstate. 11.2 vector autoregressive models var(p) models: Stat 510, 2024.

[48] Otext. Forecasting: Principles and practice(2nd ed), 2018.

[49] Quant. Garch models for volatility forecasting: A python-based guide, May 2024.

[50] Kamilya Smagulova and Alex Pappachen James. A survey on lstm memristive neural network architectures and applications - the european physical journal special topics, Oct 2019.

[51] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. A review on the long short-term memory model - artificial intelligence review, May 2020.

[52] Shangshang Jin. A comparative analysis of traditional and machine learning methods in forecasting the stock markets of china and the us. *International Journal of Advanced Computer Science and Applications*, 15(4), 2024.

[53] Sepp Hochreiter, Jürgen Schmidhuber, Technische Universität München Sepp Hochreiter, Fakultät für Informatik, and Corso Elvezia 36 Jürgen Schmidhuber, IDSIA. Long short-term memory, Nov 1997.

[54] Zhaoyang Niu, Guoqiang Zhong, Guohua Yue, Li-Na Wang, Hui Yu, Xiao Ling, and Junyu Dong. Recurrent attention unit: A new gated recurrent unit for long-term memory of important parts in sequential data. *Neurocomputing*, 517:1–9, 2023.

[55] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[56] Rahul Dey and Fathi M. Salem. Gate-variants of gated recurrent unit (gru) neural networks, 2017.

[57] Jason Brownlee. Time series prediction with lstm recurrent neural networks in python with keras, Aug 2022.

[58] Uday Tripurani. Stock market predictions using lstm and gru models with python, Apr 2024.

[59] Soukaina Ouhame and Youssef Hadi. *Multivariate workload prediction using Vector Autoregressive and Stacked LSTM models*. Association for Computing Machinery, New York, NY, USA, 2019.

[60] Fan Zhang and Lauren J. O'Donnell. Chapter 7 - support vector regression. In Andrea Mechelli and Sandra Vieira, editors, *Machine Learning*, pages 123–140. Academic Press, 2020.

[61] Mariette Awad and Rahul Khanna. *Support Vector Regression*, pages 67–80. Apress, Berkeley, CA, 2015.

[62] Lars Rosenbaum, Alexander Dorr, Matthias R Bauer, Frank M Boeckler, and Andreas Zell. Inferring multi-target qsar models with taxonomy-based multi-task learning - journal of cheminformatics, Jul 2013.

[63] Nandini Verma. An introduction to support vector regression (svr) in machine learning, Nov 2023.

[64] Goehry, Benjamin. Random forests for time-dependent processes. *ESAIM: PS*, 24:801–826, 2020.

[65] Goehry, Benjamin. Random forests for time-dependent processes. *ESAIM: PS*, 24:801–826, 2020.

[66] Piotr Gajewski, Boris Äule, and Nevena Rankovic. Unveiling the power of arima, support vector and random forest regressors for the future of the dutch employment market. *Journal of Theoretical and Applied Electronic Commerce Research*, 18(3):1365–1403, 2023.

[67] scikit-learn documentation. Randomforestregressor.