22COC255
B925898

UK CPI Inflation Rate Prediction

by

Baileigh Cordice

A Project Report

Submitted in partial fulfilment
of the requirements for the award of

B.Sc.
in
Computer Science and Mathematics
of
Loughborough University

2022-23

Acknowledgements

I would like to thank my supervisor, Mohamad Saada, for his continued support, motivation and guidance throughout this project, and my friends and family for their endless encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Inflation is the rate at which prices for goods and services rise, and the purchasing power of money declines, i.e. when prices increase, consumers can buy less with the same amount of money (Wulandari 2022).

Recent Inflationary pressures in the UK have been evident. For example, Food prices have reached a 45-year high of 19% and the UK also only just avoided recession in 2022, with an average yearly inflation of 9.1%. Fortunately, inflation is predicted to ease over 2023, with the average inflation forecasted to drop to 6.8% according to IMF. Despite this, the UK is said to be the worst-performing compared to other G7 economies (IMF 2022).

Some factors contributing to the high UK inflation rate include: The increase in the cost of consumer goods, reinforced by strong demand from consumers and supply chain bottlenecks from Covid-19 and an increase in energy prices, mainly due to Russia's invasion of Ukraine - from September 2021 to September 2022, gas prices increased by 96% and electricity prices by 54% (Brigid Francis-Devine 2022).

The main way the UK predicts inflation is the Consumer Price Index (CPI). The CPI is an estimate of the average change in prices for a basket of common goods and services purchased by households (Payne 2016).

Due to the increased computing power and technical capabilities (including data harvesting and storage), there has been a rise in the use of machine learning techniques in the financial industry. Ranging from algorithmic trading models used in quantitative finance to portfolio optimisation via dynamic programming, research has been conducted to suggest that the public sector and policymakers should also benefit from machine learning and the utilisation of large amounts of data available (Liran Einav 2013). In my literature review, there will be an

in-depth analysis of the machine learning techniques used to forecast UK CPI Inflation in this project.

1.2 Aim & Objectives

In this report, I will be forecasting the UK CPI Inflation Rate using implementations of the Long-Short Term Memory (LSTM), Random Forest and ARIMA machine learning models and comparing their forecasting performance on a 12-month time horizon against each other. I will treat this forecast as a regression problem.

1.3 Structure

My report will be structured as follows:

- Chapter 2 - Literature Review
  Illustrating papers that utilise the mentioned machine learning models for CPI forecasting as well as other economic forecasting tasks and their respective results.
- Chapter 3 - Technical Background
  Outlining the background behind the LSTM, Random Forest and ARIMA models.
- Chapter 4 - Exploratory Data Analysis
  Discuss the CPI Dataset used for my report and a time-series decomposition for further analysis.
- Chapter 5 - Multi-variate Models: Implementation
  The implementation of the LSTM and Random Forest models and an evaluation of their training performance.
- Chapter 6 - Uni-variate Models: Implementation
  The implementation of the ARIMA models and an evaluation of their training performance.
- Chapter 7 - Results
  Evaluating the results of all the models on the test data, as well as defining error metrics used to evaluate forecast performance.

- Chapter 8 - Conclusion
  Finalising my report with discussions of potential improvements and personal development.

Chapter 2

Literature Review

The literature on both neural and non-neural network time series forecasting in economic research and otherwise is quite extensive. This section will contain a breakdown of relevant work regarding each model implemented in this project and examples of other areas used.

2.1 LSTM Research

CPI Forecasting

Riofrıo et al. 2020 compared a collection of models SVR, LSTM, SARIMA, Exponential Smoothing and Facebook's Prophet to forecast CPI inflation in Ecuador. The study found that LSTM was second best at predicting CPI according to its' MAPE score (0.00173) on a 1-year time horizon. SVR (using a polynomial kernel) was the best predictor with a MAPE score of 0.00171. Despite the second-best performance, it was noted that the graphical performance of LSTM accurately captured the dynamic behaviour (suggested by the peaks and troughs the LSTM graph showed) of CPI movement more so than the SVR graph, which was a straight line. Improvements suggested for this project were to use grid search for hyper-parameter tuning, which has been implemented in my report.

Asati, Learning and Bangalore 2022 was another comparative study, comparing XGBoost, Theta, ARIMA, Facebook Prophet and LSTM in uni-variate CPI forecasting over a 1-year time horizon. LSTM was the second best at forecasting in terms of RMSE (2.05), outperformed by Facebook's Prophet (0.78). Future work suggested analysing multi-variate performance as it gives deeper insight into the effects of a variety of factors on CPI performance, more relevant to real-life scenarios. This idea will be implemented in this project.

Swanson and White 1995 Analysed whether forward interest rates can be useful in predicting future spot rates using a linear model and a simple ANN over time

horizons t=2...6. They found that ANN models were a promising forecasting method, and suggested that future work be done on larger windows, which this project aims to achieve. Wu et al. 2018 created a hybrid model of LSTM and ARIMA models, with the aim of overcoming the problem of input variable selection of LSTM. They found that the hybrid LSTM model only slightly outperformed the standard LSTM model. They also discovered that ARIMA models were ineffective for bitcoin price forecasting due to the inability of making the data stationary via differencing.

2.2 Random Forest Research

Behrens, Pierdzioch and Risse 2018 was a study that analysed annual growth and CPI inflation in Germany from 4 economic research institutes between 1970-2016, and compared forecast errors for Random Forest over short and long time horizons (t=2 and 4 respectively). The study utilised the R "MultivariateRandomForest" package. The study found that the RMSE and RMAE are larger for long-term forecasts, and the RMSE and RMAE are on average larger for growth compared to inflation, regardless of time horizon. The study noted that every bootstrapped tree uses two-thirds of the data, and results for the inflation forecast depend on the institution. For example, institute 3, "Ifo Institute Munich" provides the lowest RMSE for a long horizon, and Institute 2, "Hamburg World Economic Institute" provides the lowest RMSE for short horizons. Future work suggests applying multivariate forecasting to other countries' macroeconomic data, and comparing the effectiveness of forecasts, which is the overall aim of this project.

Baybuza 2018 , a Russian inflation study, utilised 92 macroeconomic series with CPI chosen to measure inflation. Observations ranged from February 2002 - January 2018, containing 169 data points. The study noted that machine learning methods improved forecasting performance over benchmark auto-regressive models but at a cost of losing economic interpretability compared to standard econometric models. Random Forest predicts inflation better than other models, most notably a 60% smaller RMSE over a two-year forecast (t=24) compared to AR(1) model. They also noted that out of the 3 Random Forest models used, the best result was with untransformed data, with the random forest model with block bootstrap (dividing the data into several blocks) having the worst results for both forecasting tasks (specific month inflation and inflation over a specified time horizon) This result suggests that time series data tends to have a high degree of correlation, so methods such as block bootstrapping negatively affect model performance. Future work encouraged testing macroeconomic forecasting on non-linear algorithms such as neural networks, which this project aims to achieve.

Maehashi and Shintani 2020 a Japanese forecasting study, compared financial and multiple machine learning models (including ensemble methods such as Random Forest, Boosting and Bagging) for 7 target variables between 1973-2018. The study found that machine learning forecasts perform better for medium and long time horizons, and allowing for non-linearity improved forecasting performance. Out of the ensemble methods, Random Forests performed the best for t=3,6,18 time horizons in terms of the MFSE score.

2.3 ARIMA Models Research

Pufnik, Kunovac et al. 2006 Conducted a forecasting study using the ARIMA models over a time horizon of 12 months. The paper highlighted problems when modelling inflation, such as structural breaks and insufficient length of time series data. The paper notes that price stability (low and stable inflation rate) is favourable when evaluating economic growth using statistical models. The paper found that seasonal ARIMA models (SARIMA) had improved performance over standard ARIMA models as they capture seasonal components of CPI data more accurately. The paper also tested two forecasting approaches: forecasting the average CPI, and forecasting individual components and averaging their results. The results suggested that the first method offered better results for time horizon t=1..4 months. The second method (averaging individual forecasts) performed better for t=5...12 months, with an RMSE of 0.82 compared to 1.12. My paper will implement the first method and compare the results over a 12-month period.

Hassani and Silva 2018 compared uni-variate models: ARIMA, ETS, Feed-forward neural networks, and TBATS at forecasting inflation over various time horizons. They found that for short-term horizons (t=3) ETS is the most appropriate, ARIMA has better uni-variate forecasts for long-term horizons (t=24,36), and BATS is best at t=12 with an MSE of 0.201, which I will use as a comparison for my model's predictive performance.

Meyler, Kenny and Quinn 1998 Conducted a study of ARIMA models against structural models for short-term forecasting. They found that a relatively simple SARIMA model (0,0,0)x(1,0,1) which is essentially a straight line that accounts for seasonality was best at forecasting inflation when there was a relatively simple pattern, compared to the multi-variate BVAR model. The report suggested that despite this result, uni-variate ARIMA models cannot replace multivariate models, as they struggle to perform well with volatile series.

Chapter 3

Technical Background

3.1 Time Series Forecasting

Before we begin discussing the models used, it is important to quickly explain the concept of time series analysis and forecasting.

Time series analysis (explored further in Chapter 4.2) is closely related to time series forecasting. Time series analysis observes the patterns in time series data and reveals potential relationships within the data. Time series forecasting can be seen as the next step after this analysis has taken place.

Time series forecasting is the process of making predictions based on time series data. The process uses statistical and machine learning techniques to make future predictions by extrapolating patterns from past data (Brockwell and Davis 2002).

The CPI dataset used for this project is time series data, the algorithms described (and implemented) in the following chapters are both commonly used for time series forecasting.

3.2 Introduction to LSTM

The first algorithm implemented for this project is the Long-Short-Term Memory Network (LSTM). Before explaining this algorithm, it is important to explain what Artificial and Recurrent Neural Networks are.

3.2.1 Feed-Forward Networks

To explain what the LSTM model is, it would be useful to provide a brief explanation of the feed-forward network, as the LSTM model (a type of recurrent neural network) is derived from this architecture.

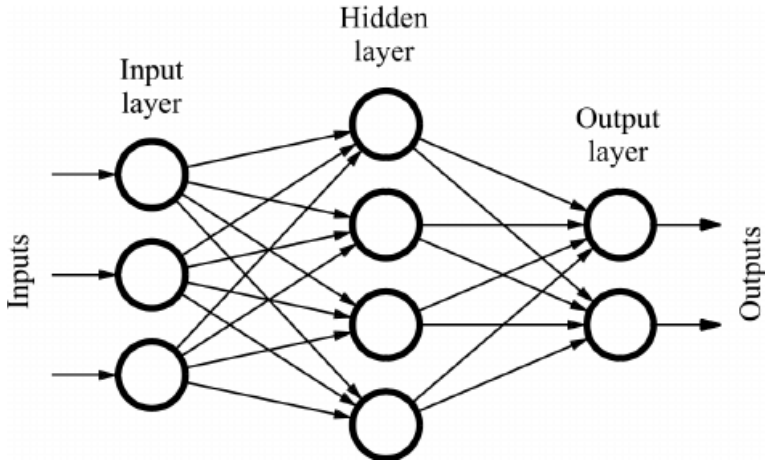

Figure 3.1: An image of a simple Feed Forward Network (Gurunathan and Krishnan 2021)

A feed-forward network is a type of artificial neural network and is the simplest type of neural network, where connections between nodes do not form a cycle. The input layer contains the initial data that is fed to the network, the hidden layer is where computation is done, and the output layer shows the results from the given inputs. Dawani 2020 The network also contains weights, biases and activation functions. Weights decide how much influence one node will have on another node; biases are a constant which is used to offset the result; activation functions are typically used in the hidden layer to introduce non-linearity to the neural network and provide a result within a specific range (depending on the function, i.e. sigmoid) from input values fed to the node. The above image shows a Multi-Layer-Perceptron, a simple feed-forward neural network.

Information is carried from the input layer in a single direction towards the output layer. Therefore, a feed-forward network can be described as a directed acylic graph ibid.

There are 3 main steps in the training of a neural network :

- Forward Propagation
- Loss Function
- Back Propagation

Forward Propagation is a standard forward pass through the network. In this step, you take the input training data and feed it through the network to get an output. The hidden layer will take the weights and biases from the input connections and apply the specified activation function, and this information is then used to calculate the output ibid.

The loss function calculates the error between the predicted result from the training output node against the actual value. Mean squared error is an example loss function used in regression tasks ibid.

Back Propagation enables the minimisation of errors. It's responsible for changing the weights and biases for each iteration, using the gradient of the loss function, calculated using Gradient Descent, with respect to the weights. This is an optimisation algorithm that looks for the local minimum of differentiable functions. The gradient that is calculated provides the amount that you change the weights and biases by ibid.

3.2.2 Recurrent Neural Networks

The first RNN model was invented by John Hopfield in 1982. He created the Hopfield Network - a single-layer RN that accesses and stores information inspired by how we believe human brains work. (Hopfield 1982).

In the previous section, the feed-forward model was described as a directed acyclic graph. The main difference between feed-forward and recurrent neural networks is that the latter can create cycles, often called closed-loop connections. The cyclic property of recurrent networks allows them to respond better to sequential data (data that depends on time/order) as it means the network is essentially given a memory to store and be influenced by previous information. (Cook 2019)

The cyclic connection allows the output information from nodes in the hidden layer to affect the input to nodes in the hidden layer:

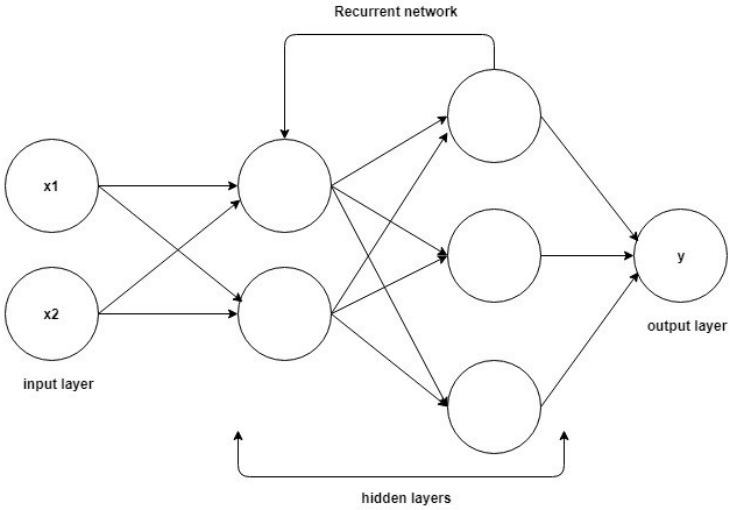

Figure 3.2: An image of a Recurrent Neural Network

This is an extremely useful feature, and highlights why RNNs are useful for sequential tasks such as sentence structure and translation, and more suitably, economic forecasting.

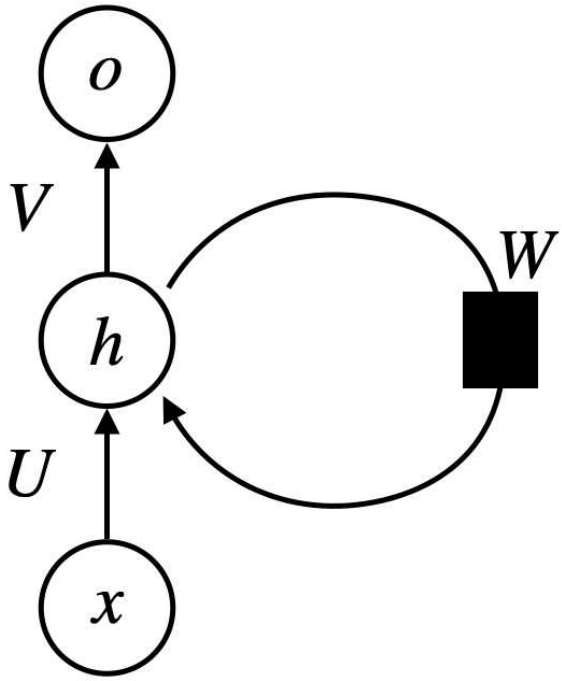

Figure 3.3: A single RNN loop
(Dawani 2020)

Outlined below are the calculations required for forward propagation for each time step:

Let U,V, and W be the weights from the input-hidden layer; hidden-hidden layer; and hidden-output layer respectively. The hidden node h_t at time step t is calculated by the following equation (Dawani 2020):

$$h_t = f_1(Wh_{t-1} + Ux_t + b) \tag{3.1}$$

where:

- x_t is the input at time t
- h_{t-1} is the previous hidden node
- b is the bias value
- f_1 is an activation function

For the hidden layer, the activation functions typically used are non-linear. Non-linearity is useful for analysing complex patterns within data. Examples of non-linear activation functions for RNN hidden layers are tanh and sigmoid ibid.

The output node o_t at time step t is defined by (ibid.):

$$o_t = f_2(Vh_t + a) \tag{3.2}$$

where:

- f_2 represents the softmax function
- a represents a bias value

A vector of probability scores is created from the raw outputs from the RNN. This is usually calculated using the softmax activation function, denoted as f_2.

The limitation of RNNs occurs at the back-propagation step. Namely, the vanishing gradient problem. As discussed, back-propagation is used to adjust weights and biases. The vanishing gradient problem occurs when the activation function causes the gradient to rapidly shrink. The weights' value tends to 0 which means no learning occurs in each iteration. As the weight between two nodes becomes very small, it prevents the later nodes from firing. This results in simple RNNs being disadvantageous in capturing long-term dependencies ibid.

The LSTM Algorithm provides a good solution to this vanishing/exploding gradient problem, by modifying the standard structure of the RNN ibid.

3.2.3 Long-Short Term Memory

The previous section discusses recurrent neural networks and their limitations regarding the vanishing gradient problem. The main consequence of this problem is the inability of standard recurrent neural networks to learn long-term dependencies.

A long-term dependency refers to the relationship between inputs in a sequence where a (relatively) distant input can impact the current state or output (ibid.). For sequential data (such as stock price data) this can mean the influence that past events can have on prediction. For example, considering the impact of COVID-19 on predicting the current inflation rate.

LSTMs are a type of RNN that can learn long-term dependencies and therefore overcome the vanishing gradient problem. LSTMs use gates to regulate the flow of information through the network. Each gate contains a sigmoid activation that signifies the amount of information passed through a gate, ranging between 0 and 1. 0 means the gate is closed, no information can pass through, 1 means the gate is completely open and all information is passed through. LSTMs have three types of gates: input, forget, and output gates, and they work together to update the cell state, which is the memory variable that stores information, enabling the 'memory' component of LSTMs by selectively retaining or forgetting information at past time steps (ibid.).

The forget gate f_t controls the flow of information out of the cell state (ibid.):

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f) \tag{3.3}$$

The input gate i_t controls the flow of new information into the cell state (Dawani 2020):

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \tag{3.4}$$

The output gate o_t controls the flow of information from the cell state to the hidden state(ibid.):

$$o_t = (w_o[h_{t-1}, x_t] + b_o) \tag{3.5}$$

where:

- σ is the sigmoid function
- w_n is the weight at gate n
- h_{t-1} is the output of the previous hidden state
- x_t is the input at the current step
- b_n is the bias at the gate n

These gates are controlled by learned parameters, which essentially work as the memory of the network. A learned parameter is a weight value updated during the training process to optimise the performance of the network, these parameters are updated during training using optimisation algorithms such as gradient descent.

The candidate layer at time t is defined as (ibid.):

$$\hat{c}_t = tanh(W_c[h_{t-1}, x_t] + b_c) \tag{3.6}$$

where $\hat{c}_t$ is a vector of new potential values for the cell state, using the input from the previous state.

The updated cell state c_t is defined as (ibid.):

$$c_t = f_t c_{t-1} + i_t \hat{c}_t \tag{3.7}$$

where c_t combines the previous cell state with the forget gate, and the candidate cell state with the input gate (ibid.).

The hidden state h_t is defined as:

$$h_t = o_t tanh(c_t) \tag{3.8}$$

The hidden state is effectively a vector of output values combined with the updated cell state passed through the tanh activation function (ibid.).

Combining these gates and states allows the LSTM to selectively preserve or block information that passes through the network, and update the model over many time steps. This allows LSTMs to model long-term dependencies, and avoid the vanishing gradient problem (ibid.).

Below shows a diagram of the LSTM model.

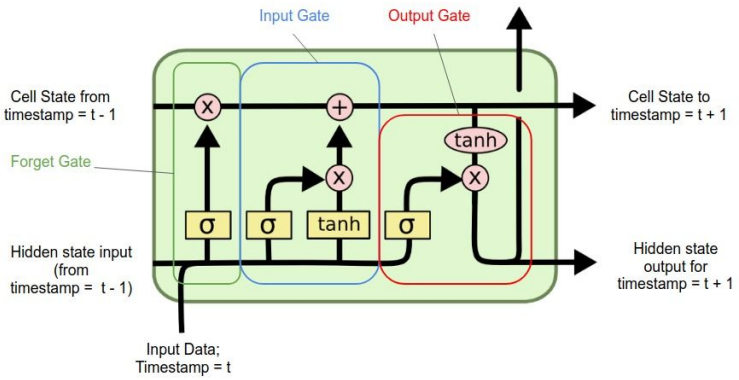

Figure 3.4: The LSTM Model
(Al-Shabi and Abuhamdah 2021)

3.3 Random Forests

3.3.1 Ensemble Methods

An ensemble learning method can be informally described as "the will of the crowd". It is an umbrella term for machine learning models that use multiple models (algorithms such as neural networks and decision trees) to make a prediction through classification or regression techniques. It is commonly used in supervised learning tasks. The main goal is to create several models with similar bias and average their outputs to reduce variance. (Sagi and Rokach 2018) Ensemble methods are also known to improve performance in prediction for the following reasons (Dietterich et al. 2002):

1. Avoids Overfitting

   Overfitting occurs when a model performs well on training data, but poorly on unseen test data. Ensemble methods avoid overfitting by averaging the results of multiple models.

2. Avoids Local Minima

A single model may get stuck in a local minima/optima. A combination of models reduces the chance of being in the same situation.

3. Larger search space

   The optimal output for a single model may be achieved outside the space in which the single model searches. A combination of different models can span a larger search space and therefore fit the data more accurately. This idea is prevalent in classification tasks.

Ensemble methods can also be used to ease difficult challenges that arise within machine learning algorithms. These include class imbalance, where one class has more data points than others which causes algorithms to have a bias towards these classes over smaller ones. Ensemble methods can be used to create an equal sub-sample of data points (Nikulin, McLachlan and Ng 2009); concept drift, where relationships within the data change over time, Kolter 2003 method uses eases this challenge by using individual decision trees that are created and deleted dynamically after changes in performance.

Bagging

Bagging (Bootstrap Aggregation) is an example of a common ensemble learning method. A sample of data from the training set is taken with replacement, (meaning a single data point can be selected multiple times) and multiple, diverse samples are generated and trained independently from each other, with the average of these predictions after training used as an estimate. (Breiman 1996).

The three steps of a typical bagging algorithm are (ibid.):

1. **Bootstrapping:** Multiple samples of the training data set are generated by taking points within the data with replacement

2. **Parallel Training:** The samples are then trained independently in parallel.

3. **Aggregation / Majority Voting:** In regression problems (such as this one) the final output is the average of all samples outputs. In classification problems, the class that appears in the majority of samples output is determined as the result

The Random Forest model is seen as an extension of bagging, by utilising a forest of decision trees that are uncorrelated.

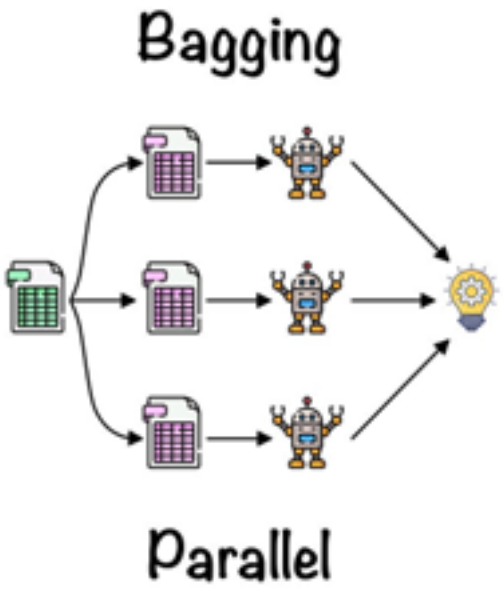

Figure 3.5: Bagging Algorithm Process

3.3.2 Random Forest

Random Forest was discovered by Leo Breiman (Breiman 2001), and as mentioned, extends the idea of bagging through multiple decision tree models and can be used for classification or regression tasks. For regression problems, each tree is bootstrapped with a sample from the dataset containing a random subset of features (feature randomness) and is then trained in parallel. The final output is an average of each tree's predictions.

Random Forest has important features such as a reduced risk of overfitting because of the utilisation of bagging - as there is a large number of decision trees within a random forest, the variance and prediction errors are lowered due to the average of these uncorrelated trees being taken.

3.4 (Seasonal) Auto-Regressive Integrated Moving Average

ARIMA models are a class of statistical models that can be used to analyse and forecast time series data that is non-stationary and can be made stationary via a technique called differencing. The SARIMA models are quite similar, with the main difference being that the SARIMA models account for seasonality within data by providing seasonal differencing within the model. Before we break down the (S)ARIMA model, it would be useful to explain what stationarity is, as it's a

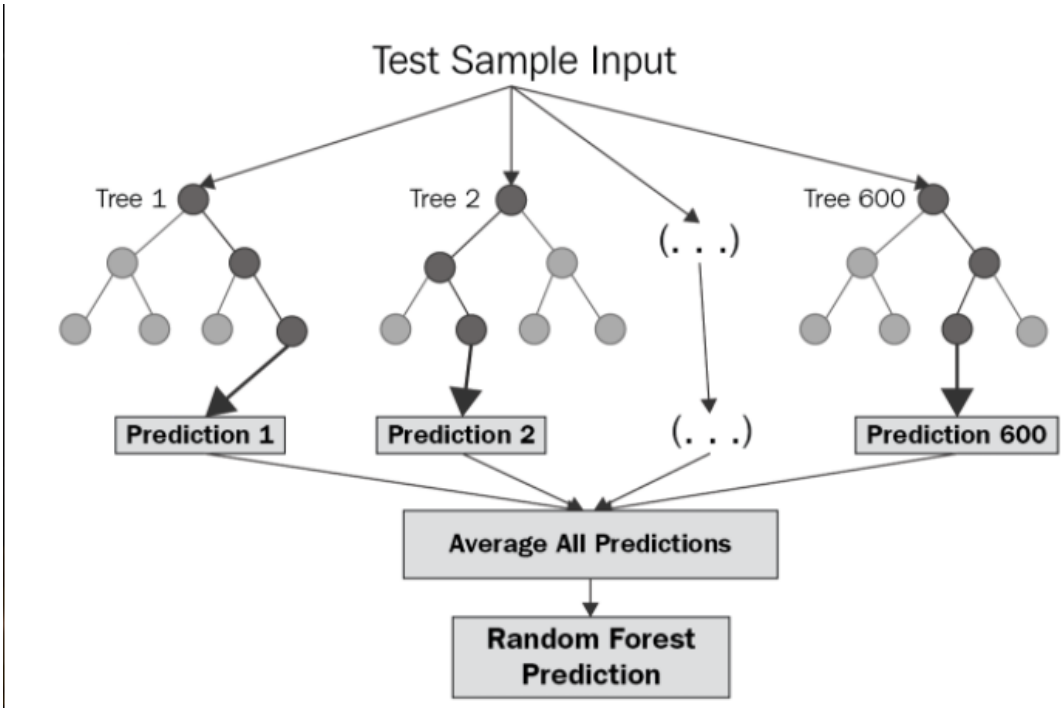

Figure 3.6: Random Forest Regression Model

key assumption in these models. It is also a uni-variate model.

3.4.1 Stationarity

I will outline the definitions of mean and covariance of a time series and use them to define stationarity (Brockwell and Davis 2002).

Definition 3.4.1. *Let X_t be a time series with $E(X_t^2) < \infty$. The **mean** of X_t is:*
$\mu_X(t) = E(X_t)$

Definition 3.4.2. *The **covariance function** of X_t is:*
$\gamma_X(r,s) = Cov(X_r, X_s) = E[(X_r - \mu_X(r))(X_s - \mu_X(s))]$ *for all integers r and s*

Combining these two definitions allows us to define stationarity:

Definition 3.4.3. *X_t is (weakly) stationary if:*

- *$\mu_X(t)$ is independent of t*
- *$\gamma_X(t+h,t)$ independent of t for each h.*

The definition above is for weak stationarity. The difference between weak and strict stationary is the condition that the original time series and the shifted time series have the same joint distribution. For this project, it is sufficient to use the definition of weak stationarity as the definition for stationarity.

Stationarity, therefore, is a property of a time series where the mean and covariance of each observation are independent (ibid.).

As mentioned, stationarity of non-stationary data is achieved through a technique called differencing. The explanation and implementation of differencing will be discussed further in Chapter 4.

3.4.2 ARIMA Models

The names of the models are acronyms that describe the model's key features. The ARIMA models are represented by parameters p,d,q and the SARIMA models are characterised by P, D, Q, s. Different (S)ARIMA models can be defined by the number of auto-regressive, moving average, and integrated terms.

The notation for an ARIMA is ARIMA(p,d,q) and Seasonal ARIMA (SARIMA) notation is SARIMA(p,d,q)(P,D,Q,s).

S - Seasonal

This term only applies to SARIMA, as ARIMA does not support seasonal data. Seasonality refers to patterns that repeat at regular intervals, including variations. In the case of economic forecasting, there may be quarterly or yearly seasonality within the data. Specific seasonality with regard to this project is explored in Chapter 4. In SARIMA models, this is represented by the S component.

AR – "Auto Regression"

Auto-Regression is a method to model time series data. The time series is modelled as a linear combination of past values.

The Auto-Regressive component is a type of auto-regressive model. An auto-regressive model is a statistical model that uses past values of a time series to predict future values. It is modelled using the auto-regressive function, denoted as AR(p) (ibid.):

$$y_t = c + \Sigma_{i=1}^{p} \varphi_i y_{t-i} + \varepsilon_t \tag{3.9}$$

- y_t is the value of the time series at time t,
- c is a constant,
- φ_1, ..., φ_p are the coefficients of the auto-regressive component, y_{t-1}, ..., y_{t-p}, respectively

- ε_t is the random error term at time t.

In ARIMA models, this represents the p component. It refers to the number (n) of lagged observations included in the model, i.e. the relationship between the current observation and n previous observations. (the number of auto-regressive terms) In SARIMA models, this represents the P component. It refers to the number of lags in the seasonal component included in the model. (the number of seasonal auto-regressive terms)

I - "Integrated"

This represents the order of differencing. Differencing is the act of removing trends and seasonality from the data. In ARIMA models, this represents the d component and is the order of differencing. In SARIMA models, this is the D component and is the order of seasonal differencing.

MA – "Moving Average"

Moving average is a method used to analyse time series data. It calculates the average value of a subset of data points over a certain period with a forward-moving "window" of the subset over time.

The Moving Average component in ARIMA represents parts of the model that contain short-term variations in the time series that are not explained by the AR component. It is a weighted sum of residuals, which makes it useful in improving the models' ability to capture the dynamic nature of a time series (Brockwell and Davis 2002):

$$y_t = c + \varepsilon_t + \Sigma_{i=1}^{q} \theta_i e_{t-i} \tag{3.10}$$

- where y_t is the value of the time series at time t
- c is a constant,
- ε_t is the random error term at time t
- θ_1, ..., θ_q are the coefficients of the moving average component, e_{t-1}, ..., e_{t-q}, respectively.

In ARIMA models, this represents the q component. It represents the number of moving average terms in the model

In SARIMA models, this represents the Q component. It represents the number of seasonal moving average terms in the model.

Methods for finding these parameter values will be discussed in Chapter 5.

3.4.3 Forecasting Equations

ARIMA Forecasting Equation

The ARIMA forecasting equation is essentially a combination of the equations listed above and is used to predict future values of a time series (Farsi et al. 2021):

$$y'_t = c + \Sigma_{i=1}^{p} \phi_i y_{t-i} - \Sigma_{i=1}^{q} \theta_i e_{t-i} + \varepsilon_t \tag{3.11}$$

- y'_t is the value of the differenced time series at time t
- c is a constant
- ϕ_1, ..., ϕ_p are the coefficients of the auto-regressive component, y_{t-1}, ..., y_{t-p}, respectively
- θ_1, ..., θ_q are the coefficients of the moving average component, e_{t-1}, ..., e_{t-q}, respectively
- ε_t is the random error term at time t.

This equation can also be written using the backshift operator, B. The backshift operator is used in time series analysis to represent the lag operator. The lag operator shifts a time series back in time by one unit.

$$(1 - \phi_1 B - \cdots - \phi_p B^p)(1 - B)^d y_t = c + (1 + \theta_1 B + \cdots + \theta_q B^q)\varepsilon_t \tag{3.12}$$

Parameter	Description	Equation
p	order of auto-regression	$(1 - \phi_1 B - \cdots - \phi_p B^p)$
d	order of differencing	$(1 - B)^d$
q	order of moving average	$(1 + \theta_1 B + \cdots + \theta_q B^q)\varepsilon_t$

Table 3.1: ARIMA Forecasting Parameters

SARIMA Forecasting Equation

As mentioned, SARIMA is an extension of ARIMA that accounts for seasonality within time series data. The SARIMA forecasting equation reflects these additional features as well as the standard ARIMA equation Farsi et al. 2021:

$$\Phi_p(B^s)\phi_p(B)(1-B^s)^d(1-B)^D y_t = c + \Theta_q(B^s)\theta_q(B)\varepsilon_t \tag{3.13}$$

- s is the seasonal period (e.g., 12 for monthly data with yearly seasonality)
- B is the backshift operator
- d and D are the non-seasonal and seasonal differences, respectively
- $\Phi_p(B^s)$ and $\phi_p(B)$ are the seasonal and non-seasonal autoregressive polynomial operators, respectively
- $\Theta_q(B^s)$ and $\theta_q(B)$ are the seasonal and non-seasonal moving average polynomial operators, respectively
- c is a constant
- ε_t is the error term at time t.

The seasonal autoregressive polynomial operators $\Phi_p(B^s)$ and the seasonal moving average polynomial operators $\Theta_q(B^s)$ are defined as:

$$\Phi_p(B^s) = 1 - \phi_1(B^s) - \phi_2(B^2 s) - ... - \phi_p(B^p s) \tag{3.14}$$

$$\Theta_q(B^s) = 1 + \theta_1(B^s) + \theta_2(B^2 s) + ... + \theta_q(B^q s) \tag{3.15}$$

respectively (ibid.).

Chapter 4

Exploratory Data Analysis

This section contains a breakdown of the data used, how it was cleaned to prepare for model implementation, and some initial observations.

4.1 The UK CPI Dataset

The dataset utilised for this project comes from the Office of National Statistics (ONS). The ONS is the official producer of statistics for the UK. The dataset used is titled 'Consumer Prices Index including owner occupiers' housing costs (CPIH): time-series'. As the name suggests, it's a time series data set starting from 1988-2022 and features the monthly CPI(H) of a variety of goods and services listed below.

Each feature is broken down into its' sub-features. For the implementation of the algorithm, I decided to only use the main features, which are a summary (calculated using the geometric mean) of the sub-features.

The features I used for implementation are listed below.

After establishing the data to be used, I debated whether to use COVID-19 data (2019-2021) within my prediction. I ultimately decided to remove it from the dataset as the ARIMA model ARIMA model assumes constant variance. To maintain consistency, I removed it for the LSTM and Random Forest implementations also.

The below plots illustrate the CPI data before and after the removal of Covid-19 data.

There is a clear spike after 2020, probably influenced by COVID-19 and other factors such as the Ukrainian War.

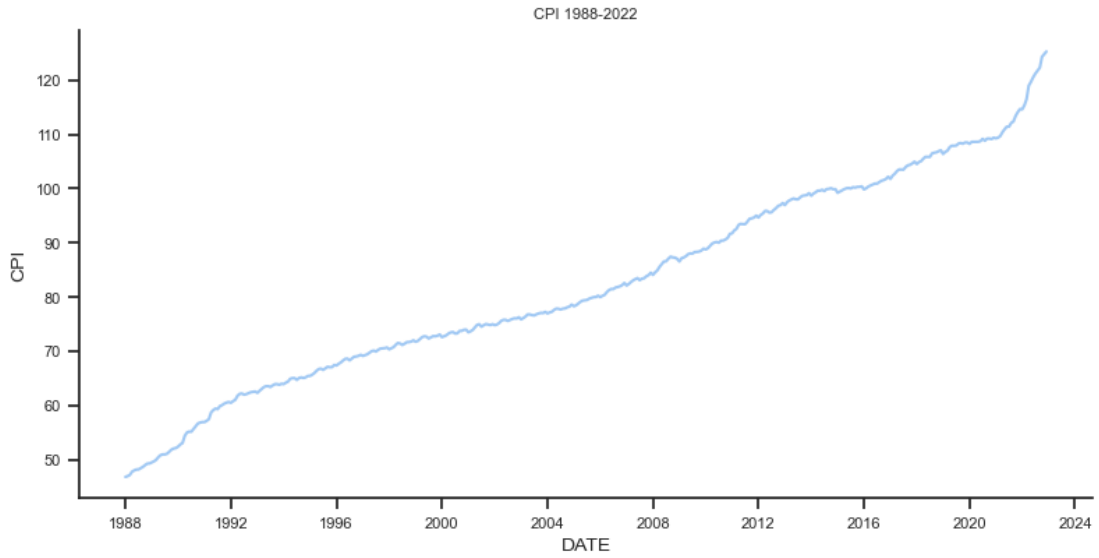

Figure 4.1: CPI Data 1988-2022

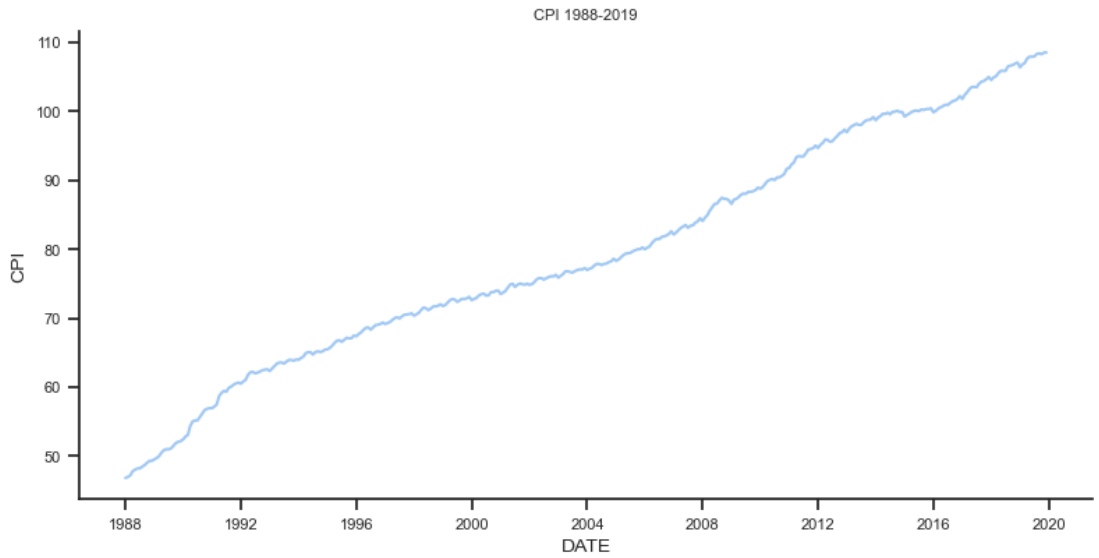

Figure 4.2: CPI Data 1988-2019

Features
CPIH All Items (Monthly CPI)
Food and Non-Alcoholic Beverages
Alcoholic Beverages and Tobacco
Clothing and Footwear
Furniture, Household Equipment and Maintenance
Health
Transport
Electricity, Gas and Other Fuels
Communication
Recreation and Culture
Education
Restaurant and Hotels
Misc. Goods and Services

Table 4.1: CPI Dataset Features

4.1.1 Visual Analysis of Key Variables

The following graphs will show a breakdown of the original CPI data by its' features.

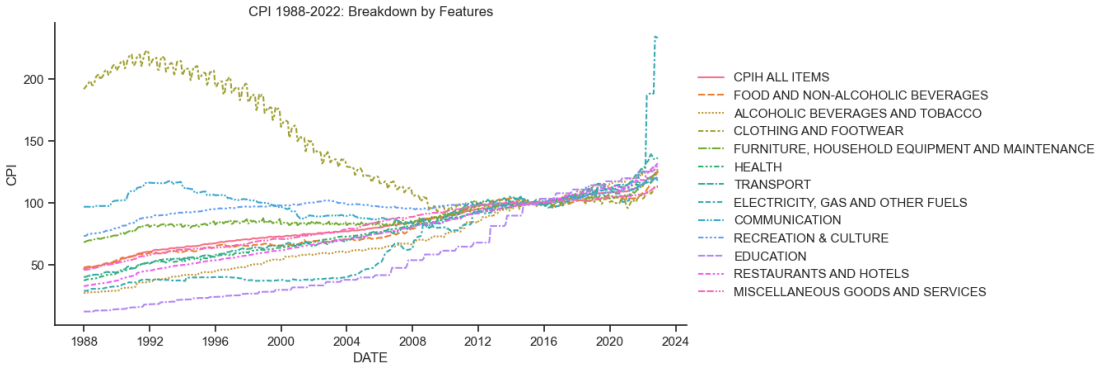

Figure 4.3: CPI Trend By Individual Features

Key Observations:

1. The CPI of 'Clothing and Footwear' experienced a much higher CPI than all other features between 1998-2008, and after 1992, appears to be the feature with the largest CPI decrease over time.

2. The 'Electricity, Gas and Other Fuels' feature, which experiences fluctuations over time, experiences a dramatic increase during 2022. This could be attributed to the Russia-Ukraine War, which resulted in Oil and Gas prices rapidly increasing as Russia is one of the largest oil exporters in the world.

3. There is less variety in the CPI of each feature after 2008. This could be because of the 2008 financial crisis.

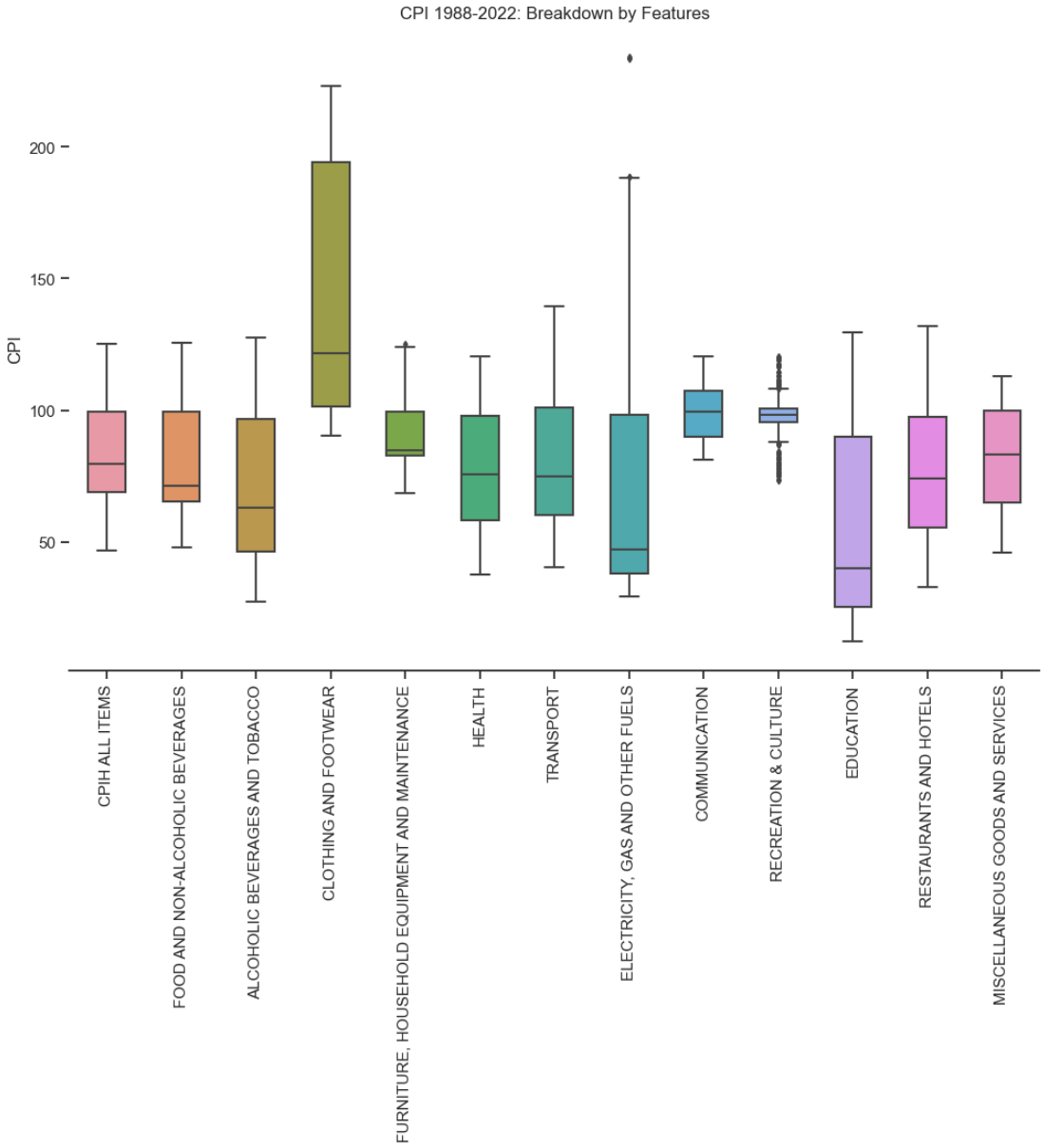

Figure 4.4: CPI Boxplot of Individual Features

Key Observations:

1. 'Clothing and Footwear' has the largest IQR, implying the most variability between the data points.

2. The 'Electricity, Gas and Other Fuels' feature has the largest difference between the maximum value and the third quartile, probably due to reasons suggested in the previous graph.

3. 'Recreation and Culture' contain the most outliers and the smallest IQR.

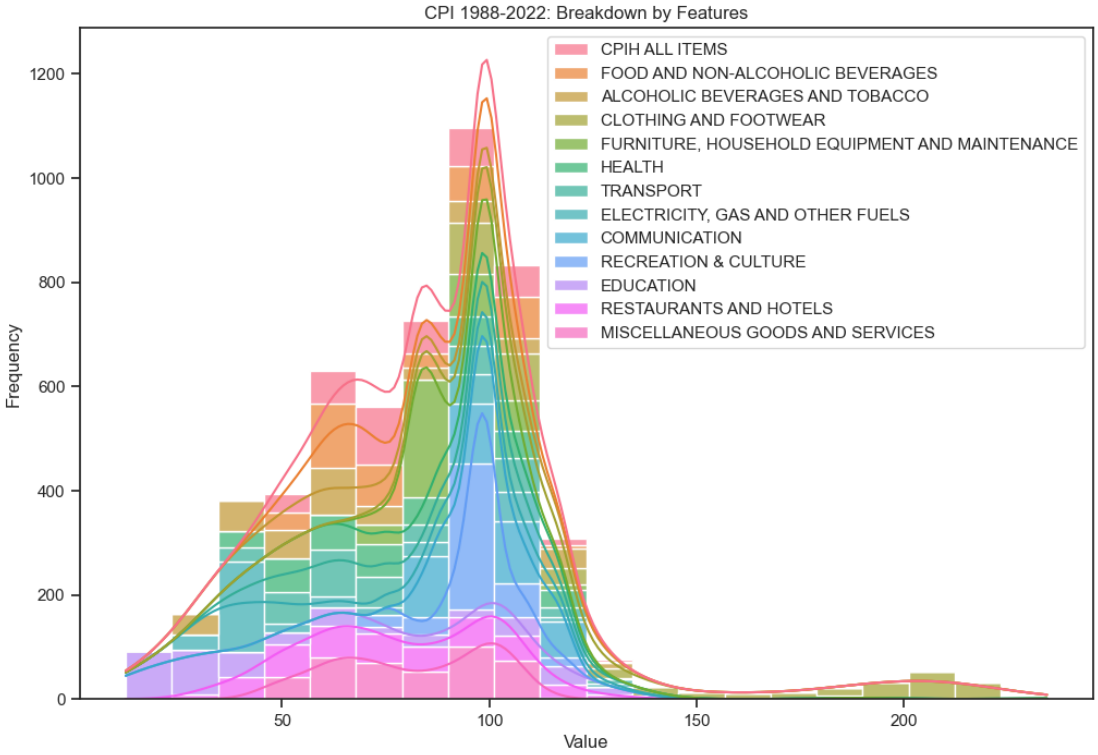

Figure 4.5: CPI Histogram of Individual Features

Key Observations:

1. The central tendency of most features appears to be around 90-100.
2. 'Food and Non-Alc Beverages', 'Alcoholic Beverages and Tobacco', 'Clothing and Footwear' and 'Furniture, Household Equipment and Maintenance' follow the distribution of the average CPI the most.
3. None of the features follows a normal distribution, meaning statistical forecasting methods that rely on this property cannot be used.
4. Clothing and Footwear could have potential outlier values as they fall outside the main range of data.

Final Points

1. The dataset had no NAN values in any of the columns.
2. The 'Footwear and Clothing' column was removed before implementing LSTM and Random Forest (ARIMA is a uni-variate model, so it only utilises the core CPI Data).

4.2 Time Series Decomposition

The (S)ARIMA models utilise a time series decomposition statistical technique. This technique splits time series data into components such as trend, seasonal variation, and residuals.

The trend represents the long-term pattern of the time series, the seasonal component represents a repeating pattern within the data, and the residual component represents randomness - variation in the series that cannot be explained by the trend or seasonal components Hyndman and Athanasopoulos 2018.

`sm.tsa.seasonal_decompose` and `statsmodels.tsa.seasonal import STL` were the libraries I used for time series decomposition. I will refer to them as Classical Decomposition and Seasonal and Trend Decomposition using Loess (STL) respectively.

4.2.1 Classical Decomposition

For my first attempt at decomposition, I used the additive model of decomposition. The additive model is a model that suggests that a time series is a function of the sum of its' trend (T), seasonal (S), and residual components (R) (ibid.):

$$Y = T + S + R \tag{4.1}$$

where Y is the CPI data.

Trend (T) is calculated using a moving average. Seasonal (S) is calculated by detrending the data, and the residual (R) is the remaining data unaccounted for by the trend and seasonal components.

The trend is shown as expected. CPI gradually increases in a (relatively) linear fashion.

My main takeaway from this graph was scepticism about the seasonal component. It confused me at first - I assumed that economic data should not show a constant seasonal pattern like this. CPI Inflation does not depend on regular intervals compared to tourists to a holiday destination during summer versus winter for instance. Upon further research, I discovered that classical decomposition assumes that data provided to the model is stationary and that the seasonal component is constant yearly (ibid.). The CPI data provided to the model exhibit non-stationary behaviour (the statistical properties change over time) which could suggest why the classical decomposition seemed inaccurate.

This led me to implement other decomposition methods, mainly Seasonal and Trend decomposition using Loess (STL) and analyse their results also.

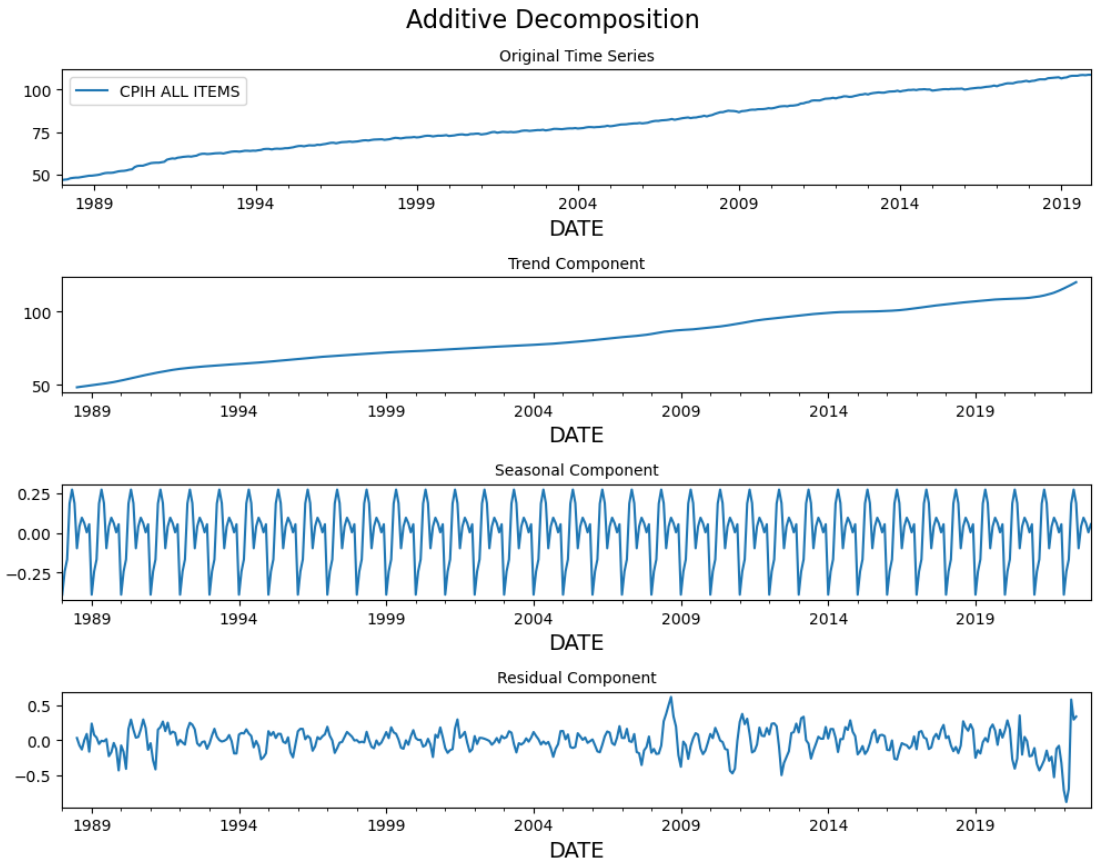

Figure 4.6: Additive Decomposition

4.2.2 Seasonal and Trend decomposition using Loess (STL)

STL differs from classical decomposition with regard to Classical decomposition in many ways. It is a non-parametric method based on locally weighted regression, allowing it to process non-stationary time series data, compared to classical decomposition which assumes stationarity. This also means that it can handle seasonal component that is not constant, as is the case for my data (ibid.).

The main point of interest in this graph is the seasonal component. It exhibits different behaviour over time and is no longer dependent on constant, yearly periods. There is more insight into the data's various seasonal, cyclic nature, which better describes economic conditions over time, rather than a constant seasonality suggested by the classical decomposition model. The peaks of seasonality are positive from 1989-1999 and have negative peaks after 1999.

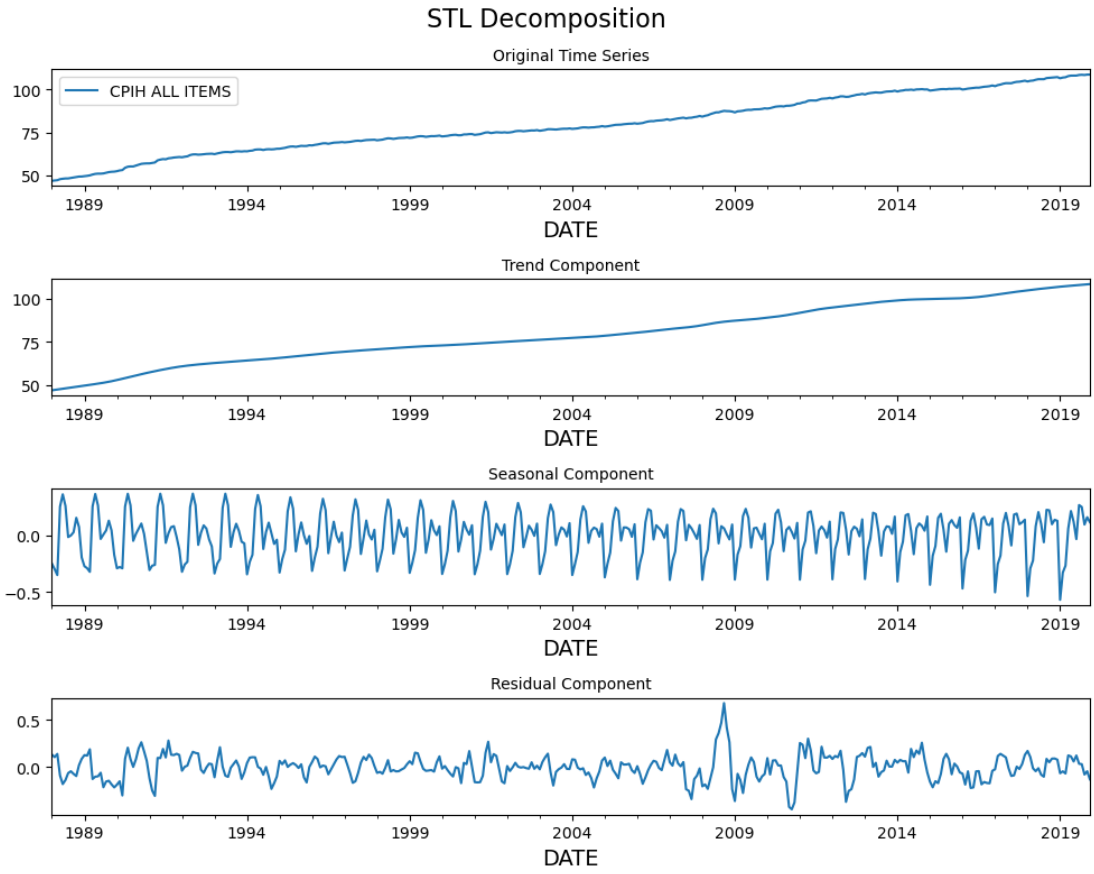

Figure 4.7: STL Decomposition

Chapter 5

Multi-variate Models: Implementation

5.1 LSTM

LSTM was the first model implemented for the project, and the performance of the algorithm was used as a baseline to compare to the other models.

5.1.1 Preparing the Data

Data Scaling

Scaling is important to ensure the effectiveness of gradient descent; which, as discussed is a method to update the model parameters. Scaling is an important feature when using gradient descent as it ensures that steps are updated at the same rate for all the features. The CPI dataset used has similar ranges for each feature, and I implemented standardisation as a precaution. It is most commonly used when features of datasets vary in their scale, such as different magnitudes or units of measurement.

```
# Scale the Data
scaler = StandardScaler()
scaler = scaler.fit(cpi_data_pre_covid)
df_scaled_cpi = scaler.transform(cpi_data_pre_covid)
```

Figure 5.1: Standardisation Process

I used `StandardScaler` to scale the data. It standardises features within a dataset by removing the mean and scaling to the unit variance, and therefore re-scaling the distribution of data points to a Gaussian distribution (a bell curve),

providing a mean of 0 and a standard deviation of 1. A data point is standardised using the following equation:

$$y = (x - u)/s \tag{5.1}$$

where y is the standardised value, x is the original value, u is the mean, and s is the standard deviation. Note that subtracting the mean from the data point is a technique called centering.

`scaler.fit()` is used to calculate the mean and standard deviation of the dataset, and `scaler.transform()` performs the standardisation process by centering and scaling the data.

Sliding Window

As this is a multivariate model, I used a sliding window approach to transform the data into samples of past and future input observations, with n_past = 4 and n_future = 1 being used to transform the data. This means that if given the past 4 months' values, we need to forecast the next month's values. This enabled me to effectively set up my data for testing and training.

5.1.2 Model Architecture

The LSTM model I created was relatively simple. This was influenced by Gu, Kelly and Xiu 2020, where asset prices were forecasted using machine learning, and found that increasing the number of hidden layers did not necessarily improve the accuracy of forecasts. The model was built with the `Sequential` from `TensorFlow`.

Rectified Linear Unit Function (ReLu)

I activated each with the ReLu function, which is a non-linear function. The input is outputted directly if its' value is positive, otherwise, the output is zero (Mercioni and Holban 2020):

$$f(x) = max(0, x) \tag{5.2}$$

ReLu is a relatively simple function, meaning that it does not require much processing power to compute, contributing to a reduction in overall training and testing time compared to other activation functions. It has the following advantages (ibid.):

- Representational Sparsity: A form of regularisation, meaning it is capable of outputting a zero value. This is useful as it simplifies the model as it learns

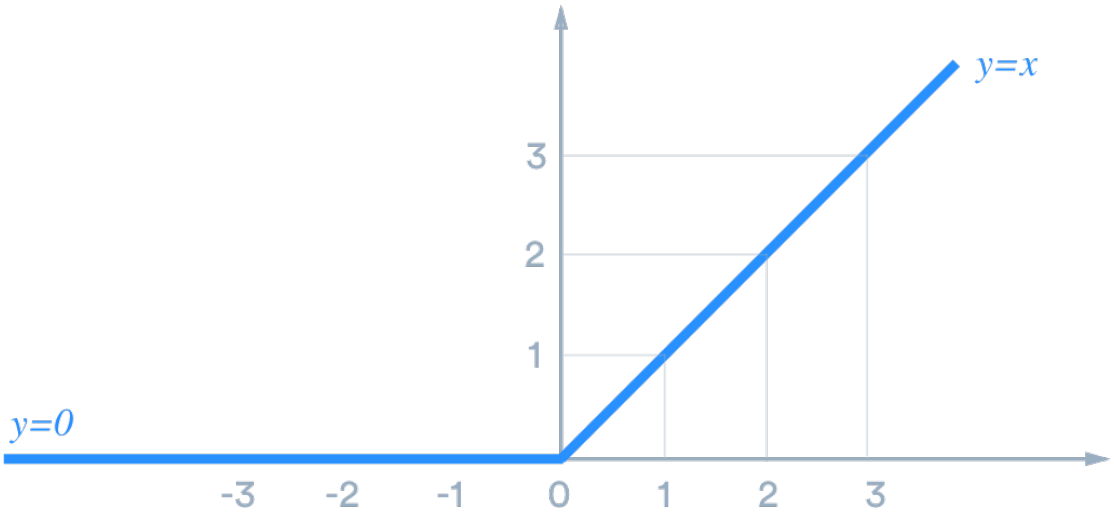

Figure 5.2: Graphical Representation of ReLu

what parameters can be dropped, as well as contributing to lower memory usage.

- Simple Computation: as mentioned, ReLu is a simple function. Namely, it has a constant derivative (1) for positive inputs, reducing model learning time and minimising errors.

Some disadvantages of ReLu include (ibid.):

- Exploding Gradient: This is a phenomenon that occurs when there are large differences between weight updates after an accumulation of gradients. This causes instability during learning and convergence towards global minima.
- Dying ReLu: This occurs when neurons constantly output zero as they are stuck in a negative space. This means the neuron is unlikely to recover.

ReLu is a preferred activation function over tanh and sigmoid as they both contribute to the vanishing gradient problem, which occurs during back-propagation, which is discussed in the earlier chapter.

LSTM Layer

I used 2 LSTM layers with 64 units for my model, the first and fourth layers. Details of the LSTM layer are discussed in the previous sections. The argument `return_sequence` was set to True for the first LSTM, meaning that the full sequence is returned from the output sequence, enabling the following RNN layers to have the full sequence as the input (including the hidden states of all time steps, resulting in a 3D array of real numbers). The same argument was set to False for the final LSTM layer meaning that only the last output of the output sequence is returned.

Dense Layer

I used 2 Dense layers for my model, my second and sixth (final) layers, with units of 32 and 1 respectively, which define the output of the layer. Dense layers are commonly used layers in any neural network model and are often called fully connected layers. Neurons within a dense layer are connected to those in the previous layer, with the relationship established via matrix-vector multiplication within the dense layer. This process reduces the dimensionality of the output of the dense layer and eventually provides a single output from the model when the Dense unit is set to 1.

Dropout Layer

Two dropout layers were used for my model, as the third and fifth layers, with arbitrary values of 0.25 and 0.3 respectively. Dropout Layers are used to reduce the overfitting of neural networks. Overfitting occurs when the error on training data is small, but unseen data produces a large error. It essentially means the model only knows how to perform well on training data, and does not generalise well to new data. Dropout reduces overfitting by dropping nodes within layers in a neural network. Forward and backward connections of dropped nodes are also removed in the process, resulting in a new architecture of the overall network. The parameters of 0.25 and 0.3 represent the fraction of input units to drop at each step during training.

Model Compiling

The final step before training a model, and the final step for creating a model, is the model compilation. This step defines the optimiser and loss function. Optimisers are algorithms to optimise the weights and learning rates of neural networks in order to reduce loss. I used the Adaptive Moment Estimation (Adam) optimiser for my model, which extends stochastic gradient descent (Kingma and Ba 2014). Adam combines the benefits of two other stochastic gradient descent-type algorithms (ibid.) :

- Adaptive Gradient Algorithm (AdaGrad): Ability to auto-tune the learning rate, rather than relying on a default value.
- Root Mean Square Propagation (RMSProp): Auto-tuning of learning rates via the average first moment (mean) of gradients, resulting in a strong performance with noisy problems (non-stationary data)

Adam adapts parameter learning rates using the average of second moments (variance) of the gradients. This process takes place via the calculation of the

exponential moving average of the gradient and the squared gradients, and using them to re-scale the learning rate for each parameter. Adam optimisation is a highly favourable method, due to its ability to perform well quickly (ibid.).

The loss parameter in the compile method defines the loss function to be used during training, which measures how well the model performs during training. The loss function used for my model is the mean squared error (MSE) (Goodfellow, Bengio and Courville 2016):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5.3}$$

where:

- n = The total number of samples in the dataset
- y_i = The actual target value of the i-th sample
- $\hat{y}_i$ = The predicted value of the i-th sample

The MSE function has significant advantages (ibid.):

- Suited to regression tasks: Large errors are penalised more heavily than smaller errors, which is ideal as the magnitude of error is important in regression problems, as the goal is to predict a continuous output value.
- Suitable for gradient-based optimisation algorithms (Adam): The MSE is a differentiable function, which means it can be used with algorithms that utilise the gradient of the loss function.
- Convenience: The MSE is a convex function, meaning that it has a global minimum, this results in easier optimisation of the loss function during training.

The mean squared error (and loss) values defined in the metrics parameter will be discussed in the next section.

```
model = Sequential()
model.add(LSTM(64,activation='relu', input_shape=(trainX.shape[1],trainX.shape[2]), return_sequences=True))
model.add(Dense(32, activation ='relu'))
model.add(Dropout(0.25))
model.add(LSTM(64, activation='relu', input_shape=(trainX.shape[1],trainX.shape[2]), return_sequences=False))
model.add(Dropout(0.3))
model.add(Dense(trainY.shape[1]))

model.compile(optimizer='adam', loss='mse', metrics=['mean_squared_error'])

model.summary()
```

Figure 5.3: LSTM Model Architecture

5.1.3 Training Performance

After fitting my model to the training data, I measured its' performance by checking for overfitting. Overfitting is when a model performs well on the training data, but poorly on unseen validation data. It usually occurs when a model is too complex compared to the data it's trained on and starts to memorise noise and outliers within training data rather than learn the underlying patterns as a result.

Loss Performance

The training loss against validation loss shows how well the model generalises to new data during training. If the training loss is much lower than the validation loss, this means the model is overfitting to the training data, and is unable to understand patterns in the data.

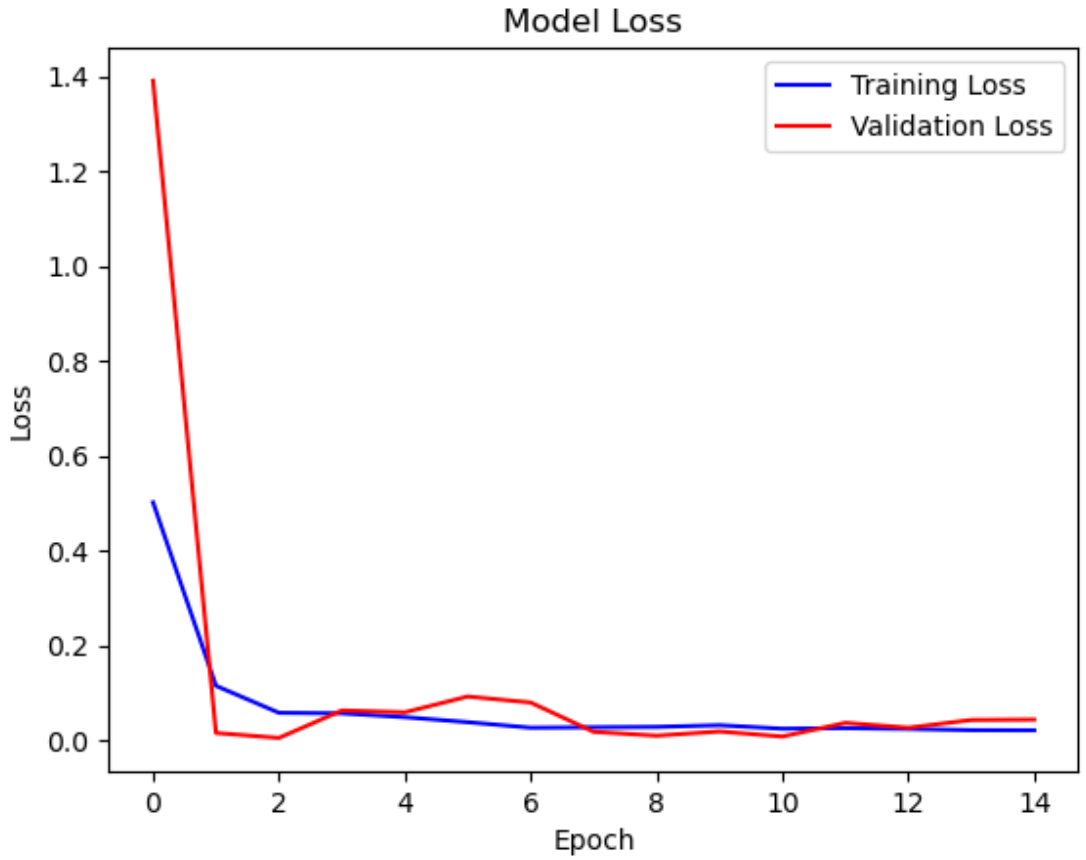

Figure 5.4: Training and Validation Loss Graph

This graph suggests that the model is generalising well to unseen data, albeit with a slight overfit, however not significant enough to re-examine the structure of the model. The training and validation loss both converge to around the same rate after 15 epochs.

Mean Squared Error Performance

As discussed, the MSE measures the difference between the predicted and actual values. If the training MSE decreases fasters than the validation MSE after each epoch, this implies the model is overfitting to the training data. An equal, steady decrease between training and validation MSE implies good model performance as it is learning patterns and improving performance on unseen data.

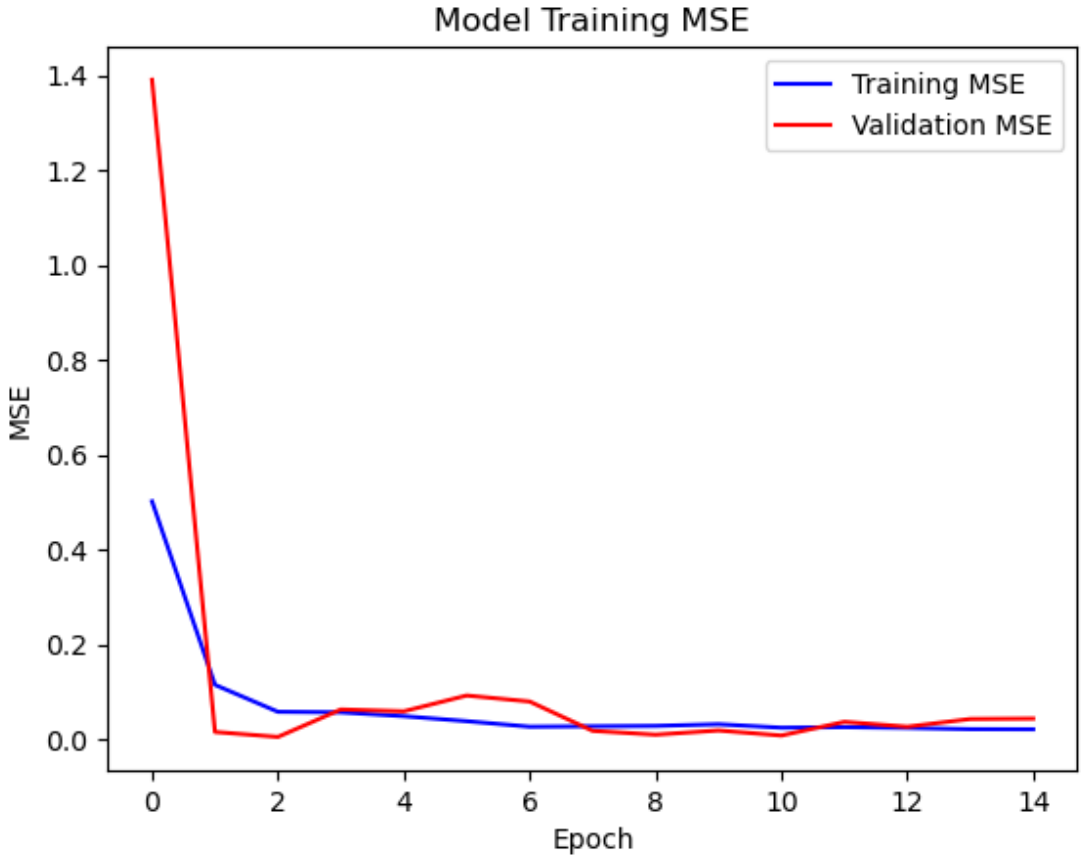

Figure 5.5: Training and Validation MSE Graph

Similarly to the Loss graph, the MSE suggests a good model performance as the training and validation MSE both decrease to the same value after 15 epochs.

5.2 Random Forest

This chapter will contain a breakdown of the implementation of the random forest model for my dataset, It will involve discussing the parameters, how they were chosen, and an evaluation of its' performance.

5.2.1 Model Architecture

The model was built using the `RandomForestRegressor` class from `sklearn.ensemble`. In comparison to the LSTM model, much less preparation was required before building the model, i.e. no scaling of the data is required due to the tree-based model of Random Forest. Two Random Forest models were built for this project, one with the default parameter values (standard model), and one with hyper-parameter tuned values.

Model Parameters

The default parameters are defined as follows (Pedregosa et al. 2011):

- **Bootstrap = True**: Bootstrapping, a technique discussed in earlier sections, is used. This means multiple samples of the dataset are taken with replacement.
- **ccp alpha = 0.0**: This is a parameter to control overfitting. The complexity parameter (alpha), controls the 'Minimal Cost-Complexity Pruning' which is a technique to avoid overfitting by easing the generalisation process of decision trees. It iteratively prunes trees by removing sub-trees that lead to a smaller increase in error rate on the validation data. A value of 0 means no pruning happens.
- **criterion = 'squared error'**: This measures the quality of a split, which is the process where a node in a tree is divided into two or more sub-nodes. 'squared error' suggests the quality is measured by the mean squared error, where the variance is reduced as a feature selection.
- **max depth & max-leaf nodes = None**: max depth represents the maximum depth of the tree, 'None' means nodes are expanded until no further expansion is possible. max-leaf nodes = 'None' mean there are an unlimited number of leaf nodes.
- **max features = 1**: This means that 1 feature is considered when achieving the best split.
- **max samples = None**: The number of samples to train each base estimator. 'None' means that the shape of the dataset is taken (372 training points)
- **min impurity decrease = 0**: This means that a node will be split if there is a decrease in impurity greater (or equal to) 0. Impurity is a measure of how well the tree splits the data.

- **min samples leaf = 1**: This means 1 is the minimum number of samples needed to be at a leaf node. This can prevent a split (at any depth) if there is less than 1 sample in the left and right branches. This assists with smoothing the model.
- **min samples split = 2**: This means that a minimum of 2 samples are required to split a node.
- **min weight fraction leaf = 0**: The weight of each sample, 0 means that each sample has equal weight.
- **n estimators = 100**: There are 100 trees in the forest.
- **n jobs = None**: There are no jobs to run in parallel.
- **oob score = False**: If bootstrap is true, this parameter means there are no out-of-bag samples used to measure generalisation. Out-of-bag samples are samples not included in the bootstrap sample used to build a tree and are usually used to evaluate the performance of the random forest without using a validation dataset.
- **random state = 42**: Randomness parameter for the bootstrapping samples used to build trees.
- **verbose = 0**: This controls verbosity, which is the amount of information outputted during training and prediction of a model.

Hyper-parameter Tuning

In order to potentially improve results, I implemented a Randomised Search (a hyper-parameter tuning method) using `RandomizedSearchCV` in order to find optimal parameters. A randomised search (as the name suggests) randomly searches a specified set of hyper-parameters and returns an estimator with parameters that provide the best score. `RandomizedSearchCV` takes the following parameters (ibid.):

- **estimator**: The Random Forest Model.
- **param distributions**: A dictionary of parameters available to choose from.
- **cv = 3**: The number of splits used for cross-validation, which is a technique used to evaluate the generalisation ability of a model and overall performance. The randomised search will have 3 splits.

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Figure 5.6: Dictionary of Parameters

The dictionary of parameters passed into the 'param distributions' parameter is shown below:

After fitting the random forest model on the training data with parameters from the randomised search, the best parameters found are listed below:

```
{'n_estimators': 1800,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 20,
 'bootstrap': False}
```

Figure 5.7: Best Random Forest Parameters

I found it interesting that Bootstrap was set to false, as it meant the whole dataset was used when building each tree. It also means that out-of-bag samples are used to measure generalisation, which potentially contributed to the relatively slow run time compared to the standard model.

5.2.2 Training Performance

I evaluated the training performance for the random forest models using the R^2 Score.

R^2 Score

The R^2 Score is a statistical measure that evaluates how well a regression line approximates the true data values (Gelman et al. 2019).

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5.4}$$

where:

- y_i is the actual value of the target variable for the ith observation.
- $\hat{y}_i$ is the predicted value of the target variable for the ith observation.

- $\bar{y}$ is the mean value of the target variable.
- n is the total number of observations.

The numerator in the equation above represents the sum of residuals squared and the denominator is the squared value of the sum of the distance each data point is away from the mean.

The R^2 score for both the standard and tuned model is shown below:

Random Forest Model	R^2 **Score**
Standard Model	0.9999552523248721
Hyper-Parameter Tuned Model	0.999999462573409

Table 5.1: Random Forest Training R^2 Score

The closer the R^2 score is to 1, the better the model fits the data. This suggests that both models perform very well on the training data, with the hyper-parameter-tuned model performing slightly better.

Chapter 6

Uni-variate Models: Implementation

I used the uni-variate ARIMA model for my project. Therefore the only column of data that will use to forecast CPI inflation is the past CPI values themselves.

6.1 Stationarity

Stationarity was discussed in Chapter 3. In this section, I will demonstrate the process of making the non-stationary data stationary. We start with a test called the Augmented-Dickey Fuller Test (ADF), as a test for stationarity within a time series.

6.1.1 Augmented Dickey-Fuller Test

The ADF Test was discovered by Dickey & Fuller in 1979 (Dickey and Fuller 1979). In summary, the ADF test is a hypothesis test that utilises T Statistics to detect a unit root within a time series. If there is a unit root, the time series is non-stationary. The null hypothesis suggests there is a unit root, and is rejected at a level of 0.05. i.e. if the ADF test provides a p-value < 0.05 then the time series data is stationary.

In order to implement ARIMA, the data needs to be stationary. Running an Augmented Dickey-Fuller Test on the raw CPI data provides a p-value of 0.83, which suggests our data is non-stationary. The next section will introduce differencing, a method to make non-stationary data stationary.

6.1.2 Differencing

Differencing is a method to remove trend and seasonality from data, both of which were found to be present in the data after time series decomposition in the previous chapter. Referring to the decomposition graph(s), the removal of trend and seasonality leaves us only with the residual component of the data. (Lag-n) Differencing essentially subtracts the current observation in the time series from the n-th previous observation. I used lag-1 differencing for my data:

$$\nabla X_t = X_t - X_t - 1 = (1 - B)X_t \tag{6.1}$$

where B is the backward shift operator,

$$BX_t = X_t - 1 \tag{6.2}$$

∇ is the difference operator, and n=1 (Brockwell and Davis 2002).

I applied this difference operation twice to my data, as 1st differencing (∇^1) resulted in a p-value of 0.054 in the ADF test. After 2nd differencing (∇^2), the p-value was 0, suggesting the data is now stationary.

Below shows a graph of the same data in terms of its' stationary residuals after 2nd differencing.

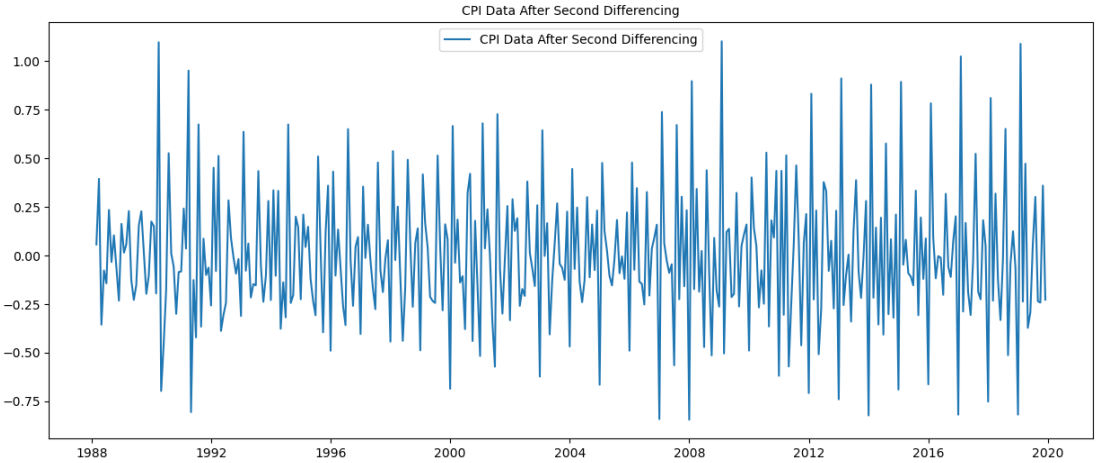

Figure 6.1: CPI Data After Second Differencing

Consequently, when fitting the ARIMA(p,d,q) and SARIMA(p,d,q)x(P,D,Q) models, we know that d, the degree of differencing = 2.

The next section will show how the p,q,P,Q parameters were chosen.

6.2 Hyper-Parameter Tuning

To find the optimal parameters for the (S)ARIMA models, I utilised `auto_arima` from the `pmdarima` library. This is essentially a grid search to find the optimal orders for the model. The optimal order is determined by the set of parameters with the minimal Akaike Information Criterion (AIC) value, which evaluates how well a model fits the data based on its prediction error. AIC is calculated using the following equation (Brockwell and Davis 2002):

$$AIC = 2K - 2\ln(L) \tag{6.3}$$

where:

- K is the number of independent variables.
- L is the likelihood estimate, which measures how well the model fits the data, a higher value implies a better fit.

Tuning the parameters for ARIMA and SARIMA models provides the following results:

Model	p	d	q	P	D	Q	AIC
ARIMA	0	2	1	0	0	0	37.030
SARIMA	4	2	1	2	0	2	-337.828

Table 6.1: (S)ARIMA Model Parameters

I then fit my models using these parameter values on the training data. The next section will show the models' performance on the testing data.

6.3 Training Performance

I evaluated the optimal ARIMA and SARIMA model's training performance using the `plot_diagnostics()` function from `pmdarima`. This function breaks down the performance into 4 plots and allows for the investigation of unusual behaviour from the models. A linear model (such as the ARIMA models) has the following assumptions (Peña and Slate 2006):

1. **Linearity**: The relationship between the (CPI) data and the mean of residuals is linear.
2. **Homoscedasticity**: The variance of the residuals is constant.
3. **Normal Distribution**: The residuals follow a normal distribution.

4. **Independence**: The residuals are independent of the data and display no trend or pattern.

Below shows the diagnostics of the ARIMA and SARIMA models respectively (Vincent n.d.):

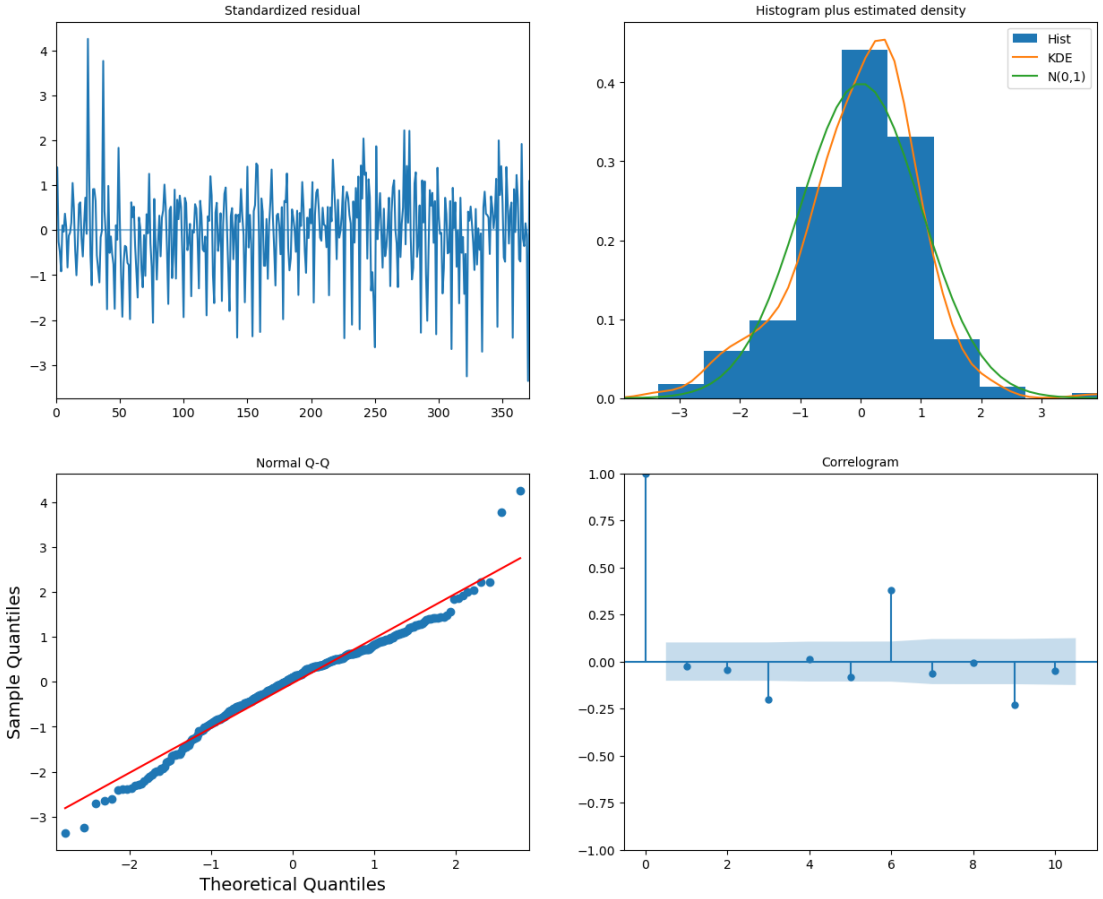

Figure 6.2: ARIMA Diagnostic Plot

This function breaks down the performance into 4 plots:

1. Standardised Residual Plot

A residual is the difference between observed and predicted data points and shows the error of prediction. A standardised residual is the residual divided by an estimate of the standard deviation and helps identify outliers. Both the ARIMA and SARIMA models display no pattern within the standardised residuals and have a linear mean of 0. Both linear models can be seen as a good fit because the standardised residual plot follows the assumption of homoscedasticity, independence, and linearity, as the mean of both residual plots is 0.

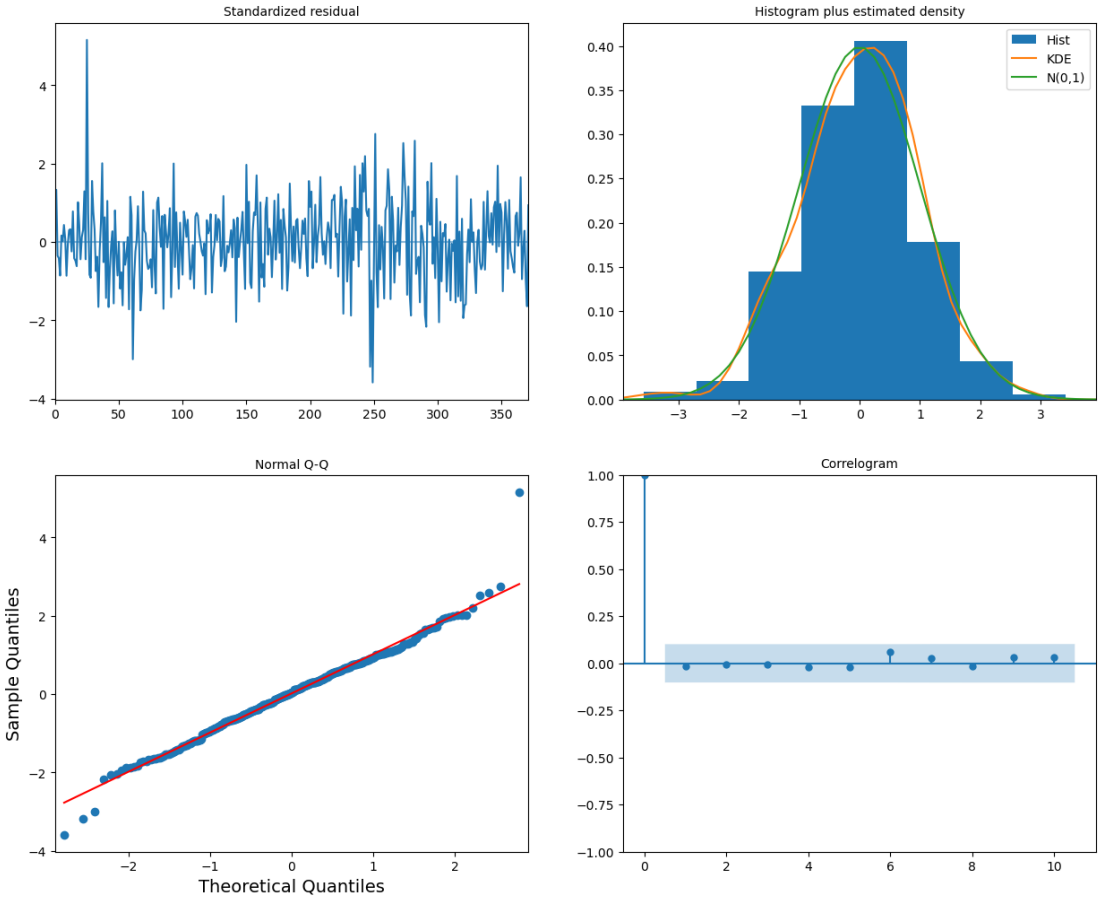

Figure 6.3: SARIMA Diagnostic Plot

2. Histogram Plus Standardised Density

This is a combination of two plots and shows the residuals' distribution. The histogram shows the distribution of the residuals, and the kernel density estimate (KDE) shows an estimate of the expected (probability) density of the residuals, using a kernel function at each point within the range. The KDE is denoted by the orange line, and is under the assumption of a normal distribution, the green line. This means that the closer the KDE line follows the normal distribution, the better a fit the model is to the data. The KDE plot for the SARIMA model fits the normal distribution better than the ARIMA model, indicating that the SARIMA model is a better fit for the data, although the ARIMA model still performs fairly well, and confirms that residuals for both plots roughly follow a normal distribution.

3. Normal Q-Q Plot

This plot is another representation of the distribution of residuals against a normal distribution. The normal distribution is denoted by the red line. The SARIMA model follows the normal distribution better than the ARIMA model as most of the residual points lie on the normal distribution line, with a few variations at either end. In comparison, the ARIMA model slightly deviates from the normal distribution, which is also suggested in the histogram plus density plot. Therefore, the SARIMA model seems to better fit the data.

4. Correlogram

A correlogram is a way to visualise autocorrelation in time series data by summarising correlations at different lags. Autocorrelation (in this instance) occurs when there is a correlation between residuals, resulting in residuals that are not independent, which is a requirement for residuals of a linear model). Significant autocorrelation in the correlogram is denoted by peaks that occur outside of the shaded blue area.

For example, the ARIMA model shows autocorrelation peaks at equal lags, t=3,6,9 indicating that there is a seasonal correlation/trend within the residuals that are not captured by the model. This intuitively makes sense, as a pure ARIMA(p,d,q) model (as discussed) fails to capture the seasonal component of data. Alternatively, the SARIMA model shows no seasonal autocorrelation peaks (or any peaks for other reasons) within the residuals, implying that the SARIMA model is a better fit for the data, as the residuals follow the assumption of independence more closely.

Chapter 7

Results

7.1 Error Metrics

This section will outline the error metrics used to evaluate model performance.

7.1.1 Mean Squared Error (MSE)

The MSE is a metric used to measure the squared difference between predicted and actual values of a continuous variable (making it useful for regression tasks), and helps evaluate the accuracy of a regression model (Plevris et al. 2022):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{7.1}$$

where:

- n is the total number of samples in the dataset
- y_i is the actual target value of the i-th sample
- $\hat{y}_i$ is the predicted value of the i-th sample

The purpose of squaring the values means that large errors are penalised more than small errors, which helps identify outliers.

7.1.2 Root Mean Square Error (RMSE)

The RMSE serves the same purpose as the MSE, with the difference being it has the same unit of measurement as the dependent variable (CPI data) ibid.:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{7.2}$$

where:

- n is the total number of samples in the dataset
- y_i is the actual target value of the i-th sample
- $\hat{y}_i$ is the predicted value of the i-th sample

7.1.3 Mean Absolute Error (MAE)

The MAE takes the absolute value of the difference between the predicted and actual values and is less sensitive to outliers than MSE and RMSE ibid.:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{7.3}$$

where:

- n is the total number of data points in the dataset
- y_i is the actual value of the dependent variable for the i-th data point
- $\hat{y}_i$ is the predicted value of the dependent variable for the i-th data point
- $|\cdot|$ denotes the absolute value operator.

7.2 Results

7.2.1 Multivariate Models

The results of the LSTM and Random Forest models will be discussed in terms of a visual aid of predictive performance against test data and error metrics.

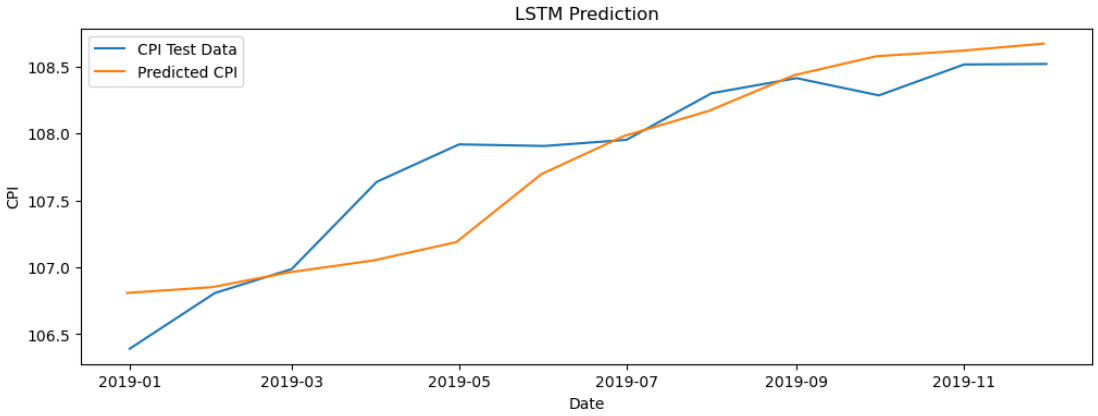

Figure 7.1: LSTM Predictive Performance

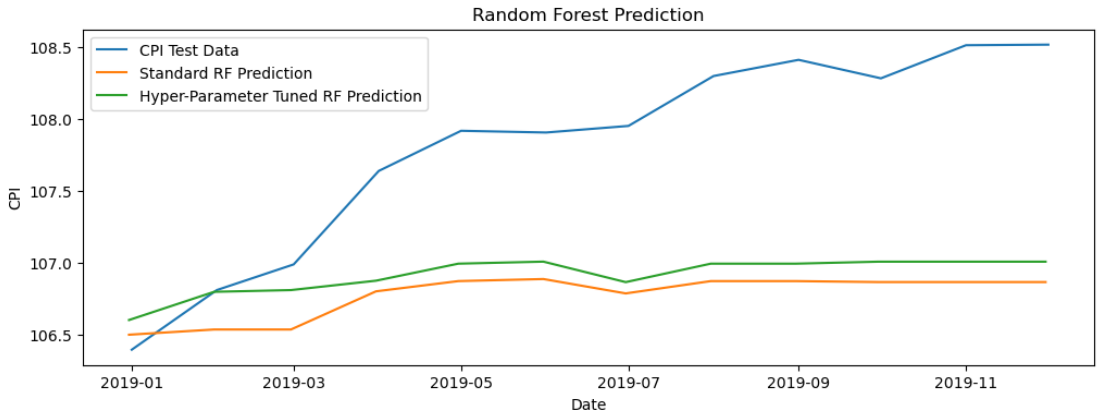

Figure 7.2: Random Forest Predictive Performance

LSTM performs extremely well in predicting the 2019 CPI Inflation rates, with predictive performance improving roughly halfway through, indicated by following a similar pattern to the test data.

In comparison, the random forest model performs poorly with both the standard model (green) and the hyper-parameter tuned model (red), suggesting that the Random Forest model is not a good fit for this prediction task, due to their inability to capture the increasing trend in CPI values over time. From these results, I believe the Random Forest model failed to extrapolate to new data, which could be a limitation of the model itself. Training and testing the tuned model also took a comparably longer time to the standard model, potentially due to the randomised search determining that the bootstrap parameter be set to false. This meant that the size of each decision tree was significantly larger (the size of the dataset). Given that the performance of the tuned model is only marginally better, there is a trade-off when considering the increased runtime compared to the standard model, which leads me to believe that the standard model is a better fit for my data in comparison.

Below shows a comparison of the models in terms of error metrics:

Model	MSE	RMSE	MAE
LSTM	0.10	0.32	0.22
Standard Model	1.36	1.17	1.05
Hyper-Parameter Tuned Model	1.11	1.05	0.92

Table 7.1: Multi-variate Model Error Metrics

LSTM completely outperforms the random forest models, suggesting this model is a better fit for the data. The tuned model performs slightly better than the standard model.

7.2.2 Uni-Variate Models

The results of the ARIMA and SARIMA models will be discussed in terms of a visual aid of predictive performance against test data and error metrics.

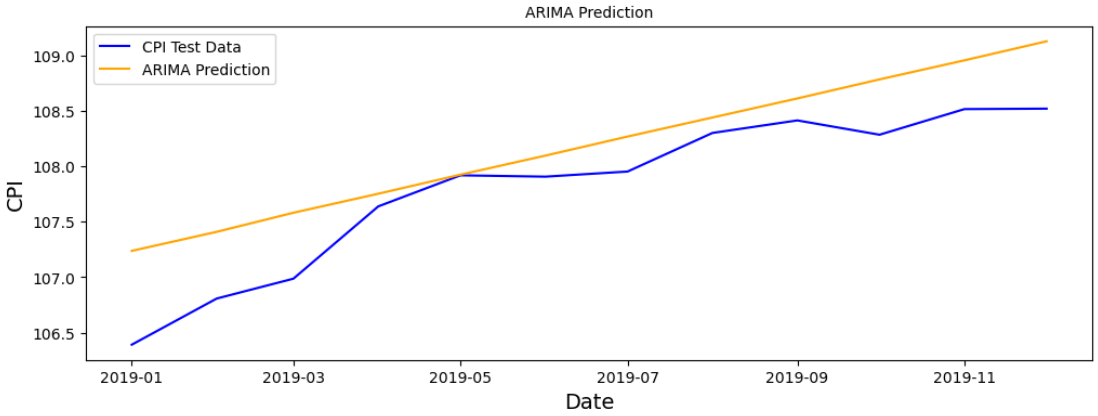

Figure 7.3: ARIMA Predictive Performance

The ARIMA model accurately captures the trend (especially in comparison to the random forest model) but fails to capture the seasonal trend component of the data, which is suggested by the straight prediction line.

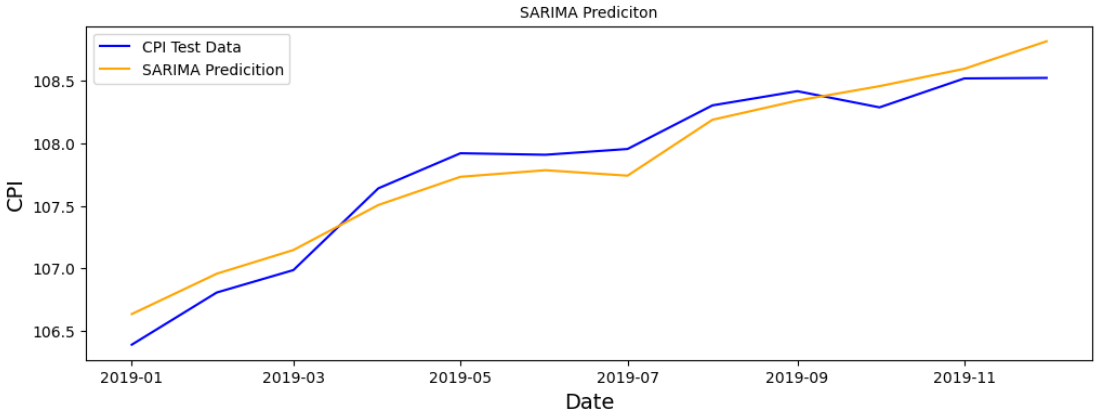

Figure 7.4: SARIMA Predictive Performance

In comparison, the SARIMA model accurately captures the trend and seasonal component of the data, as the prediction accurately follows the pattern of the test data, more so than the LSTM model.

Model	MSE	RMSE	MAE
ARIMA	0.2	0.45	0.45
SARIMA	0.03	0.17	0.17

Table 7.2: Uni-variate Model Error Metrics

These metrics further emphasise that the SARIMA model outperforms the ARIMA model, as the graphs above suggest. The SARIMA model also outperforms LSTM, meaning that the SARIMA model performs the best for predicting the CPI Inflation of the dataset.

Chapter 8

Conclusion

The aim of this project was to forecast the UK CPI Inflation rate using implementations of four different machine-learning algorithms, and following a regression approach. I discussed relevant literature, provided a technical background of each model, explored the CPI Dataset and discussed how it was to be processed to prepare for the models, showed the implementation of each model and their training performance, and evaluated their results. The ARIMA models (an extension of the linear regression algorithm) performed the best in terms of MSE, MAE, and RMSE, followed by LSTM and finally Random Forest.

8.1 Improvements & Future Work

Below are potential improvements to my project:

1. **Larger Dataset**: My models were inevitably limited by the size of the CPI Dataset. I would be curious to measure the performance of these models on a larger dataset, potentially not even related to CPI, perhaps some other economic forecasting tasks with larger available data, i.e. forecasting stock prices.

2. **Different Models**: The Random Forest model performed poorly in retrospect. It would have been interesting to observe how other models perform, namely Monte Carlo and Exponential Smoothing

3. **LSTM Hyper-parameter Tuning**: The standard LSTM model performed well, although the performance could have been enhanced with the implementation of hyper-parameter tuning, similar to the Random Forest model.

4. **Faster Computer**: My computer's power is relatively low, implementation time could have greatly increased if I had more powerful hardware.

Future work could involve exploring the performance across different time horizons, similar to some papers referenced in the literature review. Future work could also include the inclusion of Covid-19 in forecasting, to get a truer representation of the UK Economic climate in terms of CPI inflation.

8.2 Personal Development

My technical and mathematical abilities improved greatly over the course of this project and report, as well as my ability to self-teach. The ARIMA models, in particular, cemented my interest in statistics, with time series decomposition, differencing and stationarity being the concepts I enjoyed learning and implementing the most. I am strongly considering furthering my education in statistics as a result of this project. I also familiarised myself with the TensorFlow libraries and LaTex which I thoroughly enjoyed and appreciate the benefits this will have on my future career.

Bibliography

Asati, Akshita, PGD-Machine Learning and IIIT Bangalore (2022). *A Comparative Study On Forecasting Consumer Price Index Of India Amongst XGBoost, Theta, ARIMA, Prophet And LSTM Algorithms.* Tech. rep. Center for Open Science.

Baybuza, Ivan (2018). "Inflation forecasting using machine learning methods". In: *Russian Journal of Money and Finance* 77.4, pp. 42–59.

Behrens, Christoph, Christian Pierdzioch and Marian Risse (2018). "A test of the joint efficiency of macroeconomic forecasts using multivariate random forests". In: *Journal of Forecasting* 37.5, pp. 560–572.

Breiman, Leo (1996). "Bagging predictors". In: *Machine learning* 24, pp. 123–140.

— (2001). "Random forests". In: *Machine learning* 45.1, pp. 5–32.

Brigid Francis-Devine Daniel Harari, Matthew Keep (2022). *Rising cost of living in the UK.* URL: https://commonslibrary.parliament.uk/research-briefings/cbp-9428/#:~:text=Consumer%5C%20prices%5C%2C%5C%20as%5C%20measured%5C%20by,one%5C%20factor%5C%20causing%5C%20rising%5C%20inflation./.

Brockwell, Peter J. and Richard A. Davis (Apr. 2002). *Introduction to time series and forecasting.* English. Springer-Verlag. ISBN: 0387953515. URL: https://www.proquest.com/books/introduction-time-series-forecasting/docview/38152624/se-2.

Cook, Thomas (2019). "Macroeconomic Indicator Forecasting with Deep Neural Networks". In: *Society for Economic Dynamics.* URL: https://ideas.repec.org/p/red/sed019/402.html.

Dawani, Jay (2020). *Hands-On Mathematics for Deep Learning: Build a solid mathematical foundation for training efficient deep neural networks.* Packt Publishing Ltd.

Dickey, David A and Wayne A Fuller (1979). "Distribution of the estimators for autoregressive time series with a unit root". In: *Journal of the American statistical association* 74.366a, pp. 427–431.

Dietterich, TG et al. (2002). "Ensemble learning. The handbook of brain theory and neural networks". In: *Arbib MA* 2, pp. 110–125.

Farsi, Mohammed et al. (2021). "Parallel genetic algorithms for optimizing the SARIMA model for better forecasting of the NCDC weather data". In: *Alexandria Engineering Journal* 60.1, pp. 1299–1316.

Gelman, Andrew et al. (2019). "R-squared for Bayesian Regression Models". In: *The American Statistician* 73.3, pp. 307–309. DOI: 10.1080/00031305.2018.1549100. eprint: https://doi.org/10.1080/00031305.2018.1549100. URL: https://doi.org/10.1080/00031305.2018.1549100.

Goodfellow, Ian, YB Bengio and Aaron Courville (2016). *Adaptive Computation and Machine Learning Series (Deep Learning)*.

Gu, Shihao, Bryan Kelly and Dacheng Xiu (2020). "Empirical asset pricing via machine learning". In: *The Review of Financial Studies* 33.5, pp. 2223–2273.

Gurunathan, Akila and Batri Krishnan (Sept. 2021). "Detection and diagnosis of brain tumors using deep learning convolutional neural networks". In: *International Journal of Imaging Systems and Technology* 31. DOI: 10.1002/ima.22532.

Hassani, Hossein and Emmanuel Sirimal Silva (2018). "Forecasting UK consumer price inflation using inflation forecasts". In: *Research in Economics* 72.3, pp. 367–378.

Hopfield, J. J. (Apr. 1982). "Neural Networks and Physical Systems with Emergent Collective Computational Abilities". In: *Proceedings of the National Academy of Science* 79.8, pp. 2554–2558. DOI: 10.1073/pnas.79.8.2554.

Hyndman, Rob J and George Athanasopoulos (2018). *Forecasting: principles and practice*. OTexts.

IMF (Jan. 2022). URL: https://www.imf.org/en/Countries/GBR.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kolter, Maloof (2003). "A new ensemble method for tracking concept drift". In: *In Proceedings of the Third IEEE International Conference on Data Mining* 8, pp. 123–130.

Liran Einav, Jonathan D. Levin (2013). "The Data Revolution and Economic Analysis". In: *National Bureau of Economic Research*.

Maehashi, Kohei and Mototsugu Shintani (2020). "Macroeconomic forecasting using factor models and machine learning: an application to Japan". In: *Journal of the Japanese and International Economies* 58, p. 101104.

Mercioni, Marina Adriana and Stefan Holban (2020). "The most used activation functions: Classic versus current". In: *2020 International Conference on Development and Application Systems (DAS)*. IEEE, pp. 141–145.

Meyler, Aidan, Geoff Kenny and Terry Quinn (1998). "Forecasting Irish inflation using ARIMA models". In.

Nikulin, Vladimir, Geoffrey J McLachlan and Shu Kay Ng (2009). "Ensemble approach for the classification of imbalanced data". In: *AI 2009: Advances in Artificial Intelligence: 22nd Australasian Joint Conference, Melbourne, Australia, December 1-4, 2009. Proceedings 22*. Springer, pp. 1–20.

Payne, Chris (2016). *Consumer Price Indices: A brief guide: 2016*. URL: https://www.ons.gov.uk/economy/inflationandpriceindices/articles/consumerpriceindicesabriefguide/2016#consumer-price-indices.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Peña, Edsel A and Elizabeth H Slate (2006). "Global validation of linear model assumptions". In: *Journal of the American Statistical Association* 101.473, pp. 341–354.

Plevris, Vagelis et al. (June 2022). "Investigation of performance metrics in regression analysis and machine learning-based prediction models". In: DOI: 10.23967/eccomas.2022.155.

Pufnik, Andreja, Davor Kunovac et al. (2006). *Short-term forecasting of inflation in Croatia with seasonal ARIMA processes*. Tech. rep.

Riofrıo, Juan et al. (2020). "Forecasting the Consumer Price Index (CPI) of Ecuador: A comparative study of predictive models". In: *International Journal on Advanced Science, Engineering and Information Technology* 10.3, pp. 1078–1084.

Sagi, Omer and Lior Rokach (2018). "Ensemble learning: A survey". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.4, e1249.

Al-Shabi, Mohammed and Anmar Abuhamdah (Dec. 2021). "Using deep learning to detecting abnormal behavior in internet of things". In: *International Journal of Electrical and Computer Engineering* 12, pp. 2108–2120. DOI: 10.11591/ijece.v12i2.pp2108-2120.

Swanson, Norman R and Halbert White (1995). "A model-selection approach to assessing the information in the term structure using linear models and artificial neural networks". In: *Journal of Business & Economic Statistics* 13.3, pp. 265–275.

Vincent, Thomas (n.d.). *A Guide to Time Series Forecasting with ARIMA in Python 3*. URL: https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-arima-in-python-3.

Wu, Chih-Hung et al. (2018). "A new forecasting framework for bitcoin price with LSTM". In: *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, pp. 168–175.

Wulandari, Fitri (2022). *UK inflation rate: Price rises return to 40-year highs as food costs push households to brink*. URL: https://capital.com/uk-

inflation-rate-britain-price-rate-hike#:~:text=The%5C%20UK's%5C%20Office%5C%20of%5C%20National,most%5C%20comprehensive%5C%20measure%5C%20of%5C%20inflation.