# Resilience Down Under: Navigating Four Decades of Unemployment Trends in Australia (1982-2023) to predict and forecast future unemployment trends.

By Justin Grima

## *I. Abstract*

In Australia, over the last four decades, the unemployment rate has experienced significant changes, peaking at 11.15% in December 1992, reaching its lowest recorded point of 3.47% in December 2022, and currently standing at 3.58% as of the latest survey in March 2023. These trends are influenced by various factors, including global economic conditions, domestic policies, and structural shifts in the Australian economy. The Australian Bureau of Statistics (ABS) plays an important role in gathering and analyzing labor market data through a monthly labor force survey of approximately 50,000 Australians aged 15 and above, categorized into three groups: Employed, Unemployed, and Not in the Labor Force. From this data, crucial indicators such as the labor force, unemployment rate, and participation rate are calculated, providing vital insights into employment patterns.

This report focuses on the unemployment rate, a key indicator representing the percentage of the labor force actively seeking employment. It offers valuable insights into the overall economic health of Australia, where a rising rate can signal an economic slowdown or recession, while a decreasing rate implies the opposite. The purpose of this report is to analyze the 'AUS data 2023.xlsx' dataset, which contains aggregated quarterly rates of unemployment and other relevant economic factors from December 1982 to March 2023. The objectives are to: i) address any dataset issues, such as missing values, ii) identify how relevant factors influence the unemployment rate, iii) develop and test various supervised machine learning algorithms (SMLAs) to determine the best-performing algorithm, iv) build and test an Artificial Neural Network (ANN) for predicting unemployment rate, v) compare the performances of the chosen MLM with the ANN model, and vi) provide a recommended model.

The study revealed: i) a few missing data points (both Missing at Random and Missing Not at Random), requiring testing various imputation methods to find the optimal solution, ii) a thorough exploratory data analysis (EDA) showing that some variables (economic factors) influenced the unemployment rate while others exhibited spurious correlation and that other outside factors, not considered in the dataset, may also influence the unemployment rate such as global economic conditions, epidemics and government policies, iii) the decision tree boosting method was the best-performing SMLAs in predicting the Australian unemployment rate, and iv) a successful ANN was built that had slightly superior performance to the decision tree.

In summary, using a subset of the 'AUS data 2023.xlsx' and all variables as training and test data, we successfully developed, trained, and tested a variety of SMLAs and an ANN for predicting the unemployment rate in Australia. The ANN is recommended based on its superior performance. Future considerations may include exploring other SMLAs beyond the scope of this report, conducting deeper hyperparameter tuning, and incorporating additional data to enhance model training and performance, given the relatively small dataset size.

## *II. Brief overview*

Employment forms the basis of a nation's economic productivity, enabling individuals to provide for themselves, their families, and their communities (Australian Institute of Health and Welfare, 2023). In today's rapidly changing world, the uncertainty surrounding the future and its potential impact on a nation's economy is a constant concern in the minds of many. Whether facing progress or setbacks, we, as a society, work to adapt to these situations to the best of our abilities. The emergence of the COVID-19 pandemic, which brought the world to a standstill, exposed weaknesses, and dependencies in ways we

were not prepared for. The labor market, an important indicator of economic well-being, proved no exception to these transformative changes. The Australian unemployment rate, widely regarded as an important indicator for understanding labor market conditions (Reserve Bank of Australia, 2019), saw significant disruption. As the pandemic took hold, the economy endured a significant shift, presenting both challenges and opportunities.

This event serves as just one example of how economic fluctuations can significantly impact both employment and overall economic stability. The global financial crisis (GFC) of 2007-2009, triggered by a downturn in the US housing market, led to mass job losses, resulting in a surge in unemployment rates. This ripple effect was felt worldwide, including in Australia, resulting in recessions of a scale not witnessed since the Great Depression of the 1930s, due to the connection of the global financial system (Reserve Bank of Australia, n.d.). Yet, history has repeatedly demonstrated the remarkable strength of economies. In the face of hard times, we've observed time and again their ability to recover, although gradually. In the most recent economic downturn, exemplified by the COVID-19 pandemic, recent reports from the Australian Bureau of Statistics (ABS) have shown signs of recovery, with the employment rate in Australia rebounding within just one year of the pandemic (Australian Institute of Health and Welfare, 2023). This recovery serves as evidence of the nation's economic adaptability and strength, providing a positive reassurance for the future. However, the lingering question remains: What lies ahead, and how might it impact the global economy, employment, and, ultimately, the ability of individuals to provide for themselves, their families, and their communities?

This report uses the 'AUS data 2023.xlsx' dataset to utilize a selection of supervised machine learning algorithms (SMLAs). These models are thoroughly crafted, refined, trained, and tested to assess their performances and identify the best performance in predicting Australia's unemployment rate. Additionally, we pivot and focus on developing an Artificial Neural Network (ANN) tasked with the same objective. The SMLAs and ANN are then compared to determine the superior predictive model. Through this process, the report provides a thorough assessment of the predictive capabilities of the models for forecasting Australia's unemployment rate. It sheds light on their ability to capture the complex underlying forces of the labor market. Moreover, our analysis brings to light overlooked economic factors not included in the dataset that can additionally be considered significant in comprehending unemployment trends. By engaging in analytical techniques, thoroughly examining the 'AUS data 2023.xlsx,' and providing efficient and effective predictive models, this report aims to assist policymakers, economists, and stakeholders in navigating the complex landscape of the Australian labor market, specifically the unemployment rate, within an ever-evolving global framework.

## *III. Data*

### *a. Pre-processing dataset*

To conduct the study, we will be using the 'AUS data 2023.xlsx" dataset, as stated in the abstract, which contains aggregated and collected quarterly rates of unemployment (response) and other relevant economic factors including i) change in gross domestic product (GDP), ii) government final consumption expenditure, iii) final consumption expenditure of all industry sectors, iv) term of trade index (TTI), v) Consumer Price Index of all groups (CPI), vi) number of job vacancies measured and vii) estimated Resident Population, from December 1982 to March 2023 (See Appendix A for dataset variable descriptions). This dataset has 166 observations and 9 variables: 1 Date variable used as a criterion from splitting the dataset into training and testing data, 1 numerical response variable, and 7 numerical predictor variables (See Appendix A for full variable breakdown). In the initial procedure, before any machine learning machinery can be applied, we are required to complete the very first step, data preprocessing (Data Preprocessing - an Overview | ScienceDirect Topics, n.d.). To begin, we provide better clarification of the variables present in

the dataset by changing the initial names: Year_Q, Y (response), and X1-X7 (response variables), to: "DATE", "UNEMP_RATE", "GDP", "GOV_FCE", "CHNG_FCE_ALL_IND", "TTI", "CPI", "JOB_VACAN", "RES_POP" (See appendix B for raw code).

Following this, we proceed to modify the format of the "DATE" variable to one compatible with RStudio, as required for our following analyses. Initially, the "DATE" variable was structured as '1982_Dec' and was transformed to adhere to R standards, resulting in '1982-12-01'. See Appendix C for the raw code. After this conversion, we proceed with our preprocessing steps by examining the dataset for any missing values (See Appendix D for raw code). From our analysis, there are a total of 11 missing values: 'GDP' – 1, 'GOV_FCE' - 1, 'CHNG_FCE_ALL_IND' – 1, 'TTI' – 1, 'JOB_VACAN' – 5, 'RES_POP' -2 (See Appendix E for full summary). For all variables with missing data, except 'JOB_VACAN' they are assumed to be missing completely at random (MCAR): missing data is independent of the observed and unobserved data (Mack et al., 2018), where the loss was unrelated to any characteristics of the data itself. Regarding "JOB_VACAN", the ABS states that the Job Vacancies Survey (JVS) was suspended during 2008-09 and reinstated for the November 2009 survey causing a gap for five quarters between August 2008 and August 2009 inclusive and the ABS cannot produce reliable estimates by collecting this missing data retrospectively (Statistics, 2010). Therefore, it is missing not at random (MNAR): missing data is related to events or factors that are not measured by the researcher (Mack et al., 2018). With these conclusive results, one method is to remove the 'JOB_VACAN' variable from the dataset, but due to the small size of the dataset and its significance in determining the unemployment rate, we decided to keep it.

To address both the MCAR and MNAR data, we opted for a thorough approach - conducting a sensitivity analysis using multiple imputed datasets. This involves generating datasets for each imputation method and subjecting them to MLMs to see which has the lowest mean squared error score (MSE, performance metric). In our dataset, the response variable is all numerical. Based on this the following imputations conducted in R 'missForest', predictive mean matching using MICE, linear regression - predicted values using MICE, linear regression using bootstrap using MICE, a random sample from observed values, using MICE, and Median imputation (See Appendix F for imputation description and Appendix G for raw code). Upon completing the imputation process, we obtained a total of six imputed datasets. Next, each dataset underwent evaluation using three distinct MLMs: Ridge Regression, Decision Tree Boosting, and Support Vector Machine (SVM) with a radial kernel. This analysis aimed to determine which imputation dataset displayed greater performance, as indicated by the lowest MSE score. (See Appendix H for results). After conducting this imputation experiment, Median imputation had the best performance for 2/3 of the SMLAs: Decision Tree Boosting and SVM with Radial Kernel (See Appendix I for R code example of SMLAs on Median imputation dataset). Based on the results, the dataset with Median imputation is chosen for the rest of the report.

After examining the dataset, it's clear that the predictor variables have a large range of varying values. To ensure effective comparison and analysis, scaling is necessary to prevent any single variable from dominating due to its scale (refer to Appendix J for visualization). Fortunately, SMLAs incorporate scaling options in their algorithms, therefore not requiring the need for preprocess scaling and allowing for scaling to be incorporated during model execution. The next step of this report involves EDA on the median imputation training dataset, therefore requiring a split into training data (Dec 1982 - Dec 2020) with 156 observations, and test data (Mar 2021 - Jun 2023) with 10 observations (see Appendix K for R code). Following the split, the 'DATE' variable is removed. Given its absence in the original variable description, it likely served solely as a splitting criterion, holding no significance in predicting unemployment rates using the later-discussed MLMs and ANN.

### *b. EDA on training dataset from December 1982 to December 2020*

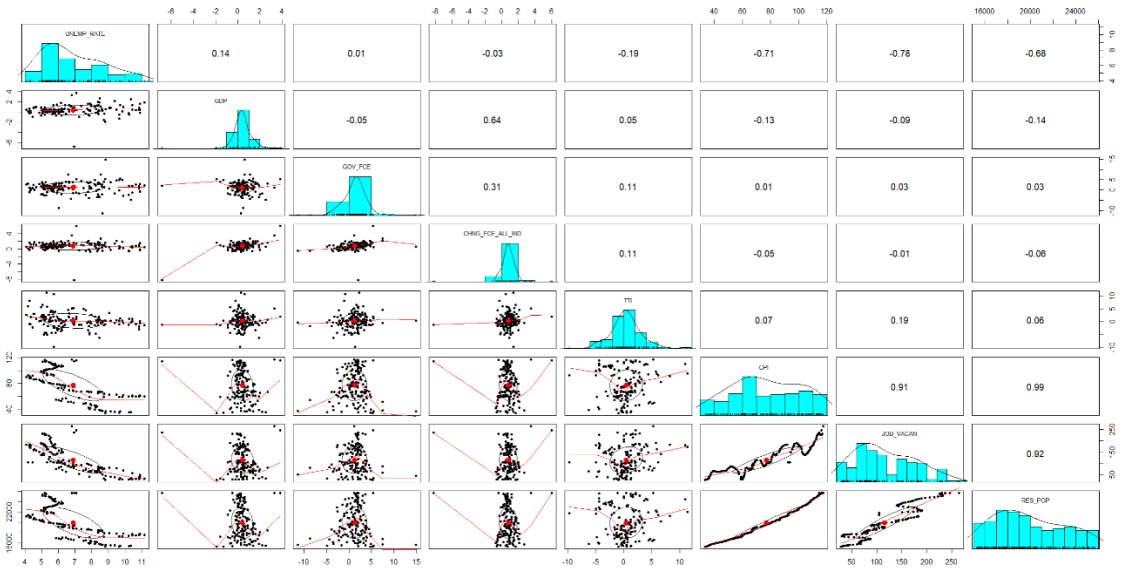



**Image 1:** pairs.panel visualization of variables in the dataset.

In the initial EDA, we explore the 'pairs.panel()' function, which produces a full visualization and summary of variable information, as seen in image 1 (refer to Appendix A for a full breakdown). From image one we state the following observations: 'UNEMP_RATE', 'JOB_VACAN', 'RES_POP' and 'CPI' are positively skewed, 'GDP', 'CHNG_FCE_ALL_IND' are slightly negatively skewed, 'GOV_FCE' and 'TTI' are normally distributed, there is a very low correlation with all predictor variables except for 'CPI', 'JOB_VACAN' and 'RES_POP' whose correlation is very high, and the predictor absolute value correlations to response variable from least to greatest are: 'GOV_FCE' (0.01), 'CHNG_FCE_ALL_IND' (-0.03), 'GDP' (0.14), 'TTI' (-0.19), 'RES_POP' (-0.68), 'CPI' (-0.71), and 'JOB_VACAN' (-0.78).

In our last point, this correlation values the linear relationship between the predictors and the response where positive correlation indicates that as one variable increases, the other tends to increase as well and vice versa for negative values. Furthermore, analyzing scatter plots allows us to understand how predictor variables impact the response variable. This sheds light on how the economic factors within this dataset affect the unemployment rate.

Referring to Appendix L, we initially, we examine the unemployment rate in Australia from December 1982 to December 2020 to discern its trends, which may help clarify other trends seen in future comparisons (Refer to Appendix M for R code). Image 2 displays fluctuations in Australia's unemployment rate over this period, showing a gradual decrease overall, with a notable spike of 5.3 % from 1989 to 1992 because of the process of Australia becoming a nation on 1 January 1901, allowing six Australian colonies to collectively govern as the Commonwealth of Australia (National Museum of Australia, 2019) with a steady decline after with another spike of 1.64% at the start of 2009. This spike is likely associated with a delayed economic response to the global financial crisis and has since remained largely unchanged. Referring to the image, specifically the scatterplots of each predictor variable and the response variable, in combination with the linear correlation values, we note that the percentage changes in gross domestic product, government final consumption expenditure, and final consumption expenditure of all industry sectors exhibit a minimal linear relationship with Australia's unemployment rate in this dataset. Therefore, it's reasonable to conclude that further visual exploration of trends or patterns may not yield significant insights.

Appendix N presents a transformed version of the scatterplots from image 1 into smooth line plots, offering a clearer understanding of the linear relationship between predictor variables: term of trade index (TTI) (percentage), Consumer Price Index (CPI) of all groups, number of job vacancies (measured in

thousands), estimated resident population (in thousands), and the response variable: unemployment rate (refer to Appendix O for raw code). The first plot (top left) illustrates the linear relationship between the CPI of all groups and the unemployment rate. It showcases a clear inverse correlation, indicating that as CPI decreases, the unemployment rate tends to increase. This aligns with the Phillips Curve, which associates lower unemployment with higher inflation and vice versa (DePersio, 2020). When examining the relationship between the estimated resident population and the unemployment rate, a similar pattern emerges. Surprisingly, as the estimated resident population decreases, unemployment increases. This contradicts the expected trend, as smaller populations generally offer more job opportunities, resulting in lower unemployment rates. This anomaly suggests a potential spurious correlation, where observed associations may not imply causation. It implies that there may be hidden, unobserved factors influencing both variables, creating a misleading impression of causation (Spurious Correlation - an Overview | ScienceDirect Topics, n.d.).

In the third image (bottom left), we observe an expected trend: as the term of trade index (TTI) decreases, there is a simultaneous increase in the unemployment rate. This aligns with historical patterns seen during terms of trade booms, where a surge in output prices reduced the real cost of labor for firms, consequently boosting labor demand (Davis et al., 2016). However, in the current scenario, the steady decline in TTI has led to reduced labor demand and increased unemployment. In the final image, we note a decrease in job vacancies correlating with an increase in the unemployment rate. This aligns with expectations, as fewer job opportunities imply more individuals struggling to find work, resulting in higher unemployment. Though I addressed the overall trends in the first two images, it's essential to note the opposite trends observed at the start and end of each graph. This highlights the fact that economic factors and their influence on unemployment rates may not always have a straightforward one-to-one correlation. It's crucial to acknowledge the potential existence of non-linear or intricate relationships beyond simple correlations alone. Furthermore, external factors not accounted for in the dataset—such as government policies, demographic shifts, technological advancements, job shortages in specific sectors, unforeseen events like COVID-19, and fluctuations in interest rates—can significantly contribute to the influence of economic factors on unemployment rates. These complexities highlight the need for a complete understanding of the economic landscape.

## *IV. Machine Learning*

Within the scope of this report, we considered a variety of SMLAs to use for predicting the unemployment rate in Australia. Since the response variable is a continuous numerical value, we will use regression machine learning algorithms to predict the unemployment rate in Australia. For this, we will use the following SMLAs: Ridge Regression, Lasso Regression, Decision Tree Bagging, Decision Tree Random Forest, Decision Tree Boosting, and SVM with linear, radial, and polynomial kernels. Decision Tree CARTs was considered as an option but due to its significant downfalls mentioned in Appendix P, it is not considered as it is an unreliable and unstable model to use to predict the unemployment rate. The decision to choose these SMLAs is based on what has been presented in our course to exemplify our comprehension. Additionally, all previously mentioned SMLAs have unique characteristics that may prove beneficial to outperforming other algorithms, which makes them all worth investigating (See Appendix P for SMLAs summaries). As previously mentioned, we will use the median imputed dataset that consists of 166 variables, a numerical response variable, and 7 numerical predictor variables, to train and test the performances of each SMLA, and determine the most effective in predicting the unemployment rate in Australia. To do this, we use three performance metrics: mean squared error (MSE), root mean squared error (RMSE), and mean absolute error (MAE) (See Appendix Q for full performance metric descriptions). Utilizing these three performance indicators provides a comprehensive evaluation, ensuring the selection of the most suitable and best-performing model among the SMLAs.

**a. Ridge Regression (See Appendix R for R code and Appendix S for visual algorithms outputs discussed)**

Ridge regression is a type of linear regression that helps when there are many features in the data. It does this by making sure no single feature has too much influence on the outcome. This is done by adding an L2 penalty (regulation term) to the cost function of the traditional linear regression to help prevent overfitting by adding a constraint on the magnitude of the coefficients. Unlike some other methods, Ridge doesn't completely remove any features, but it spreads out their importance more evenly which can be useful when some features are closely related or when there are a lot of them. In this and the preceding SMLAs, before we start training and testing the models, we need to perform tuning of each model's varying hyperparameter. As a general statement, these tuning functions in R allow us to specify a grid of hyperparameter values to search over and use cross-validation to assess model performance for different combinations to ultimately find the best configuration that maximizes the model's performance metrics. For Ridge regression and Lasso regression, the hyperparameters we conduct tuning for is a lambda (λ) value which controls the strength of the penalty applied to the coefficients of the features where a higher lambda value leads to greater regularization and potentially simpler models. In this report, both Ridge and Lasso hyperparameter tuning uses the 'cv.glmnet()' function where the lambda values tested are a grid containing a sequence of numbers ranging from 10^10 to 10^-2, with a total of 100 elements.

After running the hyperparameter cross-validation function, the optimal lambda value that results in the lowest MSE was 0.01, which is our set hyperparameter to conduct the Ridge regression algorithm. Once the model is trained, we use the 'predict ()' function on the training dataset to evaluate how well a model has learned the underlying patterns and relationships within the data. This helps us assess the model's performance before deploying it to make predictions on new, unseen data, which in our case is the test dataset that allows us to observe an unbiased evaluation of a model's performance on new, unseen data. Using the performance metrics mentioned above, the performance of Ridge regression on the training data gave the following results: MSE = 0.9232598, RMSE = 0.9608641, and MAE = 0.7242866, and its performance on the test data were: MSE = 54.37524, RMSE = 7.373957 and MAE = 6.5583333.

**b. Lasso Regression (See Appendix T for R code and Appendix U for visual algorithms outputs discussed)**

Lasso (Least Absolute Shrinkage and Selection Operator) regression is like Ridge regression but different in its regulation term. Lasso regression adds an L1 penalty (regulation term) to the cost function of the traditional linear regression to help prevent overfitting by adding a constraint on the magnitude of the coefficients. This added term to the traditional linear regression cost function is relative to the absolute values of the coefficients where the L1 penalty term drives some of the models' coefficients to exactly zero, which can be equivalent to performing feature selection. As we did in Ridge regression, we perform the same hyperparameter, and cross-validation model tuning to find out optimal (λ) value. Like Ridge regression, the lambda value that results in the lowest MSE was 0.01. Once the model is trained, we proceed to use the 'predict ()' function on the training dataset followed by the test dataset for the reasons mentioned above. Using the performance metrics mentioned above, the performance of Ridge regression on the training data gave the following results: MSE = 0.8923082, RMSE = 0.9446207, and MAE = 0.7115284, and its performance on the test data were: MSE = 66.84043, RMSE = 8.1756, and MAE = 7.1065695.

**c. Decision Tree Bagging (See Appendix V for R code and Appendix W for visual algorithms outputs discussed)**

Bagging builds upon the CART algorithm (recursively partitioning the feature space into 'purer' subspaces. This is achieved by selecting a feature and a threshold value that optimally divides the data into subsets, minimizing the variability within each subset) as its base and incorporating additional techniques to enhance the predictive capabilities of CART models. By creating numerous trees through bootstrapping, which involves drawing multiple sets of random samples with replacements from the training set, bagging moderates the instability and improves the predictability of CARTs. Each of these bootstrapped samples is used to retrain the model, resulting in a collection of estimators. The final prediction is derived by aggregating the estimates from different trees, following the principle of the 'wisdom of crowds,' which is affiliated with the majority rule concept. This method allows for more robust evaluations and reduces the risk of overfitting to specific datasets. The hyperparameter associated with bagging is 'ntree' which specifies the number of individual decision trees that are generated and combined to form a more robust predictive model, enhancing accuracy and stability in the final prediction. To find this optimal value, we run the 'randomForest()' within the function and plot the summary to identify at which ntree results in the lowest error value, which was found to be at ntree = 61. The use of bootstrapping indirectly contributes to model tuning and validation. After adjusting our hyperparameter and training our model we proceeded with performing predictions using the training and then the test data. The results are as follows: MSE = 0.07827151, RMSE = 0.2797705, and MAE = 0.1830031 using the training dataset and MSE = 3.748449, RMSE = 1.936091, and MAE = 1.7991926 using the test dataset.

**d. Decision Tree Random Forest (See Appendix X for R code and Appendix Y for visual algorithms outputs discussed)**

Like Bagging, Random Forest also builds upon the CART algorithm as its base and incorporates additional techniques to enhance the predictive capabilities of CART models. Additionally, Random Forest also utilizes the bootstrapping technique to allow for more robust evaluations and reduce the risk of overfitting to specific datasets. The difference between Bagging and Random Forest is the fact that Random Forest introduces an extra layer of randomness by considering only a subset of predictor variables at each split point, reducing overfitting, increasing robustness, addressing the multicollinearity issue, and increasing model diversity. In addition to 'ntree,' we also attempt to find the optimal mtry value which represents the number of features considered at each split point, with higher values leading to a more diverse and potentially complex ensemble of trees resulting in increased model variance, but more intricate relationships are captured. We conduct the same procedure done in Bagging to determine the ntree value and utilize the 'tuneRF()' function to determine our optimal mtry value. After conducting both procedures, the optimal hyperparameter values for Random Forest were ntree = 61, and mtry = 4 as these hyperparameters result in the lowest error score. After adjusting our hyperparameter and training our model we proceeded with performing predictions using the training and then the test data. The results are as follows: MSE = 0.06809317, RMSE = 0.2609467, and MAE = 0.1716870 using the training dataset and MSE = 3.912915, RMSE = 1.978109, and MAE = 1.8450624 using the test dataset.

**e. Decision Tree Boosting (See Appendix Z for R code and Appendix AA for visual algorithms outputs discussed)**

Like Bagging, Boosting significantly improves the predictive capabilities of tree-based methods; however, it achieves this by sequentially growing trees and fitting 'weak' classifiers to the modified data, resulting in an ensemble of weak learners. This process focuses on correcting the mistakes of previous models, ultimately creating a highly accurate predictive model. Depending on the chosen loss function and optimization algorithm, Boosting takes on various forms, with AdaBoost.M1 and Gradient Boosting being two well-known examples. For our specific model, we use Gradient Boosting, as AdaBoost.M1 is specially designed for classification tasks. The hyperparameters associated with Boosting that are considered in this

report are n.trees, interaction.depth, shrinkage, and n.minobsinnode. Having previously discussed n.tree, we focus on the discussion of the other hyperparameters. Interaction.depth is the maximum depth of each tree in the ensemble it determines how many interactions or relationships between features the tree is allowed to capture, shrinkage is associated with the learning rate to the influence of each tree, helps prevent overfitting, and improves the model's performance, and n.minobsinnode is used to specify the minimum number of observations required to create a terminal node (leaf) when creating the tree. For hyperparameter tuning, we use the expand.grid() function to specify our list of hyperparameters to test, trainControl() to specify the cross-validation characteristics, and train() as the final model for conducting the hyperparameter tuning. After hyperparameter tuning and cross-validation were complete the optimal values were n.trees = 82, interaction.depth = 10, shrinkage = 0.1, and n.minobsinnode =5. After adjusting our hyperparameter and training our model we proceeded with performing predictions using the training and then the test data. The results are as follows: MSE = 0.01930614, RMSE = 0.1389466, and MAE = 0.09133378 using the training dataset and MSE = 2.699348, RMSE = 1.642969, and MAE = 1.5544369 using the test dataset.

**f. SVM Linear Kernel (See Appendix BB for R code and Appendix CC for visual algorithms outputs discussed)**

Support Vector Machines (SVM) is an SMLA that can be used for both classification and regression tasks. The premise behind SVM is to find an optimal hyperplane that best separates classes in a high-dimensional feature space. The position of the hyperplane is based on a position that represents the largest separation or margin between two classes. Within SVMs, several kernel methods can be utilized that are effective in dealing with non-linearly separable data that maps the data into a higher-dimensional space. Specifically with a linear kernel, this assumes that there is a linear relationship between the predictor and the response variable and attempts to find a linear decision boundary that best separates classes in the original feature space. As implied previously this kernel is suitable for problems where the classes can be adequately separated by a straight line or hyperplane. Regarding hyperparameter tuning for the SVM linear kernel, we test for the optimal cost value which determines the trade-off between minimizing the error allowed on the training data and ensuring a smooth decision function. When cost is small, more error can occur on the training data which leads to a soft margin, and a larger cost value leads to less error and a narrower margin. We use the tune() function to conduct the hyperparameter tuning and specify a sequence of values to test for our cost value (1 to 50 in our case). After hyperparameter tuning and cross-validation were complete the optimal cost value was 3. After adjusting our hyperparameter and training our model we proceeded with performing predictions using the training and then the test data. The results are as follows: MSE = 0.9458506, RMSE = 0.9725485, and MAE = 0.6599424 using the training dataset and MSE = 73.51656, RMSE = 8.57418, and MAE = 7.1118548 using the test dataset.

**g. SVM Radial Kernel (See Appendix DD for R code and Appendix EE for visual algorithms outputs discussed)**

With SVM with the radial kernel, those over process remain the same as described above except the radial kernel assumes a non-linear relationship between the response variable and the predictor variables and therefore can handle non-linearly separable data and is effective in capturing complex relationships in the data. SVM radial kernel transforms the data into a higher-dimensional space, it makes groups of data points that were initially mixed up become easier to separate. With that said, SVM Radial requires careful tuning of the hyperparameters, which for this report are the cost hyperparameter discussed previously, and a gamma hyperparameter which controls the shape and flexibility of the decision boundary where smaller gamma values result in a more flexible decision boundary that leads to larger margin, but can results in overfitting, and vice-versa for larger gamma values. The same procedure is followed as we did for the SVM

linear kernel, but we specify a range of gamma values to use in the tuning function (0.01, 0.1, 1, 5). After hyperparameter tuning and cross-validation were complete the optimal cost value was 4.1 and the gamma value was 0.10. After adjusting our hyperparameter and training our model we proceeded with performing predictions using the training and then the test data. The results are as follows: MSE = 0.2806018, RMSE = 0.5297186, and MAE = 0.3448363 using the training dataset and MSE = 3.00141, RMSE = 1.732458, and MAE = 1.5637828 using the test dataset.

**h. SVM Polynomial Kernel (See Appendix FF for R code and Appendix GG for visual algorithms outputs discussed)**

With the SVM polynomial kernel, the premise is essentially the same as SVM radial except for the fact that the polynomial kernel uses a polynomial function to create decision boundaries in this higher-dimensional space to achieve non-linearity. Additionally, for conducting hyperparameter tuning, we add 'degree' as a hyperparameter which is associated with the order of the polynomial used to create decision boundaries. As we use higher degree values, this increases the complexity and intricacies of the relationship that is captured in the data. The same procedure is followed as we did for the SVM radial kernel, but we specify a range of degree values to use in the tuning function (2,3,4). After hyperparameter tuning and cross-validation were complete the optimal cost value was 3, gamma was 0.01, and degree equal to 2. After adjusting our hyperparameter and training our model we proceeded with performing predictions using the training and then the test data. The results are as follows: MSE = 3.399824, RMSE = 1.843861, and MAE = 1.41959618 using the training dataset and MSE = 3.812724, RMSE = 1.95262, and MAE = 1.8144649 using the test dataset.

After creating, tuning, training, and testing all the mentioned SMLAs, their performances were assessed based on MSE, RMSE, and MAE scores. The superior SMLA was determined by identifying the model with the lowest performance metrics score. According to the results, the most effective supervised machine learning model for predicting the unemployment rate in Australia is Decision Tree Boosting with parameters: n.trees = 82, interaction.depth = 10, shrinkage = 0.1, and n.minobsinnode = 5. Its performance metric results are MSE = 0.01930614, RMSE = 0.1389466, and MAE = 0.09133378 using the training dataset and MSE = 2.699348, RMSE = 1.642969, and MAE = 1.5544369 using the test dataset (See Appendix HH for all model's performance results on the training dataset and Appendix II for all models performance results on the testing dataset).

### *V. (Artificial) Neural Networks (ANN)*

As the name suggests neural networks are inspired by the brain where many neural components are interconnected to each other through a vast number of pathways, hence 'Neural Networks', that work together to process information and make complex calculations. Additionally, ANN utilizes its interconnected nodes and layers to learn and extract features from data, enabling them to perform tasks like pattern recognition, classification, and regression, just to name a few. Four main components make up an ANN: neurons, connection, propagation of the signal, and decision or learning rule. The neurons can be thought of as tiny processing units that receive input signals from other neurons. These units hold these inputs until an activation function decides that it should send an output signal. Common activation functions used are 'sigmoid', 'relu', and 'tanh' (Bag, 2021) (See Appendix JJ for a description of each). For Neurons to send and receive inputs they require connections to each other. Within these connections, they have a weight that indicates how important one neuron's output is for another neuron. Estimating this weight is a crucial step in training neural networks. Propagation of the signal refers to the signal from one neuron to another as a weighted sum of the outputs from the first neuron (output function) and the learning rule refers to rules that help us estimate all the parameters, including weights and connection thresholds and requires the use of optimization-based methods.

Appendix KK offers a simplified glimpse into an ANN known as a single hidden layer backpropagation neural network. Backpropagation is a crucial algorithm employed in training neural networks. It operates by adjusting the model's parameters (weights and biases) during a backward pass, following a forward pass through the network. This process, based on the chain rule, forms a fundamental building block of neural network training (Simeon Kostadinov, 2019). It has an input layer (X), an output layer (Y), and a hidden layer (Z) where features are transformed. This hidden layer is where the network does its complex computations, translating the input features into derived features. The process involves fitting non-linear models to the input layers and then constructing probabilities for the output classes. Finally, we can mathematically demonstrate the function that demonstrates the representation of the connection between the nodes in the neural network as '$a_0^{(1)} = \sigma(a_0^{(1)} = w_{0,0}\, a_0^{(0)} + w_{0,1}\, a_1^{(0)} \ldots. w_{0,n}\, a_n^{(0)} + b_0)$', or more simplified version $a^{(1)} = \sigma(Wa^{(0)} + b)$, where '$a^{(1)}$' is the first neuron in the first hidden layer, 'σ' is the sigmoid activation, 'W' is all of the weights organized into a matrix, were one row represents all of the connections between one layer and a particular neuron in the next layer, and 'b' represents all of the biases in a vector format (backpropagation). This equation represents the full translation of activations from one layer to the next. (But What Is a Neural Network? | Chapter 1, Deep Learning, n.d. 13:26)

In line with the machine learning section, we present a short overview of the features and hyperparameters utilized in constructing our ANN for this report. Given the dataset's modest scale (166 observations and 8 variables, with 153 in training and 10 in testing), we opt for shallow neural networks with three configurations: 1 input layer (X) and one output layer (Y), 2 layers: 1 input layer (X), 1 hidden layer (Z) and one output layer (Y), and 3 layers: 1 input layer (X), 2 hidden layers (Z) and one output layer (Y). To prevent overfitting, we avoid implementing further deeper neural networks. The activation function 'relu' is chosen for its non-linearity. For optimization, we consider 'adam' and 'rmsprop' as they relate to learning rate adjustment. Adam uses an adaptive learning rate approach combining different elements, while rmsprop focuses on adapting the learning rate individually for each parameter based on recent gradient magnitudes. Given the dataset's size, we prioritize MAE as our primary performance metric. This choice is influenced by the relatively limited training data points and MAE's resilience to outliers. Additionally, we consider three crucial hyperparameters: 'epoch': which refers to a single cycle through the entire training dataset, during which the model's parameters are updated based on the error calculated from the predictions on the training data. A higher value means that the model is exposed to the entire training dataset more times and vice versa for lower (baeldung, 2020). The second hyperparameter considered is 'units': the number of artificial neurons present in a particular layer, and the third is batch_size: how many training examples the neural network processes at once during training. The description above provides the overall framework for the structures of the ANN that will be conducted.

As with the SMLAs previously discussed, we follow the same pattern where we conduct hyperparameter tuning to find the most optimal hyperparameter values. Before this is done we must consider the iterations of testing that will be conducted. We will tune using the 'epoch', 'units', and 'batch_size' hyperparameters, along with the different optimizers ('adam' and 'rmsprop') and a variety of layers to which we then check the trained models' performances on the training data to identify the best model (based on the lowest MAE score). This process is iterative and demands careful attention. To improve efficiency, I created a thorough for-loop used for hyperparameter tuning, cross-validation, and model performance evaluation. This loop is employed for each layer type and optimizer (refer to Appendix LL for the complete R code). This approach provides six distinct for-loop tests, one for each layer type. Within each layer type, a separate run is conducted with a different optimizer. As a result, we obtain six datasets, each corresponding to a specific layer and optimizer type. These datasets contain a range of hyperparameter values and the corresponding performance results from each iteration of the training data. From this comprehensive dataset, we can identify the most effective ANN by evaluating its performance on

the training data. Before initiating any for-loop, we establish predefined values for each hyperparameter to be used in the tuning process: 'epoch' values of 100, 200, and 300, 'units' set at 32, 64, and 128, and 'batch_size' options of 8, 16, and 32. Additionally, for cross-validation, we set the ‘k’ value to 4, considering the time constraints that led us to avoid using the standard value of 10. These chosen hyperparameter values align with established norms for datasets of similar size, aiming to prevent the risk of overfitting in our models.

Before utilizing the for-loop, let's discuss the actual process of the ANN. First, we are required to split the training and test data even further where we make a new dataset with all the training predictor variables and another for the test predictor variables. We repeat this again but for the response variables where there is one for training data and one for test. From our initial EDA, we know that scaling needs to be done on the test and training predictor variables datasets as it would be problematic to feed into a neural network value that all take wildly different ranges. Once this is complete, we can create the network layers, which we discussed earlier. The network layers include the activation type ‘relu’ and the combination of units used in the hyperparameter tuning. Additionally, we can note that the last layer only consists of unit = 1 which since we are using regression, and the last layer is purely linear, the network is free to learn to predict values in any range. Following this we proceed to train the ANN using the ‘compile’ function where we state the type of optimizer (adam and rmsprop are used) the type of loss function (MSE) and the type of performance metric (MAE). After these have been specified, we proceed to use the ‘fit()’ function to train the ANN. The ‘validation_split()’ specifies a random fraction of the training data to be with-held during each epoch to be used as a pseudo-validation data set and the remaining fraction of the training data is used to train the network parameters during the epoch. Once this is all complete, we use the ‘evaluate()’ function to conduct a performance test where the trained ANN uses the training data to perform predictions.

After completing all the hyperparameter tuning, cross-validation, and model performance for-loops on 1 - 3 layers using ‘adam’ and ‘relu’ optimizers, the best hyperparameters and ANN features were the 2 layers ANN using a 'rmsprop' optimizer, epochs = 200, units = 32, and batch = 16, with an MAE score of 0.3679621 (See Appendix MM for full results). Now that we have obtained our optimal features and hyperparameter values, we test our optimally trained ANN on the test data (See Appendix NN for full R code). The results are as follows: MSE = 2.330682, RMSE = 1.526657, and MAE = 1.220769 using the test dataset. Due to the nature of the for-loop used, it is hard to decipher the impact that different layers and neurons used in the model. After extensive research into these considerations, and considering the size of our dataset, deepening the layers of the ANN can improve capturing complex relationships in the data, as each layer can progressively learn more abstract features, but can lead to overfitting and may not effectively capture the limited underlying patterns. If we increase the number of neurons, this allows the network to represent more complex patterns and relationships in the data, enhancing its learning capacity. But when the number of neurons becomes too much this may result in the ANN memorizing noise rather than learning meaningful relationships.

### *VI. Comparison and Suggestion*

When comparing the ANN and Decision Tree Boosting, we can consider the following discussions: i) Model performance, ii) Computational time to train models, iii) Interpretability, iv) model complexity, and v) data requirements. From our analysis of both models, we can see that for MSE, Decision Tree: Boosting was 2.699348 and ANN WAS 2.330682, for RMSE, Decision Tree: Boosting was 1.642969 and ANN was 1.526657, and lastly, we have MAE where Decision Tree: Boosting was 1.554437 and ANN was 1.220769. Overall, as previously mentioned A lower MSE indicates better performance, and a lower RMSE and MAE suggest better accuracy in all scenarios ANN had the better results and suggests that the ANN model is

more accurate in predicting the unemployment rate in Australia compared to the Decision Tree Boosting model. When comparing the computational time to train the models ANN took approximately 52 minutes, while the Decision Tree Boosting model took only 43.43202 seconds leading to better computational efficiency from the Decision Tree Boosting model. Due to the nature of the Decision tree models, they are known for being a generally more interpretable model type, compared to complex neural networks. Decision trees provide a clear and understandable representation of the decision-making process, which can be crucial for understanding the underlying relationships in the data. In contrast to this, ANN, especially with multiple hidden layers, can become very complex and could lead to overfitting, particularly with limited data as we experienced for this report.

Furthermore, Decision Tree Boosting tends to be more robust to overfitting. As a final discussion when comparing both models, we should consider the data requirements. With ANN the amount of data required for the model to train itself and perform well needs to be significantly large in comparison to Decision Tree Boosting where it can often perform well with smaller datasets. In conclusion, the choice between the Artificial Neural Network and Decision Tree Boosting model depends on the specific priorities of the problem at hand. The ANN offers superior predictive accuracy but at the cost of higher computational complexity and time. The Decision Tree Boosting model is computationally efficient and interpretable but may sacrifice some predictive accuracy and the decision should be made based on a trade-off between these factors and the specific requirements of the problem. With this said, I believe that with such a small dataset, the performance of the ANN needs to be accepted with caution as normally, ANN more times than not would not perform as well compared to other models that are more catered to handling smaller datasets. Overall, based on the results presented, if we are looking for a more efficient and less computationally expensive model at the cost of performing slightly worse than we would consider the decision tree boosting model. If we are looking solely for performance, I would suggest ANN. However, as time continues, and more data is collected ANN may be the more reliable model to adopt as it can handle larger, more complex data.

## *VII. Conclusion*

This study examines a variety of modeling techniques to predict the unemployment rate in Australia. By exploring various machine learning algorithms and ANNs, we aimed to determine the most effective approach. After a thorough evaluation of supervised machine learning algorithms, Decision Tree Boosting was the most accurate model, achieving an MAE of 0.091 on the training data and 1.554 on the test data and highlighting its superior ability to predict the unemployment rate in Australia accurately. On the other hand, the ANN model demonstrated competitive performance, showcasing an MAE of 0.367 on the training data and 1.220 on the test data with optimal hyperparameters. Based on these findings, we can recommend that, given its computational efficiency and interpretability, Decision Tree Boosting stands as an excellent choice for predicting the unemployment rate, especially in scenarios where computational resources are constrained. If we are looking for better predictive performance and computational resources are available, ANN should be considered, as it demonstrates superior predictive accuracy. Furthermore, from this study, we grasp the importance of adapting modeling techniques to the specific context and data available and the importance of continuous model evaluation and fine-tuning to ensure the best performance. To further improve unemployment rate predictions in Australia, we can introduce additional applicable features or identify new variables to uncover deeper relationships and other underlying factors that may influence the unemployment rate. This includes possibly integrating economic indicators, more demographic data, or policy variables for additional insights. In this report, we discuss only a handful of supervised machine learning algorithms, while many others can be considered that were outside the scope of this report and may be more effective and suitable models for predicting the unemployment rate in Australia. Overall, this study provides a comprehensive analysis of modeling techniques for predicting the

unemployment rate in Australia. By considering the unique strengths and limitations of each method, informed decisions can be made to enhance the accuracy of unemployment rate predictions. Additionally, the recommendations and lessons learned serve as valuable guidelines for future predictive modeling undertakings within the context of this domain.

# Appendix

**Appendix A:** Table Summary for Dataset Variables.

| Variable Name | Type | Description | Linear Correlation to Response Variable | Distribution |
|---|---|---|---|---|
| DATE | Date | Dates of quarterly report releases | NA | NA |
| UNEMP_RATE | Numerical | Unemployment rate measured in percentage | NA | Slightly right skewed |
| GDP | Numerical | Percentage change in Gross domestic product | Positive: 0.13 | Slightly left skewed |
| GOV_FCE | Numerical | Percentage change in the Government's final consumption expenditure | Positive: 0.01 | Normal |
| CHNG_FCE_ALL_IND | Numerical | Percentage change in final consumption expenditure of all industry sectors | Negative: -0.04 | Slightly left skewed |
| TTI | Numerical | Term of trade index | Negative: -0.19 | Normal |
| CPI | Numerical | Consumer Price Index of all groups (CPI) | Negative: -0.73 | Bimodal |
| JOB_VACAN | Numerical | Number of job vacancies | Negative: -0.74 | Right skewed |
| RES_POP | Numerical | Estimated Resident Population | Negative: -0.70 | Bimodal |

**Appendix B:** Change variable names for clarity.

```{r}
colnames(abs_unemply) = c("DATE", "UNEMP_RATE", "GDP", "GOV_FCE", "CHNG_FCE_ALL_IND", "TTI", "CPI", "JOB_VACAN", "RES_POP")
head(abs_unemply)
```

**Appendix C:** Converting current date format to meet R standards.

```{r}
#Date Conversion
# Convert dates to the correct format to then split into training and testing datasets.
convert_month <- function(month) {
  month_names <- c(Jan = "01", Feb = "02", Mar = "03", Apr = "04", May = "05", Jun = "06", Jul = "07", Aug = "08", Sep = "09", Oct = "10", Nov = "11", Dec = "12")
  return(month_names[month])
}
#Split current date format
(months_numeric <- sapply(strsplit(abs_unemply$DATE, "_"), function(x) convert_month(x[2])))
# Create a new date string with '/' separator and add a day (1st) to the date.
(abs_unemply$DATE <- paste(substr(abs_unemply$DATE, 1, 4), months_numeric, "01", sep = "/"))
# Convert to Date format
(abs_unemply$DATE <- as.Date(abs_unemply$DATE))
#View changes
```

head(abs_unemply)

```

## **Appendix D:** Check for missing values

```{r}

*# Check missing data in training*

(sum(is.na(abs_unemply))) *# 11*

(missing_values = colSums(is.na(abs_unemply))) *# Due to the small size of the dataset, it is not recommended to remove observations or variables. Instead, we can conduct imputation of the missing values.*

*#DATE UNEMP_RATE GDP GOV_FCE CHNG_FCE_ALL_IND TTI CPI JOB_VACAN RES_POP*

*# 0 0 1 1 1 1 0 5 2*

```

## **Appendix E:** Missing value table summary

| Variables | Number of missing data points | Type of missing data |
|---|---|---|
| **GDP** | 1 | Missing Completely at Random |
| **GOV_FCE** | 1 | Missing Completely at Random |
| **CHNG_FCE_ALL_IND** | 1 | Missing Completely at Random |
| **TTI** | 1 | Missing Completely at Random |
| **JOB_VACAN** | 5 | Missing Not at Random |
| **RES_POP** | 2 | Missing Completely at Random |

## **Appendix F:** Brief description of imputation methods.

| Imputation method | Variable types allowed | Description |
|---|---|---|
| **missForest** | Any | Utilizing a random forest framework to impute missing values. |
| **MICE: Predictive Mean Matching** | Any | Imputes missing values and then selects observed values from the dataset that have similar predicted values. |
| **MICE: Linear regression, predicted values** | Numerical | Using linear regression models, for each variable with missing data, to estimate the missing values based on the relationships observed in the rest of the data. |
| **MICE: Linear regression using bootstrap** | Numerical | Using linear regression models, for each variable with missing data, and the resampling technique of bootstrapping to construct multiple imputed datasets and capturing uncertainty in the imputation process. |
| **MICE: A random sample from observed values** | Numerical | Choosing a value randomly from observed data for imputation which introduces variation and accounts for uncertainty in the imputation process. |
| **Median** | Numerical and Ordinal | Missing values are replaced with the median of the available data for that variable, |

## **Appendix G:** Raw Code for missForrest and MICE imputation methods.

```{r}

*# Pull all predictor variables used for imputation.*

pred_var = data.frame(abs_unemply[,c(3:9)])

*#----- missForest imputation-----*

set.seed(1234)

mf_imputed <- data.frame(

 original = pred_var,

```
 imputed_missForest = missForest(pred_var)$ximp
)
# Summary
dim(mf_imputed) # 166  14
mf_imputed [c(104:166),]

#-----MICE Imputation-----
# Define the variables to impute
var_impute <- c("GDP", "GOV_FCE", "CHNG_FCE_ALL_IND", "TTI", "CPI", "JOB_VACAN", "RES_POP")

# Set the seed for reproducibility
set.seed(1234)
# Combine the imputed variables with the original data
mice_imputed <- data.frame(
 original = pred_var,
 imputed_pmm = complete(mice(data.frame(pred_var[var_impute]), method = "pmm")), # Predictive Mean Matching
 imputed_lrpv = complete(mice(data.frame(pred_var[var_impute]), method = "norm.predict")), # Linear regression, predicted values
 imputed_lrb = complete(mice(data.frame(pred_var[var_impute]), method = "norm.boot")), # Linear regression using bootstrap
 imputed_rs = complete(mice(data.frame(pred_var[var_impute]), method = "sample")) # Random sample from observed values
)
# Summary
dim(mice_imputed) #166  35
mice_imputed[c(104:166),]
#-----Median Imputation-----
# Apply the function to replace missing values
abs_unemply_med <- abs_unemply
(missing_values = colSums(is.na(abs_unemply_med)))
abs_unemply_med <- abs_unemply_med %>% mutate(across(where(is.numeric), ~replace_na(., median(., na.rm=TRUE))))
(missing_values = colSums(is.na(abs_unemply_med)))
head(abs_unemply_med)
dim(abs_unemply_med) #  166   9
```

## **Appendix H:** Imputation Methods Performance on Machine Learning Algorithms.

| | missForest | Predictive Mean Matching | Linear regression, predicted values | Linear regression using bootstrap | A random sample from observed values | Median |
|---|---|---|---|---|---|---|
| **MSE: Ridge Regression** | 37.95831 | 40.84981 | 33.27483 | 30.80063 | 18.47223 | 54.37524 |
| **RMSE: Ridge Regression** | 6.161032 | 6.39139 | 5.76843 | 5.54983 | 4.29793 | 7.37396 |
| **MSE: Decision Tree - Boosting** | 2.884317 | 2.92512 | 2.62349 | 2.89498 | 3.01816 | 2.50152 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **RMSE: Decision Tree - Boosting** | 1.698328 | 1.71030 | 1.61972 | 1.70146 | 1.73729 | 1.58162 |
| **MSE: SVM - Radial** | 3.961182 | 4.28291 | 3.94795 | 4.29278 | 3.98683 | 3.00141 |
| **RMSE: SVM - Radial** | 1.990272 | 2.06952 | 1.9869 | 2.07190 | 1.99670 | 1.73246 |

**Appendix I:** Raw R code for median imputation dataset run on Ridge regression, Decision tree boosting method and SVM with radial kernel.

```{r}

A.vi) Ridge Regression: Median imputation

#_____Ride Regression_____

# model.matrix():

  #  organises the predictors in a matrix

  #  standardizes the predictors- removes scale and mean set to 0.

x_med <- model.matrix(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP, data = unemp_train_med)[,-1]# All 7 predictors

head(x_med) # Scaling did not occur

dim(x_med) # 153   7

y_med <- unemp_train_med$UNEMP_RATE # Response

head(y_med)

length(y_med) # 153

# Fit original ridge regression model

ridge_mod_med <- glmnet(x = x_med,

                    y = y_med,

                    alpha = 0,

                    standardize = T)

# Produce Ridge trace plot

plot(ridge_mod_med, xvar = "lambda")

# We can visualize how the coefficient estimates changed as a result of increasing lambda.

#view summary of model

summary(ridge_mod_med)

# Create grid of possible lambda values

grid <- 10^ seq (10 , -2, length = 100)

# Variable grid contains a sequence of numbers ranging from 10^10 to 10^-2, with a total of 100 elements

# Choosing lambda using cross-validation

# Choose lambda with least Cross-validation error

cv_ridge_med <- cv.glmnet(x = x_med,

                      y = y_med,

                      alpha = 0, # Ridge Regression

                      lambda = grid) # Does a 10-fold cross validation by default

#View model

summary(cv_ridge_med)
```

```
#View plot

plot(cv_ridge_med)

#Determine best lambda value

(best_lambda_ridge_med = cv_ridge_med$lambda.min) #0.01

set.seed(1234)

# Model

best_ridge_mod_med <- glmnet(x = x_med,

                    y = y_med,

                    lambda = best_lambda_ridge_med,

                    alpha = 0, # Ridge regression

                    standardize = T)

# View model summary

summary(best_ridge_mod_med)

# View coefficients

coef(best_ridge_mod_med)

# Predict

x1_med <- model.matrix(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP, data = unemp_test_med)[,-1]# All 7
predictors

head(x1_med) # Scaling did not occur

dim(x1_med) #10   7

ridge_pre_med <- predict(best_ridge_mod_med,

                  newx = x1_med,

                  alpha=0,

                  s= best_lambda_ridge_med)

# View prediction

ridge_pre_med

# Performance

ridge_mse_med = mean((ridge_pre_med - unemp_test_med$UNEMP_RATE) ^2)

ridge_mse_med #54.37524

ridge_rmse_med = sqrt(mean((ridge_pre_med - unemp_test_med$UNEMP_RATE) ^2))

ridge_rmse_med #7.373957
```

B.vi) Boosting: Median imputation

```
#-----Boosting-----

set.seed(1234)

# Model

boost_model_med <- gbm(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,

                  data = unemp_train_med,

                  distribution = "gaussian",

                  n.trees = 5000 ,

                  interaction.depth = 4)

# Summary
```

```
summary(boost_model_med)

boost_model_med

# Predict

boost_pred_med <- predict(boost_model_med, unemp_test_med)

# MSE

boost_mse_med <- mean((boost_pred_med - unemp_test_med$UNEMP_RATE)^2)

boost_mse_med #2.501521

# RMSE

boost_rmse_med <- sqrt(mean((boost_pred_med - unemp_test_med$UNEMP_RATE)^2))

boost_rmse_med #1.58162

# Tuning parameters

boost_grid_med <-  expand.grid(interaction.depth = c(2,5,10,15),  # Max Tree Depth

                  n.trees = (1:100), # Boosting Iterations

                  shrinkage = c(0.001, 0.01, 0.1, 1),# Shrinkage = Learning Rate

                  n.minobsinnode = c(5,10,20) # minimum number of training set samples in a node to commence splitting

                  )

# Cross-validation

fitControl_med <- trainControl(method = "cv",

                       number = 10,

                       savePredictions = "final")

set.seed(1234)

# Hyperparameter tuning

boost_grid_train_med <- train(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,

                      data = unemp_train_med,

                      method = "gbm",

                      trControl = fitControl_med,

                      verbose = FALSE,

                      tuneGrid = boost_grid_med)

# Best tuning parameters

(best <- boost_grid_train_med$bestTune)

# n.trees interaction.depth shrinkage n.minobsinnode

#   82             10      0.1            5

# Optimal parameter

set.seed(1234)

# Model with optimal parameters

best_boost_model_med <- gbm(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,

                    data = unemp_train_med,

                    distribution = "gaussian",

                    n.trees = 82,

                    interaction.depth = 10,
```

```
                shrinkage = 0.1,
                n.minobsinnode = 5)
# Summary
summary(best_boost_model_med)
best_boost_model_med
# Predict
best_boost_pred_med <- predict(best_boost_model_med, unemp_test_med)
# MSE
best_boost_mse_med <- mean((best_boost_pred_med - unemp_test_med$UNEMP_RATE)^2)
best_boost_mse_med #2.699348
# RMSE
best_boost_rmse_med <- sqrt(mean((best_boost_pred_med - unemp_test_med$UNEMP_RATE)^2))
best_boost_rmse_med #1.642969
```

C.vi) SVM, Radial: Median imputation

```
#-----SVM, Radial-----
# Create sequences for cost and gamma
rad_cost_values_med <- seq(0.1, 5, by = 0.25)
rad_gamma_values_med <- c(0.01, 0.1, 1, 5)
# Generate all combinations of the cost values
svm_rad_tgrid_med <- expand.grid(cost = rad_cost_values_med, gamma = rad_gamma_values_med)
start_time_rad_med = Sys.time() #Time how long it takes to run the svm() function
# Ensure the same results
set.seed(1234)
# Conduct 'cost' and 'gamma' hyperparameter tuning for cost Aand radial SVM model. Fit the support vector classifier with 'scale = TRUE' for scaling.
svm_rad_tune_med <- tune(svm,
                UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,
                data = unemp_train_med,
                kernel = "radial",
                ranges = svm_rad_tgrid_med,
                scale = TRUE)
end_time_rad_med = Sys.time()# End timer.
runtime_rad_med = end_time_rad_med - start_time_rad_med # Calculate run time.
print(runtime_rad_med) # Time difference of 6.549102 mins
# View the cross-validation errors of the models
summary(svm_rad_tune_med) # Based on detailed performance results, cost = 3.35 and gamma = 0.1 are the most optimal.
# View the best model from tuning.
(rad_best_mod_med <- svm_rad_tune_med$best.model)
# Parameters:
#   SVM-Type:  eps-regression
# SVM-Kernel:  radial
```

```
#      cost:  4.10

#     gamma:  0.1

#   epsilon:  0.1

#Number of Support Vectors:  108

# Model

rad_svm_med <- svm(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,

                   data = unemp_train_med,

                   kernel = "radial",

                   cost = 4.1,

                   gamma = 0.1,

                   epsilon = 0.1,

                   scale = TRUE)

# We can determine the support vector identities

rad_svm_med$index

#View summary of svm model

summary(rad_svm_med)

#This tells us, for instance, that a linear kernel was used with cost = 4.1, gamma = 0.1, epsilon = 0.1 and that there were 108 support vectors.

# Predict

rad_pre_med <- predict(rad_svm_med, unemp_test_med)

# MSE

rad_mse_med <- mean((rad_pre_med - unemp_test_med$UNEMP_RATE)^2)

rad_mse_med #3.00141

# RMSE

rad_rmse_med <- sqrt(mean((rad_pre_med - unemp_test_med$UNEMP_RATE)^2))

rad_rmse_med #2.071902

```

## **Appendix J:** Visualisation of dataset to showing varying ranges of values for each predictor variable.

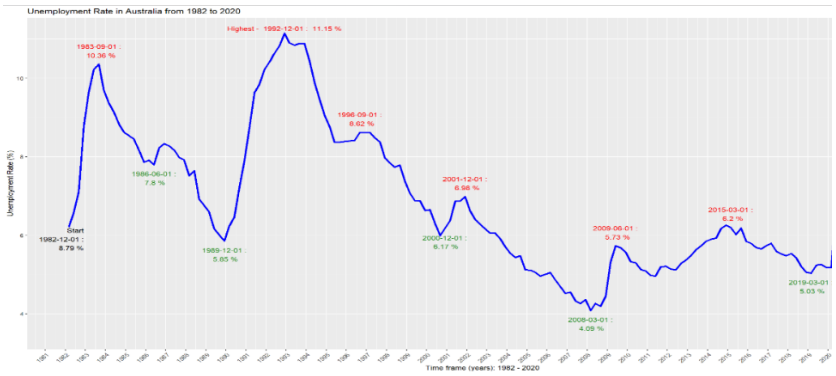
| | DATE | UNEMP_RATE | GDP | GOV_FCE | CHNG_FCE_ALL_IND | TTI | CPI | JOB_VACAN | RES_POP |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1982-12-01 | 8.793362 | -1.9 | 5.3 | 1.7 | -1.0 | 33.6 | 28.6 | 15288.9 |
| 2 | 1982-06-01 | 6.565219 | 0.5 | 7.9 | 3.4 | 2.5 | 31.5 | 31.2 | 15184.2 |
| 3 | 1982-03-01 | 6.205354 | -1.2 | -3.2 | -0.1 | -2.9 | 30.8 | 36.4 | 15121.7 |
| 4 | 1982-09-01 | 7.109055 | -1.0 | -5.9 | -1.3 | -0.5 | 32.6 | 28.4 | 15239.3 |
| 5 | 1983-12-01 | 9.707645 | 1.4 | 5.5 | 0.5 | 1.3 | 36.5 | 37.6 | 15483.5 |
| 6 | 1983-06-01 | 10.227507 | -0.5 | 2.1 | -0.6 | 1.2 | 35.0 | 31.7 | 15393.5 |
| 7 | 1983-03-01 | 9.637351 | -1.3 | 4.8 | 0.7 | 0.5 | 34.3 | 30.3 | 15346.2 |
| 8 | 1983-09-01 | 10.357458 | 2.5 | -4.9 | 0.6 | -0.2 | 35.6 | 34.3 | 15439.0 |
| 9 | 1984-12-01 | 8.632269 | 0.3 | -11.4 | -0.5 | -0.3 | 37.4 | 52.3 | 15677.3 |
| 10 | 1984-06-01 | 9.135483 | 0.8 | 3.7 | -0.2 | 0.5 | 36.4 | 43.5 | 15579.4 |
| 11 | 1984-03-01 | 9.364533 | 2.2 | -1.7 | 1.4 | -0.4 | 36.3 | 40.9 | 15531.5 |
| 12 | 1984-09-01 | 8.825721 | 0.6 | 14.9 | 0.6 | 0.5 | 36.9 | 46.5 | 15628.5 |
| 13 | 1985-12-01 | 7.861877 | -0.7 | 1.8 | 1.3 | -4.4 | 40.5 | 66.9 | 15900.6 |
| 14 | 1985-06-01 | 8.450666 | 1.9 | -3.5 | 0.9 | -6.3 | 38.8 | 66.0 | 15788.3 |
| 15 | 1985-03-01 | 8.536715 | 1.1 | 7.5 | 2.7 | -0.5 | 37.9 | 60.2 | 15736.7 |
| 16 | 1985-09-01 | 8.172109 | 1.0 | 4.9 | 1.3 | -1.3 | 39.7 | 67.9 | 15839.7 |
| 17 | 1986-12-01 | 8.331205 | 1.3 | -3.6 | -0.6 | -0.2 | 44.4 | 65.6 | 16138.8 |
| 18 | 1986-06-01 | 7.801279 | -0.6 | 3.3 | 2.1 | 0.1 | 42.1 | 64.0 | 16018.4 |
| 19 | 1986-03-01 | 7.913269 | 0.3 | 1.3 | -0.7 | -0.4 | 41.4 | 65.2 | 15961.5 |
| 20 | 1986-09-01 | 8.230243 | -0.1 | 2.2 | 1.0 | -4.6 | 43.2 | 64.2 | 16075.0 |
| 21 | 1987-12-01 | 7.915380 | 1.6 | 3.8 | 1.4 | 0.1 | 47.6 | 68.3 | 16394.6 |
| 22 | 1987-06-01 | 8.171202 | 1.2 | 2.6 | 0.7 | 2.1 | 46.0 | 68.7 | 16263.9 |
| 23 | 1987-03-01 | 8.279050 | 0.6 | -0.5 | 0.2 | 0.6 | 45.3 | 67.7 | 16204.0 |
| 24 | 1987-09-01 | 7.980163 | 1.4 | 1.0 | 1.4 | 2.7 | 46.8 | 68.2 | 16328.9 |
| 25 | 1988-12-01 | 6.754569 | 1.1 | -4.4 | 0.9 | 0.6 | 51.2 | 85.0 | 16687.1 |
| 26 | 1988-06-01 | 7.646345 | -0.3 | -2.3 | -0.5 | 4.4 | 49.3 | 74.9 | 16532.2 |
| 27 | 1988-03-01 | 7.518644 | 0.0 | 0.9 | 0.9 | 5.1 | 48.4 | 70.4 | 16471.8 |
| 28 | 1988-09-01 | 6.921147 | 0.3 | 2.1 | 1.5 | 5.4 | 50.2 | 79.8 | 16612.6 |
| 29 | 1989-12-01 | 5.853084 | -0.7 | 3.3 | 0.9 | 0.3 | 55.2 | 77.2 | 16936.7 |
| 30 | 1989-06-01 | 6.164607 | 1.8 | 3.2 | 2.0 | -0.6 | 53.0 | 86.7 | 16814.4 |
| 31 | 1989-03-01 | 6.594889 | 0.6 | 7.5 | 2.1 | 4.7 | 51.7 | 87.7 | 16764.0 |

## **Appendix K:** Data split of median imputation dataset.

```{r}

#-----Split data into training and testing dataset Median -----
```

```
# Create training dataset from 1982 - 2020

(abs_unemply_train_med <- subset(abs_unemply_med, DATE >= as.Date("1982-12-01") & DATE <= as.Date("2020-12-01")))

head(abs_unemply_train_med)

dim(abs_unemply_train_med)#153   9

str(abs_unemply_train_med)

# Create test dataset from 2020 to 2023

(abs_unemply_test_med <- subset(abs_unemply_med, DATE > as.Date("2020-12-01")))

head(abs_unemply_test_med)

dim(abs_unemply_test_med)#10 9

str(abs_unemply_test_med)

#Training

unemp_train_med = abs_unemply_train_med[,-1]

head(unemp_train_med)

dim(unemp_train_med)#153   8

str(unemp_train_med)

#Testing

unemp_test_med = abs_unemply_test_med[,-1]

head(unemp_test_med)

dim(unemp_test_med)#10   8

str(unemp_test_med)
```

## **Appendix L:** Unemployment rate from December 1982 to December 2020



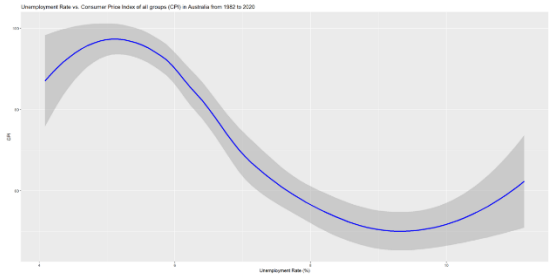

## **Appendix M:** R code unemployment rate over time visualisations

```
# -----Unemployment Rate over time-----

# Identify the highest and lowest unemployment rates

high_unemply <- abs_unemply_med[abs_unemply_med$UNEMP_RATE == max(abs_unemply_med$UNEMP_RAT), ]

low_unemply <- abs_unemply_med[abs_unemply_med$UNEMP_RATY == min(abs_unemply_med$UNEMP_RATE), ]

# Local Max

local_max <- abs_unemply_med[which(diff(sign(diff(abs_unemply_med$UNEMP_RATE))) == -2) + 1, ]

# Pull specific local max

local_max <- local_max[c(2,18,23,32,40,45),]

# Local Min
```

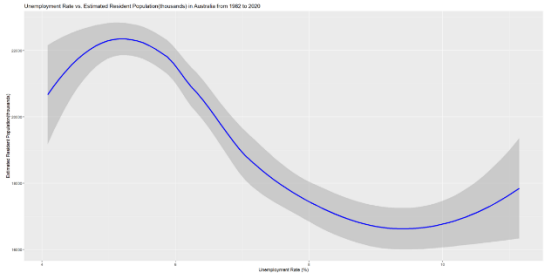
```
local_min <- abs_unemply_med[which(diff(sign(diff(abs_unemply_med$UNEMP_RATE))) == 2) + 1, ]
# Pull specific local max
local_min <- local_min[c(6,9,22,32,45,48),]
#First data entry: December 1982
(dec_1982 <- abs_unemply_med[1,]) # % 8.793362
#Last data entry: September 2020
(march_2023 <- abs_unemply_med[166,]) #  % 3.581322
#Histogram of response
hist(abs_unemply_med$UNEMP_RATE) #Slightly positive skewed.
# ggplot
ggplot(abs_unemply_med, aes(x = DATE, y = UNEMP_RATE)) +
 geom_line(colour = "blue", size = 1.2) +
 geom_text(data = high_unemply, aes(label = paste("Highest - ", DATE,": ", round(UNEMP_RATE,2),"%")),
           vjust = -0.5, hjust = 0.5, color = "red") +
 geom_text(data = low_unemply, aes(label = paste("Lowest - ", DATE,": ", round(UNEMP_RATE,2),"%")),
           vjust = 1.5, hjust = 0.5, color = "#228B22") +
 geom_text(data = local_max, aes(label = paste(DATE,":\n ", round(UNEMP_RATE,2),"%")),
           vjust = -0.5, hjust = 0.5, color = "red") +
 geom_text(data = local_min, aes(label = paste(DATE,":\n", round(UNEMP_RATE,2),"%")),
           vjust = 1.5, hjust = 0.5, color = "#228B22") +
  geom_text(data = dec_1982, aes(label = paste("Start\n", DATE,":\n", round(UNEMP_RATE,2),"%")),
          vjust = 5.6, hjust = 1, color = "black") +
geom_text(data = march_2023, aes(label = paste("End\n", DATE,":\n", round(UNEMP_RATE,2),"%")),
          vjust = -0.2, hjust = 0.08, color = "black") +
 labs(title = "Unemployment Rate in Australia from 1982 to 2020", x = "Time frame (years): 1982 - 2020", y = "Unemployment Rate (%)") +
 scale_x_date(date_labels = "%Y", date_breaks = "1 year") +
 theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

**Appendix N:** Visualisation of linear relationship between term of trade index (percentage), CPI of all groups, number of job vacancies (measured in thousands), and Estimated Resident Population (in thousands), versus the unemployment rate.

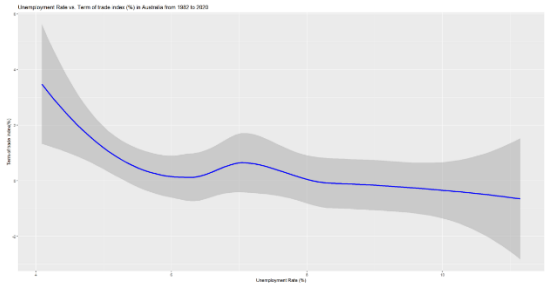
Unemployment Rate (%) vs Consumer Price Index of all groups



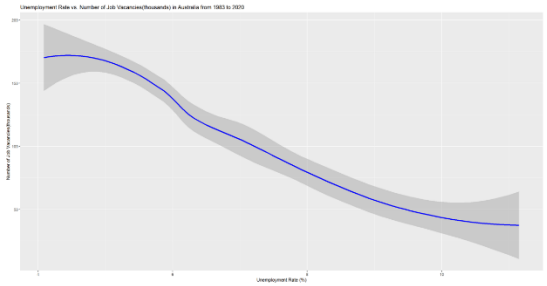
Unemployment Rate (%) vs Estimated Resident Population (thousands)



Unemployment Rate (%) vs Term of Trade Index (%)



Unemployment Rate (%) vs Job Vacancies (thousands)



## **Appendix O:** R code for smooth line, training data variable visualisations

```
#_________________Visualization for chosen imputation dataset:_________________

(all_train_var <- pairs.panels(unemp_train_med)) #view all variables from training dataset for comparison

# -----Unemployment Rate over time-----

# Identify the highest and lowest unemployment rates

high_unemply <- abs_unemply_med[abs_unemply_med$UNEMP_RATE == max(abs_unemply_med$UNEMP_RAT), ]

low_unemply <- abs_unemply_med[abs_unemply_med$UNEMP_RATY == min(abs_unemply_med$UNEMP_RATE), ]

# Local Max

local_max <- abs_unemply_med[which(diff(sign(diff(abs_unemply_med$UNEMP_RATE))) == -2) + 1, ]

# Pull specific local max

local_max <- local_max[c(2,18,23,32,40,45),]

# Local Min

local_min <- abs_unemply_med[which(diff(sign(diff(abs_unemply_med$UNEMP_RATE))) == 2) + 1, ]

# Pull specific local max

local_min <- local_min[c(6,9,22,32,45,48),]

#First data entry: December 1982

(dec_1982 <- abs_unemply_med[1,]) # % 8.793362

#Last data entry: September 2020

(march_2023 <- abs_unemply_med[166,]) #  % 3.581322

#Histogram of response

hist(abs_unemply_med$UNEMP_RATE) #Slightly positive skewed.

# ggplot

ggplot(abs_unemply_med, aes(x = DATE, y = UNEMP_RATE)) +

  geom_line(colour = "blue", size = 1.2) +

  geom_text(data = high_unemply, aes(label = paste("Highest - ", DATE,": ", round(UNEMP_RATE,2),"%")),

            vjust = -0.5, hjust = 0.5, color = "red") +
```

```
  geom_text(data = low_unemply, aes(label = paste("Lowest - ", DATE,": ", round(UNEMP_RATE,2),"%")),
            vjust = 1.5, hjust = 0.5, color = "#228B22") +
  geom_text(data = local_max, aes(label = paste(DATE,":\n ", round(UNEMP_RATE,2),"%")),
            vjust = -0.5, hjust = 0.5, color = "red") +
  geom_text(data = local_min, aes(label = paste(DATE,":\n", round(UNEMP_RATE,2),"%")),
            vjust = 1.5, hjust = 0.5, color = "#228B22") +
   geom_text(data = dec_1982, aes(label = paste("Start\n", DATE,":\n", round(UNEMP_RATE,2),"%")),
          vjust = 5.6, hjust = 1, color = "black") +
geom_text(data = march_2023, aes(label = paste("End\n", DATE,":\n", round(UNEMP_RATE,2),"%")),
          vjust = -0.2, hjust = 0.08, color = "black") +
  labs(title = "Unemployment Rate in Australia from 1982 to 2020", x = "Time frame (years): 1982 - 2020", y = "Unemployment Rate (%)") +
  scale_x_date(date_labels = "%Y", date_breaks = "1 year") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
# -----Unemployment Rate vs. Term of trade index(%)-----
#ggplot - smooth line plot
ggplot(unemp_train_med, aes(x = UNEMP_RATE, y = TTI)) +
  geom_smooth(colour = "blue", size = 1.2) +
  labs(title = "Unemployment Rate vs. Term of trade index (%) in Australia from 1982 to 2020", x = "Unemployment Rate (%)", y =  "Term of trade index(%)")
# -----Unemployment Rate vs. Consumer Price Index of all groups (CPI)-----
#ggplot - smooth line plot
ggplot(unemp_train_med, aes(x = UNEMP_RATE, y = CPI)) +
  geom_smooth(colour = "blue", size = 1.2) +
  labs(title = "Unemployment Rate vs. Consumer Price Index of all groups (CPI) in Australia from 1982 to 2020", x = "Unemployment Rate (%)", y =  "CPI")
# -----Unemployment Rate vs. Number of Job Vacancies(thousands)-----
#ggplot - smooth line plot
ggplot(unemp_train_med, aes(x = UNEMP_RATE, y = JOB_VACAN)) +
  geom_smooth(colour = "blue", size = 1.2) +
  labs(title = "Unemployment Rate vs. Number of Job Vacancies(thousands) in Australia from 1983 to 2020", x = "Unemployment Rate (%)", y =  "Number of Job Vacancies(thousands)")
# -----Unemployment Rate vs. Estimated Resident Population(thousands)-----
#ggplot - smooth line plot
ggplot(unemp_train_med, aes(x = UNEMP_RATE, y = RES_POP)) +
  geom_smooth(colour = "blue", size = 1.2) +
  labs(title = "Unemployment Rate vs. Estimated Resident Population(thousands) in Australia from 1982 to 2020", x = "Unemployment Rate (%)", y =  "Estimated Resident Population(thousands)")
```

## **Appendix P:** Description of supervised machine learning algorithms used in report.

| SMLA Name | Type | Purpose | Advantages | Disadvantages | Function used | Tuned Hyperparameter | Tuning function |
|---|---|---|---|---|---|---|---|
| **Ridge Regression** | Linear Regression with L2 regularization. | Combines linear regression with a | Handles multicollinearity well, provides stable estimates. | Less interpretable, may not perform as well | glmnet() | Lambda | cv.glmnet() |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | penalty term to prevent overfitting. | | with non-linear relationships. | | | |
| **Lasso Regression** | Linear Regression with L1 regularization. | Adds a penalty term based on the absolute value of coefficients, promoting sparsity. | Can be used for feature selection, handles multicollinearity. | Tends to select only one variable from a group of highly correlated variables. | glmnet() | Lambda | cv.glmnet() |
| **Decision Tree: Bagging** | Ensemble method using multiple decision trees. | Reduces overfitting and increases model performance by aggregating multiple trees. | Reduces variance, improves predictive accuracy. | May not improve interpretability. | randomForest() | ntree | randomForest() and plot() to visually choose ntree value |
| **Decision Tree: Random Forest** | Ensemble method using multiple decision trees. | Like Bagging but adds an extra layer of randomness to further decorrelate the trees. | Highly accurate, handles high-dimensional data. | May be computationally expensive. | randomForest() | ntree<br>mtry | randomForest() and plot() to visually choose ntree value. TuneRF() for mtry value |
| **Decision Tree: Boosting** | Ensemble method using multiple decision trees. | Builds a series of trees, each correcting the errors of the previous one. | Can achieve high accuracy, reduces bias. | Sensitive to noisy data, may overfit if not tuned properly. | gbm() | interaction.depth<br>n.trees<br>shrinkage<br>n.minobsinnode | train() |
| **SVM: Linear Kernel** | Linear Kernel | Separates classes using a linear decision boundary. | Effective in high-dimensional spaces and less prone to overfitting. | May not perform well with non-linearly separable data. | svm() | cost | tune() |
| **SVM: Radial Kernel** | Radial Basis Function | Maps data into a higher-dimensional space using a Gaussian radial basis function, allowing for non-linear relationships. | Can handle complex, non-linear relationships and can handle a wide range of datasets. | May be more prone to overfitting and requires careful selection of hyperparameters | svm() | cost<br>gamma | tune() |
| **SVM: Polynomial Kernel** | Polynomial | Maps data into a higher-dimensional space using a polynomial function. | Can capture complex, non-linear relationships and may be computationally less expensive. | Sensitive to the choice of degree for the polynomial function and requires careful selection of hyperparameters. | svm() | cost<br>gamma<br>degree | tune() |

The following information has been provided and used from my weekly content material from our subject portal.

## **Appendix Q:** Performance metric descriptions

| Performance metric | Abbreviation | Description |
|---|---|---|
| **Mean Squared Error** | MSE | The average of the squared differences between actual and predicted values |
| **Root Mean Squared Error** | RMSE | The average of the squared differences between actual and predicted values which is then square rooted. |
| **Mean Absolute Error** | MAE | Sum of absolute/positive errors of all values |

(Evaluation Metrics for Regression Models- MAE vs MSE vs RMSE vs RMSLE, n.d.)

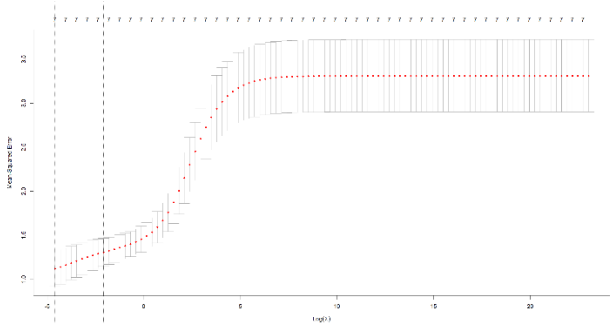
## **Appendix R:** R code for Ridge Regression.

```{r}

#_____Ride Regression_____

# model.matrix():

  #  organises the predictors in a matrix

  #  standardizes the predictors- removes scale and mean set to 0.

x_med <- model.matrix(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP, data = unemp_train_med)[,-1]# All 7 predictors

head(x_med) # Scaling did not occur

dim(x_med) # 153   7

y_med <- unemp_train_med$UNEMP_RATE # Response

head(y_med)

length(y_med) # 153

# Fit original ridge regression model

ridge_mod_med <- glmnet(x = x_med,

                    y = y_med,

                    alpha = 0,

                    standardize = T)

# Produce Ridge trace plot

plot(ridge_mod_med, xvar = "lambda")

# We can visualize how the coefficient estimates changed as a result of increasing lambda.

#view summary of model

summary(ridge_mod_med)

# Create grid of possible lambda values

grid <- 10^ seq (10 , -2, length = 100)

# Variable grid contains a sequence of numbers ranging from 10^10 to 10^-2, with a total of 100 elements

# Choosing lambda using cross-validation

# Choose lambda with least Cross-validation error

cv_ridge_med <- cv.glmnet(x = x_med,

                      y = y_med,

                      alpha = 0, # Ridge Regression

                      lambda = grid) # Does a 10-fold cross validation by default

#View model
```

```
summary(cv_ridge_med)

#View plot

plot(cv_ridge_med)

#Determine best lambda value

(best_lambda_ridge_med = cv_ridge_med$lambda.min) #0.01

set.seed(1234)

# Model

best_ridge_mod_med <- glmnet(x = x_med,

                    y = y_med,

                    lambda = best_lambda_ridge_med,

                    alpha = 0, # Ridge regression

                    standardize = T)

# View model summary

summary(best_ridge_mod_med)

# View coefficients

coef(best_ridge_mod_med)

# Performance on training data

# Predict

ridge_pre_med_training <- predict(best_ridge_mod_med,

                  newx = x_med,

                  alpha=0,

                  s= best_lambda_ridge_med)

# View prediction

ridge_pre_med_training

# Performance

ridge_mse_med_training = mean((ridge_pre_med_training - unemp_train_med$UNEMP_RATE) ^2)

ridge_mse_med_training # 0.9232598

ridge_rmse_med_training = sqrt(mean((ridge_pre_med_training - unemp_train_med$UNEMP_RATE) ^2))

ridge_rmse_med # 0.9608641

#MAE

(ride_train_mae = postResample(pred = ridge_pre_med_training, unemp_train_med$UNEMP_RATE))

#    RMSE      Rsquared       MAE

#  0.9608641   0.7174136   0.7242866

# Performance on test data

x1_med <- model.matrix(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP, data = unemp_test_med)[,-1]# All 7
predictors

head(x1_med) # Scaling did not occur

dim(x1_med) #10   7

# Predict

ridge_pre_med <- predict(best_ridge_mod_med,

                  newx = x1_med,
```

```
                alpha=0,
                s= best_lambda_ridge_med)
# View prediction
ridge_pre_med
# Performance
ridge_mse_med = mean((ridge_pre_med - unemp_test_med$UNEMP_RATE) ^2)
ridge_mse_med #54.37524
ridge_rmse_med = sqrt(mean((ridge_pre_med - unemp_test_med$UNEMP_RATE) ^2))
ridge_rmse_med #7.373957
#MAE
(ridge_test_mae = postResample(pred = ridge_pre_med, unemp_test_med$UNEMP_RATE))
#    RMSE     Rsquared      MAE
#  7.3739567  0.6689802   6.5583333
```

## **Appendix S:** Ridge Regression visualisation algorithms outputs



Graph of Error rates as lambda value increases

```
8 x 1 sparse Matrix of class "dgCMatrix"
                              s0
(Intercept)       -0.2103905984
GDP                0.3503712665
GOV_FCE            0.0464346682
CHNG_FCE_ALL_IND  -0.2513013926
TTI                0.0005791377
CPI               -0.0976987288
JOB_VACAN         -0.0312663504
RES_POP            0.0009168350
```

Estimated coefficients from model

```
> ridge_mse_med_training
[1] 0.9232598
> ridge_rmse_med # 0.9608641
[1] 7.373957
> #MAE
> (ride_train_mae = postResample(pred = ridge_pre_med_training, unemp_train_med$UNEMP_RATE))
     RMSE  Rsquared       MAE
0.9608641 0.7174136 0.7242866
```

Performance of trained Ridge Regression model on training data

```
> ridge_mse_med #54.37524
[1] 54.37524
> ridge_rmse_med #7.373957
[1] 7.373957
> #MAE
> (ridge_test_mae = postResample(pred = ridge_pre_med, unemp_test_med$UNEMP_RATE))
     RMSE  Rsquared       MAE
7.3739567 0.6689802 6.5583333
```

Performance of trained Ridge Regression model on test data

## **Appendix T:** R code for Lasso Regression.

```{r}
# Fit original lasso regression model
lasso_mod_med <- glmnet(x = x_med,
                y = y_med,
                alpha = 1,
                standardize = T)
# View summary of model
summary(lasso_mod_med)
# Produce Ridge trace plot
plot(lasso_mod_med, xvar = "lambda")
```

```
# We can visualize how the coefficient estimates changed as a result of increasing lambda.
# Perform k-fold cross-validation to find optimal lambda value
set.seed(1234)
cv_lasso_med <- cv.glmnet(x=x_med,
                          y=y_med,
                          alpha = 1,
                          lambda = grid)
# Produce plot of test MSE by lambda value
plot(cv_lasso_med)
# Find optimal lambda value that minimizes test MSE
best_lambda_lasso_med <- cv_lasso_med$lambda.min
best_lambda_lasso_med #0.01
set.seed(1234)
# Model
best_lasso_mod_med <- glmnet(x = x_med,
                             y = y_med,
                             lambda = best_lambda_lasso_med,
                             alpha = 1, # Lasso regression
                             standardize = T)
# View model summary
summary(best_lasso_mod_med)
# View coefficients
coef(best_lasso_mod_med)
# Performance on training
lasso_pre_med_training <- predict(best_lasso_mod_med,
                          newx = x_med,
                          alpha=1,
                          s= best_lambda_lasso_med)
# View prediction
lasso_pre_med_training
# MSE
lasso_mse_med_training <- mean((lasso_pre_med_training - unemp_train_med$UNEMP_RATE) ^2)
lasso_mse_med_training # 0.8923082
# RMSE
lasso_rmse_med_training <- sqrt(mean((lasso_pre_med_training - unemp_train_med$UNEMP_RATE) ^2))
lasso_rmse_med_training # 0.9446207
#MAE
(lasso_train_mae = postResample(pred = lasso_pre_med_training, unemp_train_med$UNEMP_RATE))
#     RMSE     Rsquared       MAE
#  0.9446207  0.7268324   0.7115284
```

```
# Performance on test data

lasso_pre_med <- predict(best_lasso_mod_med,

                         newx = x1_med,

                         alpha=1,

                         s= best_lambda_lasso_med)

# View prediction

lasso_pre_med

# MSE

lasso_mse_med <- mean((lasso_pre_med - unemp_test_med$UNEMP_RATE) ^2)

lasso_mse_med # 66.84043

# RMSE

lasso_rmse_med <- sqrt(mean((lasso_pre_med - unemp_test_med$UNEMP_RATE) ^2))

lasso_rmse_med # 8.1756

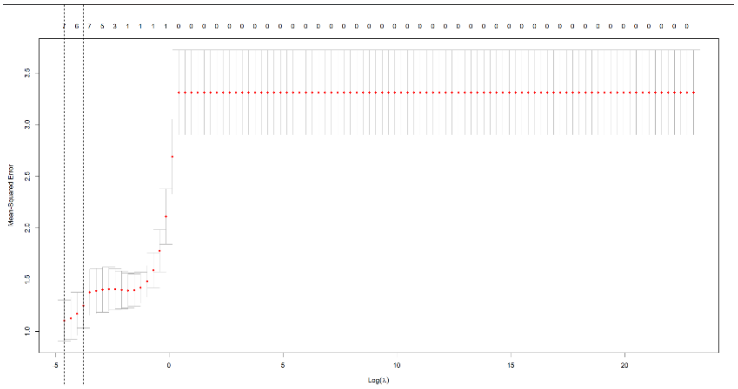
#MAE

(lasso_test_mae = postResample(pred = lasso_pre_med, unemp_test_med$UNEMP_RATE))

#    RMSE     Rsquared      MAE

#  8.1755995  0.6225268    7.1065695
```

## **Appendix U:** Lasso regression visualisation algorithms outputs.



Graph of Error rates as lambda value increases

```
> lasso_pre_med
            s1
157  -1.191131
158   2.094785
159   3.359258
160   1.248934
161  -3.413090
162  -3.043421
163  -2.368517
164  -3.507914
165 -10.815666
166 -11.088456
```

Estimated coefficients from model

```
> lasso_mse_med_training # 0.8923082
[1] 0.8923082
> lasso_rmse_med_training # 0.9446207
[1] 0.9446207
> #MAE
> (lasso_train_mae = postResample(pred = lasso_pre_med_training, unemp_train_med$UNEMP_RATE))
     RMSE  Rsquared       MAE
0.9446207 0.7268324 0.7115284
```

Performance of trained Lasso Regression model on training data

```
> lasso_mse_med # 66.84043
[1] 66.84043
> lasso_rmse_med # 8.1756
[1] 8.1756
> #MAE
> (lasso_test_mae = postResample(pred = lasso_pre_med, unemp_test_med$UNEMP_RATE))
     RMSE  Rsquared       MAE
8.1755995 0.6225268 7.1065695
```

Performance of trained Lasso Regression model on test data

## **Appendix V:** R code for Decision Tree Bagging.

```
```{r}

#-----Bagging-----

set.seed(1234)

# Model

bag_model_med <- randomForest(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,
```
```

```
                    data = unemp_train_med,
                    mtry = 7, # Use all predictors for bagging
                    importance = TRUE)
# Summary
summary(bag_model_med)
bag_model_med
# Plot
plot(bag_model_med)
abline(v= 61, col = "red") # Lowest point is around ntree=400. It plateaus arouns 320 to 500, I have chosen 400.
# Variable importance
(bg_var_imp_med <- varImpPlot(bag_model_med, scale = FALSE)
# Predict
bag_pred_med <- predict(bag_model_med, unemp_test_med)
# MSE
bag_mse_med <- mean((bag_pred_med - unemp_test_med$UNEMP_RATE)^2)
bag_mse_med #4.7291
# Run again with ntree = 61
set.seed(1234)
# Model
bag_model_med_61 <- randomForest(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,
                     data = unemp_train_med,
                     mtry = 7, # Use all predictors fro bagging
                     ntree = 61,
                     importance = TRUE)
# Summary
summary(bag_model_med_61)
bag_model_med_61
# Variable importance
(bg_var_imp_med_61 <- varImpPlot(bag_model_med_61, scale = FALSE)
# Predict on train
bag_pred_bag_med_61_train <- predict(bag_model_med_61, unemp_train_med)
# MSE
mse_bagging_med_61_train <- mean((bag_pred_bag_med_61_train - unemp_train_med$UNEMP_RATE)^2)
mse_bagging_med_61_train # 0.03185838
# RMSE
mse_bagging_med_61_train <- sqrt(mean((bag_pred_bag_med_61_train - unemp_train_med$UNEMP_RATE)^2))
mse_bagging_med_61_train # 0.1784892
# Predict on test
bag_pred_bag_med_61 <- predict(bag_model_med_61, unemp_test_med)
```

```
# MSE
mse_bagging_med_61 <- mean((bag_pred_bag_med_61 - unemp_test_med$UNEMP_RATE)^2)
mse_bagging_med_61 # 4.543146
# RMSE
mse_bagging_med_61 <- sqrt(mean((bag_pred_bag_med_61 - unemp_test_med$UNEMP_RATE)^2))
mse_bagging_med_61 # 2.131466
#-----Bagging with boostrap----
# Define the number of bootstrapped samples
B <- 100
# Initialize a list to store bagged models
bag_boot_models_med <- list()
# Set seed for reproducibility
set.seed(1234)
# Loop for bootstrapping
for (i in 1:B) {
 # Generate a bootstrap sample
 boot_sample_med <- sample(nrow(unemp_train_med), replace = TRUE)
 # Train a model on the bootstrap sample
 bag_boot_model_med <- randomForest(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,
                        data = unemp_train_rs[boot_sample_med, ],
                        mtry = 7,
                        ntree = 61,
                        importance = TRUE)
 # Store the model in the list
 bag_boot_models_med[[i]] <- bag_boot_model_med
}
# Combine predictions from all bagged models for train
bag_boot_pred_med_train <- numeric(nrow(unemp_train_med))
for (i in 1:B) {
 bag_boot_pred_med_train <- bag_boot_pred_med_train + predict(bag_boot_models_med[[i]], unemp_train_med)
}
bag_boot_pred_med_train <- bag_boot_pred_med_train / B
# Plot
plot(bag_model_med)
abline(v= 61, col = "red") # Lowest point is around ntree=400. It plateaus arouns 320 to 500, I have chosen 400.
# MSE
bag_boot_mse_med_train <- mean((bag_boot_pred_med_train - unemp_train_med$UNEMP_RATE)^2)
bag_boot_mse_med_train # 0.07827151
# RMSE
bag_boot_rmse_med_train <- sqrt(mean((bag_boot_pred_med_train - unemp_train_med$UNEMP_RATE)^2))
```

```
bag_boot_rmse_med_train # 0.2797705

#MAE

(bag_boot_train_mae = postResample(pred = bag_boot_pred_med_train, unemp_train_med$UNEMP_RATE))

#     RMSE     Rsquared      MAE

#  0.2797705  0.9774491    0.1830031

# Combine predictions from all bagged models for test

bag_boot_pred_med <- numeric(nrow(unemp_test_med))

for (i in 1:B) {

  bag_boot_pred_med <- bag_boot_pred_med + predict(bag_boot_models_med[[i]], unemp_test_med)

}

bag_boot_pred_med <- bag_boot_pred_med / B

# MSE

bag_boot_mse_med <- mean((bag_boot_pred_med - unemp_test_med$UNEMP_RATE)^2)

bag_boot_mse_med # 3.748449

# RMSE

bag_boot_rmse_med <- sqrt(mean((bag_boot_pred_med - unemp_test_med$UNEMP_RATE)^2))

bag_boot_rmse_med # 1.936091

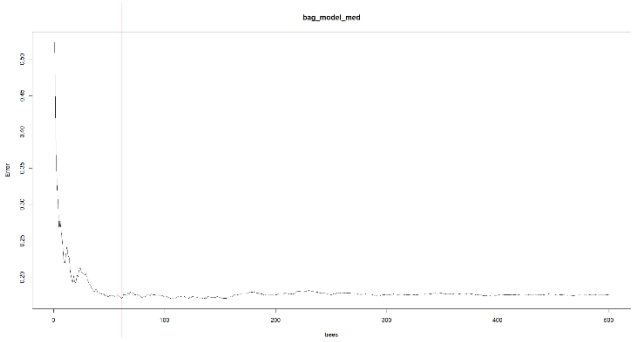
#MAE

(bag_boot_test_mae = postResample(pred = bag_boot_pred_med, unemp_test_med$UNEMP_RATE))

#     RMSE     Rsquared      MAE

#  1.9360913  0.2484545    1.7991926
```

## **Appendix W:** Decision Tree Bagging visualisation algorithms outputs.




Identifying optimal ntree value for lowest error rate

```
> summary(bag_model_med)
                Length Class  Mode
call              5    -none- call
type              1    -none- character
predicted       153    -none- numeric
mse             500    -none- numeric
rsq             500    -none- numeric
oob.times       153    -none- numeric
importance       14    -none- numeric
importanceSD      7    -none- numeric
localImportance   0    -none- NULL
proximity         0    -none- NULL
ntree             1    -none- numeric
mtry              1    -none- numeric
forest           11    -none- list
coefs             0    -none- NULL
y               153    -none- numeric
test              0    -none- NULL
inbag             0    -none- NULL
terms             3    terms  call
> bag_model_med

Call:
 randomForest(formula = UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND +      TTI + CPI + JOB_VACAN + RES_POP, data = unemp_tra
in_med,      mtry = 7, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 7

          Mean of squared residuals: 0.1759837
                    % Var explained: 94.59
```

Summary of Bagging model

```
[[99]]

Call:
 randomForest(formula = UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND +      TTI + CPI + JOB_VACAN + RES_POP, data = unemp_tra
in_rs[boot_sample_med,      ], mtry = 7, ntree = 61, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 61
No. of variables tried at each split: 7

          Mean of squared residuals: 0.1224747
                    % Var explained: 96.37

[[100]]

Call:
 randomForest(formula = UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND +      TTI + CPI + JOB_VACAN + RES_POP, data = unemp_tra
in_rs[boot_sample_med,      ], mtry = 7, ntree = 61, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 61
No. of variables tried at each split: 7

          Mean of squared residuals: 0.1947599
                    % Var explained: 93.67
```

Summary of Bagging models with bootstrap

```
> bag_boot_mse_med_train # 0.07827151
[1] 0.07827151
> bag_boot_rmse_med_train # 0.2797705
[1] 0.2797705
> #MAE
> (bag_boot_train_mae = postResample(pred = bag_boot_pred_med_train, unemp_train_med$UNEMP_RATE))
     RMSE  Rsquared       MAE
0.2797705 0.9774491 0.1830031
>
```

Performance of trained Decision Tree Bagging Bootstrap model on training data

```
> (bag_boot_train_mae = postResample(pred = bag_boot_pred_med_train, unemp_train_med$UNEMP_RATE))
     RMSE  Rsquared       MAE
0.2797705 0.9774491 0.1830031
> bag_boot_mse_med # 3.748449
[1] 3.748449
> bag_boot_rmse_med # 1.936091
[1] 1.936091
> #MAE
> (bag_boot_test_mae = postResample(pred = bag_boot_pred_med, unemp_test_med$UNEMP_RATE))
     RMSE  Rsquared       MAE
1.9360913 0.2484545 1.7991926
```

Performance of trained Decision Tree Bagging Bootstrap model on test data

## **Appendix X:** R code for Decision Tree Random Forest.

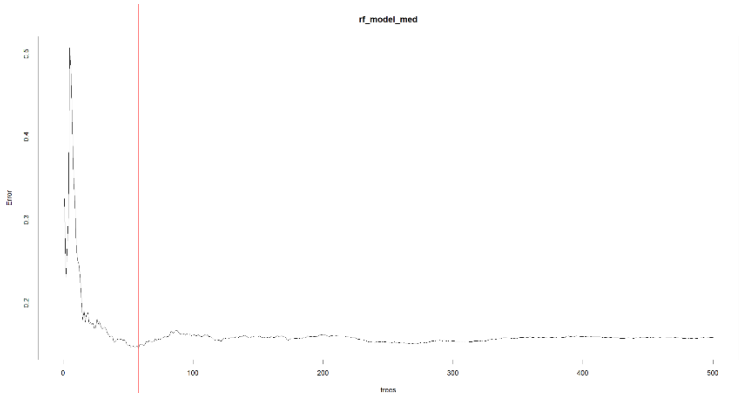
```{r}

#-----Random Forest-----

set.seed(1234)

# Model

rf_model_med <- randomForest(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,

                             data = unemp_train_med,

                             mtry = sqrt(7), # Use square root of all predictors for random forest

                             importance = TRUE)

# Summary

summary(rf_model_med)

rf_model_med

# Variable importance

(bg_var_imp_med <- varImpPlot(rf_model_med, scale = FALSE))

# Performance on train data

rf_pred_med_train <- predict(rf_model_med, unemp_train_med)

# MSE

mse_rf_med_train <- mean((rf_pred_med_train - unemp_train_med$UNEMP_RATE)^2)

mse_rf_med_train # 0.03002241

# RMSE

mse_rf_med_train <- sqrt(mean((rf_pred_med_train - unemp_train_med$UNEMP_RATE)^2))

mse_rf_med_train # 0.1732698

# Performance on test data

rf_pred_med <- predict(rf_model_med, unemp_test_med)

# MSE

mse_rf_med <- mean((rf_pred_med - unemp_test_med$UNEMP_RATE)^2)

mse_rf_med # 4.279245

# RMSE

mse_rf_med <- sqrt(mean((rf_pred_med - unemp_test_med$UNEMP_RATE)^2))

mse_rf_med # 2.068634

# Plot

plot(rf_model_med)

abline(v=58, col = "red") # ntree = 58 is when the error is the lowest.

set.seed(1234)


# Tuning parameters
```

```
mtry_med <- tuneRF(unemp_train_med[-1],
                   unemp_train_med$UNEMP_RATE,
                   ntreeTry=1000,
                   stepFactor=1.5,
                   improve=0.01,
                   trace=TRUE,
                   plot=TRUE)
# Best mtry values
best_m_med <- mtry_med[mtry_med[, 2] == min(mtry_med[, 2]), 1]
print(mtry_med)
print(best_m_med) # 4
# Implement optimal parameters ntree = 58, mtry = 4.
set.seed(1234)
# Model
rf_model_best_med <- randomForest(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,
                          data = unemp_train_med,
                          mtry = 4,
                          ntree = 58,
                          importance = TRUE)
# Summary
summary(rf_model_best_med)
rf_model_best_med
# Variable importance
(bg_var_imp_med <- varImpPlot(rf_model_best_med, scale = FALSE))
# Performance on train data
rf_model_best_pre_med_train <- predict(rf_model_best_med, unemp_train_med)
# MSE
rf_mse_best_med_train <- mean((rf_model_best_pre_med_train - unemp_train_med$UNEMP_RATE)^2)
rf_mse_best_med_train #0.02955353
# RMSE
rf_rmse_best_med_train <- sqrt(mean((rf_model_best_pre_med_train - unemp_train_med$UNEMP_RATE)^2))
rf_rmse_best_med_train #0.1719114
# Performance on test data
rf_model_best_pre_med <- predict(rf_model_best_med, unemp_test_med)
# MSE
rf_mse_best_med <- mean((rf_model_best_pre_med - unemp_test_med$UNEMP_RATE)^2)
rf_mse_best_med #4.79764
# RMSE
rf_rmse_best_med <- sqrt(mean((rf_model_best_pre_med - unemp_test_med$UNEMP_RATE)^2))
rf_rmse_best_med #2.183034
```

```
#-----Random Forest with bootstrap-----
# Define the number of bootstrapped samples (B)
B <- 100
# Initialize a list to store bagged random forest models
bagged_rf_models_med <- list()
# Set seed for reproducibility
set.seed(1234)
# Loop for bootstrapping
for (i in 1:B) {
 # Generate a bootstrap sample
 rf_boot_sample_med <- sample(nrow(unemp_train_med), replace = TRUE)
 # Train a random forest model on the bootstrap sample
 rf_boot_model_med <- randomForest(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,
                                   data = unemp_train_med[rf_boot_sample_med, ],
                                   mtry = 4,
                                   ntree = 58,
                                   importance = TRUE)
 # Store the model in the list
 bagged_rf_models_med[[i]] <- rf_boot_model_med
}
# Summary
summary(rf_boot_model_med)
rf_boot_model_med
# Combine predictions from all bagged models on train
rf_boot_pred_med_train <- numeric(nrow(unemp_train_med))
for (i in 1:B) {
 rf_boot_pred_med_train <- rf_boot_pred_med_train + predict(bagged_rf_models_med[[i]], unemp_train_med)
}
rf_boot_pred_med_train <- rf_boot_pred_med_train / B
# MSE
rf_boot_mse_med_train <- mean((rf_boot_pred_med_train - unemp_train_med$UNEMP_RATE)^2)
rf_boot_mse_med_train # 0.06809317
# RMSE
rf_boot_rmse_med_train <- sqrt(mean((rf_boot_pred_med_train - unemp_train_med$UNEMP_RATE)^2))
rf_boot_rmse_med_train # 0.2609467
#MAE
(rf_boot_train_mae = postResample(pred = rf_boot_pred_med_train, unemp_train_med$UNEMP_RATE))
#    RMSE    Rsquared      MAE
```

```
#  0.2609467 0.9805702   0.1716870

# Combine predictions from all bagged models on test

rf_boot_pred_med <- numeric(nrow(unemp_test_med))

for (i in 1:B) {

  rf_boot_pred_med <- rf_boot_pred_med + predict(bagged_rf_models_med[[i]], unemp_test_med)

}

rf_boot_pred_med <- rf_boot_pred_med / B

# MSE

rf_boot_mse_med <- mean((rf_boot_pred_med - unemp_test_med$UNEMP_RATE)^2)

rf_boot_mse_med # 3.912915

# RMSE

rf_boot_rmse_med <- sqrt(mean((rf_boot_pred_med - unemp_test_med$UNEMP_RATE)^2))

rf_boot_rmse_med # 1.978109

#MAE

(rf_boot_test_mae = postResample(pred = rf_boot_pred_med, unemp_test_med$UNEMP_RATE))

#     RMSE     Rsquared      MAE

#  1.9781090  0.2782989    1.8450624
```

## **Appendix Y:** Decision Tree Random Forest visualisation algorithms outputs.




Identifying optimal ntree value for lowest error rate

| | mtry | OOBError |
|---|---|---|
| 2 | 2 | 0.2237841 |
| 3 | 3 | 0.1628380 |
| 4 | 4 | 0.1480637 |
| 6 | 6 | 0.1531391 |

Identifying optimal mtry value for lowest error rate

```
[[98]]

Call:
 randomForest(formula = UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND +      TTI + CPI + JOB_VACAN + RES_POP, data = unemp_tra
in_med[rf_boot_sample_med,      ], mtry = 4, ntree = 58, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 58
No. of variables tried at each split: 4

          Mean of squared residuals: 0.1089318
                    % Var explained: 96.25

[[99]]

Call:
 randomForest(formula = UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND +      TTI + CPI + JOB_VACAN + RES_POP, data = unemp_tra
in_med[rf_boot_sample_med,      ], mtry = 4, ntree = 58, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 58
No. of variables tried at each split: 4

          Mean of squared residuals: 0.1791289
                    % Var explained: 94.32

[[100]]

Call:
 randomForest(formula = UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND +      TTI + CPI + JOB_VACAN + RES_POP, data = unemp_tra
in_med[rf_boot_sample_med,      ], mtry = 4, ntree = 58, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 58
No. of variables tried at each split: 4

          Mean of squared residuals: 0.1074918
                    % Var explained: 96.57
```

Summary of Random forest models with bootstrap

```
> bagged_rf_models_med
list()
> rf_boot_mse_med_train # 0.06809317
[1] 0.06809317
> rf_boot_rmse_med_train # 0.2609467
[1] 0.2609467
> #MAE
> (rf_boot_train_mae = postResample(pred = rf_boot_pred_med_train, unemp_train_med$UNEMP_RATE))
     RMSE  Rsquared       MAE
0.2609467 0.9805702 0.1716870
```

Performance of trained Decision Tree Random Forest Bootstrap model on training data

```
> rf_boot_mse_med # 3.912915
[1] 3.912915
> rf_boot_rmse_med # 1.978109
[1] 1.978109
> #MAE
> (rf_boot_test_mae = postResample(pred = rf_boot_pred_med, unemp_test_med$UNEMP_RATE))
     RMSE  Rsquared       MAE
1.9781090 0.2782989 1.8450624
>
```

Performance of trained Decision Tree Random Forest Bootstrap model on test data

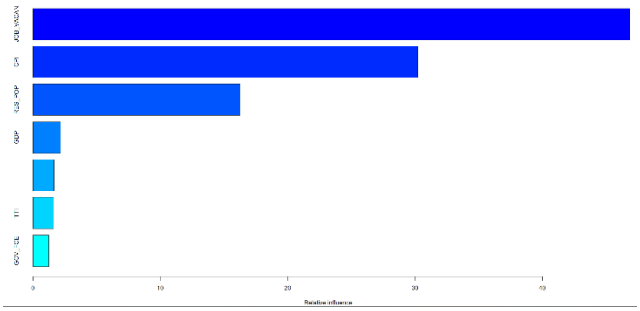
## **Appendix Z:** R code for Decision Tree Boosting.

```{r}

#-----Boosting-----

set.seed(1234)

# Model

boost_model_med <- gbm(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,

                       data = unemp_train_med,

                       distribution = "gaussian",

                       n.trees = 5000 ,

                       interaction.depth = 4)

# Summary

summary(boost_model_med)

boost_model_med

# Predict

boost_pred_med <- predict(boost_model_med, unemp_test_med)

# MSE

boost_mse_med <- mean((boost_pred_med - unemp_test_med$UNEMP_RATE)^2)

boost_mse_med #2.50152

# RMSE

boost_rmse_med <- sqrt(mean((boost_pred_med - unemp_test_med$UNEMP_RATE)^2))

boost_rmse_med #1.58162

# Tuning parameters

boost_grid_med <-  expand.grid(interaction.depth = c(2,5,10,15),  # Max Tree Depth

                       n.trees = (1:100), # Boosting Iterations

                       shrinkage = c(0.001, 0.01, 0.1, 1),# Shrinkage = Learning Rate

                       n.minobsinnode = c(5,10,20) # minimum number of training set samples in a node to commence splitting

                       )

# Cross-validation

fitControl_med <- trainControl(method = "cv",

                           number = 10,

                           savePredictions = "final")

start_time_boost = Sys.time()  # Start the timer

set.seed(1234)

# Hyperparameter tuning

boost_grid_train_med <- train(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,

                          data = unemp_train_med,
```

```
                    method = "gbm",
                    trControl = fitControl_med,
                    verbose = FALSE,
                    tuneGrid = boost_grid_med
end_time_boost = Sys.time()  # End the timer
(time_diff_boost = end_time_boost - start_time_boost) # Time difference of 43.43202 secs
# Best tuning parameters
(best <- boost_grid_train_med$bestTune)
# n.trees interaction.depth shrinkage n.minobsinnode
#   82             10      0.1            5
# Optimal parameter
set.seed(1234)
# Model with optimal parameters
best_boost_model_med <- gbm(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,
                    data = unemp_train_med,
                    distribution = "gaussian",
                    n.trees = 82,
                    interaction.depth = 10,
                    shrinkage = 0.1,
                    n.minobsinnode = 5
# Summary
summary(best_boost_model_med)
best_boost_model_med
# Performance on train data
best_boost_pred_med_train <- predict(best_boost_model_med, unemp_train_med)
# MSE
best_boost_mse_med_train <- mean((best_boost_pred_med_train - unemp_train_med$UNEMP_RATE)^2)
best_boost_mse_med_train #0.01930614
# RMSE
best_boost_rmse_med_train <- sqrt(mean((best_boost_pred_med_train - unemp_train_med$UNEMP_RATE)^2))
best_boost_rmse_med_train #0.1389466
#MAE
(boost_train_mae = postResample(pred = best_boost_pred_med_train, unemp_train_med$UNEMP_RATE))
#      RMSE     Rsquared       MAE
#  0.13894655  0.99411041   0.09133378
# Performance on test data
best_boost_pred_med <- predict(best_boost_model_med, unemp_test_med)
# MSE
best_boost_mse_med <- mean((best_boost_pred_med - unemp_test_med$UNEMP_RATE)^2)
best_boost_mse_med #2.699348
```

```
# RMSE

best_boost_rmse_med <- sqrt(mean((best_boost_pred_med - unemp_test_med$UNEMP_RATE)^2))

best_boost_rmse_med #1.642969

#MAE

(boost_test_mae = postResample(pred = best_boost_pred_med, unemp_test_med$UNEMP_RATE))

#    RMSE    Rsquared      MAE

#  1.6429694  0.6030349   1.5544369
```

## **Appendix AA:** Decision Tree Boosting visualisation algorithms outputs.



Boosting variable importance chart

```
> # Best tuning parameters
> (best <- boost_grid_train_med$bestTune)
     n.trees interaction.depth shrinkage n.minobsinnode
3082      82                10       0.1              5
>
```

Boosting hyperparameter tuning optimal values

```
> summary(best_boost_model_med)
                                 var    rel.inf
JOB_VACAN                   JOB_VACAN 46.873110
CPI                               CPI 30.218441
RES_POP                       RES_POP 16.262551
GDP                               GDP  2.145523
CHNG_FCE_ALL_IND CHNG_FCE_ALL_IND  1.656912
TTI                               TTI  1.607648
GOV_FCE                       GOV_FCE  1.235816
> best_boost_model_med
gbm(formula = UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND +
    TTI + CPI + JOB_VACAN + RES_POP, distribution = "gaussian",
    data = unemp_train_med, n.trees = 82, interaction.depth = 10,
    n.minobsinnode = 5, shrinkage = 0.1)
A gradient boosted model with gaussian loss function.
82 iterations were performed.
There were 7 predictors of which 7 had non-zero influence.
>
```

Summary of trained Boosting model with optimal values

```
> best_boost_mse_med_train #0.01930614
[1] 0.01930614
> best_boost_rmse_med_train #0.1389466
[1] 0.1389466
> #MAE
> (boost_train_mae = postResample(pred = best_boost_pred_med_train, unemp_train_med$UNEMP_RATE))
      RMSE   Rsquared        MAE
0.13894655 0.99411041 0.09133378
>
```

Performance of trained Decision Tree Boosting model on training data

```
> best_boost_mse_med #2.699348
[1] 2.699348
> best_boost_rmse_med #1.642969
[1] 1.642969
> #MAE
> (boost_test_mae = postResample(pred = best_boost_pred_med, unemp_test_med$UNEMP_RATE))
     RMSE  Rsquared       MAE
1.6429694 0.6030349 1.5544369
```

Performance of trained Decision Tree Boosting model on test data

## **Appendix BB:** R code for SVM Linear kernel.

```{r}

#-----SVM, Linear-----

# Create a sequence of values from 1 to 50

ln_cost_values_med <- seq(1, 50, by = 1)

# Generate all combinations of the cost values

svm_lin_tgrid_med <- expand.grid(cost = ln_cost_values_med)

# Ensure the same results

set.seed(1234)
```

```
# Conduct 'cost' hyperparameter tuning for cost on radial SVM model. Fit the support vector classifier with 'scale = TRUE'. When 'scale = FALSE' resulted in the software to crash and no convergence.

svm_lin_tune_med <- tune(svm,

            UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,

            data = unemp_train_med,

            kernel = "linear",

            ranges = svm_lin_tgrid_med,

            scale = TRUE)

# Summary

summary(svm_lin_tune_med)

# View the best model from tuning.

(lin_bestmod_med <- svm_lin_tune_med$best.model)

#Parameters:

#   SVM-Type:  eps-regression

# SVM-Kernel:  linear

#       cost:  3

#      gamma:  0.1428571

#    epsilon:  0.1

#Number of Support Vectors:  110

# Model

lin_svm_med <- svm(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,

            data = unemp_train_med,

            kernel = "linear",

            cost = 3,

            scale = TRUE

# We can determine the support vector identities

lin_svm_med$index

# View summary of svm model

summary(lin_svm_med)

# This tells us, for instance, that a linear kernel was used with cost = 10,and that there were 115 support vectors.

# Performance on train

lin_pre_med_train <- predict(lin_svm_med, unemp_train_med)

# MSE

lin_mse_med_train = mean((lin_pre_med_train - unemp_train_med$UNEMP_RATE)^2)

lin_mse_med_train #0.9458506

# RMSE

lin_rmse_med_train = sqrt(mean((lin_pre_med_train - unemp_train_med$UNEMP_RATE)^2))

lin_rmse_med_train #0.9725485

#MAE

(lin_train_mae = postResample(pred = lin_pre_med_train, unemp_train_med$UNEMP_RATE))

#    RMSE     Rsquared      MAE
```

```
# 0.9725485   0.7308884    0.6599424

# Performance on test

lin_pre_med <- predict(lin_svm_med, unemp_test_med)

# MSE

lin_mse_med = mean((lin_pre_med - unemp_test_med$UNEMP_RATE)^2)

lin_mse_med #73.51656

# RMSE

lin_rmse_med = sqrt(mean((lin_pre_med - unemp_test_med$UNEMP_RATE)^2))

lin_rmse_med #8.57418

#MAE

(line_test_mae = postResample(pred = lin_pre_med, unemp_test_med$UNEMP_RATE))

#    RMSE     Rsquared      MAE

#  8.5741799  0.5410879   7.1118548
```

## **Appendix CC:** SVM Linear kernel visualisation algorithms outputs.

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
    3

- best performance: 1.144047

- Detailed performance results:
   cost    error dispersion
1     1 1.171127  0.5021385
2     2 1.163342  0.5123025
3     3 1.144047  0.5255424
4     4 1.147867  0.5252411
5     5 1.153207  0.5252454
6     6 1.154780  0.5205115
7     7 1.153839  0.5153826
8     8 1.153035  0.5156331
9     9 1.150815  0.5139399
10   10 1.150112  0.5140846
11   11 1.152595  0.5161358
12   12 1.153224  0.5168661
```

Hyperparameter tuning summary for cost value

```
Call:
svm(formula = UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP, data = unemp_train_med,
    kernel = "linear", cost = 3, scale = TRUE)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  linear
       cost:  3
      gamma:  0.1428571
    epsilon:  0.1


Number of Support Vectors:  110
```

Final SVM linear kernel model summary

```
> lin_mse_med_train #0.9458506
[1] 0.9458506
> lin_rmse_med_train #0.9725485
[1] 0.9725485
> #MAE
> (lin_train_mae = postResample(pred = lin_pre_med_train, unemp_train_med$UNEMP_RATE))
     RMSE  Rsquared       MAE
0.9725485 0.7308884 0.6599424
```

Performance of trained SVM Linear model on training data

```
> lin_mse_med #73.51656
[1] 73.51656
> lin_rmse_med #8.57418
[1] 8.57418
> #MAE
> (line_test_mae = postResample(pred = lin_pre_med, unemp_test_med$UNEMP_RATE))
     RMSE  Rsquared       MAE
8.5741799 0.5410879 7.1118548
```

Performance of trained SVM Linear model on test data

## **Appendix DD:** R code for Radial kernel.

```{r}

#-----SVM, Radial-----

# Create sequences for cost and gamma

rad_cost_values_med <- seq(0.1, 5, by = 0.25)

rad_gamma_values_med <- c(0.01, 0.1, 1, 5)

# Generate all combinations of the cost values

svm_rad_tgrid_med <- expand.grid(cost = rad_cost_values_med, gamma = rad_gamma_values_med)
```

```
start_time_rad_med = Sys.time() #Time how long it takes to run the svm() functio

# Ensure the same results

set.seed(1234)

# Conduct 'cost' and 'gamma' hyperparameter tuning for cost Aand radial SVM model. Fit the support vector classifier with 'scale = TRUE' for scaling.

svm_rad_tune_med <- tune(svm,

                         UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,

                         data = unemp_train_med,

                         kernel = "radial",

                         ranges = svm_rad_tgrid_med,

                         scale = TRUE)

end_time_rad_med = Sys.time()# End timer.

runtime_rad_med = end_time_rad_med - start_time_rad_med # Calculate run time.

print(runtime_rad_med) # Time difference of 11.17665 mins

# View the cross-validation errors of the models

summary(svm_rad_tune_med) # Based on detailed performance results, cost = 3.35 and gamma = 0.1 are the most optimal.

# View the best model from tuning.

(rad_best_mod_med <- svm_rad_tune_med$best.model)

# Parameters:

#   SVM-Type:  eps-regression

# SVM-Kernel:  radial

#       cost:  4.10

#      gamma:  0.1

#    epsilon:  0.1

#Number of Support Vectors:  108

# Model

rad_svm_med <- svm(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,

                   data = unemp_train_med,

                   kernel = "radial",

                   cost = 4.1,

                   gamma = 0.1,

                   epsilon = 0.1,

                   scale = TRUE)

# We can determine the support vector identities

rad_svm_med$index

#View summary of svm model

summary(rad_svm_med)

#This tells us, for instance, that a linear kernel was used with cost = 4.1, gamma = 0.1, epsilon = 0.1 and that there were 108 support vectors.
```

```
# Performance on train

rad_pre_med_train <- predict(rad_svm_med, unemp_train_med)

# MSE

rad_mse_med_train <- mean((rad_pre_med_train - unemp_train_med$UNEMP_RATE)^2)

rad_mse_med_train #0.2806018

# RMSE

rad_rmse_med_train <- sqrt(mean((rad_pre_med_train - unemp_train_med$UNEMP_RATE)^2))

rad_rmse_med_train #0.5297186

#MAE

(rad_train_mae = postResample(pred = rad_pre_med_train, unemp_train_med$UNEMP_RATE))

#     RMSE     Rsquared      MAE

#  0.5297186  0.9145497    0.3448363

# Performance on test

rad_pre_med <- predict(rad_svm_med, unemp_test_med)

# MSE

rad_mse_med <- mean((rad_pre_med - unemp_test_med$UNEMP_RATE)^2)

rad_mse_med #3.00141

# RMSE

rad_rmse_med <- sqrt(mean((rad_pre_med - unemp_test_med$UNEMP_RATE)^2))

rad_rmse_med #1.732458

#MAE

(rad_test_mae = postResample(pred = rad_pre_med, unemp_test_med$UNEMP_RATE))

#     RMSE     Rsquared      MAE

#  1.7324577  0.1283899   1.5637828
```

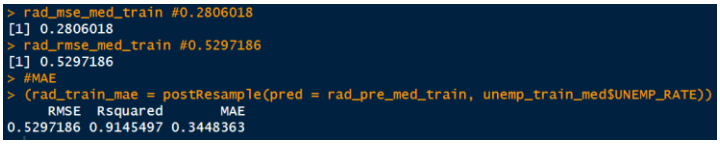
## **Appendix EE:** SVM Radial kernel visualization algorithms outputs.

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
  4.1   0.1

- best performance: 0.6400469

- Detailed performance results:
    cost gamma     error dispersion
1   0.10  0.01 2.0608134  1.0496129
2   0.35  0.01 1.4257868  0.6683199
3   0.60  0.01 1.2875866  0.5994118
4   0.85  0.01 1.2190898  0.5761869
5   1.10  0.01 1.1509708  0.5458016
6   1.35  0.01 1.0913611  0.5251955
7   1.60  0.01 1.0469945  0.5060202
8   1.85  0.01 1.0061729  0.4886313
9   2.10  0.01 0.9685606  0.4675341
10  2.35  0.01 0.9489326  0.4639325
11  2.60  0.01 0.9286977  0.4592639
12  2.85  0.01 0.9091029  0.4530778
```

Hyperparameter tuning summary for cost and gamma value

```
Call:
best.tune(METHOD = svm, train.x = UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN +
    RES_POP, data = unemp_train_med, ranges = svm_rad_tgrid_med, kernel = "radial", scale = TRUE)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  4.1
      gamma:  0.1
    epsilon:  0.1


Number of Support Vectors:  108
```

Final SVM Radial kernel model summary




Performance of trained SVM Radial model on training data

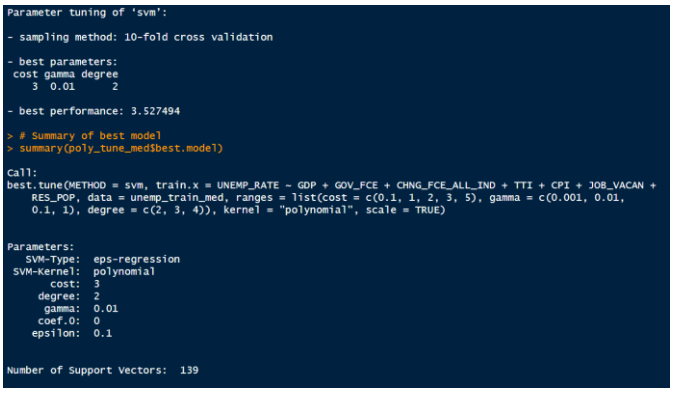

Performance of trained SVM Radial model on test data

## **Appendix FF:** R code for Radial Polynomial kernel.

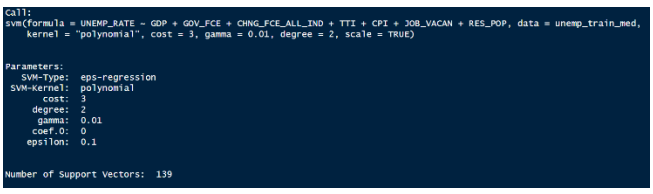
```{r}

#-----SVM, polynomial -----

#Ensure the same results

set.seed(1234)

#Conduct 'cost', 'gamma', and 'degree' hyperparameter tuning for polynomial SVM model. Fit the support vector classifier with 'scale = TRUE'.

poly_tune_med <- tune(svm,

UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,

data = unemp_train_med,

kernel = "polynomial",

ranges = list(cost = c(0.1, 1, 2, 3, 5),

gamma = c(0.001, 0.01, 0.1, 1),

degree = c(2, 3, 4)),

scale = TRUE)

# Summary of best model

summary(poly_tune_med$best.model)

# Parameters:

#  SVM-Type:  eps-regression

# SVM-Kernel:  polynomial

#  cost:  3

#  degree:  2

#  gamma:  0.01

#  coef.0:  0

#  epsilon:  0.1

# Number of Support Vectors:  139

# Model with best parameters

poly_svm_med <- svm(UNEMP_RATE ~ GDP + GOV_FCE + CHNG_FCE_ALL_IND + TTI + CPI + JOB_VACAN + RES_POP,

data = unemp_train_med,

kernel = "polynomial",

cost = 3,

gamma = 0.01,

degree = 2,

scale = TRUE)

# We can determine the support vector identities

poly_svm_med$index
```

```
# Summary

summary(poly_svm_med)

# This tells us, for instance, that a linear kernel was used with cost = 3, degree = 2, gamma = 0.01 and that there were 139 support vectors.

# Performance on train

poly_pre_med_train <- predict(poly_svm_med, unemp_train_med)

# MSE

poly_mse_med_train <- mean((poly_pre_med_train - unemp_train_med$UNEMP_RATE)^2)

poly_mse_med_train #3.39982

# RMSE

poly_rmse_med_train <- sqrt(mean((poly_pre_med_train - unemp_train_med$UNEMP_RATE)^2))

poly_rmse_med_train #1.843861

#MAE

(poly_train_mae = postResample(pred = poly_pre_med_train, unemp_train_med$UNEMP_RATE))

#      RMSE    Rsquared       MAE

#   1.84386110 0.08718197   1.41959618

# Performance on test

poly_pre_med <- predict(poly_svm_med, unemp_test_med)

# MSE

poly_mse_med <- mean((poly_pre_med - unemp_test_med$UNEMP_RATE)^2)

poly_mse_med #3.812724

# RMSE

poly_rmse_med <- sqrt(mean((poly_pre_med - unemp_test_med$UNEMP_RATE)^2))

poly_rmse_med #1.95262

#MAE

(poly_test_mae = postResample(pred = poly_pre_med, unemp_test_med$UNEMP_RATE))

#    RMSE   Rsquared      MAE

# 1.9526198 0.1693279    1.8144649
```

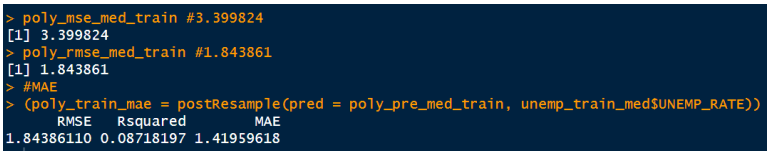
## **Appendix GG:** SVM Polynomial kernel visualisation algorithms outputs.



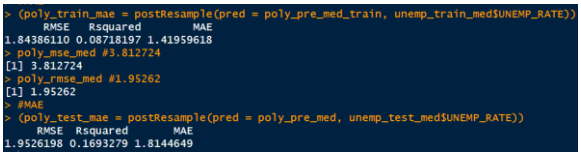

Hyperparameter tuning summary for cost, gamma, and degree value




Final SVM Polynomial kernel model summary




Performance of trained SVM Polynomial model on training data




Performance of trained SVM Polynomial model on test data

**Appendix HH:** Performance of Machine Learning Algorithms on Training Data.

| ML Algorithms | MSE | RMSE | MAE |
|---|---|---|---|
| **Ridge Regression** | 0.923260 | 0.960864 | 0.724287 |
| **Lasso Regression** | 0.892308 | 0.944621 | 0.711528 |
| **Decision Tree: CARTs (Not considered)** | 0.308784 | 0.555684 | 0.419869 |
| **Decision Tree: Bagging** | 0.078271 | 0.279771 | 0.183003 |
| **Decision Tree: Random Forest** | 0.068093 | 0.260947 | 0.171687 |
| **Decision Tree: Boosting** | 0.019306 | 0.138947 | 0.0913338 |
| **SVM: Linear Kernel** | 0.945851 | 0.972549 | 0.6599424 |
| **SVM: Radial Kernel** | 0.280602 | 0.529719 | 0.3448363 |
| **SVM: Polynomial Kernel** | 3.399824 | 1.843861 | 1.4195962 |

**Appendix II:** Performance of Machine Learning Algorithms on Test Data.

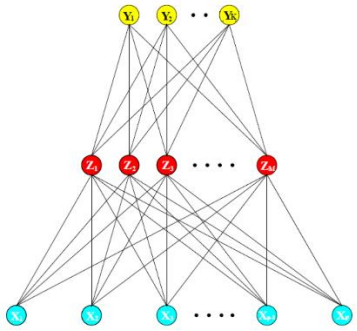
| ML Algorithms | MSE | RMSE | MAE |
|---|---|---|---|
| **Ridge Regression** | 54.375237 | 7.37395665719194 | 6.5583333 |
| **Lasso Regression** | 66.8404285 | 8.17559953525553 | 7.1065695 |
| **Decision Tree: CARTs (Not considered)** | 1.901695 | 1.37901958906499 | 1.239968 |
| **Decision Tree: Bagging** | 3.748449 | 1.93609125187114 | 1.7991926 |
| **Decision Tree: Random Forest** | 3.91291520812168 | 1.9781089980387 | 1.8450624 |
| **Decision Tree: Boosting** | 2.69934834698782 | 1.64296936885257 | 1.5544369 |
| **SVM: Linear Kernel** | 73.5165611616787 | 8.57417991190287 | 7.1118548 |
| **SVM: Radial Kernel** | 3.0014095243612 | 1.73245765442079 | 1.5637828 |
| **SVM: Polynomial Kernel** | 3.81272405275726 | 1.95261979216571 | 1.8144649 |

**Appendix JJ:** Summary of activation functions.

| Activation function | Description | Equation |
|---|---|---|
| **sigmoid** | Reduces the input values between 0 and 1,providing suitability for classification. | f(z) = 1/(1+ e^-z) |
| **relu** | If input is positive it directly return it, if it is less that zero it will return 0. | f(x)={max(0,x), x >=0} { 0 , x <0} |
| **tanh** | Similar to sigmoid but the input values are between -1 and 1. | $f(x)=( 2/ (1 + e^{-2x})) -1$ |

(Bag, 2021)

**Appendix KK:** Simplified visual overview of ANN



## **Appendix LL:** ANN full code

Artificial Neural Network: Median

```{r}

# Separate response variables for train and test

unemp_train_nn_y_med = unemp_train_med$UNEMP_RATE

unemp_test_nn_y_med = unemp_test_med$UNEMP_RATE

# Separate predictor variables for train and test

unemp_train_nn_x_med = unemp_train_med[,-1]

unemp_test_nn_x_med = unemp_test_med[,-1]

# Scale

mean_med <- apply(unemp_train_nn_x_med, 2, mean)

std_med <- apply(unemp_train_nn_x_med, 2, sd)

unemp_train_nn_x_med <- scale(unemp_train_nn_x_med, center = mean_med, scale = std_med)

head(unemp_train_nn_x_med)

dim(unemp_train_nn_x_med) # 153   7

sum(is.na(unemp_train_nn_x_med)) # 0

unemp_test_nn_x_med <- scale(unemp_test_nn_x_med, center = mean_med, scale = std_med)

head(unemp_test_nn_x_med)

dim(unemp_test_nn_x_med) # 10  7

sum(is.na(unemp_test_nn_x_med)) # 0

Using forloop for hyperparameter tuning (1 layers)

```

```{r}

#------ NN adam ------

# Define lists of parameters to tune

epochs_list <- c(100, 200, 300)

units_list <- c(32, 64, 128)

batch_sizes <- c(8, 16, 32)

#Set cross validation amount

k <- 4  # Number of folds for cross-validation

# Create an empty list to store the results

results_adam_1 <- list()

# Iterate through parameter combinations
```

```
for (epochs in epochs_list) {
 for (units in units_list) {
  for (batch_size in batch_sizes) {
   cat("Training with epochs =", epochs, ", units =", units, ", batch size =", batch_size, "\n")
   # Create a vector to store cross-validation results
   cv_scores_adam_1 <- numeric(k)
   for (i in 1:k) {
    cat("Processing fold #", i, "\n")
    # Define and compile the model
    nn_model_adam_1 <- keras_model_sequential() %>%
     layer_dense(units = units, activation = "relu", input_shape = c(ncol(unemp_train_nn_x_med))) %>%
     layer_dense(units = 1)
    nn_model_adam_1 %>% compile(
     optimizer = "adam",
     loss = "mse",
     metrics = c("mae"))
    set.seed(1234)
    # Split data into training and validation sets for this fold
    val_indices_adam_1 <- which(fold_id == i)
    val_data_adam_1 <- unemp_train_nn_x_med[val_indices_adam_1, ]
    val_targets_adam_1 <- unemp_train_nn_y_med[val_indices_adam_1]
    partial_train_data_adam_1 <- unemp_train_nn_x_med[-val_indices_adam_1, ]
    partial_train_targets_adam_1 <- unemp_train_nn_y_med[-val_indices_adam_1]
    # Train the model
    nn_model_hist_adam_1 <- nn_model_adam_1 %>%
     fit(partial_train_data_adam_1,
         partial_train_targets_adam_1,
         epochs = epochs,
         batch_size = batch_size,
         verbose = 1)
    # Evaluate the model on the validation data for this fold
    val_metrics_adam_1 <- nn_model_adam_1 %>% evaluate(val_data_adam_1, val_targets_adam_1, verbose = 0)
    cv_scores_adam_1[i] <- val_metrics_adam_1[['mae']]
   }
   # Store the cross-validation results for this combination of hyperparameters
   results_adam_1[[paste("epochs", epochs, "_units", units, "_batch", batch_size, sep = "_")]] <- mean(cv_scores_adam_1)
  }
 }
}
```

```
View(results_adam_1)

Best_results_adam_1 = results_adam_1[["epochs_300__units_32__batch_8"]]

Best_results_adam_1 # epochs = 300, units = 32, batch = 8, MAE = 0.5002812

#------ NN rmsprop ------

# Create an empty list to store the results

results_rmsprop_1 <- list()

# Iterate through parameter combinations

for (epochs in epochs_list) {

 for (units in units_list) {

  for (batch_size in batch_sizes) {

   cat("Training with epochs =", epochs, ", units =", units, ", batch size =", batch_size, "\n")

   # Create a vector to store cross-validation results

   cv_scores_rmsprop_1 <- numeric(k)

   for (i in 1:k) {

    cat("Processing fold #", i, "\n")

    # Define and compile the model

    nn_model_rmsprop_1 <- keras_model_sequential() %>%

     layer_dense(units = units, activation = "relu", input_shape = c(ncol(unemp_train_nn_x_med))) %>%

     layer_dense(units = 1)

    nn_model_rmsprop_1 %>% compile(

     optimizer = "rmsprop",

     loss = "mse",

     metrics = c("mae"))

    set.seed(1234)

    # Split data into training and validation sets for this fold

    val_indices_rmsprop_1 <- which(fold_id == i)

    val_data_rmsprop_1 <- unemp_train_nn_x_med[val_indices_rmsprop_1, ]

    val_targets_rmsprop_1 <- unemp_train_nn_y_med[val_indices_rmsprop_1]

    partial_train_data_rmsprop_1 <- unemp_train_nn_x_med[-val_indices_rmsprop_1, ]

    partial_train_targets_rmsprop_1 <- unemp_train_nn_y_med[-val_indices_rmsprop_1]

    # Train the model

    nn_model_hist_rmsprop_1 <- nn_model_rmsprop_1 %>%

     fit(partial_train_data_rmsprop_1,

         partial_train_targets_rmsprop_1,

         epochs = epochs,

         batch_size = batch_size,

         verbose = 1)

    # Evaluate the model on the validation data for this fold

    val_metrics_rmsprop_1 <- nn_model_rmsprop_1 %>% evaluate(val_data_rmsprop_1, val_targets_rmsprop_1, verbose = 0)

    cv_scores_rmsprop_1[i] <- val_metrics_rmsprop_1[['mae']]
```

```
    }
    # Store the cross-validation results for this combination of hyperparameters
    results_rmsprop_1[[paste("epochs", epochs, "_units", units, "_batch", batch_size, sep = "_")]] <- mean(cv_scores_rmsprop_1)
   }
  }
}
View(results_rmsprop_1)
Best_results_rmsprop_1 = results_rmsprop_1[["epochs_300__units_128__batch_16"]]
Best_results_rmsprop_1 #epochs = 300, units = 128, batch = 15, MAE 0.4770097
```

Using forloop for hyperparameter tuning (2 layers)

```{r}
#----- NN adam-----
# Create an empty list to store the results
results_adam <- list(
# Iterate through parameter combinations
for (epochs in epochs_list) {
 for (units in units_list) {
  for (batch_size in batch_sizes) {
    cat("Training with epochs =", epochs, ", units =", units, ", batch size =", batch_size, "\n")
    # Create a vector to store cross-validation results
    cv_scores <- numeric(k)
    for (i in 1:k) {
      cat("Processing fold #", i, "\n"
      # Define and compile the model
      nn_model_adam <- keras_model_sequential() %>%
        layer_dense(units = units, activation = "relu", input_shape = c(ncol(unemp_train_nn_x_med))) %>%
        layer_dense(units = units, activation = "relu") %>%
        layer_dense(units = 1
      nn_model_adam %>% compile(
        optimizer = "adam",
        loss = "mse",
        metrics = c("mae")
      set.seed(1234)
      # Split data into training and validation sets for this fold
      val_indices <- which(fold_id == i)
      val_data <- unemp_train_nn_x_med[val_indices, ]
      val_targets <- unemp_train_nn_y_med[val_indices]
      partial_train_data <- unemp_train_nn_x_med[-val_indices, ]
      partial_train_targets <- unemp_train_nn_y_med[-val_indices]
```

```
      # Train the model
      nn_model_hist_adam <- nn_model_adam %>%
       fit(partial_train_data,
          partial_train_targets,
          epochs = epochs,
          batch_size = batch_size,
          verbose = 1)
      # Evaluate the model on the validation data for this fold
      val_metrics <- nn_model_adam %>% evaluate(val_data, val_targets, verbose = 0)
      cv_scores[i] <- val_metrics[['mae']]
     }
     # Store the cross-validation results for this combination of hyperparameters
     results_adam[[paste("epochs", epochs, "_units", units, "_batch", batch_size, sep = "_")]] <- mean(cv_scores)
   }
  }
}
View(results_adam)
Best_results_adam_2 = results_adam[["epochs_200__units_32__batch_8"]]
Best_results_adam_2 # epochs = 200, units = 32, batch = 8, MAE = 0.4569995
#------ NN rmsprop ------
start_time_nn = Sys.time()  # Start the timer
# Create an empty list to store the results
results_rmsprop_2 <- list(
# Iterate through parameter combinations
for (epochs in epochs_list) {
 for (units in units_list) {
  for (batch_size in batch_sizes) {
   cat("Training with epochs =", epochs, ", units =", units, ", batch size =", batch_size, "\n")
   # Create a vector to store cross-validation results
   cv_scores_rmsprop_2 <- numeric(k)
   for (i in 1:k) {
    cat("Processing fold #", i, "\n")
    # Define and compile the model
    nn_model_rmsprop_2 <- keras_model_sequential() %>%
     layer_dense(units = units, activation = "relu", input_shape = c(ncol(unemp_train_nn_x_med))) %>%
     layer_dense(units = units, activation = "relu") %>%
     layer_dense(units = 1)
    nn_model_rmsprop_2 %>% compile(
     optimizer = "rmsprop",
```

```
      loss = "mse",
      metrics = c("mae"))
    set.seed(1234)
    # Split data into training and validation sets for this fold
    val_indices_rmsprop_2 <- which(fold_id == i)
    val_data_rmsprop_2 <- unemp_train_nn_x_med[val_indices_rmsprop_2, ]
    val_targets_rmsprop_2 <- unemp_train_nn_y_med[val_indices_rmsprop_2]
    partial_train_data_rmsprop_2 <- unemp_train_nn_x_med[-val_indices_rmsprop_2, ]
    partial_train_targets_rmsprop_2 <- unemp_train_nn_y_med[-val_indices_rmsprop_2]
    # Train the model
    nn_model_hist_rmsprop_2 <- nn_model_rmsprop_2 %>%
      fit(partial_train_data_rmsprop_2,
        partial_train_targets_rmsprop_2,
        epochs = epochs,
        batch_size = batch_size,
        verbose = 1)
    # Evaluate the model on the validation data for this fold
    val_metrics_rmsprop_2 <- nn_model_rmsprop_2 %>% evaluate(val_data_rmsprop_2, val_targets_rmsprop_2, verbose = 0)
    cv_scores_rmsprop_2[i] <- val_metrics_rmsprop_2[['mae']]
   }
   # Store the cross-validation results for this combination of hyperparameters
   results_rmsprop_2[[paste("epochs", epochs, "_units", units, "_batch", batch_size, sep = "_")]] <- mean(cv_scores_rmsprop_2)
  }
 }
}
end_time_nn = Sys.time()  # End the timer
(time_diff_nn = end_time_nn - start_time_nn) # Time difference of 54.46687 mins
# A plot of the bootstrapped validation MAE loss from network model training history.
nn_model_hist_rmsprop_2_plot <- data.frame(x = c(nn_model_hist_rmsprop_2$params$epochs),
                        y = nn_model_hist_rmsprop_2$metrics)
# Visualisation
ggplot(nn_epoch_plot_rmsprop_med, aes(x=x, y=y)) +geom_smooth() +xlab("epoch") +ylab("Estimated Validation MAE loss") + geom_vline(xintercept = 30,
linetype="dotted", color = "red", size=1.5)
plot(nn_model_hist_rmsprop_2)
View(results_rmsprop_2)
Best_results_rmsprop_2 = results_rmsprop_2[["epochs_200__units_32__batch_16"]]
Best_results_rmsprop_2 # epochs = 200, units = 32, batch = 16, MAE 0.3679621
```

Using forloop for hyperparameter tuning (3 layers)

```{r}
#------ NN adam ------
# Create an empty list to store the results
results_adam_3 <- list()
# Iterate through parameter combinations
for (epochs in epochs_list) {
 for (units in units_list) {
  for (batch_size in batch_sizes) {
   cat("Training with epochs =", epochs, ", units =", units, ", batch size =", batch_size, "\n")
   # Create a vector to store cross-validation results
   cv_scores_adam_3 <- numeric(k)
   for (i in 1:k) {
    cat("Processing fold #", i, "\n")
    # Define and compile the model
    nn_model_adam_3 <- keras_model_sequential() %>%
     layer_dense(units = units, activation = "relu", input_shape = c(ncol(unemp_train_nn_x_med))) %>%
     layer_dense(units = units, activation = "relu") %>%
     layer_dense(units = units, activation = "relu") %>%
     layer_dense(units = 1)
    nn_model_adam_3 %>% compile(
     optimizer = "adam",
     loss = "mse",
     metrics = c("mae"))
    set.seed(1234)
    # Split data into training and validation sets for this fold
    val_indices_adam_3 <- which(fold_id == i)
    val_data_adam_3 <- unemp_train_nn_x_med[val_indices_adam_3, ]
    val_targets_adam_3 <- unemp_train_nn_y_med[val_indices_adam_3]
    partial_train_data_adam_3 <- unemp_train_nn_x_med[-val_indices_adam_3, ]
    partial_train_targets_adam_3 <- unemp_train_nn_y_med[-val_indices_adam_3]
    # Train the model
    nn_model_hist_adam_3 <- nn_model_adam_3 %>%
     fit(partial_train_data_adam_3,
         partial_train_targets_adam_3,
         epochs = epochs,
         batch_size = batch_size,
         verbose = 1)
    # Evaluate the model on the validation data for this fold
    val_metrics_adam_3 <- nn_model_adam_3 %>% evaluate(val_data_adam_3, val_targets_adam_3, verbose = 0)
    cv_scores_adam_3[i] <- val_metrics_adam_3[['mae']]
```

```
    }
    # Store the cross-validation results for this combination of hyperparameters
    results_adam_3[[paste("epochs", epochs, "_units", units, "_batch", batch_size, sep = "_")]] <- mean(cv_scores_adam_3)
   }
  }
}
View(results_adam_3)
Best_results_adam_3 = results_adam_3[["epochs_200__units_32__batch_8"]]
Best_results_adam_3 #epochs = 200, units = 32, batch = 8,  MAE = 0.4894649
#------ NN rmsprop ------
# Create an empty list to store the results
results_rmsprop_3 <- list()
# Iterate through parameter combinations
for (epochs in epochs_list) {
 for (units in units_list) {
  for (batch_size in batch_sizes) {
   cat("Training with epochs =", epochs, ", units =", units, ", batch size =", batch_size, "\n")
   # Create a vector to store cross-validation results
   cv_scores_rmsprop_3 <- numeric(k)
   for (i in 1:k) {
    cat("Processing fold #", i, "\n")
    # Define and compile the model
    nn_model_rmsprop_3 <- keras_model_sequential() %>%
     layer_dense(units = units, activation = "relu", input_shape = c(ncol(unemp_train_nn_x_med))) %>%
     layer_dense(units = units, activation = "relu") %>%
     layer_dense(units = units, activation = "relu") %>%
     layer_dense(units = 1)
    nn_model_rmsprop_3 %>% compile(
     optimizer = "rmsprop",
     loss = "mse",
     metrics = c("mae"))
    set.seed(1234)
    # Split data into training and validation sets for this fold
    val_indices_rmsprop_3 <- which(fold_id == i)
    val_data_rmsprop_3 <- unemp_train_nn_x_med[val_indices_rmsprop_3, ]
    val_targets_rmsprop_3 <- unemp_train_nn_y_med[val_indices_rmsprop_3]
    partial_train_data_rmsprop_3 <- unemp_train_nn_x_med[-val_indices_rmsprop_3, ]
    partial_train_targets_rmsprop_3 <- unemp_train_nn_y_med[-val_indices_rmsprop_3]
    # Train the model
    nn_model_hist_rmsprop_3 <- nn_model_rmsprop_3 %>%
```

```
      fit(partial_train_data_rmsprop_3,
          partial_train_targets_rmsprop_3,
          epochs = epochs,
          batch_size = batch_size,
          verbose = 1)
      # Evaluate the model on the validation data for this fold
      val_metrics_rmsprop_3 <- nn_model_rmsprop_3 %>% evaluate(val_data_rmsprop_3, val_targets_rmsprop_3, verbose = 0)
      cv_scores_rmsprop_3[i] <- val_metrics_rmsprop_3[['mae']]
    }
    # Store the cross-validation results for this combination of hyperparameters
    results_rmsprop_3[[paste("epochs", epochs, "_units", units, "_batch", batch_size, sep = "_")]] <- mean(cv_scores_rmsprop_3)
  }
 }
}
View(results_rmsprop_3)
Best_results_rmsprop_3 = results_rmsprop_3[["epochs_200__units_32__batch_16"]]
Best_results_rmsprop_3 # epochs = 200, units = 32, batch = 16, MAE 0.5315303
```

```{r}
print("Based on the forloop hyperparameter tuning and evaluating their performances on training, the best model was the 2 layer NN using an 'rmsprop', epochs = 200, units = 32, batch = 16, MAE 0.3679621")
```

**Appendix MM:** Performance of Artificial Neural Networks using a Forloop K-fold Cross-Validation with Hyperparameter Tuning on Training Training Data.

| Artificial Neural Network | MSE | MAE |
|---|---|---|
| **adam optimizer, 1 layer: Train** | Error in calculating. Refer to MAE. | 0.500281 |
| **rmsprop optimizer 1 layer: Train** | Error in calculating. Refer to MAE. | 0.477010 |
| **adam optimizer, 2 layers: Train** | Error in calculating. Refer to MAE. | 0.457000 |
| **rmsprop optimizer 2 layers: Train** | Error in calculating. Refer to MAE. | 0.367962 |
| **adam optimizer, 3 layers: Train** | Error in calculating. Refer to MAE. | 0.489465 |
| **rmsprop optimizer 3 layers: Train** | Error in calculating. Refer to MAE. | 0.531530 |

**Appendix NN:** Comparison of Optimal Decision Tree Boosting vs Artificial Neural Network on test data.

| Method | MSE | RMSE | MAE |
|---|---|---|---|
| **Decision Tree: Boosting** | 2.699348 | 1.642969 | 1.554437 |
| **Artificial Neural Network** | 2.330682 | 1.526657 | 1.220769 |

# Bibliography


Australian Institute of Health and Welfare. (2023, September 16). *Employment and unemployment*. Australian Institute of Health and Welfare. Accessed October 20th, 2023, from https://www.aihw.gov.au/reports/australias-welfare/employment-unemployment

Baeldung. (2020, December 29). *Epoch in Neural Networks | Baeldung on Computer Science*. Www.baeldung.com. Accessed October 21st, 2023, https://www.baeldung.com/cs/epoch-neural-networks#:~:text=An%20epoch%20means%20training%20the

Bag, S. (2021, April 26). *Activation Functions — All You Need To Know!* Analytics Vidhya. Accessed October 21st, 2023, https://medium.com/analytics-vidhya/activation-functions-all-you-need-to-know-355a850d025e

*But what is a neural network? | Chapter 1, Deep learning*. (n.d.). Www.youtube.com. Retrieved October 25, 2023, from https://www.youtube.com/watch?v=aircAruvnKk&t=3s

*Data Preprocessing - an overview | ScienceDirect Topics*. (n.d.). Www.sciencedirect.com. Accessed October 19th, 2023, https://www.sciencedirect.com/topics/engineering/data-preprocessing#:~:text=Data%20preprocessing%20is%20a%20required

Davis, K., McCarthy, M., & Bridges, J. (2016). The Labour Market during and after the Terms of Trade Boom | Bulletin – March Quarter 2016. *Bulletin*, *March*. Accessed October 19th, 2023, from, https://www.rba.gov.au/publications/bulletin/2016/mar/1.html

*Evaluation Metrics for Regression models- MAE Vs MSE Vs RMSE vs RMSLE*. (n.d.). Akhilendra.com. Accessed October 21st, 2023, from, https://akhilendra.com/evaluation-metrics-regression-mae-mse-rmse-rmsle/

National Museum of Australia. (2019). Federation | National Museum of Australia. *Nma.gov.au*. Accessed October 20th, 2023, from, https://www.nma.gov.au/defining-moments/resources/federation

Reserve Bank of Australia. (n.d.). *The Global Financial Crisis*. Reserve Bank of Australia. Accessed October 19th, 2023, from, https://www.rba.gov.au/education/resources/explainers/the-global-financial-crisis.html

Reserve Bank of Australia. (2019). *Unemployment: Its Measurement and Types*. Reserve Bank of Australia. Accessed October 20th, 2023, from https://www.rba.gov.au/education/resources/explainers/unemployment-its-measurement-and-types.html

Simeon Kostadinov. (2019, August 8). *Understanding Backpropagation Algorithm*. Medium; Towards Data Science. Accessed October 20th, 2023, from https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

*Spurious Correlation - an overview | ScienceDirect Topics*. (n.d.). Www.sciencedirect.com. Accessed October 20th, 2023, from https://www.sciencedirect.com/topics/mathematics/spurious-correlation

Statistics, c=AU; o=Commonwealth of A. ou=Australian B. of. (2010, February 4). *Main Features - Main Features*. Www.abs.gov.au. Accessed October 21st, 2023, from https://www.abs.gov.au/ausstats/abs@.nsf/mf/6354.0.55.001