Predicting Australia’s Unemployment Rate: A Machine Learning and Neural Network Approach

Rijo Kuruvilla (14171011)

I. Abstract

Unemployment is a measure of a country's economic health and has a significant impact on its population. Unemployment is a key indicator of how jobs, people, and government policies interact in a country's labour market, influencing people's lives and the economy. The Australian unemployment rate has fluctuated over the past four decades, influenced by economic conditions and government policies. The purpose of this report is to examine the Australian unemployment rate between 1982 and 2023, providing insights into its dynamics and analyzing the factors that have influenced it over the years. The study involves applying data wrangling methodologies to the dataset and comparing the performance of Machine Learning (ML) models and Artificial Neural Networks (ANN) to assess and predict the factors driving the unemployment rate in Australia. The data used in this report is aggregated and collected from the Australian Bureau of Statistics (ABS) Labour Force Survey. The ABS Labour Force Survey is a monthly survey that measures the employment and unemployment status of people aged 15 and above. The data in this report includes important information, such as the country's overall economic output (Gross Domestic Product), government and industry spending, terms of trade index consumer price trends, job opportunities (number of job vacancies), and the estimated population of residents. For the analysis a diverse set of modelling techniques were deployed which included Regularization (Shrinkage) models, Tree-based models, Support Vector Machines (SVM) and ANN to predict the unemployment rate.

Since the early 1980s, Australia's unemployment rate has dropped significantly, from a high of 11.15% in December 1992 to a record low of 3.4% in October 2022. However, the unemployment rate has recently increased slightly, rising to 3.6% in September 2023. The study's findings also revealed that the Estimated Resident Population, Consumer Price Index, and Job Vacancies were the most important factors in determining the unemployment rate. Stochastic Gradient Boosting and SVM with a polynomial kernel outperformed other ML models, though Neural Networks have the potential to outperform ML models, particularly if more data is available. The findings of the report can help shape government policies and programs aimed at reducing unemployment and promoting full employment. For example, based on the key predictors identified, the government could invest in infrastructure and education to boost economic growth and job creation. Additionally, assisting job seekers and businesses in adapting to a changing economy can improve job prospects. Furthermore, the government could create policies that encourage skill development and lifelong learning, as well as address the root causes of unemployment, such as discrimination and social exclusion. Other factors, such as technological advancement, global economic conditions, natural disasters, and government policies, can all have an impact on unemployment rates, in addition to those mentioned in this report.

II. Brief overview of the Australian unemployment rate

Employment and unemployment are fundamental determinants of a nation's socio-economic strength. They underpin a country's economic performance, providing individuals with the means to support themselves, their families, and their communities. Employment is closely tied to both physical and mental well-being, playing a pivotal role in shaping the overall quality of life for individuals (Australian Institute of Health and Welfare, 2023).

The Australian unemployment rate has shown significant variability from December 1982 to December 2020. In the early 1980s, Australia faced high unemployment due to economic restructuring and global economic challenges (Australian Institute of Health and Welfare, 2023). The unemployment rate for individuals aged 15 and above reached a peak of 11.15% in 1992, primarily due to a recession and the decline of traditional industries in the early 1990s (Australian Institute of Health and Welfare, 2023). However, as the Australian economy rebounded, the unemployment rate gradually declined, reaching its lowest point at 4.1% in 2008 (Australian Bureau of Statistics, 2008).The Global Financial Crisis (GFC) in 2009 had a significant impact on the Australian labour market. While the unemployment rate did increase during this time (approximately 6%), it remained relatively low compared to many other countries, largely due to stimulus packages and policy responses. In the years following the GFC, Australia experienced a resource-driven economic boom, driven by high demand for its natural resources, particularly from China (Bishop et al., 2013). This period was characterized by low unemployment, with the rate hovering around 5%.

Technological advancements and automation significantly affected the labour market, creating new opportunities and displacing workers in traditional roles. The rise of the digital age transformed the employment landscape. The Australian labour market continued to evolve, driven by shifts in industries, the profound influence of technology, and the importance of education and skills development. Additionally, the COVID-19 pandemic had a significant impact on the Australian employment market. In 2020, the unemployment rate reached a peak of 7.5% between March and July 2020, the highest level since 1992. However, the unemployment rate has since recovered to pre-pandemic levels, reaching 3.6% in September 2023 . This recovery underscores the resilience of the Australian labour market and the effectiveness of government responses in mitigating the impact of a global crisis.

Looking ahead, with global uncertainties, rising inflation, internal conflicts, wars, and other external factors, questions arise about Australia's ability to sustain employment opportunities for its population. This paper aims to conduct a comprehensive analysis of the Australian unemployment rate, identify its determinants, and employ various ML models and Neural Networks to predict future unemployment rate trends. The insights and predictions generated by this study have the potential to provide valuable assistance to relevant organizations and policymakers, enabling them to make informed decisions to reduce unemployment rates and ensure the stability of the labour market.

III. Data

The data "AUS_data_2023.xlsx" , used in this report is aggregated and collected from the Australian Bureau of Statistics (ABS). This data consists of 166 observations and 9 variables and is available quarterly from Dec 1982 to March 2023. The data includes the response variable (unemployment rate = Y) and 7 predictors variables ,Percentage change in GDP (X1), Percentage change in the Government final consumption expenditure (X2), Percentage change in final consumption expenditure of all industry sectors (X3), Term of trade index (X4), Consumer Price Index (X5), Number of job vacancies (X6) and Estimated Resident Populations (X7). The data also contained of the Year and Month (Year_Q) column which determines the time-period during which the data was collected. To make the analysis more meaningful, the original variable names were ***transformed*** into new column names. Table 1 shows the column names before and after transformation. After that, the Year_Month column was converted to a date format and split into separate columns. This was done to split the data and then separately examine the unemployment rate between December (1982-2020) (Training Data and March (2021-2023) (Test Data).

Table 1:

Variable names : Before vs After Transformation

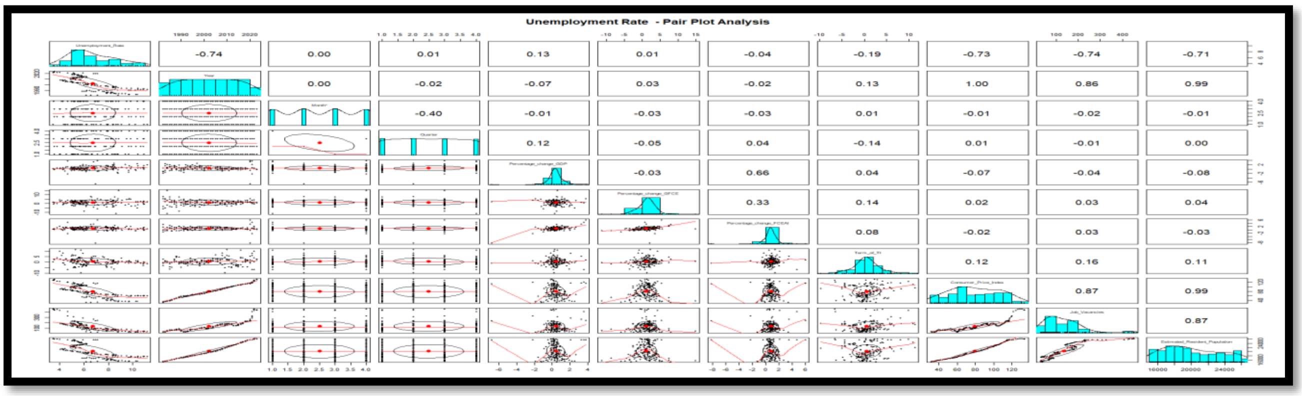
Sr. No	**Original Column Names**	**Transformed Column Names**
1	Year_Q	Year_Month
2	Y : unemployment rate measured in percentage	Unemployment_Rate
3	X1: Percentage change in Gross domestic product	Percentage_change_GDP
4	X2: Percentage change in the Government final consumption expenditure	Percentage_change_GFCE
5	X3: Percentage change in final consumption expenditure of all industry sectors	Percentage_change_FCEAI
6	X4: Term of trade index (percentage)	Term_of_TI
7	X5: Consumer Price Index of all groups (CPI)	Consumer_Price_Index
8	X6: Number of job vacancies measured in thousands	Job_Vacancies
9	X7: Estimated Resident Population in thousands	Estimated_Resident_Population

Note: This table displays the variable names before and after the application of column name transformation

Following this, the dataset was checked for ***missing values***. Missing values were discovered in the variables Percentage_change_GDP(1), Percentage_change_FCEAI(1), Term_of_TI(1), Job_Vacancies(5), and Estimated_Resident_Popoluation(2). The high number of missing Job values is due to the discontinuation of the Job Vacancy Survey (JVS) for five quarters, spanning from August 2008 to August 2009 (Australian Bureau of Statistics, 2009). The JVS, a key data source, ceased during this period. The ABS has noted that attempting to collect this missing data retrospectively is not feasible due to the lack of reliability. Additionally, alternative data sources and modeling techniques have not proven effective in filling the data gap. Consequently, these missing values can be classified as 'Missing not at Random' (MNAR), signifying that they are not randomly distributed but have a specific underlying pattern or reason for being missing. To address this issue, one option is to exclude the Job_Vacancies variable from the remaining portion of the study. Nevertheless, given its significance in determining the unemployment rate and the substantial amount of data available, the Job Vacancy variable has been retained for the entirety of our analysis. Before using the imputation techniques to handle missing values, the distribution (spread) of the original data was checked after which a variety of missing value imputation techniques, namely, Median Imputation, Predictive Mean Matching (PMM), CART (Classification and Regression Trees), Lasso Regression, and Miss Forest (RF- Random Forest based imputation) were used to handle the missing data. The distributions before and after imputation were nearly identical (see Appendix A1, for the data distributions of all imputation techniques,).Miss Forrest imputation, on the other hand, performed best in dealing with missing variables (Kokla et al., 2019), and the imputed values from the Miss Forest imputation method were used in the subsequent analysis. The Miss Forest imputation technique is based on the RF algorithm. It's a non-parametric imputation method, which means it doesn't make explicit assumptions about the function form, but instead tries to estimate the function in the most accurate way possible (Radečić, 2023). Miss Forrest has the following advantages: it can work with both numerical and categorical data, it can handle outliers without the need for feature scaling, it is robust to outliers, and it can handle data non-linearity (Dash, 2022).

Figure 1:

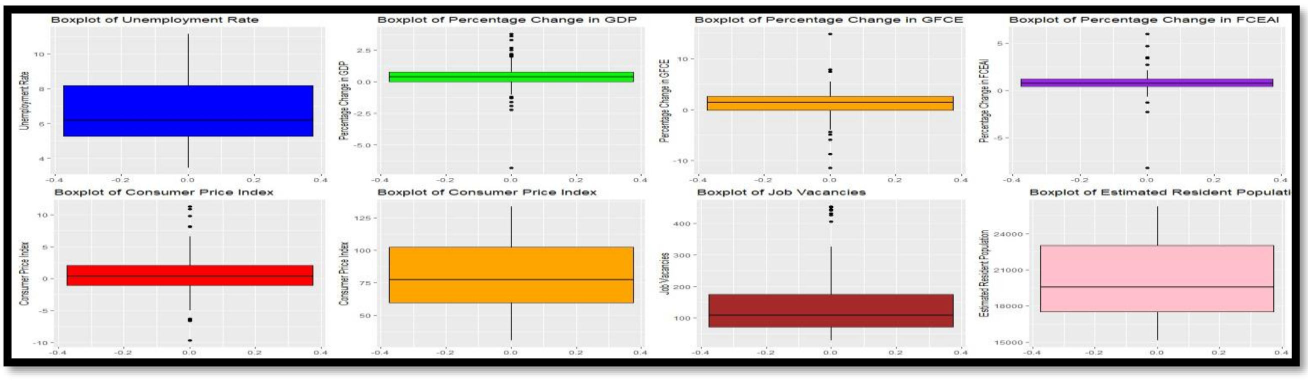
Pair Plot Analysis – All Variables



Note: This plot displays the correlation between the numerical variables

The pair plot in Figure 1 depicts the relationship between the numerical variables. Notably, the Consumer Price Index, Job Vacancies, and Resident Population exhibit a strong negative correlation with the response variable, the Unemployment Rate. Conversely, the Consumer Price Index and Resident Population display an evident positive correlation. Figure 2 includes the box plots for each predictor variable. The percentage change in Gross Domestic Product (GDP) data has a relatively normal distribution. The percentage change in Government Final Consumption Expenditure (GFCE) on the other hand shows positive skewness, indicating potential higher outliers. The percentage change in Final Consumption Expenditure of All Industries (FCEAI) exhibits a similar positive skewness. The Term of Trade Index (TI) and Estimated Resident Population appears to follow a normal distribution. Job vacancies exhibit significant positive skewness, as evidenced by the significant difference between the mean and median, indicating the presence of higher outliers.

Figure 2:
Box Plots – All Variables



Note: Box plots for all variables

IV. Analysis and Investigation of Machine Learning (ML) method

Table 2 presents a summary of the machine learning algorithms and their corresponding hyperparameters employed for predicting the unemployment rate. The methodology for assessing these ML models followed a structured process involving several key steps:

- The initial phase of our analysis involved building base models for each algorithm using the training dataset (Dec 1982 – Dec 2020). These base models served as a starting point for further refinement.
- To enhance the performance of each model, hyperparameter tuning was conducted. Cross-validation was utilized to systematically explore various hyperparameter settings. The aim was to identify the optimal combination of hyperparameters that produced the best predictive results.
- After hyperparameter tuning, the models' performance on the training data was evaluated utilizing the optimal parameters. Subsequently, the models' performance was assessed on a separate test dataset (March 2021 – March 2023). This step is crucial to gauge how well the models generalize to new, unseen data. It helps ensure that the selected algorithms are robust and can provide accurate predictions beyond the training data.

➢ ***Shrinkage Methods***

The investigation began with implementing the (Shrinkage) Regularization models on the training data. *Ridge regression* and *Lasso Regression* models were the shrinkage methods used. The hyperparameters for both Ridge and Lasso Regression is the penalty term Lambda (λ). Ridge Regression and Lasso Regression are both regularization techniques used in linear regression to prevent overfitting and enhance model generalization. Ridge Regression introduces an L2 penalty term to the linear regression objective function, constraining the size of coefficients and mitigating multicollinearity without forcing coefficients to zero. In contrast, Lasso Regression uses an L1 penalty term, which can force some coefficients to exactly zero during model training, effectively conducting feature selection and creating simpler, more interpretable models (James et al., 2021, p. 241). A 10-fold cross-validation over a range of λ values (10^2 to 10^{-2}) was used to determine the optimal lambda values for both Ridge and Lasso Regression (See Appendix C2 for raw code). The minimum lambda value was 0.01. Utilizing this optimal lambda value, the model was built on training data and the performance of the model on both training and test data was analyzed. To assess the performance of the model, the metrics Mean Squared Error and the Residual Mean Squared Error were used. In evaluating both Ridge and Lasso Regression models, Ridge Regression achieved a low training error, with an MSE of 0.8782 and RMSE of 0.9371, indicating a good fit to the training data, but struggled when applied to the test data, with a notably higher MSE of 37.4098 and RMSE of 6.1164, suggesting a potential overfitting issue. On the other hand, the Lasso Regression model exhibited commendable performance on both the training and test datasets, with MSE and RMSE values around 0.8735 and 0.9346, respectively. It is important to mention that Lasso Regression, despite its effective feature selection capabilities, did not drive any coefficients to zero in this analysis, indicating that all predictor variables retained some degree of influence in the model's predictions (See Appendix B for Summary Outputs and Results). On analyzing the coefficients, positive coefficients suggest a positive relationship with the target variable, while negative coefficients suggest a negative relationship.

Ridge and Lasso regression produced nearly identical outcomes, revealing consistent trends in the relationships between predictor variables and the unemployment rate. Specifically, percentage changes in GDP, GFCE, and Estimated Resident Population exhibited positive correlations, while the remaining variables displayed negative associations This implies that a one-unit increase in GDP, GFCE, or Estimated Resident Population corresponds to an increase in the unemployment rate of approximately 0.39, 0.05, and 0.0007 units, respectively.

➢ ***Tree- Based Models***

Tree-based methods are simple and useful for interpretation and therefore is the go-to ML model for industry related problems. Building a training model that can predict value of the target variable by extracting simple decision rules from the data is the main goal when using a decision tree (Chauhan, 2022). The following tree-based models were utilized to predict the Australian unemployment rate (Appendix B for summary outputs and results) :

-*CART* : CART is applicable to tasks involving regression as well as classification. In order to minimise the variance of the target variable within each subset, it recursively divides the dataset into subsets based on the feature that offers the best separation. The mean of the target variable in the leaf nodes is usually the final prediction for regression.

-*Bagging* : Bagging (Bootstrap Aggregating) is an ensemble technique that combines multiple decision trees to reduce variance and improve model stability. Bagging works by training multiple instances of the same base model on different subsets of the bootstrapped sampled training data. These models are combined for a regression problem by averaging the generated outcomes. According to James et al. (2013), p. 348, bagging contributes to increased model robustness and a decrease in data variance.

-*Random Forest*: Random Forest Random Forest is an ensemble model that extends bagging by introducing an additional element of randomness in the tree-building process. By building a collection of decorrelated trees, random forests provide an improvement over bagged trees (James et al., 2013, p. 343). In regression, the average of the predictions made by each tree in the forest is usually the final prediction.

-*Boosting*: Boosting is another ensemble method, but it builds trees sequentially in a way that corrects the errors of the previous trees. It fits a series of shallow trees to the residuals of the previous trees, gradually improving the model's predictive power. Stochastic Gradient Boosting (SGBM) is an ensemble method that enhances traditional Gradient Boosting by introducing randomness. It builds a sequence of decision trees, each trained on random subsets of the data and features (Friedman, 1999). This randomness adds robustness and prevents overfitting, making SGBM a powerful and efficient technique for regression and classification tasks, particularly on large datasets, while maintaining high predictive accuracy.

The respective hyperparameters for each tree-model were tuned to obtain the optimal parameters from which the final model was built and its performance evaluated on both training and test data (See Table 2 for Hyperparameters). CART performed the best amongst the tree-based models. However, the major problem with CART is the instability of the trees and the lack of smoothness in prediction of regression trees. This is because the CART algorithm partitions the feature space into a relatively small number of subspaces and only one predicted value is assigned to each subspace (the arithmetic mean). Subsequently the predictions of the regression tree are no longer smooth. Ensemble techniques, such as Bagging, Random Forest, and Boosting, are typically preferred over single decision trees like CART due to their enhanced performance and robustness.

However, it's essential to note that ensembles can be computationally intensive. Among the ensemble methods, Stochastic Gradient Boosting (SGBM) stood out with superior model performance on both the training and test datasets compared to other tree-based models. SGBM constructs a sequence of decision trees, each trained on random data and feature subsets, which adds robustness and prevents overfitting. The key hyperparameters for Boosting, including the number of iterations (n.trees), max tree depth (interaction.depth), learning rate (shrinkage), and the minimum number of samples for splitting (n.minobsinnode), were systematically tuned using 10-fold cross-validation. Interaction depth controlled the maximum tree depth with values of 2, 5, 10, and 15. The number of trees ranged from 1 to 150, while shrinkage was evaluated at 0.001, 0.1, and 0.15, influencing the contribution of each tree to the ensemble. The minimum observations in a node dictated the minimum sample size for initiating a split, with options of 5, 10, or 20. These hyperparameters collectively shaped the boosting model's behavior, offering flexibility to balance model complexity and predictive accuracy.

Model Summary and Variable Importance: The summary of the best model reveals the relative importance of predictor variables concerning the unemployment rate (see Appendix B for model summary statistics and variable importance chart). Notably, "Consumer_Price_Index" emerges as the most influential predictor, with a substantial relative importance of around 49.37%. It is followed by "Job_Vacancies" at approximately 38.65%. "Estimated_Resident_Population" and "Percentage_change_GDP" each contribute to a lesser extent, with around 3.18% and 3.55% relative importance, respectively. "Percentage_change_GFCE" exhibits a similar influence. Meanwhile, "Term_of_TI" plays a moderate role with a relative importance of approximately 1.93%, while "Percentage_change_FCEAI" holds the lowest relative importance among the variables, at roughly 1.32%. These figures illuminate the key contributors to variations in the unemployment rate.

Hyperparameter Tuning and Model Evaluation: Following comprehensive hyperparameter tuning, the optimal configuration was determined: a model composed of 63 trees, with an interaction depth of 10, a learning rate of 0.15, and a minimum number of observations in nodes set at 10. Using these optimal hyperparameters, the final model was constructed and assessed using Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). The training data yielded a low MSE of 0.14284784, indicating accurate predictions compared to actual values, with an RMSE of 0.3779522. However, the model's performance on the test data was less satisfactory, revealing a significantly higher MSE of 2.4602118 and an RMSE of 1.5685062. This divergence suggests potential overfitting to the training data, emphasizing the need for further model refinement to enhance performance on unseen test data.

➢ ***Support Vector Machines***

The Support Vector Machine (SVM) is an extension of the intuitive maximal margin classifier, providing enhanced capabilities through the enlargement of feature space using kernels (James et al., 2021, p. 380). To assess SVM's performance, various kernels including *linear*, *radial*, and *polynomial* were examined. Hyperparameters such as *cost,* and *gamma* were fine-tuned to identify the optimal combination of kernels and their respective hyperparameter settings. For linear and radial kernels, the range of cost and gamma values included 0.001, 0.01, 0.1, 1, 5, and 10. Meanwhile, the polynomial kernel involved adjustments to cost (0.001, 0.01, 0.1, 1, 5), gamma (0.001, 0.01, 0.1, 1), and degree (2, 3, 4).This extensive investigation was carried out in order to identify the best model and assess predictive performance on both the training and test datasets.

Using the optimal hyperparameters derived from the parameter tuning process, the final SVM model was constructed with each kernel. Among these, the SVM model with a polynomial kernel demonstrated the most robust performance. The best hyperparameters identified for the SVM with a polynomial kernel were cost = 1, gamma = 0.1, and degree = 2 (See Appendix B for Model Summary and Appendix C2– for Raw Code).This configuration led to the generation of 137 Support Vectors and achieved the best performance with a value of 3.18572. This value signifies that applying the SVM with a polynomial kernel using cost = 1, gamma = 0.1, and degree = 2 resulted in the lowest cross-validation error rate of 3.18572. The SVM model with a polynomial kernel performed well on both the training and test datasets. The Mean Squared Error (MSE) for the training data was 1.6533952, indicating a moderate prediction error. This value was mirrored by the Root Mean Squared Error (RMSE), which was 1.6533952, denoting the average error in the same units as the target variable. The model performed better on the test data, with an MSE of 0.5671939, indicating a lower prediction error compared to the training data. The RMSE for the test data was 0.7531228, indicating that the model can make reasonably accurate predictions. The lower error metrics on the test dataset indicate that the SVM model with a polynomial kernel generalizes well to unknown test data.

Table 2:

Machine Learning Models : Hyper-parameters and Performance

Sr No	**Model Name**	**Hyperparameters (R programming)**	**Training Performance**	**Test Performance**
1.	Ridge Regression	lambda (λ)	MSE : 0.87822031 RMSE : 0.9371341	MSE : 37.4097622 RMSE : 6.1163520
2.	Lasso Regression	lambda (λ)	MSE : 0.87348606 RMSE : 0.9346048	MSE : 39.2432839 RMSE : 6.2644460
3.	CART	Cp (Cost Complexity Pruning Parameter)	MSE: 0.40822132 RMSE : 0.6389220	MSE :1.9016950 RMSE : 1.3790196
4.	Bagging + Decision Trees	Number of trees (ntrees), Number of features (mtry)	MSE : 0.04688094 RMSE : 0.2165201	MSE : 5.2907663 RMSE : 2.3001666
5.	Random Forest	Number of trees (ntree), Maximum Depth of the tree (max_depth), Number of features (mtry)	MSE : 0.04358691 RMSE : 3.7763147	MSE : 0.2087748 RMSE : 1.9432742
6.	Gradient Boosting	Number of Iterations (n.trees), Max Tree Depth(interaction.depth), Learning Rate (shrinkage), Minimum Number of training samples to be considered for split (n.minobsinnode)	MSE : 0.14284784 RMSE : 0.3779522	MSE : 2.4602118 RMSE : 1.5685062
7.	SVM – linear	cost , gamma	MSE : 0.90881243 RMSE : 0.9533165	MSE : 31.7155557 RMSE : 5.6316566
8.	SVM – radial	cost, gamma	MSE : 0.24769742 RMSE : 0.4976921	MSE : 2.7059839 RMSE : 1.6449875

9.	SVM – polynomial	cost, gamma, degree	MSE : 1.6533952 RMSE : 1.6533952	MSE : 0.5671939 RMSE : 0.7531228

V. Analysis and Investigation of Neural Network (NN) method

Neural networks (NN) are the foundation of deep learning algorithms. They are also known as Artificial Neural Networks (ANNs) and are subset of machine learning. Both their names and structure are derived from the human brain, emulating the communication between biological neurons (IBM, n.d.). There are 4 main components of ANNs:

- ***Neurons***: the context of artificial neural networks, a neuron can be visualized as a fundamental unit that plays a pivotal role in information processing. The neuron is characterized by a node that functions as a recipient for incoming signals transmitted through connections originating from preceding neurons. Once these incoming signals reach the current neuron, they serve as the impetus for an activation function to come into play. This *activation function* essentially determines how the neuron responds to its inputs, thus enabling the neural network to process and transform information.
- ***Connection***: Within the neural network framework, the strength and significance of interactions between neurons are encapsulated by the concept of connections. Each neuron's output is tied to a weight, which essentially functions as a numerical measure quantifying the intensity of its connection to a subsequent neuron. The magnitude of this weight serves as a key indicator of the current neuron's significance within the network concerning the successor neurons.
- ***Propagation of the signal***: The idea of signal propagation captures the dynamic information flow inside the neural network. The outputs from preceding neurons are mathematically aggregated in this process. The combined output serves as the composite input for the network's next neuron. This output is also known as the "output function," which describes its primary function in transferring data between neurons and facilitating the neural network's decision-making processes.
- ***Decision rules***: At the heart of the neural network's operation lies a set of decision rules that leverage optimization-based mathematical principles. These rules guide the estimation of crucial parameters established in the preceding stages of neural processing. These parameters encompass not only the pivotal connection weights but also the thresholds governing the neurons' activation. The estimation process is a critical component and is integral to the formulation of these decision rules. These decision rules are central to empowering the neural network's capacity to learn, adapt, and effectively address the specific problems at hand.

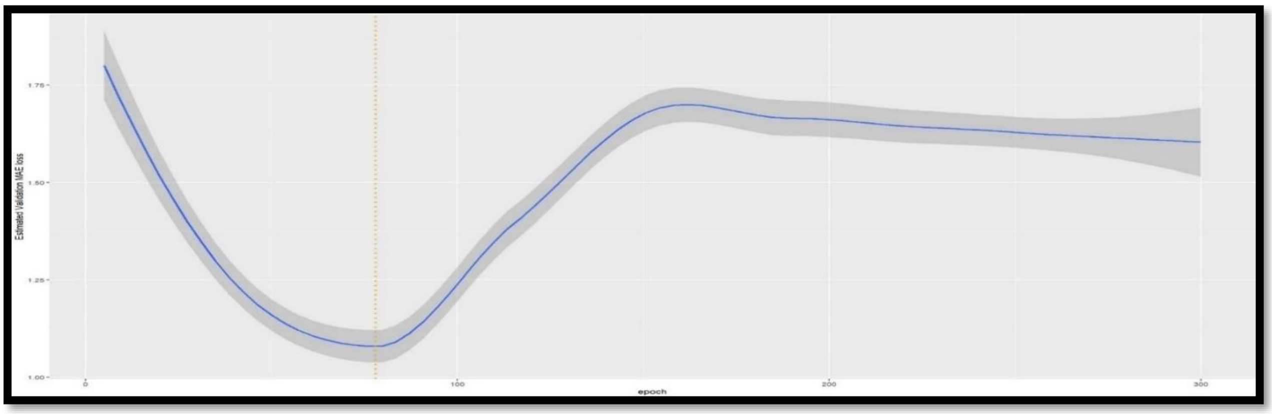
The Neural Network model construction process is quite similar to the ones used in ML model building process. To make sure that each feature variable's mean is centered around 0 with a standard deviation of 1, the data first went through a standardization process. The process began by defining the structure of the neural network model. Given the relatively small dataset size (156 observations with 8 variables in the training set and 10 observations with 8 variables in the test set), a compact network was chosen with two shallow hidden layers, each comprising 64 units. This choice aligns with the general guideline that with limited training data, there's a heightened risk of overfitting. Using a small network aimed to mitigate this risk. The network culminated with a single unit (units = 1) and omitted any activation function, implying a linear layer. This is a typical setup for a scalar regression problem, i.e. when trying to predict a single continuous value, in this case the unemployment rate in

Australia. To assess the NN model's performance, two key metrics were utilized: Mean Squared Error (MSE) and Mean Absolute Error (MAE). The 'relu' activation function was utilized for its non-linearity, and the 'adam' optimizer for its adaptive learning rate, which is often more effective than other optimization methods.

Leveraging the model's training history, the 'fit()' function was applied to the standardized training data. Several parameters were employed, including *'epoch'* set at 300, *'batch_size*' at 8, and *'validation_split'* at 1/3. Epoch indicates the number of passes of the entire training dataset the NN has completed, while the batch size is the number of samples processed before updating the model (Hossain, 2021).The 'validation_split' parameter specifies the portion of the training data held out during each epoch for use as a pseudo-validation dataset, while the remaining fraction is dedicated to updating the network's parameters throughout each epoch. Figure 3 displays the line plot of the bootstrapped validation MAE loss from the model history .It was observed that epochs above 78 resulted in overfit on the training data.

Figure 3

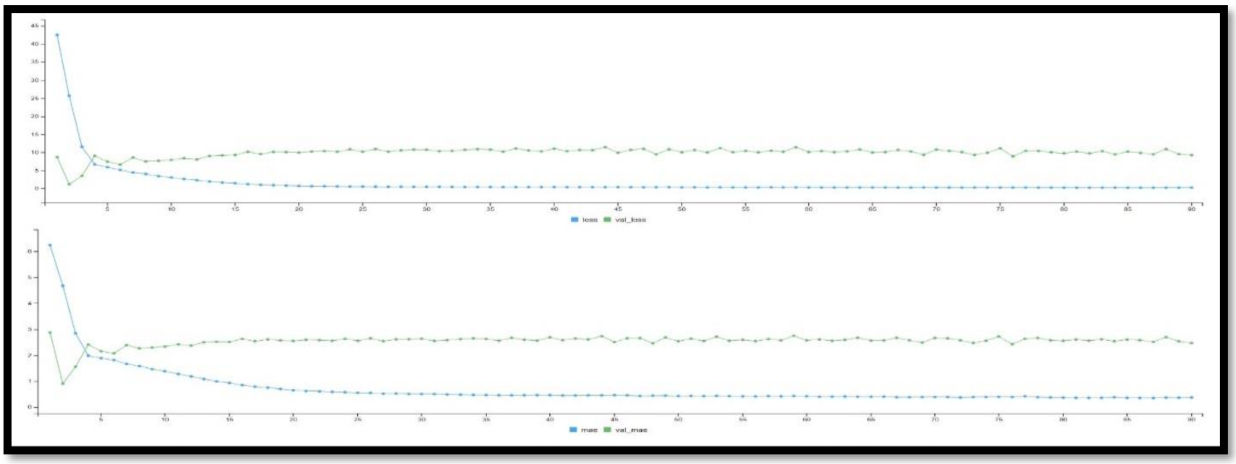
NN Base Model History



Note: This value determines the cut-off (epoch value) after which the NN model overfits the data

Figure 4

NN results on test data using Optimal Hyperparameters



Note: NN results using optimal parameters on the test data

Building on this the epoch value the model was rebuilt again. The model results show that on the training data, the neural network achieved a loss of approximately 0.561 and a Mean Absolute Error (MAE) of around 0.561. However, on the test data, the model's performance decreased significantly, with a loss of approximately 10.342 and a MAE of approximately 2.633. The neural network architecture comprises two layers, with 64 units in the first and second layers, and a single unit in the final layer. The total number of trainable parameters in the model is 4737, occupying roughly 18.50 kilobytes of memory. These results suggest that while the model performs well on the training data, it exhibits a considerable drop in performance when applied to unseen test data, indicating potential overfitting.(See Appendix B for Model Summary and Appendix C2 – for Raw Code).

To enhance the model's performance, hyperparameters such as the number of epochs (ranging from 70 to 100), training units (30, 60, or 90), and batch sizes (10, 30, 50, or 70) were meticulously fine-tuned to pinpoint the most effective combination of settings for predicting the unemployment rate (for detailed results, refer to Appendices for Model Summary and Raw Code). The tuning process was facilitated by a robust 10-fold cross-validation approach. Throughout this optimization process, the activation function remained 'relu,' and the 'adam' optimizer was retained. Following this rigorous procedure, the analysis revealed that the optimal set of hyperparameters consisted of 90 epochs, 90 training units, and a batch size of 10. The model summary generated using these optimal settings disclosed a total of 9001 trainable parameters, emphasizing their adaptability during the training process. In terms of performance, the Mean Squared Error (MSE) was found to be approximately 9.271559, providing insight into the average squared difference between the model's predictions and the actual values. The Mean Absolute Error (MAE) stood at approximately 2.466631, representing the model's average prediction error.

VI. Comparison and Contrasting NN and ML Models

Table 3 displays the comparison of the NN and ML models obtained after extracting the conducting the hyper-parameter tuning using a 10-fold cross validation and extracting the optimal hyper-parameters. The comparison of Machine Learning (ML) models and Neural Network (NN) models provides insights into their performance and computational efficiency in predicting the unemployment rate in Australia.

Among the models, the Stochastic Gradient Boosting exhibited the best predictive accuracy, achieving the lowest Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). This accuracy makes it a strong contender for precise unemployment rate predictions. However, its disadvantage lies in the relatively longer computational time of 3.9 minutes, which may not be suitable for real-time applications or situations where quick predictions are required. Additionally, Stochastic Gradient Boosting offers relatively lower interpretability. The SVM model with a polynomial kernel, while not as accurate as the Stochastic Gradient Boosting, still delivers good predictive performance with a lower MSE and RMSE than the Neural Network. Its key advantage is its quick computational time of 36.8 seconds, making it an attractive option for applications that demand relatively fast results. Moreover, SVM models tend to be more interpretable due to their reliance on support vectors.

The Neural Network model exhibits the highest MSE and RMSE, indicating lower prediction accuracy in this particular context. However, it has the potential to yield superior results with an increased volume of data and the addition of more layers, making it a valuable choice for handling complex data patterns. Neural Networks, despite their longer execution time, offer a unique advantage in capturing intricate relationships within the data, adding a layer of interpretability when model complexity is warranted.

Ultimately, the model choice should be guided by the precise demands of the task at hand. Stochastic Gradient Boosting excels in accuracy, the SVM model strikes a balance between accuracy and computational efficiency, while the Neural Network emerges as a promising option, particularly when more data and layers are available, with the potential to yield improved results. Thus, for predicting the unemployment rate in Australia, Neural Networks stand as the recommended model, especially when tackling complex, multifaceted data patterns, despite their current limitations in this specific context.

Table 3:

ML and NN model comparison

Sr No.	Model Name	MSE	RMSE	Computational Time
1	Stochastic Gradient Boosting	2.4602118	1.5685062	3.900011 mins
2	SVM – polynomial	0.5671939	0.7531228	36.7868 seconds
3	Neural Networks	9.271559	3.044923	12.48 minutes

Note: Comparison of ML models and NN models

VII. Conclusion

Unemployment not only has a negative impact on people's mental and physical health, but it also has serious socioeconomic consequences. As a key economic indicator, the unemployment rate has a significant impact on a country's economic stability and overall health. This report utilizes the "AUS data 2023.xlsx" data and with the application of data transformation , missing value imputation, uses a variety of Machine Learning (ML) models and Artificial Neural Networks (ANNs) to predict the unemployment rate in Australia from 1982 – 2023. The Australian unemployment rate has declined significantly from an average of 10.7% in 1992 to approximately 3.6% in 2023, with some fluctuations over time. This trend highlights the dynamic nature of Australia's labor market, with periods of both economic growth and challenges that have influenced these changes. Nonetheless, the overall direction has been a positive one, suggesting an overall improvement in Australia's unemployment landscape during this period. Estimated Resident Population, Job Vacancies and Consumer Price Index were found to be the most important features in predicting the unemployment rate. Stochastic Gradient Boosting and Support Vector Machines were the best performers amongst ML models with a MSE of 2.4602118 and 0.5671939 and RMSE score of 0.5671939 and 0.7531228, respectively. ANN also provided an excellent performance with RMSE of 3.044923. ANN has the potential to outperform ML models due to its flexibility in handling complex data structures and learning on intricate patterns in data. To further enhance the prediction of the unemployment rate in Australia, more variables like economic indicators and industry demographics can be included by which models can learn on. Additionally, part of the data in this dataset is in aggregated form. The incorporation of real-time data such as job-listings and the impact of external factors like government policies and environmental conditions can significantly aid in predicting the unemployment rate.

References

[1] Australian Bureau of Statistics. (2008, March 13). *6202.0 - Labour Force, Australia, Feb 2008*. https://www.abs.gov.au/AUSSTATS/abs@.nsf/Lookup/6202.0Main+Features1Feb%202008?Ohttps://www.abs.gov.au/AUSSTATS/abs@.nsf/Lookup/6202.0Main+Features1Feb%202008?OpenDocument=penDocument=

[2] Australian Bureau of Statistics. (2009, November). *6354.0.55.001 - Information paper: Reinstatement of job vacancies survey, Nov 2009*. Retrieved October 24, 2023, from https://www.abs.gov.au/ausstats/abs@.nsf/mf/6354.0.55.001

[3]Australian Bureau of Statistics. (2009). *6354.0.55.001 - Information paper: Reinstatement of job vacancies survey, Nov 2009*. https://www.abs.gov.au/ausstats/abs@.nsf/mf/6354.0.55.001

[4] Australian Bureau of Statistics. (2020, May 26). *6102.0.55.001 - Labour statistics: Concepts, sources and methods, Feb 2018*. Retrieved October 24, 2023, from https://www.abs.gov.au/ausstats/abs@.nsf/Lookup/by%20Subject/6102.0.55.001~Feb%202018~Main%20Features~Job%20Vacancies%20Survey~35

[5] Australian Bureau of Statistics. (2023, September). *Labour force, Australia, August 2023*. https://www.abs.gov.au/statistics/labour/employment-and-unemployment/labour-force-australia/latest-release

[6] Australian Institute of Health and Welfare. (2023, December 7). *Employment and unemployment*. https://www.aihw.gov.au/reports/australias-welfare/employment-unemployment

[7] Bishop, J., Kent, C., Plumb, M., & Rayner, V. (2013, March). *The Resources Boom and the Australian Economy: A Sectoral Analysis*. Reserve Bank of Australia. https://www.rba.gov.au/publications/bulletin/2013/mar/pdf/bu-0313-5.pdf

[8] Dash, S. K. (2022, September 22). Handling missing values with random forest. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2022/05/handling-missing-values-with-random-forest/

[9] Fahrer, J., & Heath, A. (1992). *THE EVOLUTION OF EMPLOYMENT AND UNEMPLOYMENT IN AUSTRALIA* [Master's thesis]. https://www.rba.gov.au/publications/rdp/1992/pdf/rdp9215.pdf

[10] Friedman, J. H. (1999, March 26). *Stochastic Gradient Boosting*. https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/. https://jerryfriedman.su.domains/ftp/stobst.pdf

[11] Hong, S., & Lynn, H. S. (2020, July 25). *Accuracy of random-forest-based imputation of missing data in the presence of non-normality, non-linearity, and interaction*. BioMed Central. https://doi.org/10.1186/s12874-020-01080-1

[12] Hossain, E. (2021, April 8). *Difference between the batch size and epoch in neural network*. Medium. https://medium.com/mlearning-ai/difference-between-the-batch-size-and-epoch-in-neural-network-2d2cb2a16734

[13] IBM. (n.d.). *What are neural networks?* IBM - United States. https://www.ibm.com/topics/neural-networks#:~:text=Neural%20networks%2C%20also%20known%20as,neurons%20signal%20to%20one%20another

[14] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning: With applications in R* (2nd ed.). Springer Science & Business Media.

[15] Kokla, M., Virtanen, J., Kolehmainen, M., Paananen, J., & Hanhineva, K. (2019, October 11). *Random forest-based imputation outperforms other methods for imputing LC-MS metabolomics data: A comparative study*. BioMed Central. https://doi.org/10.1186/s12859-019-3110-0

[16] Radečić, D. (2023, January 10). *Imputation in R: Top 3 ways for imputing missing data*. R Shiny | Enterprise R Shiny Dashboards | R Consulting. https://appsilon.com/imputation-in-r/

[17] Reserve Bank of Australia. (2023). *Unemployment: Its measurement and types*. https://www.rba.gov.au/education/resources/explainers/unemployment-its-measurement-and-types.html

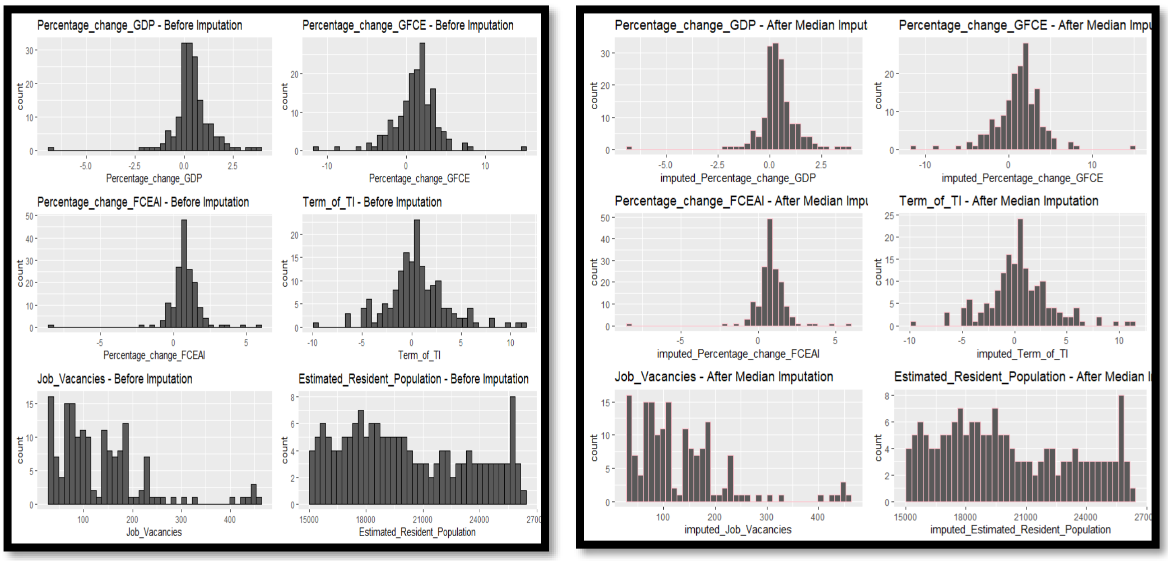
APPENDIX

A. ***EDA(Data Wrangling)***

➢ A1. Missing Value Imputation Techniques

Figure 1:

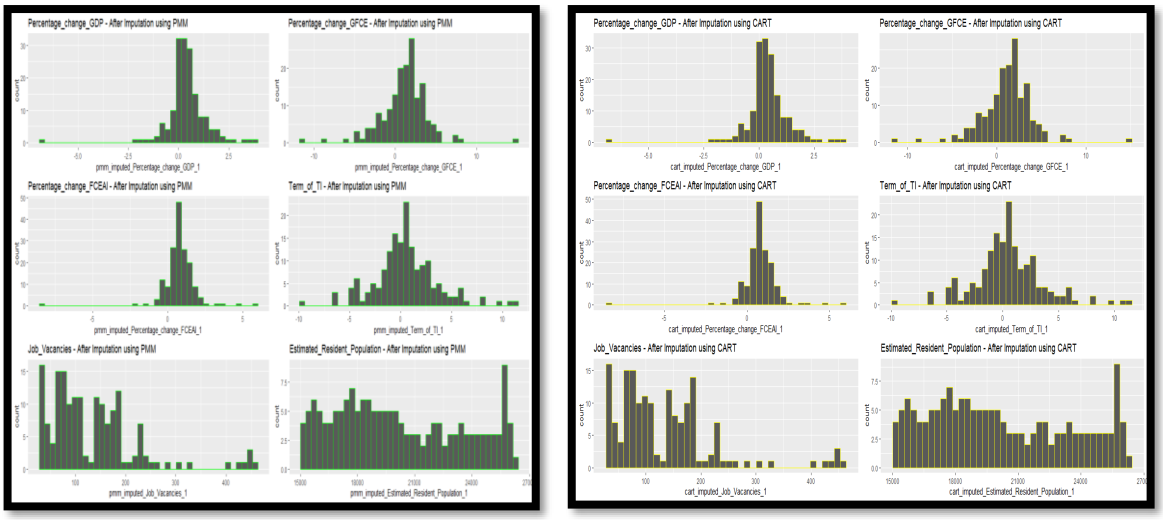
Before Imputation and After Median Imputation results



Note: Missing Value Imputation results Before Imputation and after applying Median Imputation

Figure 2:

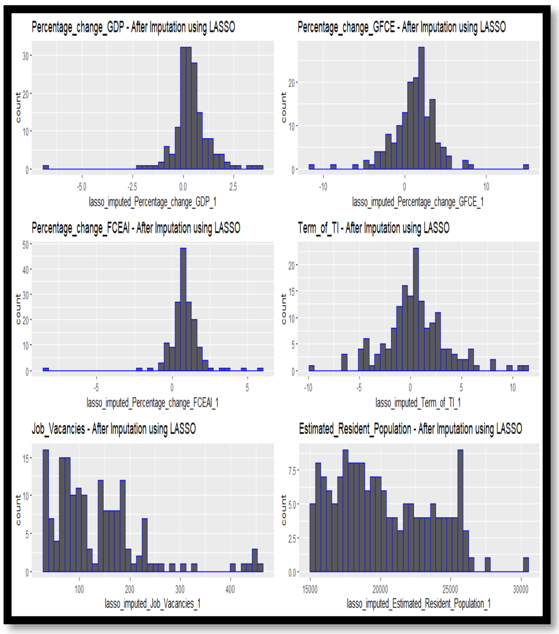
PMM and CART Imputation

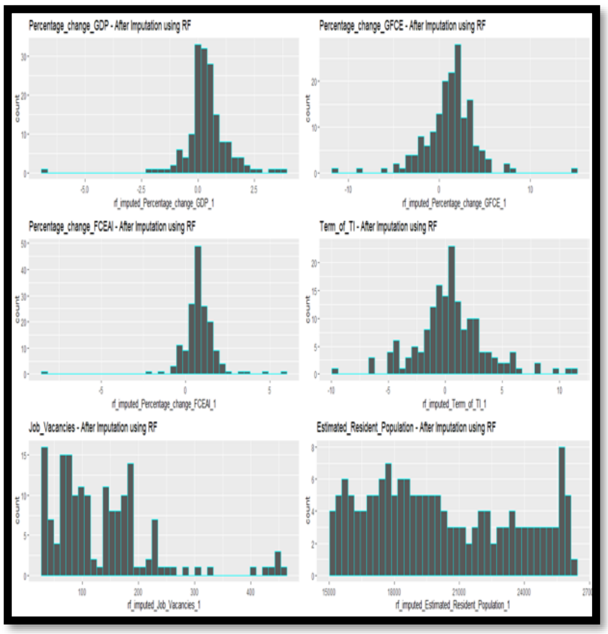


Note: Missing Value Imputation results after applying PMM and CART Imputation

Figure 3:

Lasso Regression and Miss Forest Imputation



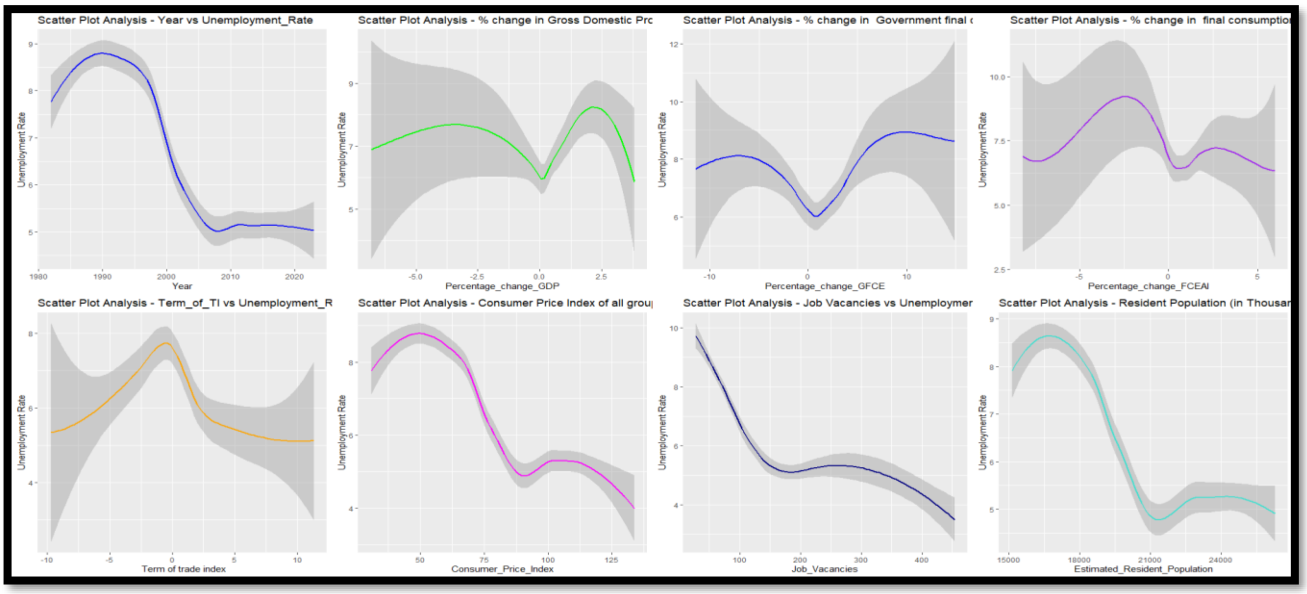


Note: Missing Value Imputation results after applying Lasso Regression and Miss Forest Imputation

➢ A2. <u>Exploratory Data Analysis</u>

Figure 1

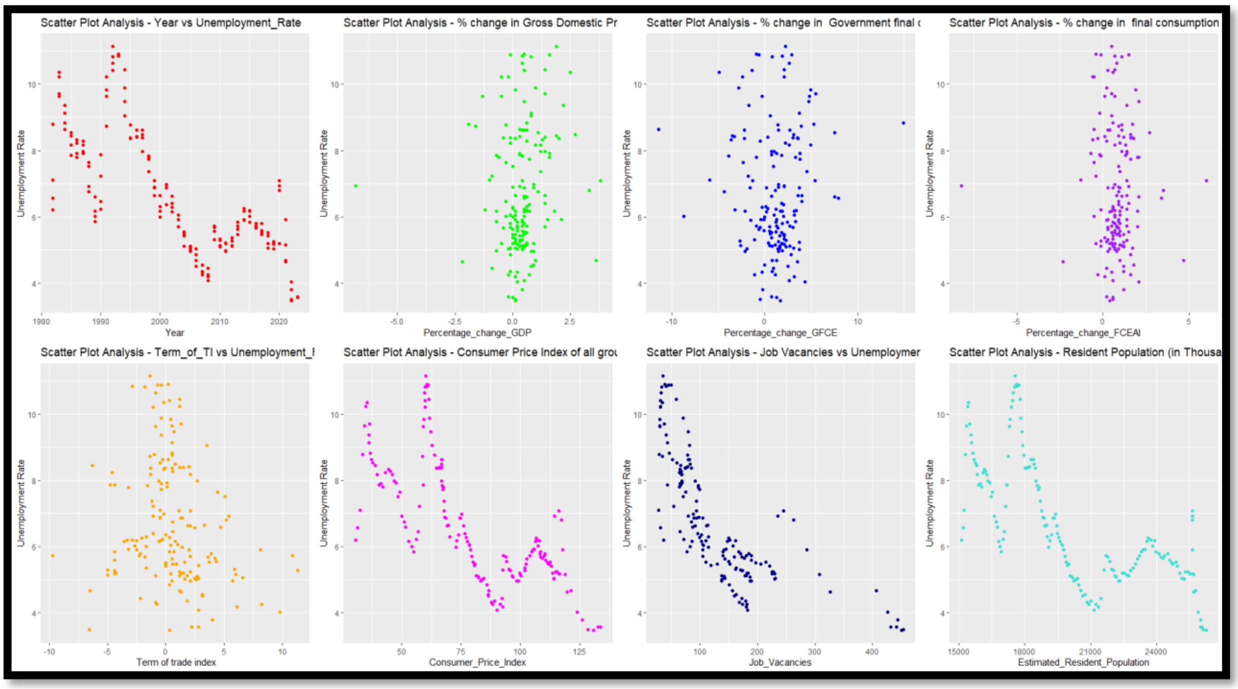
Line Plots All Variables



Note: Smooth Line plots for all variables

Figure 2

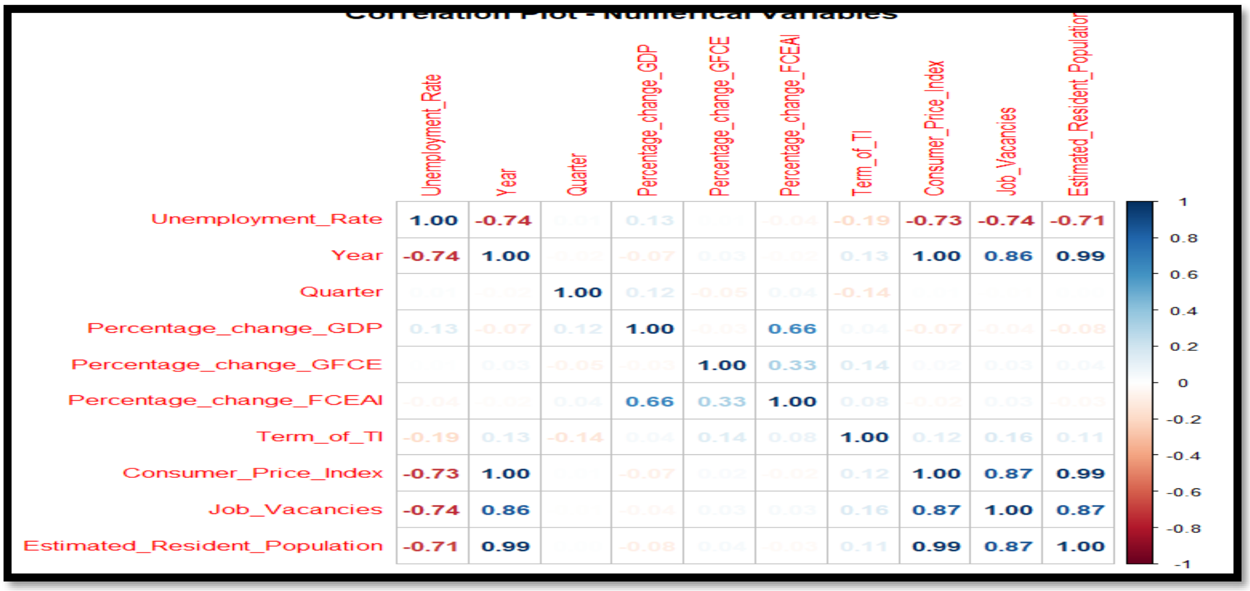
Scatter Plots All Variables



Note: Scatter Plots results for all variables

Figure 3

Correlation Plot

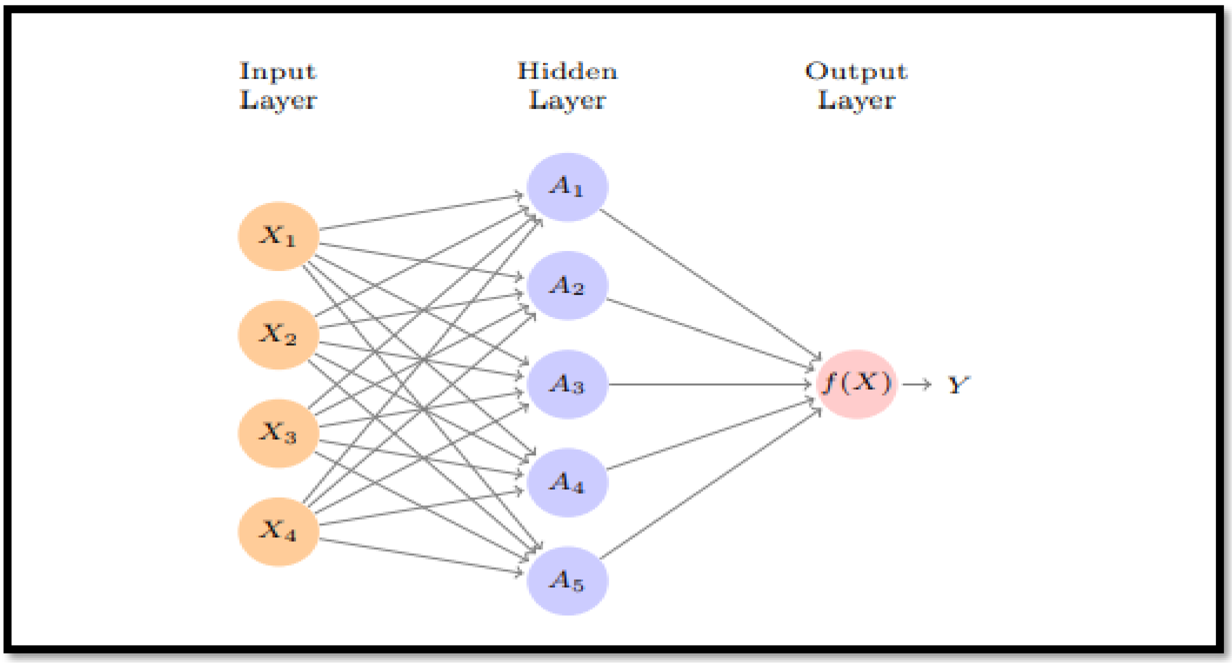


	Unemployment_Rate	Year	Quarter	Percentage_change_GDP	Percentage_change_GFCE	Percentage_change_FCEAI	Term_of_TI	Consumer_Price_Index	Job_Vacancies	Estimated_Resident_Population
Unemployment_Rate	1.00	-0.74	0.01	0.13	0.01	-0.04	-0.19	-0.73	-0.74	-0.71
Year	-0.74	1.00	-0.02	-0.07	0.03	-0.02	0.13	1.00	0.86	0.99
Quarter	0.01	-0.02	1.00	0.12	-0.05	0.04	-0.14			
Percentage_change_GDP	0.13	-0.07	0.12	1.00	-0.03	0.66	0.04	-0.07	-0.04	-0.08
Percentage_change_GFCE	0.01	0.03	-0.05	-0.03	1.00	0.33	0.14	0.02	0.03	0.04
Percentage_change_FCEAI	-0.04	-0.02	0.04	0.66	0.33	1.00	0.08	-0.02	0.03	-0.03
Term_of_TI	-0.19	0.13	-0.14	0.04	0.14	0.08	1.00	0.12	0.16	0.11
Consumer_Price_Index	-0.73	1.00		-0.07	0.02	-0.02	0.12	1.00	0.87	0.99
Job_Vacancies	-0.74	0.86		-0.04	0.03	0.03	0.16	0.87	1.00	0.87
Estimated_Resident_Population	-0.71	0.99		-0.08	0.04	-0.03	0.11	0.99	0.87	1.00

Note: Correlation plot for all variables

Figure 4

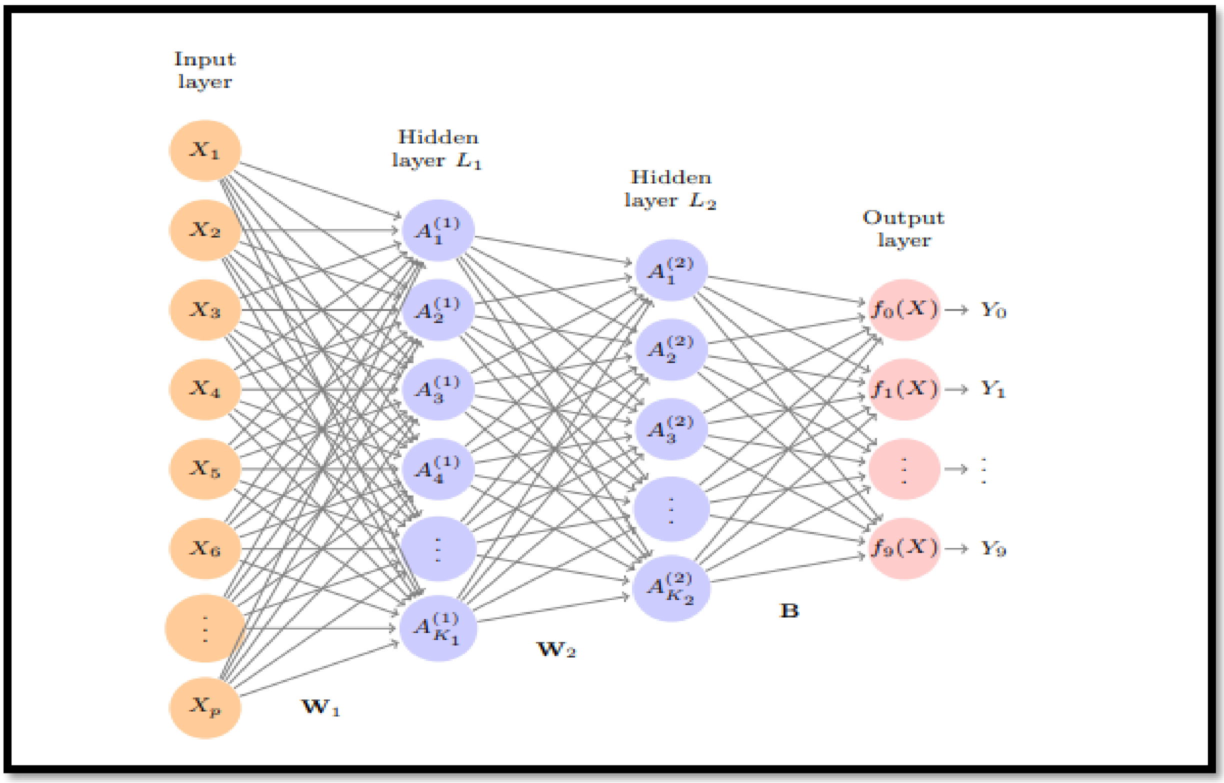
Neural Networks single-hidden layer workflow



Note: NN with a single hidden layer

Figure 4

Neural Networks multiple-hidden layer workflow



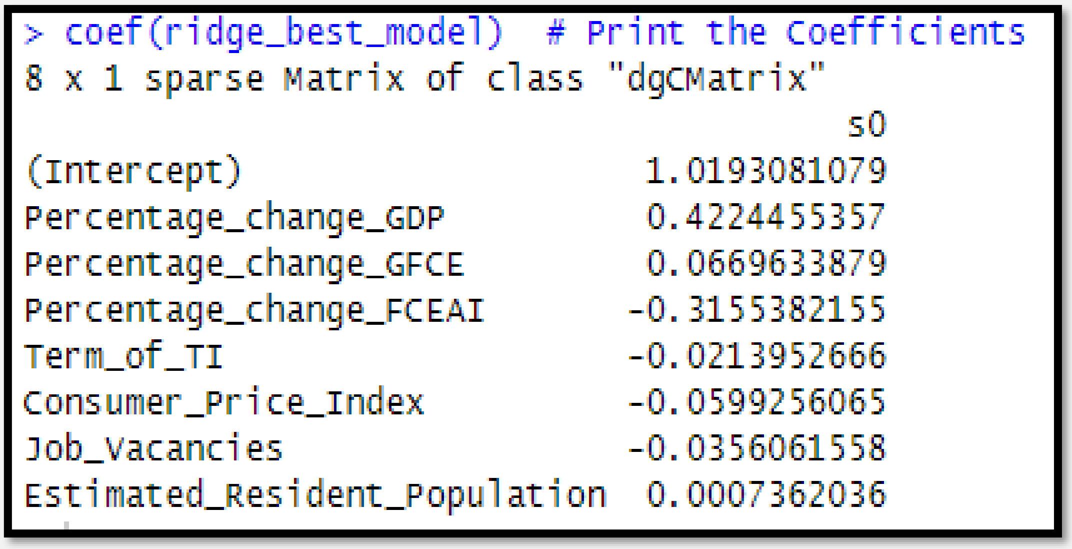
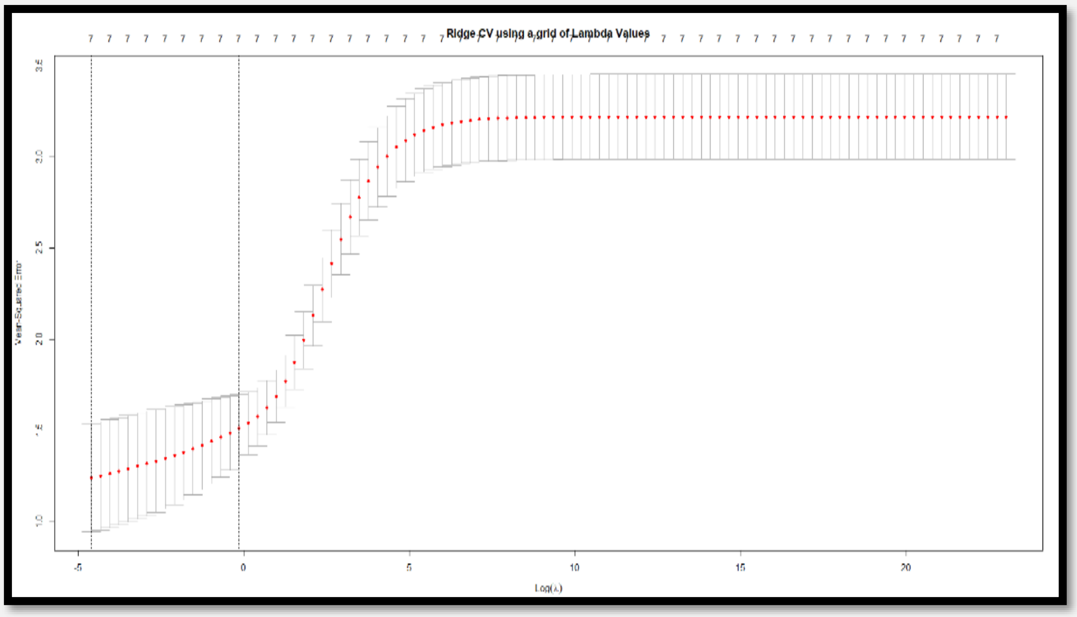
Note: NN with a multiple hidden layers

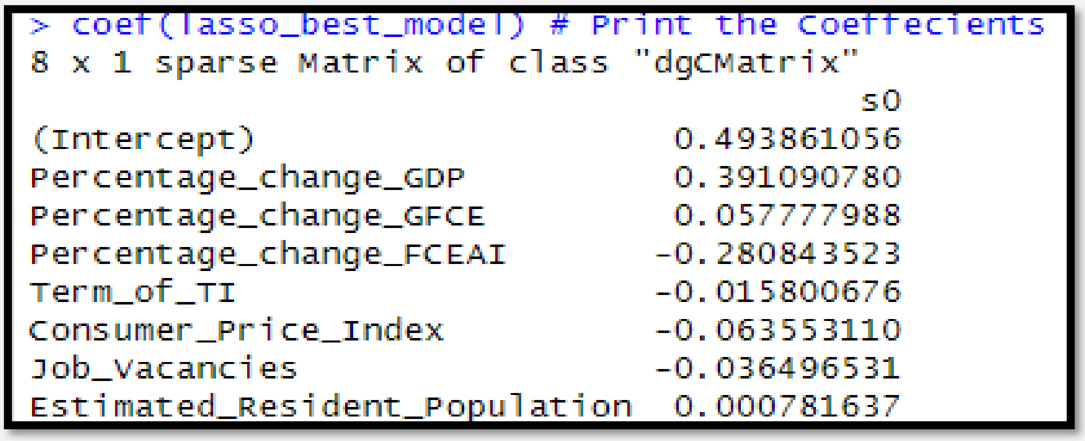
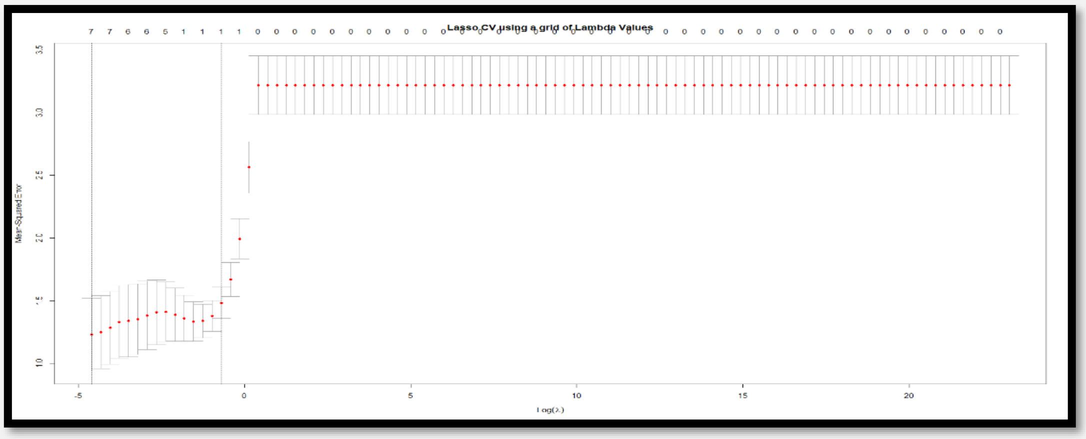
Appendix B: **ML Models Summary Outputs and Results**

Table 1:

Model Summary and Results

Model	**Summary / Results**

Ridge Regression	**Figure 1**
	Coeffecients of Ridge Regression
	
	`> coef(ridge_best_model)   # Print the Coefficients`
	`8 x 1 sparse Matrix of class "dgCMatrix"`
	`                                       s0`
	`(Intercept)                  1.0193081079`
	`Percentage_change_GDP        0.4224455357`
	`Percentage_change_GFCE       0.0669633879`
	`Percentage_change_FCEAI     -0.3155382155`
	`Term_of_TI                  -0.0213952666`
	`Consumer_Price_Index        -0.0599256065`
	`Job_Vacancies               -0.0356061558`
	`Estimated_Resident_Population  0.0007362036`
	
	Figure 2 :
	CV plot for Ridge Regression displaying the best lambda value
	

Lasso Regression	**Figure 3** *Coeffecients of Lasso Regression*  **Figure 4** *CV plot for Lasso Regression displaying the best lambda value* 
CART	**Figure 5**

```
> coef(lasso_best_model) # Print the Coeffecients
8 x 1 sparse Matrix of class "dgCMatrix"
                                        s0
(Intercept)                    0.493861056
Percentage_change_GDP          0.391090780
Percentage_change_GFCE         0.057777988
Percentage_change_FCEAI       -0.280843523
Term_of_TI                    -0.015800676
Consumer_Price_Index          -0.063553110
Job_Vacancies                 -0.036496531
Estimated_Resident_Population  0.000781637
```

Cost Complexity Parameter

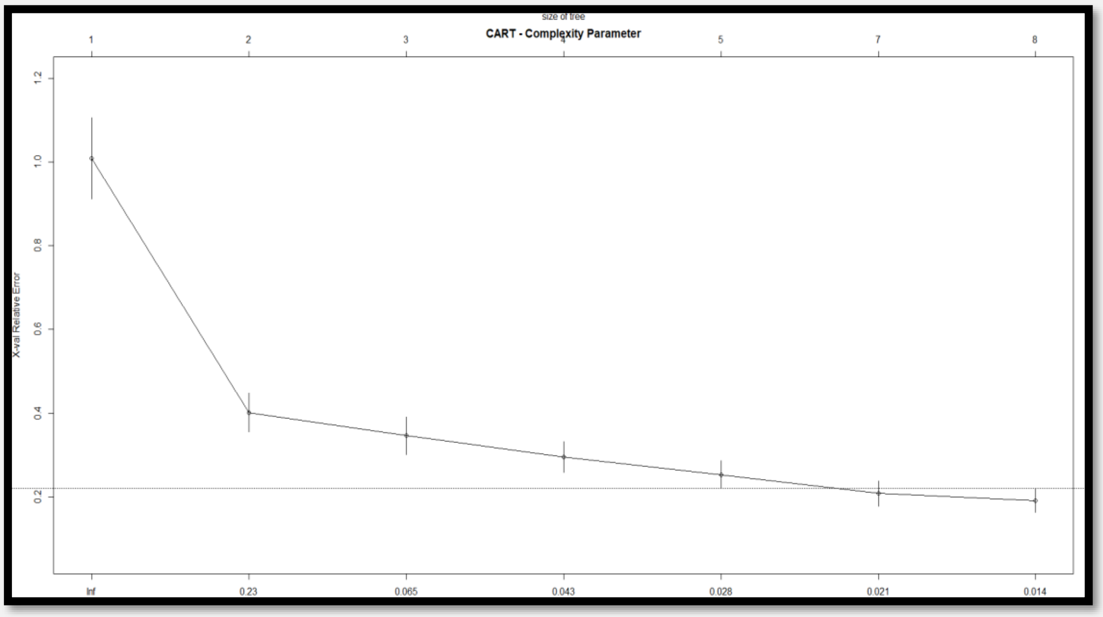


Figure 6

Pruned Tree

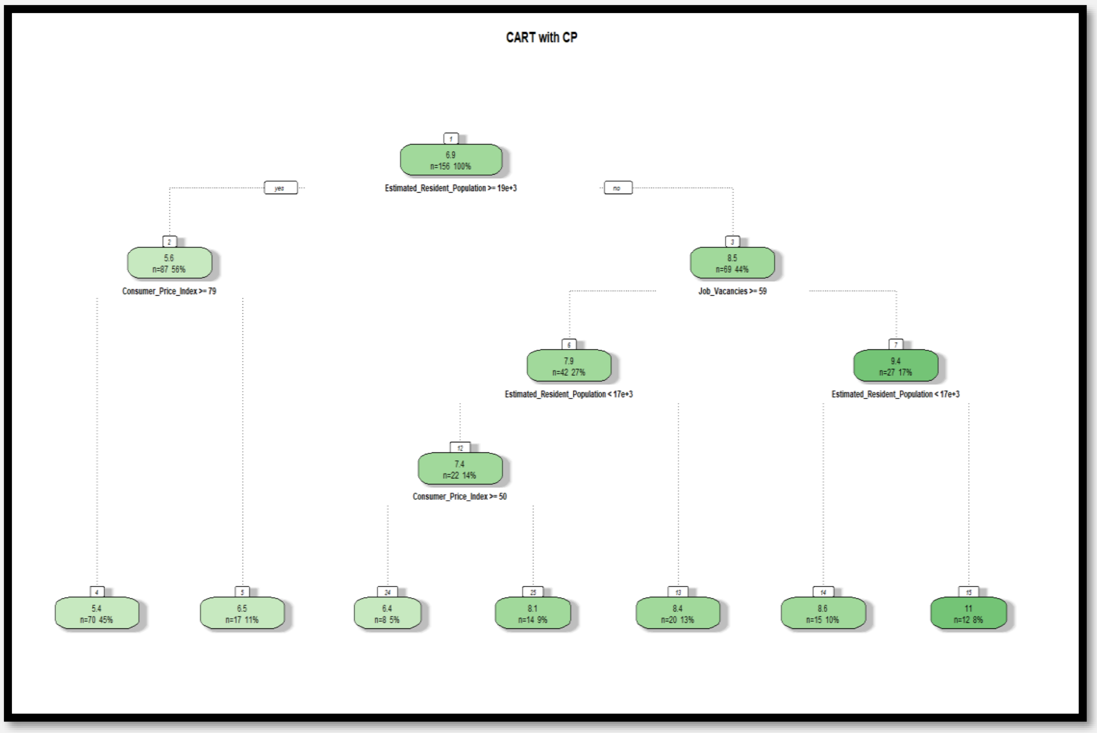
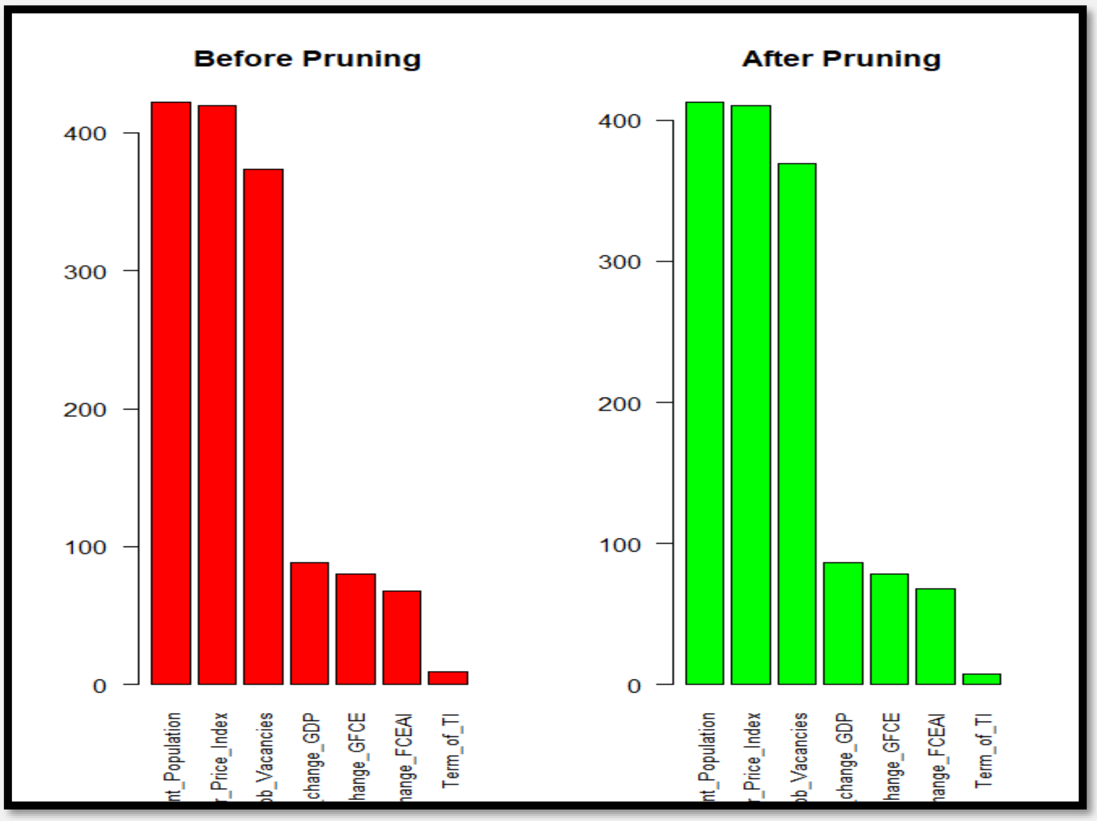
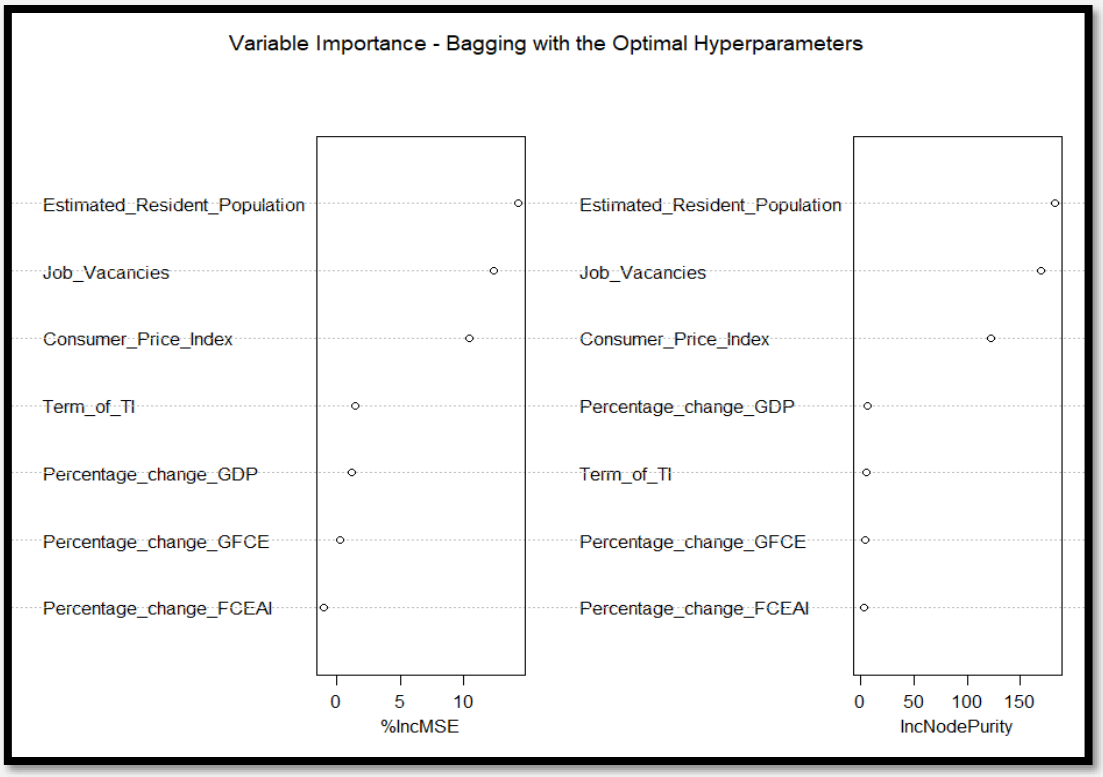
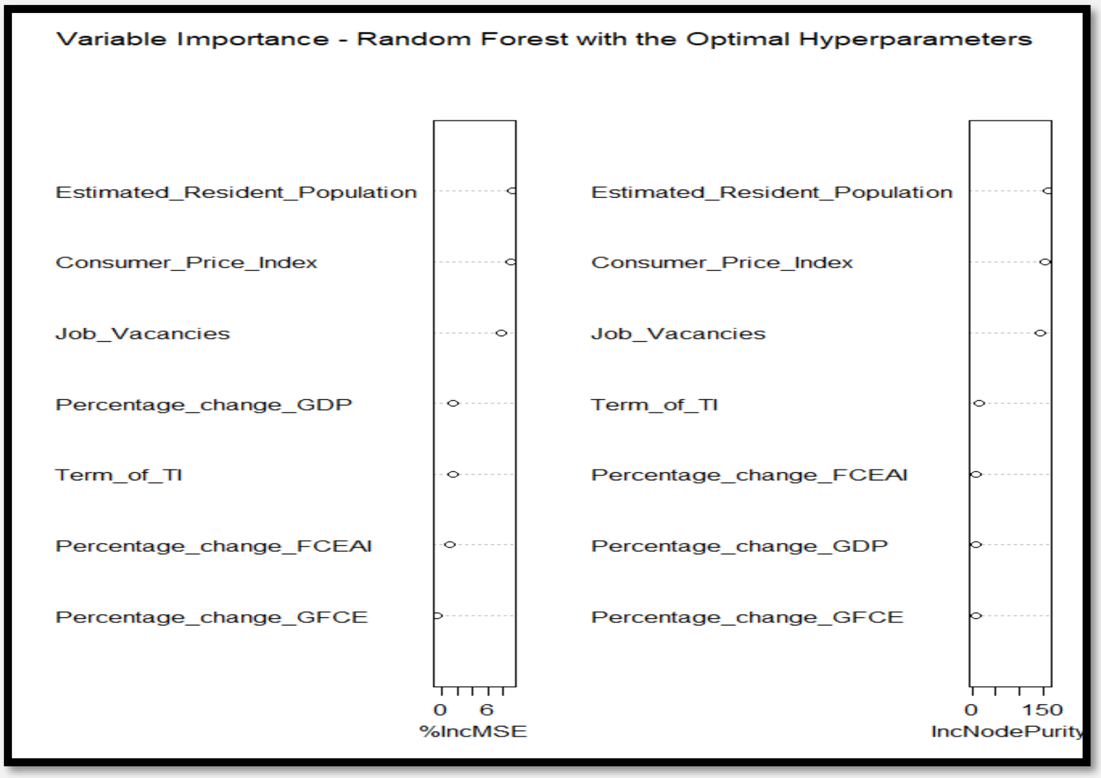
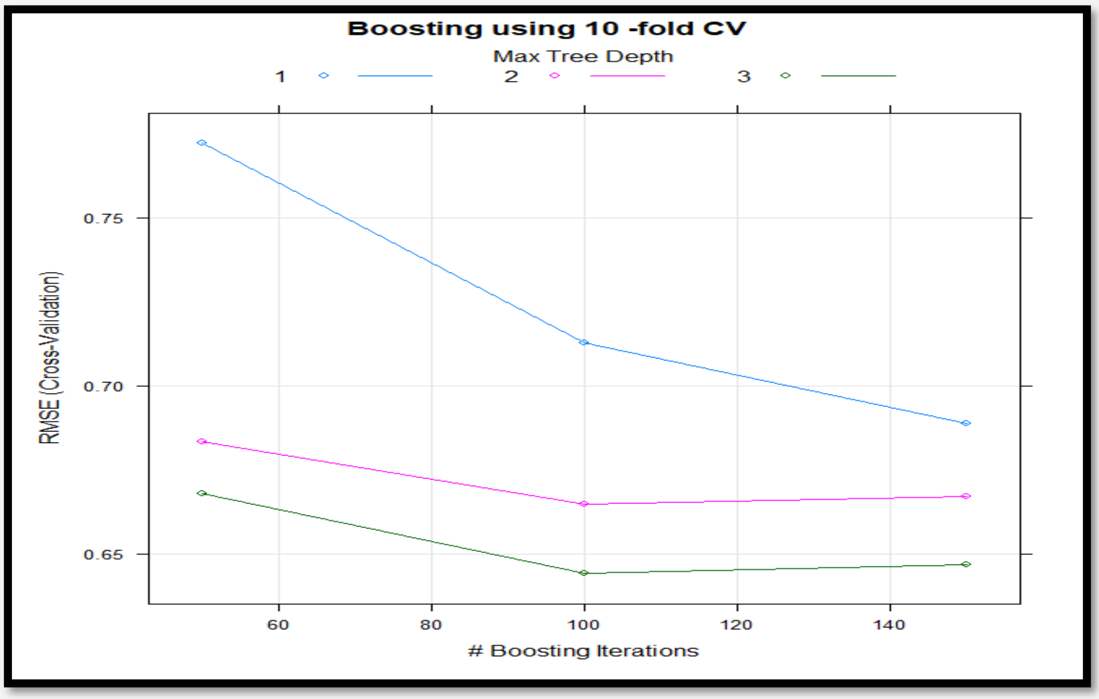


Figure 7

Variable Importance – Before vs After Pruning

	
Bagging + Decision Trees	**Figure 8** *Variable Importance – Bagging with Optimal Hyperparameters* 

Random Forrest	**Figure 9** *Variable Importance – Random Forest with Optimal Hyperparameters* 
Boosting	**Figure 10** *Boosting using 10-Fold CV*  **Figure 11** *Boosting: Hyperparameter Importance using 10 – fold CV*

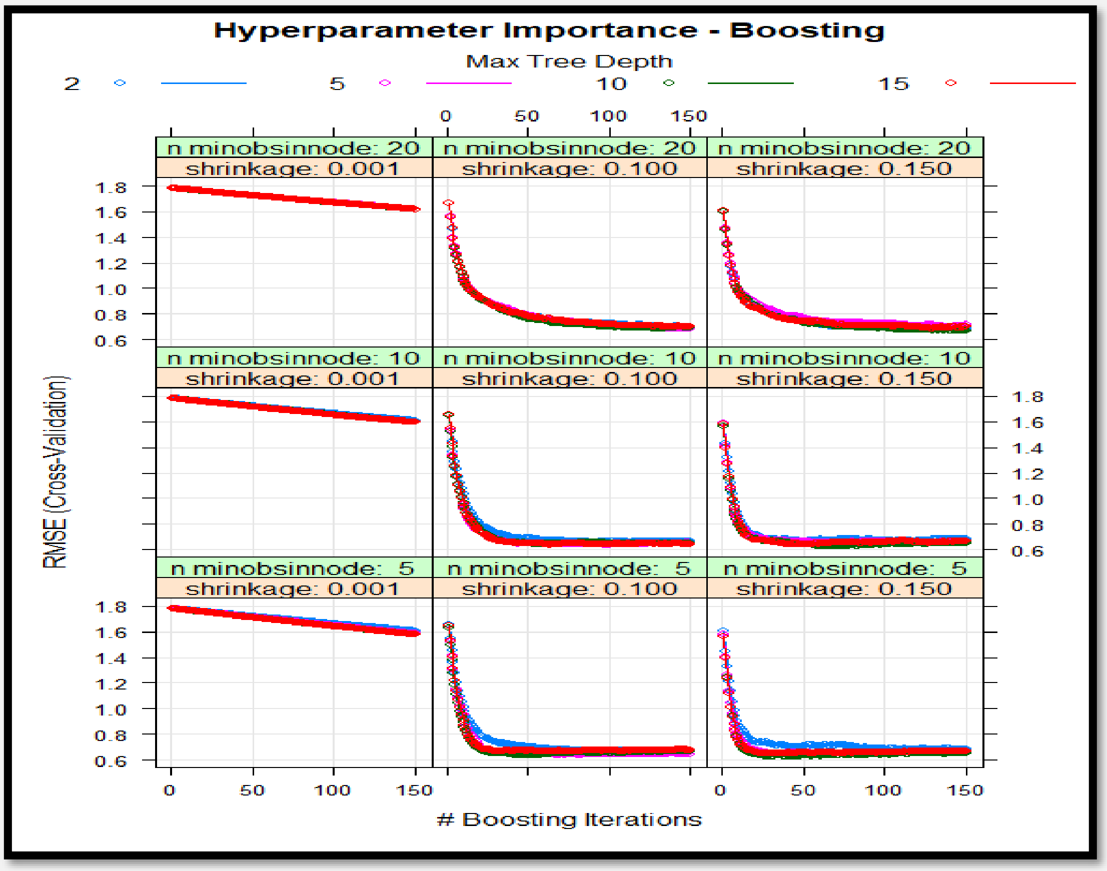
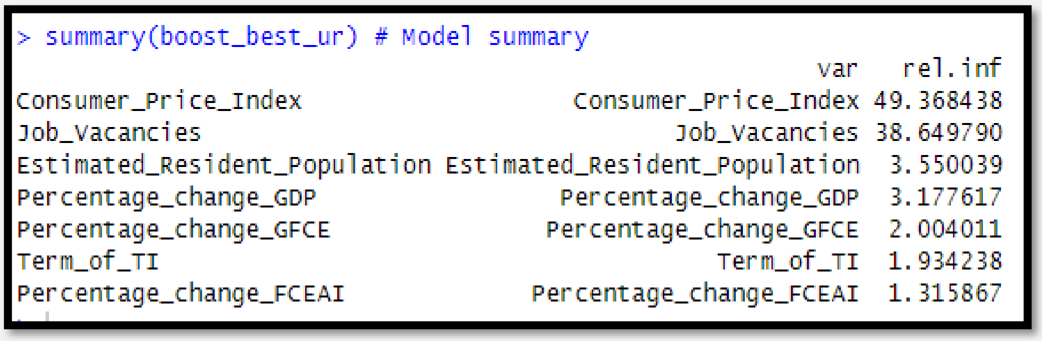
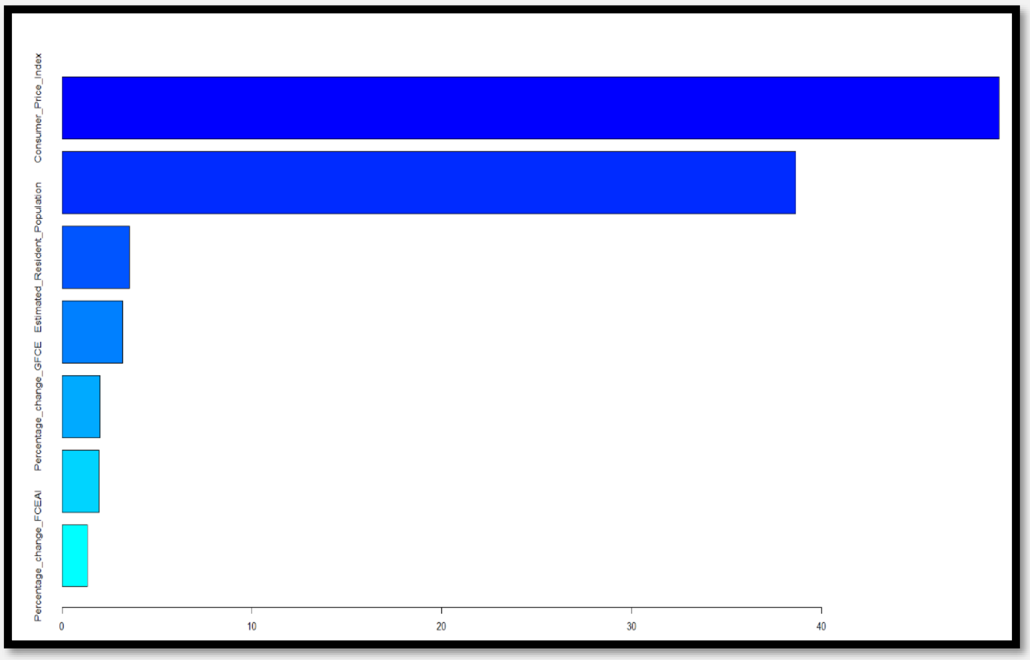


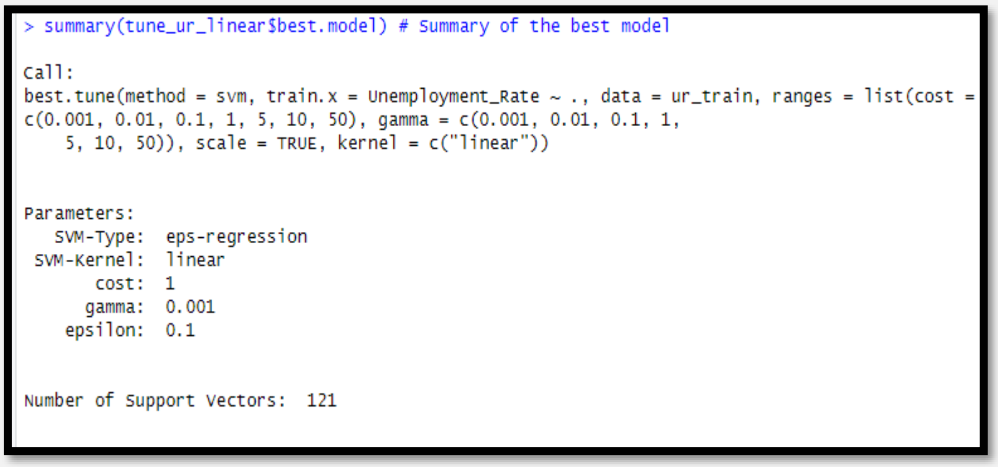
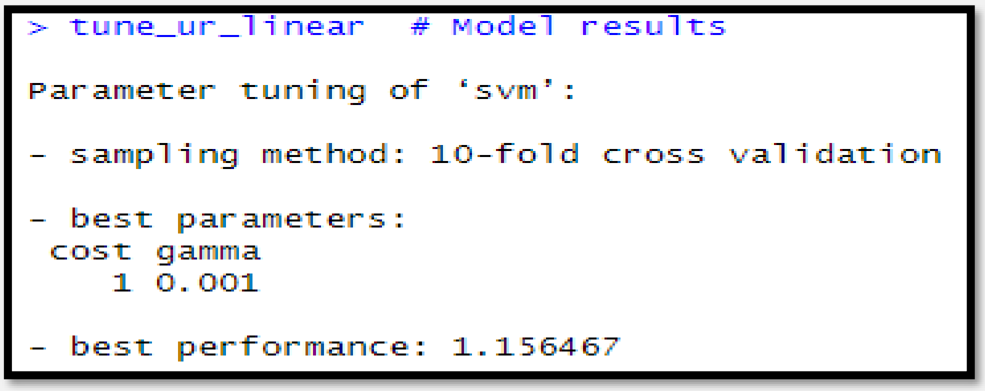
Figure 11
Model Summary– Boosting with Optimal Hyperparameters

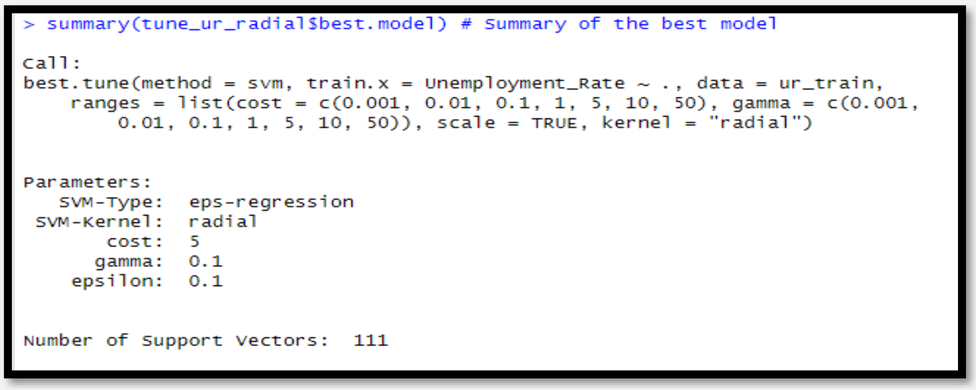
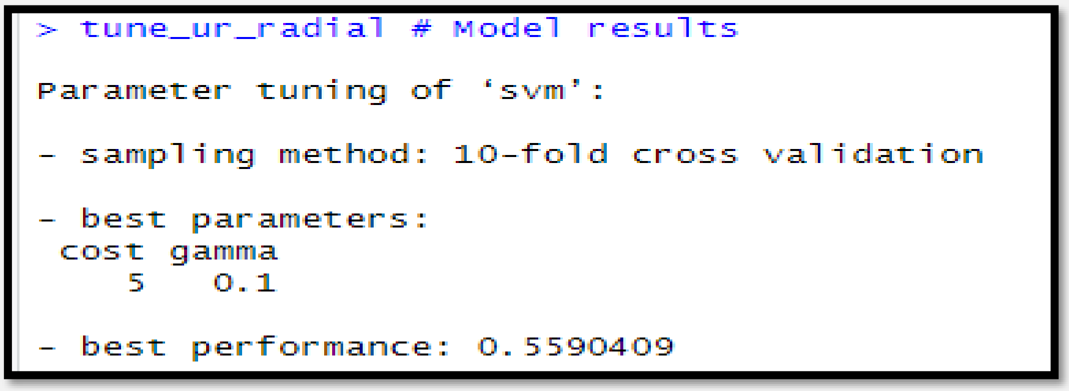


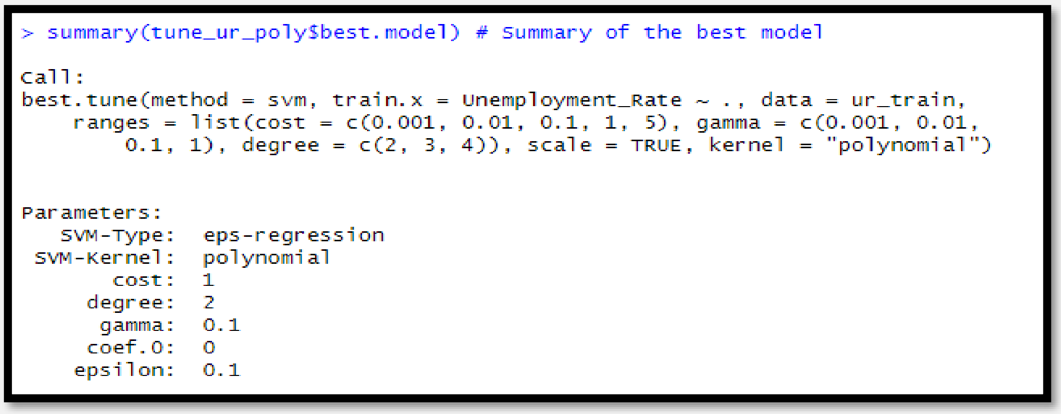
```
> summary(boost_best_ur) # Model summary
                                                      var    rel.inf
Consumer_Price_Index                 Consumer_Price_Index 49.368438
Job_Vacancies                               Job_Vacancies 38.649790
Estimated_Resident_Population Estimated_Resident_Population  3.550039
Percentage_change_GDP               Percentage_change_GDP  3.177617
Percentage_change_GFCE             Percentage_change_GFCE  2.004011
Term_of_TI                                     Term_of_TI  1.934238
Percentage_change_FCEAI           Percentage_change_FCEAI  1.315867
```

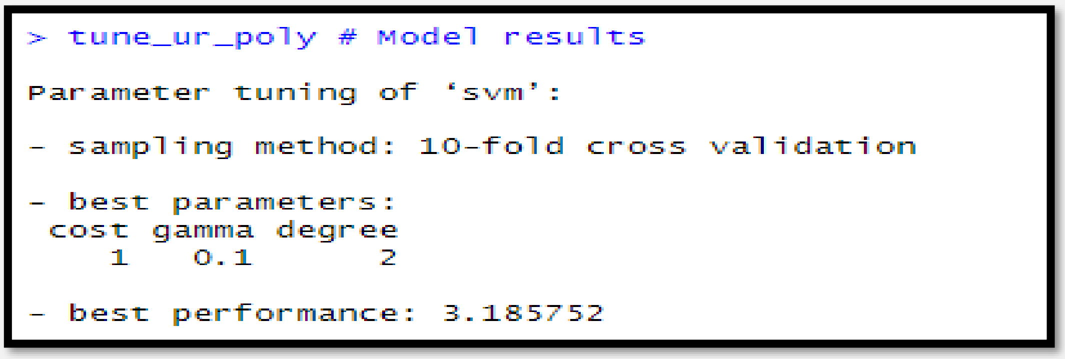
Figure 12
Variable Importance – Boosting with Optimal Hyperparameters

SVM – linear	**Figure 13**
	SVM -linear kernel model summary
	
	`> summary(tune_ur_linear$best.model) # Summary of the best model`
	
	`Call:`
	`best.tune(method = svm, train.x = Unemployment_Rate ~ ., data = ur_train, ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 50), gamma = c(0.001, 0.01, 0.1, 1, 5, 10, 50)), scale = TRUE, kernel = c("linear"))`
	
	`Parameters:`
	`   SVM-Type:  eps-regression`
	` SVM-Kernel:  linear`
	`       cost:  1`
	`      gamma:  0.001`
	`    epsilon:  0.1`
	
	`Number of Support Vectors:  121`
	
	Figure 14
	SVM linear kernel model results
	
	`> tune_ur_linear  # Model results`
	
	`Parameter tuning of 'svm':`
	
	`- sampling method: 10-fold cross validation`
	
	`- best parameters:`
	` cost gamma`
	`    1 0.001`
	
	`- best performance: 1.156467`

SVM – radial	**Figure 15**
	SVM -radial kernel model summary
	
	```
	> summary(tune_ur_radial$best.model) # Summary of the best model

	Call:
	best.tune(method = svm, train.x = Unemployment_Rate ~ ., data = ur_train,
	    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 50), gamma = c(0.001,
	        0.01, 0.1, 1, 5, 10, 50)), scale = TRUE, kernel = "radial")


	Parameters:
	   SVM-Type:  eps-regression
	 SVM-Kernel:  radial
	       cost:  5
	      gamma:  0.1
	    epsilon:  0.1


	Number of Support Vectors:  111
	```
	Figure 16
	SVM radial kernel model results
	
	```
	> tune_ur_radial # Model results

	Parameter tuning of ‘svm’:

	- sampling method: 10-fold cross validation

	- best parameters:
	 cost gamma
	    5   0.1

	- best performance: 0.5590409
	```

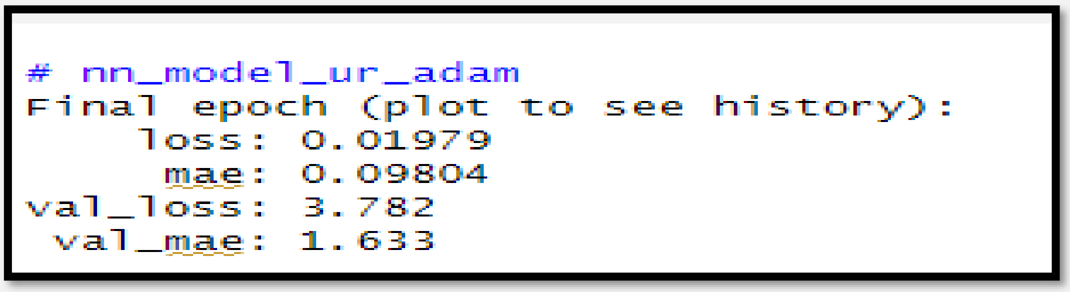
SVM - polynomial	**Figure 17**
	SVM - polynomial kernel model summary
	
	```
	> summary(tune_ur_poly$best.model) # Summary of the best model

	Call:
	best.tune(method = svm, train.x = Unemployment_Rate ~ ., data = ur_train,
	    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5), gamma = c(0.001, 0.01,
	        0.1, 1), degree = c(2, 3, 4)), scale = TRUE, kernel = "polynomial")


	Parameters:
	   SVM-Type:  eps-regression
	 SVM-Kernel:  polynomial
	       cost:  1
	     degree:  2
	      gamma:  0.1
	     coef.0:  0
	    epsilon:  0.1
	```
	Figure 18
	SVM polynomial kernel model results



```
> tune_ur_poly # Model results

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma degree
    1   0.1      2

- best performance: 3.185752
```

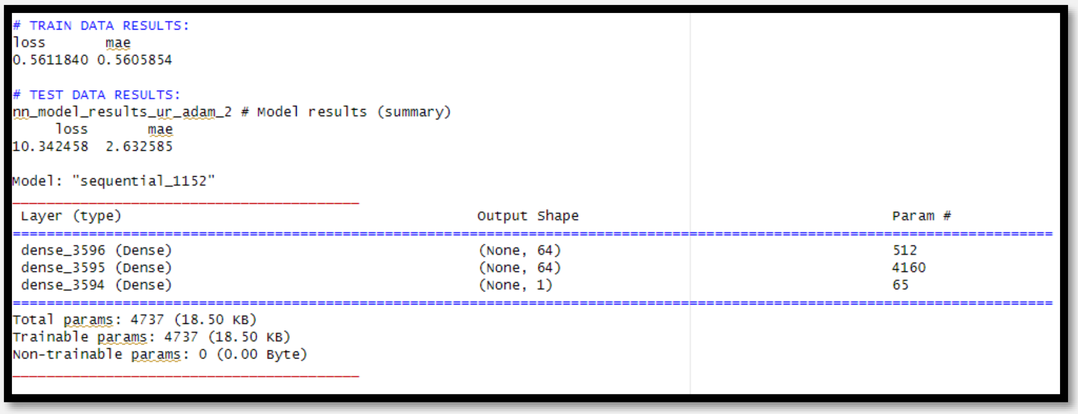
Neural Networks

Figure 19
NN Model History summary



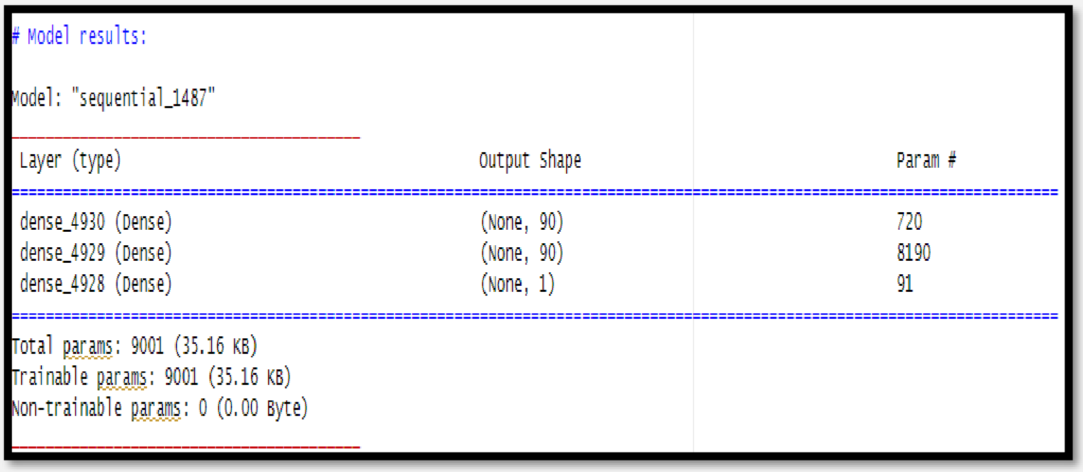
```
# nn_model_ur_adam
Final epoch (plot to see history):
     loss: 0.01979
      mae: 0.09804
 val_loss: 3.782
  val_mae: 1.633
```

Figure 20
Model summary utilizing the epoch = 78.



```
# TRAIN DATA RESULTS:
loss       mae
0.5611840 0.5605854

# TEST DATA RESULTS:
nn_model_results_ur_adam_2 # Model results (summary)
     loss       mae
10.342458  2.632585

Model: "sequential_1152"
________________________________________
 Layer (type)                          Output Shape                                    Param #
================================================================================================
 dense_3596 (Dense)                    (None, 64)                                      512
 dense_3595 (Dense)                    (None, 64)                                      4160
 dense_3594 (Dense)                    (None, 1)                                       65
================================================================================================
Total params: 4737 (18.50 KB)
Trainable params: 4737 (18.50 KB)
Non-trainable params: 0 (0.00 Byte)
________________________________________
```

Figure 21

	Model summary after building the model suing the optimal hyperparameters
	

```
# Model results:

Model: "sequential_1487"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4930 (Dense)          (None, 90)                720
 dense_4929 (Dense)          (None, 90)                8190
 dense_4928 (Dense)          (None, 1)                 91
=================================================================
Total params: 9001 (35.16 KB)
Trainable params: 9001 (35.16 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Appendix C: RAW Code

- **<u>C1 : Data Loading, Transformations and EDA</u>**

Description	**Code**

Load the Data
&
Rename Columns
	# Load the Data Set
	unemployment_rate = read.csv("Aus_data_2023.csv")
	# Rename Columns
	# Set New Column Names
	new_column_names =
	c("Year_Month","Unemployment_Rate","Percentage_change_GDP",
	
	"Percentage_change_GFCE","Percentage_change_FCEAI","Term_of_TI",
	
	"Consumer_Price_Index","Job_Vacancies","Estimated_Resident_Population")
	# Rename the old column names to new column names
	unemployment_rate = unemployment_rate %>% rename(Year_Month =
	"Year_Q",
	                              Unemployment_Rate = "Y",
	                              Percentage_change_GDP = "X1",
	                              Percentage_change_GFCE = "X2",
	                              Percentage_change_FCEAI = "X3",
	                              Term_of_TI = "X4",
	                              Consumer_Price_Index = "X5",
	                              Job_Vacancies = "X6",
	                              Estimated_Resident_Population = "X7")

Data Type
Transformations
	# 1. Transform the "Year_Q" column
	# Create a new data frame with the Year_Month column
	year_month = data.frame(Year_Month = unemployment_rate$Year_Month)
	# Split the Year_Month column into Year and Month Column
	year_month = separate(year_month, Year_Month,into = c("Year","Month"),
	sep = "_")
	#str(year_month)
	# Add this new column to the Original Data Frame
	unemployment_rate = cbind(unemployment_rate,year_month)
	# Remove the first column which consists of the "Year_Month" combined
	data
	unemployment_rate = unemployment_rate[,-1]
	#---
	# Convert "Year" to "Date" Format
	unemployment_rate$Year = as.Date(unemployment_rate$Year , format('%Y'))
	# Extract only Year
	unemployment_rate$Year = year(unemployment_rate$Year)
	str(unemployment_rate)
	#---
	# Create a new column named "Quarter"
	months = c("Mar", "Jun", "Sep", "Dec")
	quarters = c(1, 2, 3, 4)

	# Create the "Quarter" column based on the "Month" values
	unemployment_rate$Quarter = quarters[match(unemployment_rate$Month, months)]
	#View(unemployment_rate)
	str(unemployment_rate)

Check for Missing Values	# Check for Missing Values is.na(unemployment_rate) # Check for missing values colSums(is.na(unemployment_rate)) # Count the number of missing values per column # which(colSums(is.na(unemployment_rate))>0) # Identify the position of the columns with atleast one missing value names(which(colSums(is.na(unemployment_rate))>0)) # Return the column names with missing values

Miss Forrest Imputation	unemployment_rate_4 = unemployment_rate
	unemployment_rate_4 = unemployment_rate_4[,-c(10)]
	str(unemployment_rate_4)
	
	set.seed(99) # set seed to maintain model consistency
	# Impute missing values using PMM
	rf_imputed_Percentage_change_GDP_1 = missForest(unemployment_rate_4)$ximp$Percentage_change_GDP
	rf_imputed_Percentage_change_GFCE_1 = missForest(unemployment_rate_4)$ximp$Percentage_change_GFCE
	rf_imputed_Percentage_change_FCEAI_1 = missForest(unemployment_rate_4)$ximp$Percentage_change_FCEAI
	rf_imputed_Term_of_TI_1 = missForest(unemployment_rate_4)$ximp$Term_of_TI
	rf_imputed_Job_Vacancies_1 = missForest(unemployment_rate_4)$ximp$Job_Vacancies
	rf_imputed_Estimated_Resident_Population_1 = missForest(unemployment_rate_4)$ximp$Estimated_Resident_Population
	
	rf_imputed_values = data.frame(rf_imputed_Percentage_change_GDP_1,
	            rf_imputed_Percentage_change_GFCE_1,
	            rf_imputed_Percentage_change_FCEAI_1,
	            rf_imputed_Term_of_TI_1,
	            rf_imputed_Job_Vacancies_1,
	            rf_imputed_Estimated_Resident_Population_1)
	# View(rf_imputed_values)
	
	############### HISTOGRAM ######################
	
	p1_1_rf = ggplot(data = rf_imputed_values, mapping = aes(rf_imputed_Percentage_change_GDP_1)) + geom_histogram(bins = 40, colour = "cyan") + ggtitle('Percentage_change_GDP - After Imputation using RF')

	p1_2_rf = ggplot(data = rf_imputed_values, mapping = aes(rf_imputed_Percentage_change_GFCE_1)) + geom_histogram(bins = 40, colour = "cyan") + ggtitle('Percentage_change_GFCE - After Imputation using RF')
	
	p1_3_rf = ggplot(data = rf_imputed_values, mapping = aes(rf_imputed_Percentage_change_FCEAI_1)) + geom_histogram(bins = 40, colour = "cyan") + ggtitle('Percentage_change_FCEAI - After Imputation using RF')
	
	p1_4_rf = ggplot(data = rf_imputed_values, mapping = aes(rf_imputed_Term_of_TI_1)) + geom_histogram(bins = 40, colour = "cyan") + ggtitle('Term_of_TI - After Imputation using RF')
	
	p1_5_rf = ggplot(data = rf_imputed_values, mapping = aes(rf_imputed_Job_Vacancies_1)) + geom_histogram(bins = 40, colour = "cyan") + ggtitle('Job_Vacancies - After Imputation using RF')
	
	p1_6_rf = ggplot(data = rf_imputed_values, mapping = aes(rf_imputed_Estimated_Resident_Population_1)) + geom_histogram(bins = 40, colour = "cyan") + ggtitle('Estimated_Resident_Population - After Imputation using RF')
	
	gridExtra::grid.arrange(p1_1_rf,p1_2_rf,p1_3_rf,p1_4_rf,p1_5_rf,p1_6_rf, nrow = 3)

Further EDA	# Descriptive Statistics& Box Plots
	summary(unemployment_rate$Year)
	summary(unemployment_rate$Unemployment_Rate)
	summary(unemployment_rate$Percentage_change_GDP)
	summary(unemployment_rate$Percentage_change_GFCE)
	summary(unemployment_rate$Percentage_change_FCEAI)
	summary(unemployment_rate$Term_of_TI)
	summary(unemployment_rate$Consumer_Price_Index)
	summary(unemployment_rate$Job_Vacancies)
	summary(unemployment_rate$Estimated_Resident_Population)
	
	# Boxplot for Unemployment Rate
	gg_1 = ggplot(unemployment_rate, aes(y = Unemployment_Rate)) +
	  geom_boxplot(fill = "blue", color = "black") +
	  labs(title = "Boxplot of Unemployment Rate", y = "Unemployment Rate")
	
	# Boxplot for Percentage Change in GDP
	gg_2 = ggplot(unemployment_rate, aes(y = Percentage_change_GDP)) +
	  geom_boxplot(fill = "green", color = "black") +
	  labs(title = "Boxplot of Percentage Change in GDP", y = "Percentage Change in GDP")
	
	# Boxplot for Percentage Change in GFCE
	gg_3 = ggplot(unemployment_rate, aes(y = Percentage_change_GFCE)) +
	  geom_boxplot(fill = "orange", color = "black") +

	labs(title = "Boxplot of Percentage Change in GFCE", y = "Percentage Change in GFCE")
	
	# Boxplot for Percentage Change in FCEAI
	gg_4 = ggplot(unemployment_rate, aes(y = Percentage_change_FCEAI)) +
	  geom_boxplot(fill = "purple", color = "black") +
	  labs(title = "Boxplot of Percentage Change in FCEAI", y = "Percentage Change in FCEAI")
	
	# gridExtra::grid.arrange(gg_1, gg_2, gg_3, gg_4 , nrow = 2)
	
	# Boxplot for Consumer Price Index
	gg_5 = ggplot(unemployment_rate, aes(y = Term_of_TI)) +
	  geom_boxplot(fill = "red", color = "black") +
	  labs(title = "Boxplot of Consumer Price Index", y = "Consumer Price Index")
	
	# Boxplot for Consumer Price Index
	gg_6 = ggplot(unemployment_rate, aes(y = Consumer_Price_Index)) +
	  geom_boxplot(fill = "orange", color = "black") +
	  labs(title = "Boxplot of Consumer Price Index", y = "Consumer Price Index")
	
	# Boxplot for Job Vacancies
	gg_7= ggplot(unemployment_rate, aes(y = Job_Vacancies)) +
	  geom_boxplot(fill = "brown", color = "black") +
	  labs(title = "Boxplot of Job Vacancies", y = "Job Vacancies")
	
	# Boxplot for Estimated Resident Population
	gg_8 = ggplot(unemployment_rate, aes(y = Estimated_Resident_Population)) +
	  geom_boxplot(fill = "pink", color = "black") +
	  labs(title = "Boxplot of Estimated Resident Population", y = "Estimated Resident Population")
	
	gridExtra::grid.arrange(gg_1, gg_2, gg_3, gg_4 ,gg_5, gg_6, gg_7,gg_8, nrow = 2)

Data Partitioning	set.seed(99) # Set the seed for model consistency
	
	# Training Data = Dec 1982 to Dec 2020
	ur_training_filter = (unemployment_rate$Year < 2021) \| (unemployment_rate$Year == 2020 & unemployment_rate$Month == "Dec")
	
	# Test Data = Jan 2021 to March 2023
	ur_test_filter = (unemployment_rate$Year >= 2021) \| (unemployment_rate$Year == 2021 & unemployment_rate$Month %in% c("Mar"))
	
	# Create the training and test sets

	ur_train = unemployment_rate[ur_training_filter,]
	ur_test = unemployment_rate[ur_test_filter,]
	
	dim(ur_train) # 156 x 11
	dim(ur_test) # 10 x 11
	
	ur_train = ur_train[,-c(2,3,4)] # Remove Year, Month and Quarter since it's not a part of the original predictors
	ur_test = ur_test[,-c(2,3,4)] # Remove Year, Month and Quarter since it's not a part of the original predictors
	
	#View(ur_train)
	#View(ur_test)
	
	dim(ur_train) # 156 x 8
	dim(ur_test) # 10 x 8

- **C2: Code: ML and NN Models**

Model	Code

RIDGE REGRESSION	# Build a matrix of Predictors
	x = model.matrix(Unemployment_Rate ~ . , data = ur_train)[,-1] # remove the response variable
	y = ur_train$Unemployment_Rate # Response Variable
	
	# ------------------------------ CROSS VALIDATION using a grid of Lambda values-----------------------#
	# Set the lambda values
	grid = 10^seq(10, -2 , length = 100)
	
	# Applying Cross-validation
	start_time_1 = Sys.time() # Start the timer
	
	set.seed(99)
	
	ridge_model_cv = cv.glmnet(x,y, alpha = 0, family = 'gaussian', lambda = grid, scale = TRUE) # Build the 10 - fold CV Model on training data
	
	end_time_1 = Sys.time() # End the timer
	(time_diff_1 = end_time_1 - start_time_1)
	
	ridge_model_cv # Model Results
	summary(ridge_model_cv) # Model Summary
	
	# Plot the model for the CV_Lasso
	plot(ridge_model_cv, xvar = "lambda", main = "Ridge CV using a grid of Lambda Values")
	
	# Minimal Lambda

	(best_lambda_ridge = ridge_model_cv$lambda.min)# Minimum Lambda = 0.01
	
	# ------------------------------ PERFORMANCE ON TRAINING DATA ------------------------------------#
	# Prediction on Training Data using Best (Minimum Lambda)
	ridge_train_predict = predict(ridge_best_model, s = ridge_best_model$lambda, newx = x)
	
	# Mean Squared Error for Training Data
	(mse_ridge_train = mean((ridge_train_predict - y)^2))
	
	# Root Mean Squared Error for Training Data
	(rmse_ridge_train = sqrt(mse_ridge_train))
	
	# ------------------------------ PERFORMANCE ON TEST DATA -----------------------------------#
	# Prediction on Test Data using Best (Minimum Lambda)
	x_test = model.matrix(Unemployment_Rate ~ ., data = ur_test)[, -1]
	
	ridge_test_predict = predict(ridge_best_model, newx = x_test) # Predictions on the Test Data
	
	# Mean Squared Error for Test Data
	(mse_ridge_test = mean((ridge_test_predict - ur_test$Unemployment_Rate)^2))
	
	# Root Mean Squared Error for Test Data
	(rmse_ridge_test = sqrt(mse_ridge_test))

LASSO REGRESSION	# Build a matrix of Predictors
	x = model.matrix(Unemployment_Rate ~ . , data = ur_train)[,-1] # remove the response variable
	y = ur_train$Unemployment_Rate # Response Variable
	
	# ------------------------------- CROSS VALIDATION using a grid of Lambda values------------------------#
	# Set the lambda values
	grid = 10^seq(10, -2 , length = 100)
	
	# Applying Cross-validation
	start_time_3 = Sys.time() # Start the timer
	
	set.seed(99) # Set seed for model consistency
	lasso_model_cv = cv.glmnet(x,y, alpha = 1, family = 'gaussian', lambda = grid, scale = TRUE) # Build the model using the training data
	
	end_time_3 = Sys.time() # End the timer
	(time_diff_3 = end_time_3 - start_time_3) # Time difference of 0.1514041 secs
	
	lasso_model_cv # Model results
	summary(lasso_model_cv) # Model summary

	# Plot the model for the CV_Lasso
	plot(lasso_model_cv, main = "Lasso CV using a grid of Lambda Values")
	
	# Minimal Lambda
	(best_lambda_lasso = lasso_model_cv$lambda.min)# Minimum Lambda = 0.01
	
	# Build the Lasso Model using the Best Lambda Value (0.01)
	set.seed(99)
	lasso_best_model = glmnet(x,y, alpha = 1, lambda = best_lambda_lasso,scale = TRUE)

CART	set.seed(99) # Set seed for model consistency
	tree_ur = rpart(Unemployment_Rate ~. , data = ur_train) # Build the CART model on training data
	
	summary(tree_ur) # Model summary
	
	# Plot the model
	par(xpd = TRUE)
	rattle::fancyRpartPlot(tree_ur, main = "Basic CART") # using rattle
	
	# ---USING CV = 10 ---#
	start_time_5 = Sys.time() # Start the timer
	
	cv_tree_ur = rpart(Unemployment_Rate ~. , data = ur_train, xval = 10, model = TRUE) # Build the model using 10-FOld CV on training data
	
	end_time_5 = Sys.time() # End the timer
	(time_diff_5 = end_time_5 - start_time_5) # Time difference of 0.008260965 secs
	
	summary(cv_tree_ur) # Model summary
	
	rpart.plot::rpart.plot(cv_tree_ur, yesno = TRUE , main = "CART with CV") # Plot the tree
	
	varImp(cv_tree_ur) # Variable Importance
	plot(cv_tree_ur$variable.importance) # Variable Importance plot
	
	# Plot the Complexity Parameter
	plotcp(cv_tree_ur) # cp = 0.021

BAGGING	start_time_7 = Sys.time() # Start the timer
	
	set.seed(99) # Set the seed for model consistency
	# Hyperparameters for Bagging
	hyperparameters = expand.grid(ntree = c(20:500)) # Vary the number of trees

```
# Initialize variables to store the best MSE and corresponding hyperparameters
for training data
best_mse_train = Inf
best_hyperparameters_train = NULL

# Bagging loop
for (i in 1:nrow(hyperparameters))
  {
   ntree = hyperparameters$ntree[i]

   # Create a bootstrapped sample from the training data
   bootstrap_sample = ur_train[sample(1:nrow(ur_train), replace = TRUE), ]

   # Train a random forest regression model (bagging) with the current
hyperparameters
   ur_bag_cart = randomForest(Unemployment_Rate ~. ,
                             data = bootstrap_sample,
                             ntree = ntree,
                             mtry = 7,
                             scale = TRUE,
                             importance = TRUE)

   # Make predictions on the training set
   predictions_train = predict(ur_bag_cart, newdata = ur_train)

   # Calculate MSE for the training set
   mse_train = mean((predictions_train - ur_train$Unemployment_Rate)^2)

   # Check if this model has a better MSE compared to the overall best for
training data
   if (mse_train < best_mse_train)
     {
      best_mse_train = mse_train
      best_hyperparameters_train = ntree
      best_model_bag_train = ur_bag_cart
     }
}

end_time_7 = Sys.time()  # End the timer
(time_diff_7 = end_time_7 - start_time_7) # Time difference of 57.03038 secs

# Calculate RMSE for training data
(rmse_train = sqrt(best_mse_train))

# # Print the best MSE for training data
# cat("Best Mean Squared Error (MSE) for Training Data:", best_mse_train,
"\n")
#
# # Print the RMSE for training data
```

	# cat("Root Mean Squared Error (RMSE) for Training Data:", rmse_train, "\n")
	
	# Print the best hyperparameters for training data
	cat("Best Hyperparameters for Training Data:\n",
	    "Number of Trees (ntree):", best_hyperparameters_train, "\n")
	
	print(best_model_bag_train) # Print the Best model obtained from the Bagging
	
	best_model_bag_train$importance # Variable Importance
	
	# Variable Importance Plot
	varImpPlot(best_model_bag_train, main = "Variable Importance - Bagging")

RANDOM FOREST	start_time_9 = Sys.time() # Start the timer
	set.seed(99) # Set the seed for model consistency
	
	# Hyperparameters for Random Forest
	hyperparameters = expand.grid(
	  ntree = c(20:500), # Number of trees in the forest
	  max_depth = c(5, 10, 15), # Maximum depth of each tree
	  mtry = 3 # Number of features considered for splitting at each node (7/3 ~ here is 3)
	  )
	
	# Initialize variables to store the best MSE and corresponding hyperparameters
	best_mse_train_rf = Inf
	best_hyperparameters_train = NULL
	
	# Random Forest loop
	for (i in 1:nrow(hyperparameters))
	  {
	
	    # Train a random forest regression model with the current hyperparameters
	    rf_ur = randomForest(Unemployment_Rate ~ .,
	                        data = ur_train,
	                        ntree = hyperparameters$ntree[i],
	                        max_depth = hyperparameters$max_depth[i],
	                        mtry = hyperparameters$mtry[i],
	                        importance = TRUE)
	
	    # Make predictions on the Training set
	    predictions_train = predict(rf_ur, newdata = ur_train)
	
	    # Calculate MSE for the training set
	    mse_train_rf = mean((predictions_train - ur_train$Unemployment_Rate)^2)
	
	    # Check if this model has a better MSE compared to the overall best
	    if (mse_train_rf < best_mse_train_rf)
	      {
	        best_mse_train_rf = mse_train_rf

	`        best_hyperparameters_train = hyperparameters$ntree[i]  # Update the best hyperparameters`
	`        best_model_rf_train = rf_ur  # Update the best model`
	`      }`
	`}`
	
	`#start_time_9 = Sys.time()  # Start the timer`
	`end_time_9 = Sys.time()  # End the timer`
	`(time_diff_9 = end_time_9 - start_time_9)`
	
	`# Print the best hyperparameters for the training set`
	`cat("Best Hyperparameters:\n",`
	`    "Number of Trees (ntree):", best_hyperparameters_train, "\n",`
	`    "Maximum Depth:", hyperparameters$max_depth[which.min(best_mse_train_rf)], "\n",`
	`    "mtry:", hyperparameters$mtry[which.min(best_mse_train_rf)], "\n"`
	`    )`
	
	`print(best_model_rf_train) # Summary of the best model`
	`best_model_rf_train$importance # Variable Importance for Random Forest`
	`varImpPlot(best_model_rf_train, main = "Variable Importance - Random Forest") # Variable Importance`

STOCHASTIC GRADIENT BOOSTIN	`# TUNE THE BOOSTING PARAMETERS`
	`# Set the training control for Boosting`
	`ur_train_control = trainControl(method = "cv",`
	`                                number = 10,`
	`                                savePredictions = "final"`
	`                                 ) # save`
	
	`# Set the hyperparameters for Boosting`
	`gbmGrid =  expand.grid(interaction.depth = c(2,5,10,15),  # Max Tree Depth`
	`                        n.trees = c(1:150), # Boosting Iterations`
	`                        shrinkage = c(0.001, 0.1,0.15),# Shrinkage = Learning Rate`
	`                        n.minobsinnode = c(5,10,20) # minimum number of training set samples in a node to commence splitting`
	`                       )`
	
	`start_time_12 = Sys.time()  # Start the timer`
	`set.seed(99) # Set seed for Model Consistency`
	`boost_ur_tune = caret::train(Unemployment_Rate ~ . , data = ur_train,`
	`                         method = "gbm",`
	`                         trControl = ur_train_control,`
	`                         tuneGrid = gbmGrid)`
	
	`#start_time_12 = Sys.time()  # Start the timer`
	`end_time_12 = Sys.time()  # End the timer`
	`(time_diff_12 = end_time_12 - start_time_12) # Time difference of 3.900011 mins`

SVM – linear	`# With kernel = "linear" with cost and gamma`

	start_time_14 = Sys.time() # Start the timer
	set.seed(99) # Set seed for model consistency
	tune_ur_linear = tune(svm,
	              Unemployment_Rate ~. ,
	              data = ur_train,
	              scale = TRUE,
	              kernel= c("linear"),
	              ranges=list(cost= c(0.001, 0.01, 0.1, 1,5,10,50),
	                        gamma = c(0.001, 0.01, 0.1, 1,5,10,50)))
	
	# start_time_14 = Sys.time() # Start the timer
	end_time_14 = Sys.time() # End the timer
	(time_diff_14 = end_time_14 - start_time_14) # Time difference of 9.060912 secs

SVM – radial	# 2. With kernel = "radial" and hyperparameters : cost and gamma
	```
	{r}
	# With kernel = "linear" with cost and gamma

	start_time_15 = Sys.time()  # Start the timer
	set.seed(99)
	tune_ur_radial = tune(svm,
	              Unemployment_Rate ~. ,
	              data = ur_train,
	              scale = TRUE,
	              kernel= "radial",
	              ranges=list(cost= c(0.001, 0.01, 0.1, 1,5,10,50),
	                        gamma = c(0.001, 0.01, 0.1, 1,5,10,50)))

	# start_time_15 = Sys.time()  # Start the timer
	end_time_15 = Sys.time()  # End the timer
	(time_diff_15 = end_time_15 - start_time_15) # Time difference of 4.546159 secs
	```

SVM – polynomial	# 3. With kernel = "polynomial" and hyperparameters : cost and gamma
	```
	{r}
	# With kernel = "linear" with cost and gamma
	start_time_16 = Sys.time()  # Start the timer
	set.seed(99) # Set seed for model consistency
	tune_ur_poly = tune(svm,
	              Unemployment_Rate ~. ,
	              data = ur_train,
	              scale = TRUE,
	              kernel= "polynomial",
	              ranges=list(cost= c(0.001, 0.01, 0.1, 1,5),
	                        gamma = c(0.001, 0.01, 0.1, 1),
	                        degree = c(2,3,4)
	                        ))
	```

	# start_time_16 = Sys.time() # Start the timer
	end_time_16 = Sys.time() # End the timer
	(time_diff_16 = end_time_16 - start_time_16) # Time difference of 36.7868 secs

Neural Networks	# Build the model based on the Model based on the Lowest error rate (Epoch value = 78)
	#----Train performance ----
	
	# Define the neural network model
	nn_model_ur_train = keras_model_sequential() %>%
	  layer_dense(units = 64, activation = "relu", input_shape = c(ncol(ur_train_nn_x_scaled))) %>%
	  layer_dense(units = 64, activation = "relu") %>%
	  layer_dense(units = 1)
	
	# Define metrics
	nn_model_ur_train %>% compile(optimizer = "adam",
	                          loss = "mse",
	                          metric = c("mae"))
	
	# Build the model on Training Data
	
	start_time_18 = Sys.time() # Start the timer
	set.seed(99)
	nn_model_ur_train %>% fit(ur_train_nn_x_scaled,
	                      ur_train_nn_y,
	                      epochs = 78, # Optimal epoch value derived from ggplot
	                      validation_data = list(ur_train_nn_x_scaled,ur_train_nn_y),
	                      verbose = 1)
	
	end_time_18 = Sys.time() # End the timer
	(time_diff_18 = end_time_18 - start_time_18) # Time difference
	
	# Model Performance on Training Data
	# After the network has been tuned to a suitable number of epochs, the validation data can be predicted using the evaluate() function.
	nn_model_results_ur_adam_train = nn_model_ur_train %>% evaluate(ur_train_nn_x_scaled , ur_train_nn_y)
	
	nn_model_results_ur_adam_train # Model results (summary)
	
	# -------------------- Hyper parameter tuning ----------------------------------#
	# Define hyperparameters
	epochs_list_ = c(70, 80, 90, 100)
	units_list_ = c(30, 60, 90)
	batch_sizes_ = c(10, 30, 50, 70)
	
	# Create a data frame with all combinations of hyperparameters
	hyperparameter_combinations = expand.grid(epochs = epochs_list_, units = units_list_, batch_size = batch_sizes_)

```
# Initialize variables to store the best results and hyperparameters
best_train_loss = Inf
best_hyperparameters = NULL

start_time_19 = Sys.time()  # Start the timer
set.seed(99)
# Iterate through hyperparameter combinations
for (i in 1:nrow(hyperparameter_combinations))
  {
    epochs = hyperparameter_combinations$epochs[i]
    units = hyperparameter_combinations$units[i]
    batch_size = hyperparameter_combinations$batch_size[i]

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

    # Define the neural network model
    nn_model= keras_model_sequential() %>%
      layer_dense(units = units, activation = "relu", input_shape =
c(ncol(ur_train_nn_x_scaled))) %>%
      layer_dense(units = units, activation = "relu") %>%
      layer_dense(units = units, activation = "relu") %>%
      layer_dense(units = 1)

    # Compile the model
    nn_model %>% compile(optimizer = "adam",
                     loss = "mse",
                     metrics = c("mae"))

    # Train the model on Training Data
    set.seed(99)
    history = nn_model %>% fit(ur_train_nn_x_scaled,
                           ur_train_nn_y,
                           epochs = epochs,
                           batch_size = batch_size,
                           validation_data = list(ur_train_nn_x_scaled, ur_train_nn_y),
                           verbose = 1)

    # Model Performance on Test Data
    nn_model_results_train = nn_model %>% evaluate(ur_train_nn_x_scaled,
ur_train_nn_y)
    cat("Test results (summary):\n")
    print(nn_model_results_train)

    # Check if the current test results are better than the best so far
    if (nn_model_results_train[[1]] < best_train_loss)
      {
        (best_train_loss = nn_model_results_train[[1]])
        (best_hyperparameters = c(epochs, units, batch_size))
        }
}
end_time_19 = Sys.time()  # End the timer
(time_diff_19 = end_time_19 - start_time_19) # Time difference of 12.48
minutes

# Print the best hyperparameter results
cat("Best Hyperparameters:\n")
cat("Epochs:", best_hyperparameters[1], "\n")
cat("Units:", best_hyperparameters[2], "\n")
cat("Batch Size:", best_hyperparameters[3], "\n")
cat("Best Train Loss:", best_train_loss,"\n")
```