# MA5832 Capstone Assessment (Assessment 3)

Samuel Bousfield

2023-10-15

## Abstract/Executive Summary

This report will perform an analysis of the Unemployment Rate prediction over an almost 40-year period. This analysis will utilise two machine learning models, a Support Vector Machine (SVM) using a radial kernel and an Artificial Neural Network (ANN). The report examines the suitability of seven socio-economic predictors in this estimation and will explore data preprocessing, missing value imputation, model selection, hyperparameter tuning and model evaluation metrics. The key findings are:

Data Preprocessing – This report examines various methods of dealing with missing values, including dropping the observations, leaving the missingness and ultimately opting for interpolation of the missing values.

SVM – An SVM was chosen over a Forest model due to its regression capabilities and the purely numeric predictor variables. This report examines the process of hyperparameter tuning, cross-validation, and evaluation of the final model on both the training and the testing data.

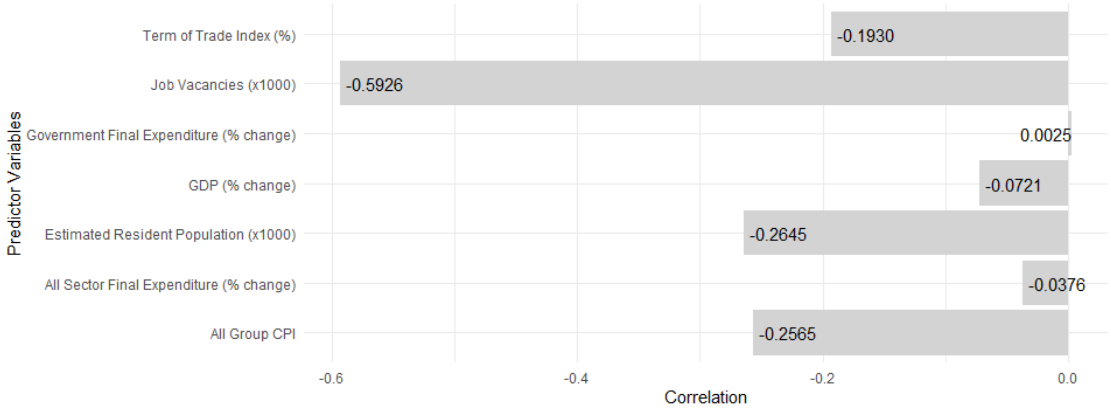
ANN – An ANN model is implemented with varying numbers of hidden layers and neurons; the ANN model in this report chose to use 2 layers of 7 neurons and 1 neuron. These values were not tuned; rather, they were chosen semi-randomly. This model was then evaluated on both the training and testing data. This report also examines varying the number of layers and varying the number of neurons and highlights the risk of overfitting and underfitting the data.

Comparison and Suggestions – This report compares the SVM and ANN models based on cross-validated accuracy, computational time, and interpretability. Suggestions for improving the methodology employed in this report include utilising cross-validation and hyperparameter tuning for the ANN model. Suggestions for improving the data include gathering more observations, and potentially gathering more predictor variables.

# Overview of Unemployment Rate

Unemployment is a critical problem within a society, leading to poverty, low living standards, and pressure on welfare and other public services, and is proven to lead to worse health outcomes, physically and mentally. This report aims to assess over 20 years of unemployment data and seven socio-economic predictors and apply a Support Vector Machine (SVM) model and an artificial neural network (ANN) to assess the suitability of using these predictors to estimate the unemployment rate. The graph below shows the correlation between each predictor variable and the Unemployment Rate (%). Figure 1 shows that the Job Vacancies predictor is the best predictor of the unemployment rate, followed by the Estimated Resident Population and All Group CPI. Government Final Expenditure appears to have almost no correlation whatsoever; this variable also has a positive correlation. The initial implications of this are that changing government expenditure does not really influence the unemployment rate, although an increase in government expenditure is correlated with an extremely small increase in the unemployment rate.

Figure 1. Predictor Correlation to Unemployment Rate (%)




# Data

## 2(a.1) - Preprocessing

Preprocessing included assessing the column types. A brief examination of the data showed that all the data is numeric; assessing the column types showed that two of the columns are of type character. This is caused by missing data. The predictor and response variables were then coerced to be of type numeric, which introduced NAs. There are 5 missing values in the Job Vacancies and Estimated Resident Population fields, which is approximately 3.2% of the data in each column, although these are at different row levels. Therefore, approximately 6.3% of the data has missing values. Three main options are available.

1) Since the observations with missing data account for less than 10% of the data, they could be dropped from the data set. This is not a good idea as the testing datasets for the models have been specifically defined, and this would drop ~50% of the testing data.

2) The missing values could be left as is.

3) Interpolate or Impute values. These three options will be explored in the next section.

## 2(a.2) - Missing Value Imputation

There are two variables with missing values: Estimated Resident Population (x1000) and Job Vacancies (x1000). Examining Figures 2 and 3 below, the population variable appears to be increasing relatively steadily. One method of missing value imputation is to find an external source of data, which should be easy in this case; population statistics are usually easy to find. however, this is inappropriate in two ways for this data:

1) This dataset treats the population of Australia as x10 greater than the real variables.

2) It cannot be properly inferred that we can take the real population statistics and multiply them by 10. This is also the case for the Job Vacancies variable.

**Figure 2. Estimated Resident Population (x1000) real values**

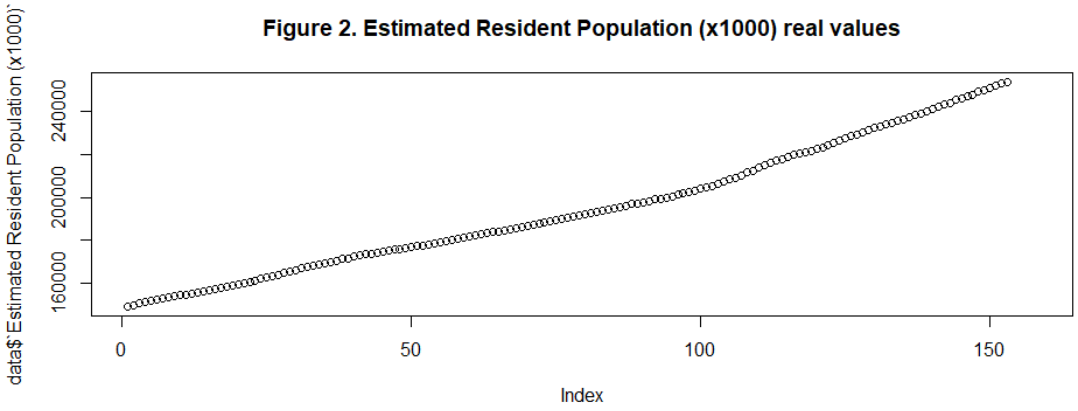
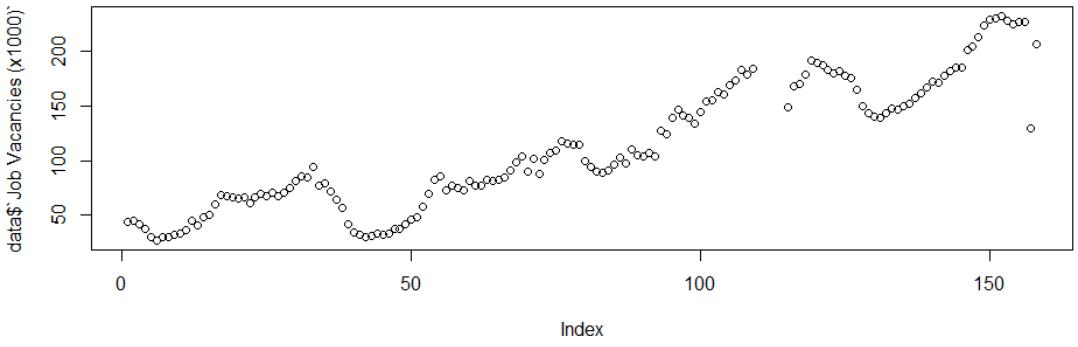



**Figure 3. Job Vacancies (x1000) real values**




Examining Figures 2 and 3 above, for the variables with missing values, we can see that the Estimate Resident Population appears to be increasing in a linear fashion, and while the Job Vacancies variable is not linear, it also appears to be following a pattern, likely a cubic function. Thus, some of the normal methods for dealing with missing data are likely to be inappropriate (knn, mean imputation). The best static value imputation for these is likely to be the previous filled value (or max value) in the case of the resident population, and either

that or the mean value of the previous filled value and the following filled value for the Job Vacancies data. However, the most appropriate option for both is likely to interpolate. A few methods will be assessed for appropriateness, using the na_interpolation() function from the imputeTS package and generating a simple linear or polynomial model. These will then be examined for what appears to fit the data best, and then the values imputed into the original data.

The methods have been completed; these will be assessed visually using the ggplot package. The geom_jitter option has been used to see individual points more clearly. This offset should be considered when assessing the best fit for the data.

Figure 4. Estimated Resident Population
Real and Interpolated Values

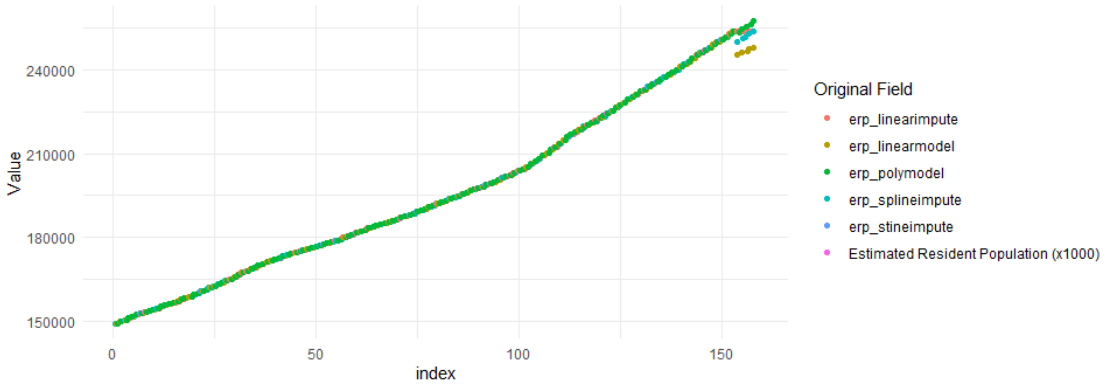
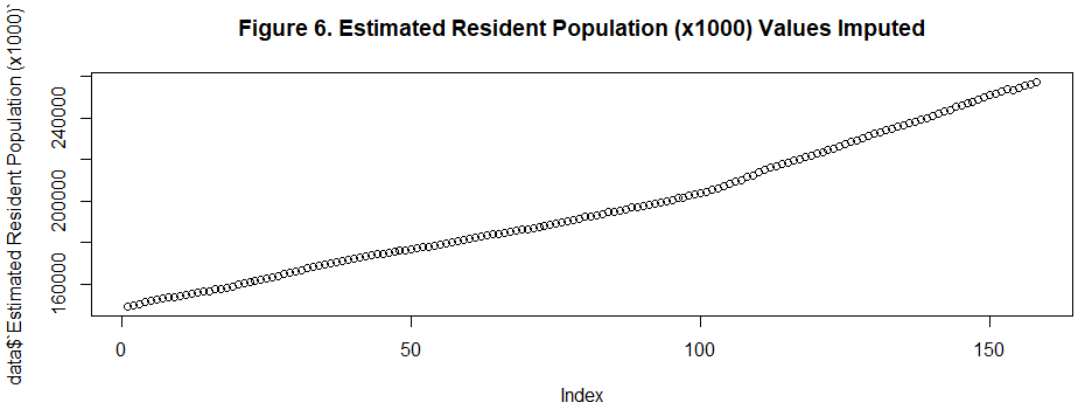


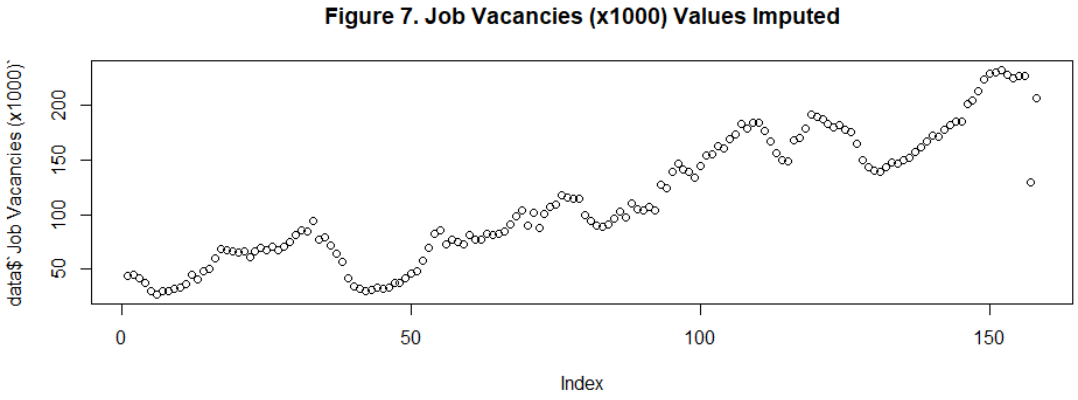

Figure 5. Job Vacancies
Real and Interpolated Values


In the Estimated Resident Population, Figure 4, the slight curve makes a significant difference, resulting in the linear model not fitting the data well, and the other imputation methods just take the max value in the field. However, the polynomial model fits the data very well.

In Figure 5, the linear and Stine imputation both appear to work almost equally well, while the spline imputation takes the lowest value beyond the next observation with a value. While this may be correct in the real data, it is better to assume a value between the two values with data. In this case, the Stine imputation option has been chosen, although the linear option provides a similar level of appropriateness.

```
##                                     Period                       Unemployment
Rate (%)
##                                          0
0
##                            GDP (% change) Government Final Expenditure (%
change)
##                                          0
0
## All Sector Final Expenditure (% change)                  Term of Trade
Index (%)
##                                          0
0
##                              All Group CPI                    Job Vacancies
(x1000)
##                                          0
0
##   Estimated Resident Population (x1000)
##                                          0
```




Figures 6 and 7 show that the chosen interpolation methods for the two variables with missing values appear to fit very well. There is an obvious dip in Figure 6, where the values were interpolated, although they follow the same trend, and in Figure 7 the interpolation is seamless.

### 2(b) - Descriptive Statistics

The summary of the data shows that the data covers the period from June 1981 to September 2020. The unemployment rate for this period is between 4.1% and 11.133%, with a mean employment rate of 6.852%. GDP change ranges include negative and positive values, ranging from -7% change to 3.3% change. Government and All Sector Expenditure % change also covers negative and positive values with a minimum of -4.6% and -8.3% change and a maximum of 7.5% and 5.9% change, respectively. The Term of Trade Index % covers a range of -8.1 to 5.9%. These values indicate there are likely to be interactions within the data between these predictors and the response variable.

## Machine Learning

Because of the choice of ML algorithm and the comparison to a neural network, it is good practice to scale the data for a few reasons. The Period variable is not a predictor variable and, thus, will not be included in the model formula.

Testing data is specified to be data from March 2018 - September 2020. Therefore, the Training data is all data up to 2017.

### 3(a) - Justify Choice

SVM has been chosen because this is a regression model, and all the variables are numeric. This means there is likely a complex relationship between all 7 predictor variables and the response variable. This is an area in which an SVM with a non-linear kernel will perform better than a Forest model. The sample size is another major reason for choosing an SVM model over a Forest. SVMS perform well with small and medium-sized datasets, while Forest's may overfit.

### 3(b) - Hyperparameters

Because of the small dataset, the Cost and Sigma values can be tuned in a reasonable amount of time. The C (cost) hyperparameter represents the trade-off between training error and testing error. A higher value here encourages the SVM to classify more training points accurately, resulting in a smaller margin, but it has the potential trade-off of overfitting. The Sigma hyperparameter influences the boundary. A smaller sigma is sensitive to local patterns, while a larger sigma is more sensitive to global patterns; this may lead to overfitting if it is too small or underfitting if it is too large.

The hyperparameters will be tuned using the cross-validation method, and the number of cross-validations and repeats can be relatively large due to the small number of observations. The optimal hyperparameters will then be taken and used to train another model; because of the high number of cross-validations and repeats, these optimal

hyperparameters can be trusted to be the optimal combination of hyperparameters in this case.

The hyperparameter tuning has been saved to RDS to be reloaded due to training time. The optimal hyperparameters will then be extracted from the tuning model and used to train the final model, which will also be saved due to training time.

### 3(c) - Predictive Performance on Training Data

```
## [1] "SVM Model on Training Data"

##      RMSE  Rsquared       MAE
## 0.5039870 0.9226239 0.3141412
```

These values are very good. It is difficult to evaluate the Root Mean Square Error (RMSE) in a vacuum unless it is used to compare to other models, so the value will not be examined too deeply, but a lower RMSE indicates a model that fits the data better. The R-squared value of 0.92263239 indicates a very well-fitted model. The R-squared value indicates the amount of variance in the data/response variable that is explained by the model (0 = 0%; 1 = 100%). Thus, this value indicates that ~92% of the variance in the data is explained by the model, which indicates a model of exceptional fit. The Mean Absolute Error (MAE), like the RMSE, is an average of the errors, and a lower value indicates a better model; however, because the MAE value is the absolute value, the result is in the original units of the predictor variable. This means that the average difference between what the model predicts the Unemployment Rate (%) to be and the actual Unemployment Rate (%) is 0.3141412%. This could be considered a very good value; the range of real results is between 4.1% and 11.133%, with a mean of 6.852% and a median of 6.250%. This indicates that the SVM model varies from the average of the real results by ~4.5%. It is important to note that these predictions are on the training data, where the model would be expected to perform well, although a poor result here is a good indicator of model unsuitability. Next, the SVM model will be assessed on the training data.

### 3(d) - Predictive Performance on Testing Data

```
## [1] "SVM Model on Testing Data"

##      RMSE  Rsquared       MAE
## 0.3822914 0.9414006 0.3310902
```

The model has performed very well on the test data; interestingly, the RMSE and R-squared values indicate that the model is a better fit on the test data than the training data (Train RMSE = 0.5039870, Test RMSE =0.3822914; Train R-squared = 0.9226239, Test R-squared = 0.9414006), although the MAE is larger (0.3310902) as expected. These results are likely the function of the small testing data set.

These values indicate that while the model fits the data better, the incorrect values are slightly more incorrect. However, this only results in an error from the average of the real results in ~4.8%. Again, it is critical to recognise that the testing data is an extremely small dataset (11 observations).

The results from both sets of predictions indicate that the SVM model performs exceptionally well.

# Neural Network

## 4(a.1) - Describe the Structure of the Model

An Artificial Neural Network (ANN),

## 4(a.2) - Implement the model

The initial ANN will be implemented using 7 hidden layers of 1 neuron each. These numbers were chosen semi-randomly. It is important to note that these values are "random". This model could be trained with any number of hidden layers and any number of neurons; the only limit would be training time.

## 4(b) - Predictive Performance on Training Data

```
## [1] "Initial Neural Network on Training Data:"

##      RMSE  Rsquared       MAE
## 0.2373404 0.9826311 0.1868678
```

These results are phenomenal. The R-squared value indicates that the model correlates to ~98% of the variation within the training data. The MAE is also very small, which, compared to the average result, is ~2.7% variation. These values indicate that this model has extreme precision. One major concern with this high correlation is the possibility of overfitting. It will be important to assess the model on the training data.

## 4(c) - Predictive Performance on Testing Data

```
## [1] "Initial Neural Network on Testing Data:"

##      RMSE  Rsquared       MAE
## 0.4547744 0.9612497 0.4220960
```

The ANN model on the testing data is also performing exceptionally well; as expected, the RMSE and MAE are higher than the training data, and the R-squared value is lower. However, the model still correlates to ~96% of the variation within the testing data (compared to ~98% in the training data). The MAE value is more than twice the value of the MAE in the training data predictions (0.4220960 vs 0.1868678).

The results from both sets of predictions indicate that the ANN model, like the SVM model, performs exceptionally well. The ANN model appears to be performing better than the SVM and certainly performs better on the current set of data.

## 4(d) - Varied Hidden Layers

```
##          CPUTime Layers Neurons train_RMSE  train_R2 train_MAE test_RMSE
## elapsed     0.68      1       1  0.8308023 0.7871742 0.6494376 1.0218865
## elapsed1    0.58      2       1  0.8308154 0.7871675 0.6498082 1.0373238
```

```
## elapsed2    4.19      3       1  1.0494602 0.6604054 0.7173622 0.7556050
## elapsed3    5.76      4       1  1.0493704 0.6604635 0.7171117 0.7556664
## elapsed4    9.86      5       1  1.0493595 0.6604705 0.7169628 0.7557888
## elapsed5   15.94      6       1  1.0495715 0.6603334 0.7177880 0.7545955
## elapsed6   17.94      7       1  1.0491610 0.6605990 0.7165245 0.7557236
## elapsed7    0.02      8       1  1.8008803 0.6420708 1.5352277 1.5723640
## elapsed8    0.02      9       1  1.8008832 0.6369178 1.5352316 1.5723732
## elapsed9    0.03     10       1  1.8008831 0.6228856 1.5352425 1.5724263
##            test_R2  test_MAE
## elapsed  0.4246709 0.8720590
## elapsed1 0.4258969 0.8911329
## elapsed2 0.3960553 0.4337242
## elapsed3 0.3963245 0.4336768
## elapsed4 0.3967623 0.4335819
## elapsed5 0.3971423 0.4345101
## elapsed6 0.3974456 0.4336329
## elapsed7 0.4079535 1.4286804
## elapsed8 0.4039243 1.4286870
## elapsed9 0.4035932 1.4287247
```

In this set of testing, it appears that with an increase in the number of layers, with a constant number of neurons (in this case 1), surprisingly, the model gets worse on the training data predictions and stays relatively stable in the testing data predictions, with a slight dip. The model training time rises, with a sharp jump at the 6 and 7-layer models, before dropping again for the 8, 9 and 10-layer models. The initial expectation was that the model would get better with a larger number of hidden layers, but this does not appear to be the case. More layers should mean that the model is better able to identify patterns within the data. The worsening results could be a result of the model overfitting the data, but that only makes sense of the predictions on the test data. It does not make sense of the training data predictions. Rather, these results appear to be an example of underfitting the data.

Ultimately, there does not appear to be a pattern in increasing layers, with a single neuron.

## 4(e) - Varied Neurons

```
##          CPUTime Layers Neurons train_RMSE  train_R2 train_MAE test_RMSE
## elapsed     0.54      1       1  0.8308023 0.7871742 0.6494376 1.0218865
## elapsed1    4.41      1       3  0.5971253 0.8900590 0.4198831 1.6247241
## elapsed2   79.34      1       4  0.2426091 0.9818519 0.1914585 0.4948335
## elapsed3   66.01      1       5  0.3638017 0.9591910 0.2624449 0.7090919
## elapsed4   52.42      1       6  0.2621706 0.9788071 0.1980213 1.2123396
## elapsed5   32.05      1       7  0.1612343 0.9919843 0.1216018 1.8286630
## elapsed6    3.61      1       8  0.2304500 0.9836251 0.1815201 1.3164815
## elapsed7    5.06      1       9  0.1649248 0.9916134 0.1259694 1.4317517
## elapsed8    2.58      1      10  0.2108954 0.9862861 0.1529243 1.0635380
##             test_R2  test_MAE
## elapsed  0.42467086 0.8720590
## elapsed1 0.39744972 0.9485760
## elapsed2 0.65425396 0.3620424
```

```
## elapsed3 0.33293900 0.6593755
## elapsed4 0.14038641 0.7690041
## elapsed5 0.35628070 1.0647542
## elapsed6 0.23273593 0.8306120
## elapsed7 0.02306045 0.6680696
## elapsed8 0.27984869 0.6769852
```

In this set of testing, a varying number of neurons have been trained on a single hidden layer. In this dataset, the set of data containing 2 neurons is missing because it would not train in the given number of steps. In this testing set, the R-squared value for the testing data predictions increases quite rapidly and is greater than 0.95 for most testing sets. However, the R-squared value on the testing data jumps all around the place, usually staying below 0.4. This is an excellent indication that the high number of neurons has overfitted the data.

Ultimately, the number of neurons and layers will depend on the complexity of the data it is intended to classify and its underlying patterns. It is difficult to assess patterns on a single set of data. However, generally, a larger number of hidden layers and neurons per layer leads to better results in complex data, while in simpler datasets, this can lead to overfitting.

## Comparison and Suggestions

### 5(a) - Cross-validated Accuracy

Cross-validated accuracy is a crucial step in machine learning that takes the training data and tests the models against a number of subsets of the data to assess the accuracy of the model. Cross-validation is done for several important reasons:

1) Cross-validation provides an estimate of the model's goodness of fit against unseen data.

2) Cross-validation allows the comparison of different hyperparameters to select the hyperparameters which perform the best.

3) Cross-validation allows for the assessment of overfitting and underfitting of the model.

4) Cross-validation allows for the assessment of model robustness by testing on multiple splits of the data. This ensures that the model's performance is consistent and not due to a specific split of the training data.

5) Cross-validation provides an estimate of the model’s overall accuracy. This helps assess whether the model is stable or not.

Repeated cross-validation was used in the hyperparameter tuning for the SVM model initially, allowing for the best cost and sigma parameters to be chosen. Cross-validation was not performed for the ANN model, which is a major limitation in trusting the model on unseen data, although its performance allows for a level of greater trust in the model for unseen data.

## 5(b) - Computational Time to Train Models

```
## Time to train SVM (seconds):  0.01

## Time to train ANN (seconds):  9.41

## Time to tune SVM Parameters (seconds):  255.75
```

One of the major considerations in choosing a machine learning model is the training time. The true training time for the model and the amount of time it takes to find the appropriate hyperparameters (tuning time). The tuning time is affected by multiple factors, including the number of observations, model type/complexity and validation measures (cross-validation, bootstrapping, etc.).

In this report, the time it took to train the SVM and ANN is negligible, 0.01 and 9.41 seconds, respectively. This is in large part due to the small number of observations in the training dataset (<150).

The big problem regarding computational time is in the hyperparameter tuning. For the SVM, this value is 255.75 seconds (~4 minutes), and no tuning was performed on the ANN model, but it is likely to be larger, given the same number of repeats.

1.96 hours (~117 minutes) respectively. These values can be adjusted by changing the hyperparameters to be tested and changing the number of cross-validations. In this report, proper hyperparameter tuning has not been performed on the ANN model, although it could be done in the same way as the SVM. Proper hyperparameter tuning of the ANN model, utilising cross-validation, would result in a longer time to tune.

## 5(c) - Interpretability

Neither the SVM nor ANN models are particularly interpretable. We can extract the hyperparameters (C and Sigma) used in the SVM and the defined number of Hidden Layers and Neurons used in the ANN. One of the drawbacks in this case for the SVM is the choice of the kernel; a linear kernel would make the result more interpretable, as the hyperplane is more easily described, but the radial kernel used here makes this difficult but not necessarily impossible. An ANN model is considered a black box in general; the process of backpropagation, used to adjust the weights of the neurons by assessing the predictions with the labelled data, is not easily understood.

## 6 - Suggestions for Prediction Improvement (Methodologies and Data)

### Methodology

One major improvement to the implemented methodologies would be to implement cross-validation of the ANN model, allowing better trusting of the ANN model on different data.

A second major improvement would be to increase the range of hyperparameters to be tested. This would allow better trust in the hyperparameters chosen to train the model.

Both suggested improvements will be significantly limited in implementation by computational constraints and the number of observations.

### Data

The major improvement to the data would be by gathering more observations and more predictors. Gathering more observations is self-explanatory. The more observations a model can be trained on, the more accurate the model can be. More predictors, however, have the side effect of introducing more noise into the data. The choice of predictors is ultimately up to domain experts. The 7 predictors in this dataset may be the most appropriate, or one should be swapped out for something more relevant. Ideally, the number of predictors is minimised, but the importance of the predictors is maximised. Whether or not this is the case in this data is unknown.

## References

C, P. (2021, November 17). Missing Values Be Gone. Medium. https://towardsdatascience.com/missing-values-be-gone-a135c31f87c1

Chugh, A. (2020, December 8). MAE, MSE, RMSE, Coefficient of Determination, Adjusted R Squared — Which Metric Is Better? Medium. https://medium.com/analytics-vidhya/mae-mse-rmse-coefficient-of-determination-adjusted-r-squared-which-metric-is-better-cd0326a5697e

https://twitter.com/cassiasamp. (2023, April 21). Understanding SVM Hyperparameters. Stack Abuse. https://stackabuse.com/understanding-svm-hyperparameters/

Machine learning - when to use random forest over SVM and vice versa? (n.d.). Data Science Stack Exchange. Retrieved October 5, 2023, from https://datascience.stackexchange.com/questions/6838/when-to-use-random-forest-over-svm-and-vice-versa

More, P. (2020, January 27). An Introduction to Missing Value Imputation in Univariate Time series. Medium. https://medium.com/@poojamore282/an-introduction-to-missing-value-imputation-in-univariate-time-series-7739a34e87e3

neuralnet: Train and Test Neural Networks Using R. (2019, September 19). DataScience+. https://datascienceplus.com/neuralnet-train-and-test-neural-networks-using-r/

Shulga, D. (2018, September 27). 5 Reasons Why You Should Use Cross-Validation in Your Data Science Projects. Medium. https://towardsdatascience.com/5-reasons-why-you-should-use-cross-validation-in-your-data-science-project-8163311a1e79

## Appendix 1 – Full Markdown

# MA5832 Capstone Assessment (Assessment 3) RMarkdown all

Samuel Bousfield

2023-10-15

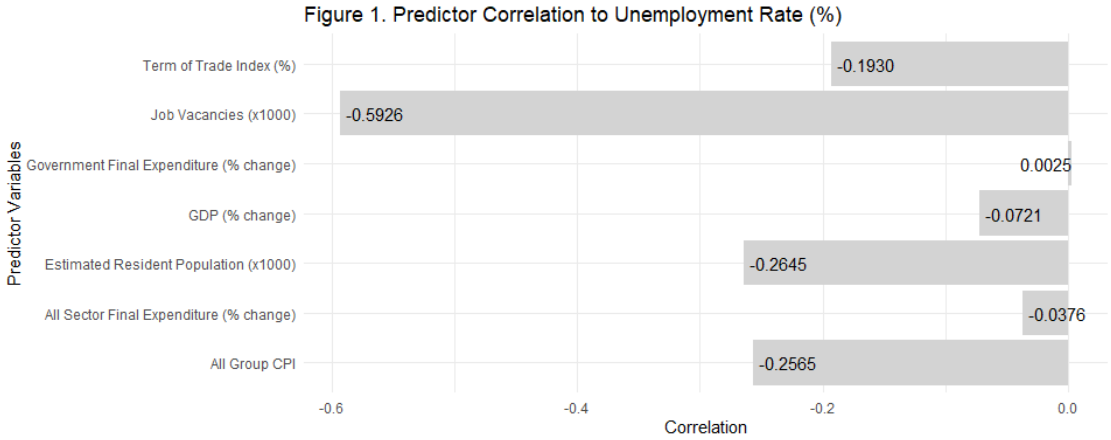
```
library(tidyverse)
library(imputeTS)
library(caret)
library(neuralnet)
library(e1071)
```

## Overview of Unemployment Rate

Unemployment is a critical problem within a society, leading to poverty, low living standards, pressure on welfare and other public services, and is proven to lead to worse health outcomes, physically and mentally. The goal of this report is to assess over 20 years of unemployment data, and seven socio-economic predictors, and apply a Support Vector Machine (SVM) model and an artificial neural network (ANN), to assess the suitability of using these predictors to estimate unemployment rate. The graph below shows the correlation between each predictor variable, and the Unemployment Rate (%). This graph shows that the Job Vacancies predictor is the best predictor of the unemployment rate, followed by Estimated Resident Population and All Group CPI. Government Final Expenditure appears to have almost no correlation whatsoever, this variable also has a positive correlation. The initial implications of this are that changing government expenditure does not really have an effect on the unemployment rate, although an increase in government expenditure is correlated with an extremely small increase in the unemployment rate.

```
unemployment <- AUS_Data[72:158,2:ncol(AUS_Data)]
for (i in 1:ncol(unemployment)){
 unemployment[[i]] <- as.numeric(unemployment[[i]])
}
unemployment_cor <- as.data.frame(cor(unemployment, use =
"pairwise.complete.obs"))
unemployment_cor <- unemployment_cor[1,]
unemployment_cor <- pivot_longer(unemployment_cor, cols = -`Unemployment Rate
(%)`, names_to = "variable", values_to = "correlation")

ggplot(unemployment_cor) +
 aes(x = variable, weight = correlation) +
 geom_bar(fill = "lightgrey") +
 labs(x = "Predictor Variables",
 y = "Correlation", title = "Figure 1. Predictor Correlation to Unemployment
```

```
Rate (%)") +
 geom_text(aes(label = sprintf("%.4f", correlation), y = correlation, hjust =
ifelse(variable == "Government Final Expenditure (% change)",1, -0.1)))+
 coord_flip() +
 theme_minimal()
```




# Data

## 2(a.1) - Preprocessing

Preprocessing included assessing the column types. A brief examination of the data showed that all the data is numeric, assessing the column types showed that two of the columns are of type character. This is caused by missing data. The predictor and response variables were then coerced to be of type numeric, which introduced NAs. there are 5 missing values in the Job Vacancies and Estimated Resident Population fields, which is approximately 3.2% of the data in each column, although these are at different row levels, therefore approximately 6.3% of the data has missing values. Three main options are available. 1) Since the observations with missing data accounts for less than 10% of the data, they could be dropped from the data set. This is not a good idea as the testing datasets for the models have been specifically defined, and this would drop ~50% of the testing data 2) The missing values could be left as is. 3) Interpolate or Impute values. These three options will be explored in the next. The summary of the data shows that the data covers the period from June 1981 to September 2020. The unemployment rate for this period is between 4.1% and 11.133%, with a mean employment rate of 6.852%. GDP change ranges includes negative and positive values, ranging from -7% change to 3.3% change. Government and All Sector Expenditure % change also covers negative and positive values with a minimum of -4.6% and -8.3% change and a maximum of 7.5% and 5.9% change respectively, likewise the Term of Trade Index % covers a range of -8.1 to 5.9%. These values indicate there is likely to be interactions within the data between these predictors and the response variable.

```
#, eval=FALSE, include=FALSE}
for (i in 1:ncol(AUS_Data)){
```

```
  print(typeof(AUS_Data[[i]]))
}
```

```
## [1] "double"
## [1] "double"
## [1] "double"
## [1] "double"
## [1] "double"
## [1] "double"
## [1] "double"
## [1] "character"
## [1] "character"
```

```
#All variables should be numeric (except for date), must convert from
character
#create new data set to avoid messing with old dataset
data <- AUS_Data
for (i in 2:ncol(data)){
 data[[i]] <- as.numeric(data[[i]])
}
#warning that NA's were introduced in conversion process in previous loop.
#check to see count of and % of NA's in each column
colSums(is.na(data))
```

```
##                                   Period                   Unemployment
Rate (%)
##                                        0
0
##                           GDP (% change) Government Final Expenditure (%
change)
##                                        0
0
## All Sector Final Expenditure (% change)                 Term of Trade
Index (%)
##                                        0
0
##                            All Group CPI                   Job Vacancies
(x1000)
##                                        0
5
##   Estimated Resident Population (x1000)
##                                        5
```

```
colSums(is.na(data))/nrow(data) * 100
```

```
##                                     Period                   Unemployment
Rate (%)
##                                   0.000000
0.000000
##                           GDP (% change) Government Final Expenditure (%
change)
```

```
##                                 0.000000
0.000000
## All Sector Final Expenditure (% change)                Term of Trade
Index (%)
##                                 0.000000
0.000000
##                           All Group CPI                  Job Vacancies
(x1000)
##                                 0.000000
3.164557
##   Estimated Resident Population (x1000)
##                                 3.164557
```

```
#5 errors present in 2 of the columns (~3.16%). check to see which rows these
are in
which(is.na(data), arr.ind=TRUE)
```

```
##       row col
##  [1,] 110   8
##  [2,] 111   8
##  [3,] 112   8
##  [4,] 113   8
##  [5,] 114   8
##  [6,] 154   9
##  [7,] 155   9
##  [8,] 156   9
##  [9,] 157   9
## [10,] 158   9
```

```
(10 / nrow(data)) * 100
```

```
## [1] 6.329114
```

```
#10 total rows are missing data, this is approximately 6.3% of the data
summary(data)
```

```
##      Period                    Unemployment Rate (%) GDP (% change)
##  Min.   :1981-06-01 00:00:00   Min.   : 4.100        Min.   :-7.0000
##  1st Qu.:1991-03-24 00:00:00   1st Qu.: 5.467        1st Qu.: 0.4000
##  Median :2001-01-15 00:00:00   Median : 6.250        Median : 0.7000
##  Mean   :2001-01-15 00:00:00   Mean   : 6.852        Mean   : 0.7247
##  3rd Qu.:2010-11-08 06:00:00   3rd Qu.: 8.250        3rd Qu.: 1.1000
##  Max.   :2020-09-01 00:00:00   Max.   :11.133        Max.   : 3.3000
##
##  Government Final Expenditure (% change)
##  Min.   :-4.6000
##  1st Qu.: 0.1000
##  Median : 1.0000
##  Mean   : 0.8532
##  3rd Qu.: 1.6000
##  Max.   : 7.5000
```

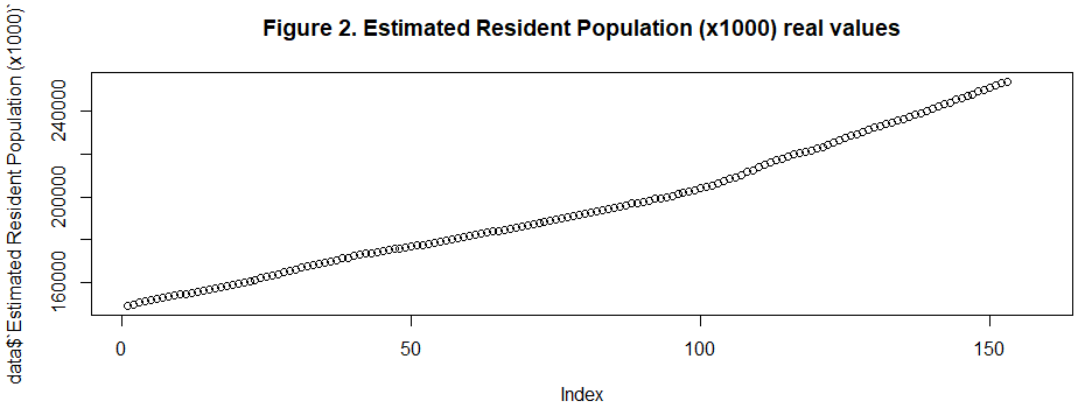
```
##
##  All Sector Final Expenditure (% change) Term of Trade Index (%)
##  Min.   :-8.300                          Min.   :-8.1000
##  1st Qu.: 0.400                          1st Qu.:-1.1000
##  Median : 0.800                          Median : 0.3000
##  Mean   : 0.762                          Mean   : 0.3456
##  3rd Qu.: 1.200                          3rd Qu.: 1.6500
##  Max.   : 5.900                          Max.   :13.2000
##
##  All Group CPI    Job Vacancies (x1000) Estimated Resident Population
(x1000)
##  Min.   : 28.40   Min.   : 26.8         Min.   :149233
##  1st Qu.: 59.00   1st Qu.: 66.9         1st Qu.:171698
##  Median : 73.50   Median : 99.7         Median :190288
##  Mean   : 75.08   Mean   :111.4         Mean   :194851
##  3rd Qu.: 96.80   3rd Qu.:160.8         3rd Qu.:218656
##  Max.   :116.60   Max.   :232.3         Max.   :253643
##                   NA's   :5             NA's   :5
```

```
data <- AUS_Data
for (i in 2:ncol(data)){
 data[[i]] <- as.numeric(data[[i]])
}
```

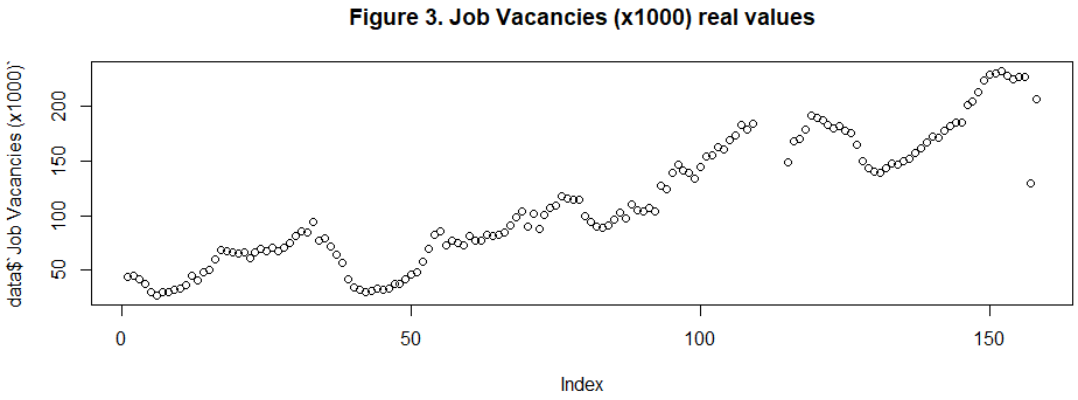
## 2(a.2) - Missing Value Imputation

There are two variables with missing values: Estimated Resident Population (x1000), and Job Vacancies (x1000). Examining a basic plot of the two variables, the population variable appears to be increasing relatively steady. One method of missing value imputation is to find an external source of data, which should be easy in this case, population statistics are usually pretty easy to find, however, this is inappropriate in two ways for this data, 1) this dataset is treating the population of Australia as x10 greater than the real variables 2) even though this is fairly obviously the case, because this is not the real data, it cannot be properly inferred that we can take the real population statistics and multiply by 10. This is also the case for the Job Vacancies variable.

```
plot(data$`Estimated Resident Population (x1000)`, main = "Figure 2.
Estimated Resident Population (x1000) real values")
```




```
plot(data$`Job Vacancies (x1000)`, main = "Figure 3. Job Vacancies (x1000)
real values")
```




Examining the plots above for the variables with missing values, we can see that the Estimate Resident Population appears to be increasing in a linear fashion, and while the Job Vacancies variable is not linear, it also appears to be following a pattern, likely a cubic function. Thus, some of the normal methods for dealing with missing data are likely to be inappropriate (knn, mean imputation). The best static value imputation for these is likely to be the previous filled value (or max value) in the case of the resident population, and either that or the mean value of the previous filled value and following filled value for the Job Vacancies data. However the most appropriate option for both is likely to interpolate. A few methods will be assessed for appropriateness, using the na_interpolation() function from imputeTS package, and in generating a simple linear or polynomial model. These will then be examined for what appears to fit the data best, and then the values imputed into the original data.

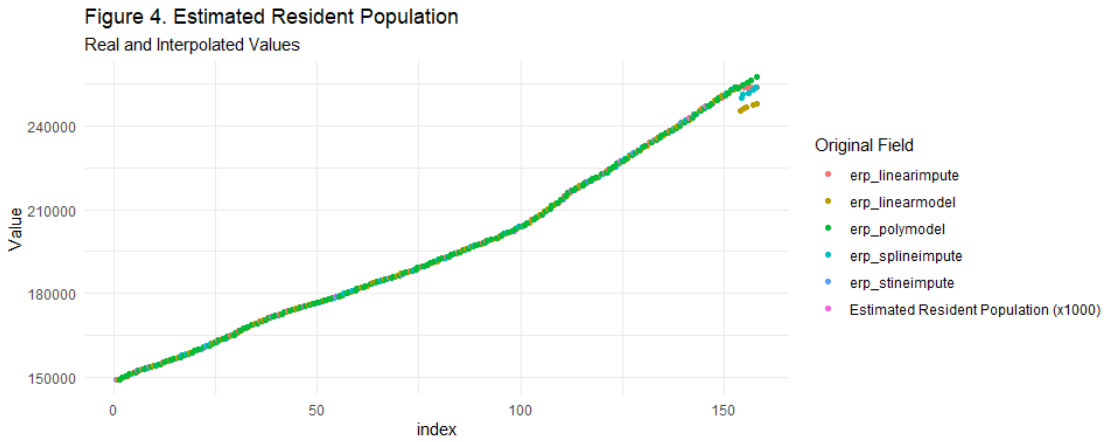
```
data_1 <- data[,c("Job Vacancies (x1000)", "Estimated Resident Population
(x1000)")]
data_1$index <- 1:nrow(data_1)
data_1$job_stineimpute <- na_interpolation(data_1$`Job Vacancies (x1000)`,
option = "stine")
data_1$job_linearimpute <- na_interpolation(data_1$`Job Vacancies (x1000)`,
```

```
option = "linear")
data_1$job_splineimpute <- na_interpolation(data_1$`Job Vacancies (x1000)`,
option = "spline")

data_1$erp_stineimpute <- na_interpolation(data_1$`Estimated Resident
Population (x1000)`, option = "stine")
data_1$erp_linearimpute <- na_interpolation(data_1$`Estimated Resident
Population (x1000)`, option = "linear")
data_1$erp_splineimpute <- na_interpolation(data_1$`Estimated Resident
Population (x1000)`, option = "spline")

linear_model <- lm(data$`Estimated Resident Population (x1000)`~ index,
data=data_1)
predicted_values_linear <- predict(linear_model, newdata = data_1)
data_1$erp_linearmodel <- data_1$`Estimated Resident Population (x1000)`
data_1$erp_linearmodel[is.na(data_1$erp_linearmodel)] <-
predicted_values_linear[is.na(data_1$erp_linearmodel)]

poly_model <- lm(data$`Estimated Resident Population (x1000)`~ poly(index,
2), data=data_1)
predicted_values <- predict(poly_model, newdata = data_1)
data_1$erp_polymodel <- data_1$`Estimated Resident Population (x1000)`
data_1$erp_polymodel[is.na(data_1$erp_polymodel)] <-
predicted_values[is.na(data_1$erp_polymodel)]

#pivot longer
data_long_job <- pivot_longer(
  data_1,
  cols = c(`Job Vacancies (x1000)`, job_stineimpute, job_linearimpute,
job_splineimpute),
  names_to = "Original Field",
  values_to = "Value"
    )
data_long_job <- data_long_job[, c("index", "Original Field", "Value")]

data_long_erp <- pivot_longer(
  data_1,
  cols = c(`Estimated Resident Population (x1000)`, erp_stineimpute,
erp_linearimpute, erp_splineimpute, erp_linearmodel, erp_polymodel),
  names_to = "Original Field",
  values_to = "Value"
    )
data_long_erp <- data_long_erp[, c("index", "Original Field", "Value")]
```

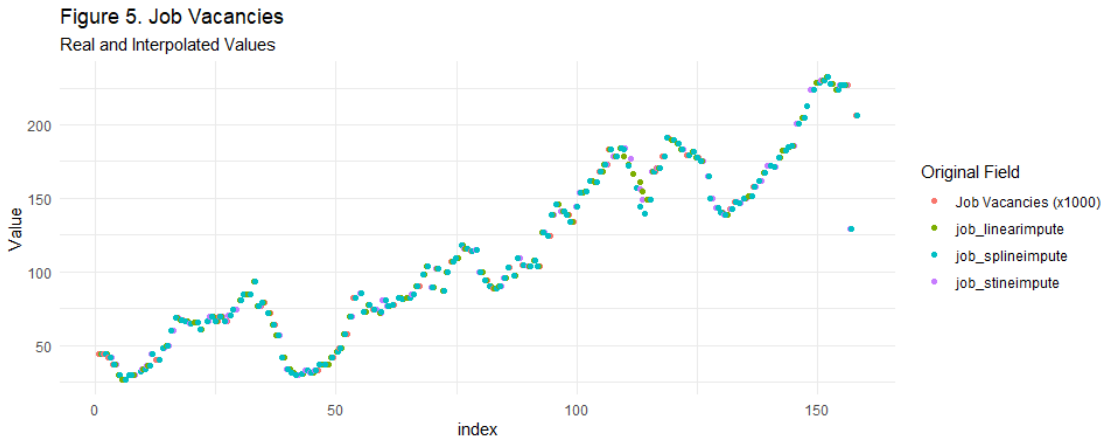
The methods have been completed, these will be assessed visually using the ggplot package. The geom_jitter option has been used so that indiviudal points can be seen more clearly. This offset should be taken into account when assessing the best fitt for the data.

```
ggplot(data_long_erp) +
 aes(x = index, y = Value, colour = `Original Field`) +
```

```
 geom_jitter(size = 1.5) +
 scale_color_hue(direction = 1) +
 labs(title = "Figure 4. Estimated Resident Population",
      subtitle = "Real and Interpolated Values") +
 theme_minimal()
```




```
ggplot(data_long_job) +
 aes(x = index, y = Value, colour = `Original Field`) +
 geom_jitter(shape = "circle",
 size = 1.5) +
 scale_color_hue(direction = 1) +
 labs(title = "Figure 5. Job Vacancies",
      subtitle = "Real and Interpolated Values") +
 theme_minimal()
```




We can see that in the Estimated Resident Population, the slight curve makes a significant difference, resulting in the linear model not fitting the data well, and the other imputation methods just take the max value in the field. However, the polynomial model fits the data very well.

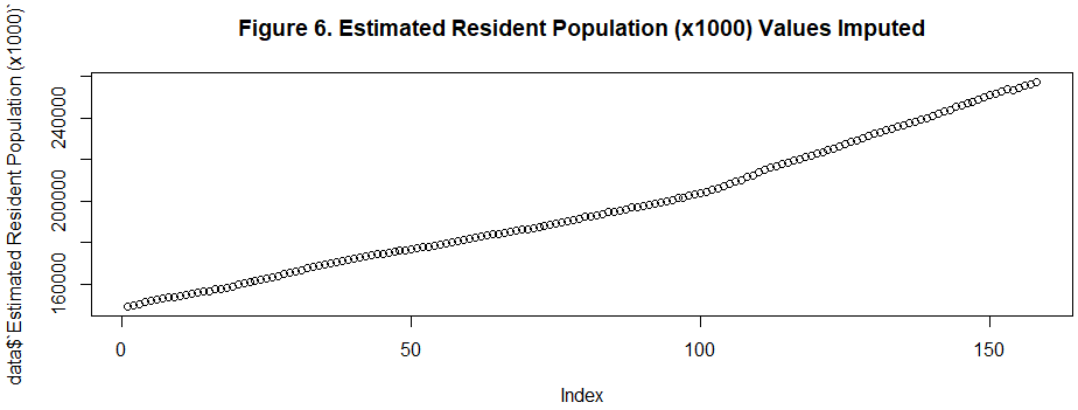
In the Job Vacancies data, the linear and Stine imputation both appear to work almost equally well, while the spline imputation takes the lowest value beyond the next

observation with a value. While this may be correct in the real data, it is better to assume a value between the two values with data. In this case, the stine imputation option has been chosen, although the linear option provides a similar level of appropriateness.

```
#impute
data$`Job Vacancies (x1000)` <- data_1$job_stineimpute
data$`Estimated Resident Population (x1000)` <- data_1$erp_polymodel
#check to ensure no more NA's, and plots look good
colSums(is.na(data))
```

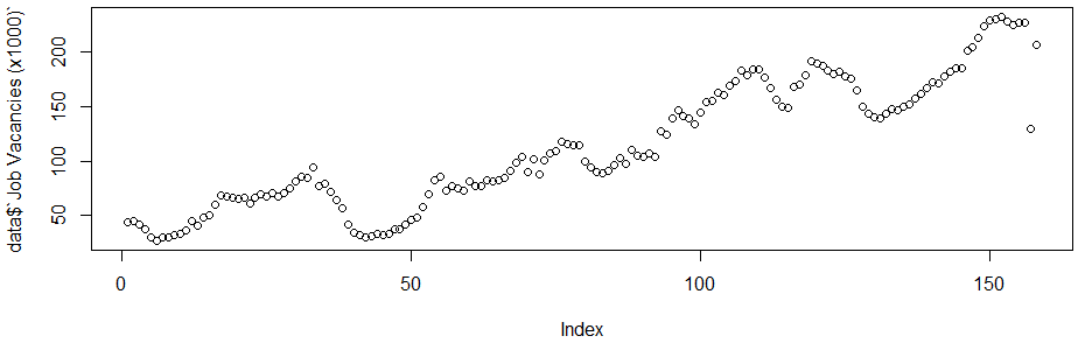
```
##                                     Period                         Unemployment
Rate (%)
##                                          0
0
##                               GDP (% change) Government Final Expenditure (%
change)
##                                          0
0
## All Sector Final Expenditure (% change)                      Term of Trade
Index (%)
##                                          0
0
##                                All Group CPI                         Job Vacancies
(x1000)
##                                          0
0
##   Estimated Resident Population (x1000)
##                                          0
```

```
plot(data$`Estimated Resident Population (x1000)`, main = "Figure 6.
Estimated Resident Population (x1000) Values Imputed")
```




```
plot(data$`Job Vacancies (x1000)`, main = "Figure 7. Job Vacancies (x1000)
Values Imputed")
```

**Figure 7. Job Vacancies (x1000) Values Imputed**




```
#drop unnecessary things from environment
rm(data_1, data_long_erp, data_long_job, linear_model, poly_model)
```

### 2(b) - Descriptive Statistics

## Machine Learning

Because of the choice of ML algorithm, and the comparison to a neural network, it is good practice to scale the data for a few reasons: . We will also be removing the "Period" variable, as this is more of an identifier, rather than

Testing data is specified to be data from March 2018 - September 2020, therefore Training data is data up to 2017

```
#scale training data
data_scaled <- data.frame(scale(data[, 3:9]))
#rebind the response variable, and rename the columns
data_scaled <- cbind(data$Period ,data$`Unemployment Rate (%)`, data_scaled)
colnames(data_scaled) <- c("x0", "Y", "X1", "X2", "X3", "X4", "X5", "X6",
"X7")

#create data partition, and split into training and testing data
training_data <- data_scaled[1:147,]
testing_data <- data_scaled[148:nrow(data),]
model_formula <- Y ~ X1 + X2 + X3 + X4 + X5 + X6 + X7
```

### 3(a) - Justify Choice

SVM has been chosen because this is a regression model, and all the variables are numeric. This means that there is likely a complex relationship between all 7 of the predictor variables, and the response variable. This is an area in which an SVM with a non-linear kernel shines over Forests. Another major reason for choosing an SVM model over a Forest is the sample size. SVMS perform well with small and medium sized datasets, while Forest's may overfit.

## 3(b) - Hyperparameters

Because of the small dataset, the Cost and Sigma values can be tuned in a reasonable amount of time. The C (cost) hyperparameter represents the trade-off between training error and testing error. A higher value here encourages the SVM to classify more training points accurately, resulting in a smaller margin, but has the potential trade off of overfitting. The Sigma hyper parameter influences the boundary. A smaller sigma is sensitive to local patterns, while a larger sigma is more sensitive to global patterns, this may lead to overfitting if it is too small, or underfitting if it is too large.

Using the cross-validation method, the hyperparameters will be tuned, the number of cross-validations and repeats can be relatively large, due to the small number of observations. The optimal hyperparameters will then be taken and used to train another model, because of the high number of cross-validations and repeats, these optimal hyperparameters can be trusted to be the optimal combination of hyperparameters in this case.

```
#C_values and Sigma
# tuning_time_svm <- system.time({
# c_values <- c(0.1, 0.5, 1, 2.5, 5, 10)
# Sigma_values <- c(0.1, 0.5, 1, 2.5, 5, 10)
# grid <- expand.grid(sigma = Sigma_values, C = c_values)
# ctrl_method <- trainControl(method = "repeatedcv", number = 100, repeats =
10)
# set.seed(0)
# hyperparameter_tuning_svm <- train(model_formula
#                         , data = training_data
#                         , method = "svmRadial"
#                         , trControl = ctrl_method
#                         , tuneGrid = grid)
# })
#
# saveRDS(hyperparameter_tuning_svm, file = "hyperparameter_tuning_svm.rds")
# saveRDS(tuning_time_svm, file = "tuning_time_svm.rds")
#commented out to prevent rerunning
```

The hyperparameter tuning has been saved to RDS to be reloaded, this is due to training time. The optimal hyperparameters will then be extracted from the tuning model, and used to train the final model. Which will also be saved, due to training time.

```
hyperparameter_tuning_svm <- readRDS(file = "hyperparameter_tuning_svm.rds")
#hyperparameter_tuning_svm
best_sigma <- hyperparameter_tuning_svm$bestTune$sigma
best_cost <- hyperparameter_tuning_svm$bestTune$C
time_svm_tuning <- readRDS(file = "tuning_time_svm.rds")
#tuning_time_svm
# set.seed(0)
# time_svm <- system.time({svm_model <- svm(model_formula,
#                                           data = training_data,
```

```
#                                                   kernel = "radial",
#                                                   sigma = best_sigma,
#                                                   cost = best_cost)
#                         })
# saveRDS(time_svm, file = "time_svm.rds")
# saveRDS(svm_model, file = "svm_model.rds")
svm_model <- readRDS(file = "svm_model.rds")
time_svm <- readRDS(file = "time_svm.rds")

#After multiple runs through, of using the SAME lines of code to get the
regression results, I decided to turn it into a function #DRY
predict_funct <- function(model, estimate_results, real_results){
    x <- predict(model, newdata = estimate_results)
    y <- real_results
    postResample(x, y)
  }
```

### 3(c) - Predictive Performance on Training Data

```
#load in best hyperparameters
# Original Lines of Code
# svm_predict_train <- predict(hyperparameter_tuning_svm, newdata =
training_data)
# #assess the svm model accuracy on the training data using the
postResample() function in caret
# svm_accuracy_train <- postResample(svm_predict_train,
training_data$`Unemployment Rate (%)`)
# svm_accuracy_train
# Use my function
print("SVM Model on Training Data")

## [1] "SVM Model on Training Data"

predict_funct(svm_model, training_data, training_data$Y)

##      RMSE  Rsquared       MAE
## 0.5039870 0.9226239 0.3141412
```

These values are very good, it is difficult to evaluate the Root Mean Square Error (RMSE) in a vacuum, unless it is used to compare to other models, so the particular value will not be examined too deeply, but a lower RMSE indicates a model that fits the data better. The Rsquared value 0.92263239 is an exceptionally good value. The Rsquared value indicates the amount of variance in the data/response variable that is explained by the model (0 = 0%; 1 = 100%). Thus this value indicates that ~92% of the variance in the data is explained by the model, which indicates a model of exceptional fit. The Mean Absolute Error (MAE), like the RMSE, is an average of the errors and a lower value indicates a better model; however, because the MAE value is the absolute values, the result is in the original units of the predictor variable. This means that the average difference between what the model predicts the Unemployment Rate (%) to be, and the actual Unemployment Rate (%) is 0.3141412%. This could be considered a very good value, the range of real results is between 4.1% and 11.133%, with a mean of 6.852% and a median of 6.250%. This

indicates that the SVM model varies from the average of the real results by ~4.5%. It is important to note that these predictions are on the training data, where the model would be expected to perform well, although a poor result here is a good indicator of model unsuitability. Next the SVM model will be assessed on the training data.

### 3(d) - Predictive Performance on Testing Data

```
print("SVM Model on Testing Data")
```

```
## [1] "SVM Model on Testing Data"
```

```
predict_funct(svm_model, testing_data, testing_data$Y)
```

```
##      RMSE  Rsquared       MAE
## 0.3822914 0.9414006 0.3310902
```

The model has performed very well on the test data, interestingly the RMSE and Rsquared values indicate that the model is a better fit on the test data than the training data (Train RMSE = 0.5039870, Test RMSE =0.3822914; Train Rsquared = 0.9226239, Test Rsquared = 0.9414006), although the MAE is larger(0.3310902) as expected. These results are likely the function of the small testing data set.

These values indicate that while the model fits the data better, the incorrect values are slightly more incorrect. Although this only results in the error from the average of the real results in ~4.8%. Again it is critical to recognise that the testing data is an extremely small dataset (11 observations).

The results from both sets of predictions indicate that the svm model performs exceptionally well.

## Neural Network

### 4(a.1) - Describe the Structure of the Model

An Artificial Neural Network (ANN),

### 4(a.2) - Implement the model

Implementation of the initial ANN will be performed using 7 hidden layers of 1 neuron each. These numbers were chosen semi-randomly. 7 layers were chosen because the data has 7 predictor variables and 1 layer was chosen because one “variable” (date) is not a predictor or response variable. It is important to note that these values are “random”. This model could be trained with any number of hidden layers and any number of neurons, the only limit would be training time.

```
#commented out to prevent rerunning
# #2 hidden layers of 7 neurons in first hidden layer, 1 neuron in second
hidden layer
# initial_ann_training_time <- system.time({
```

```
#     set.seed(0)
#     initial_ann <- neuralnet(model_formula,
#                              data = training_data,
#                              hidden = c(7,1),
#                              stepmax = 1e7)
# })
# # saveRDS(initial_ann_training_time, file =
"initial_ann_training_time.rds")
# # saveRDS(initial_ann, file = "initial_ann.rds")

time_ann <- readRDS(file = "initial_ann_training_time.rds")
initial_ann <- readRDS(file = "initial_ann.rds")
```

### 4(b) - Predictive Performance on Training Data

```
print("Initial Neural Network on Training Data:")

## [1] "Initial Neural Network on Training Data:"

predict_funct(initial_ann, training_data, training_data$Y)

##      RMSE  Rsquared       MAE
## 0.2373404 0.9826311 0.1868678
```

These results are phenomenal. The Rsquared value indicates that the model correlates to ~98% of the variation within the training data. The MAE is also very small, which compared to the average result is ~2.7% variation. These values indicate that this model has been to extreme precision. One major concern with this high correlation is the possibility of overfitting. It will be important to assess the model on the training data.

### 4(c) - Predictive Performance on Testing Data

```
print("Initial Neural Network on Testing Data:")

## [1] "Initial Neural Network on Testing Data:"

predict_funct(initial_ann, testing_data, testing_data$Y)

##      RMSE  Rsquared       MAE
## 0.4547744 0.9612497 0.4220960
```

The ANN model on the testing data is also performing exceptionally well, as expected, the RMSE and MAE are higher than the training data, and the Rsquared value is lower. However, the model still correlates to ~96% of the variation within the testing data (compared to the ~98% in the training data). The MAE value, is more than twice the value of the MAE in the training data predictions (0.4220960 vs 0.1868678).

The results from both sets of predictions indicate that the ANN model, like the SVM model, performs exceptionally well. The ANN model appears to be performing better than the SVM, and certainly performs better on the current set of data.

## 4(d) - Varied Hidden Layers

```
# #multi_layer_df <- data.frame(CPUTime = numeric(0), Layers = integer(0),
Neurons = integer(0), train_RMSE = numeric(0), train_R2 = numeric(0),
train_MAE = numeric(0), test_RMSE = numeric(0), test_R2 = numeric(0),
test_MAE = numeric(0))
# for(i in 1:10){
#   hidden_layers <- rep(1,i)
#   time_to_train <- system.time({
#     set.seed(0)
#     ann <- neuralnet(model_formula,
#                      data = training_data,
#                      hidden = hidden_layers,
#                      stepmax = 1e7
#                      )
#     })
#   train_pred <- predict_funct(ann, training_data, training_data$Y)
#   test_pred <- predict_funct(ann, testing_data, testing_data$Y)
#   new_row <- data.frame(
#     CPUTime = time_to_train[3],
#     Layers = i,
#     Neurons = 1,
#     train_RMSE = train_pred[1],
#     train_R2 = train_pred[2],
#     train_MAE = train_pred[3],
#     test_RMSE = test_pred[1],
#     test_R2 = test_pred[2],
#     test_MAE = test_pred[3])
#   multi_layer_df <- rbind(multi_layer_df, new_row)
#
# }
# saveRDS(multi_layer_df, file = "multi_layer_df.rds")
multi_layer_df <- readRDS(file = "multi_layer_df.rds")
multi_layer_df
```

```
##          CPUTime Layers Neurons train_RMSE  train_R2 train_MAE test_RMSE
## elapsed     0.68      1       1  0.8308023 0.7871742 0.6494376 1.0218865
## elapsed1    0.58      2       1  0.8308154 0.7871675 0.6498082 1.0373238
## elapsed2    4.19      3       1  1.0494602 0.6604054 0.7173622 0.7556050
## elapsed3    5.76      4       1  1.0493704 0.6604635 0.7171117 0.7556664
## elapsed4    9.86      5       1  1.0493595 0.6604705 0.7169628 0.7557888
## elapsed5   15.94      6       1  1.0495715 0.6603334 0.7177880 0.7545955
## elapsed6   17.94      7       1  1.0491610 0.6605990 0.7165245 0.7557236
## elapsed7    0.02      8       1  1.8008803 0.6420708 1.5352277 1.5723640
## elapsed8    0.02      9       1  1.8008832 0.6369178 1.5352316 1.5723732
## elapsed9    0.03     10       1  1.8008831 0.6228856 1.5352425 1.5724263
##            test_R2  test_MAE
## elapsed  0.4246709 0.8720590
## elapsed1 0.4258969 0.8911329
## elapsed2 0.3960553 0.4337242
## elapsed3 0.3963245 0.4336768
```

```
## elapsed4 0.3967623 0.4335819
## elapsed5 0.3971423 0.4345101
## elapsed6 0.3974456 0.4336329
## elapsed7 0.4079535 1.4286804
## elapsed8 0.4039243 1.4286870
## elapsed9 0.4035932 1.4287247
```

In this set of testing, it appears that an increase in the number of layers, with a constant number of neurons (in this case 1), surprisingly the model gets worse on the the training data predictions, and stays relatively stable in the testing data predictions, with a slight dip. The model training time rises, with a sharp jump at the 6 and 7 layer models, before dropping again for the 8, 9 and 10 layer models. The initial expectation was that the model would get better with a larger number of hidden layers, but this does not appear to be the case. More layers should mean that the model is better able to identify patterns within the data. The worsening results could be a result of the model overfitting the data, but that only makes sense of the predictions on the test data, it does not make sense of the training data predictions. Ultimately there does not appear to be a pattern in an increasing number of layers, with a single neuron.

### 4(e) - Varied Neurons

```
# #multi_neuron_df <- data.frame(CPUTime = numeric(0), Layers = integer(0),
Neurons = integer(0), train_RMSE = numeric(0), train_R2 = numeric(0),
train_MAE = numeric(0), test_RMSE = numeric(0), test_R2 = numeric(0),
test_MAE = numeric(0))
# for(i in 1:10){
#   time_to_train <- system.time({
#     set.seed(0)
#     ann <- neuralnet(model_formula,
#                      data = training_data,
#                      hidden = c(i),
#                      stepmax = 1e7
#                      )
#     })
#   train_pred <- predict_funct(ann, training_data, training_data$Y)
#   test_pred <- predict_funct(ann, testing_data, testing_data$Y)
#   new_row <- data.frame(
#     CPUTime = time_to_train[3],
#     Layers = 1,
#     Neurons = i,
#     train_RMSE = train_pred[1],
#     train_R2 = train_pred[2],
#     train_MAE = train_pred[3],
#     test_RMSE = test_pred[1],
#     test_R2 = test_pred[2],
#     test_MAE = test_pred[3])
#   multi_neuron_df <- rbind(multi_neuron_df, new_row)
#   print(i)
# }
# saveRDS(multi_neuron_df, file = "multi_neuron_df.rds")
```

```
multi_neuron_df <- readRDS(file = "multi_neuron_df.rds")
multi_neuron_df
```

```
##          CPUTime Layers Neurons train_RMSE  train_R2 train_MAE test_RMSE
## elapsed     0.54      1       1  0.8308023 0.7871742 0.6494376 1.0218865
## elapsed1    4.41      1       3  0.5971253 0.8900590 0.4198831 1.6247241
## elapsed2   79.34      1       4  0.2426091 0.9818519 0.1914585 0.4948335
## elapsed3   66.01      1       5  0.3638017 0.9591910 0.2624449 0.7090919
## elapsed4   52.42      1       6  0.2621706 0.9788071 0.1980213 1.2123396
## elapsed5   32.05      1       7  0.1612343 0.9919843 0.1216018 1.8286630
## elapsed6    3.61      1       8  0.2304500 0.9836251 0.1815201 1.3164815
## elapsed7    5.06      1       9  0.1649248 0.9916134 0.1259694 1.4317517
## elapsed8    2.58      1      10  0.2108954 0.9862861 0.1529243 1.0635380
##             test_R2  test_MAE
## elapsed  0.42467086 0.8720590
## elapsed1 0.39744972 0.9485760
## elapsed2 0.65425396 0.3620424
## elapsed3 0.33293900 0.6593755
## elapsed4 0.14038641 0.7690041
## elapsed5 0.35628070 1.0647542
## elapsed6 0.23273593 0.8306120
## elapsed7 0.02306045 0.6680696
## elapsed8 0.27984869 0.6769852
```

In this set of testing, a varying number of neurons has been trained on a single hidden layer. In this dataset the set of data containing 2 neurons is missing, because it would not train in the given number of steps. In this set of testing, the Rsquared value for the testing data predictions increases quite rapidly, and staying above 0.95 for most sets of testing. However the Rsquared value on the testing data jumps all around the place, usually staying below 0.4. This is an excellent indication that the high number of neurons have overfit the data.

Ultimately, the number of neurons and layers will depend on the complexity of the data it is intended to classify, and its underlying patterns. It is difficult to make assess patterns on a single set of data. However generally, a larger number of hidden layers and neurons per layer leads to better results in complex data, while in simpler datasets this can lead to overfitting.

## Comparison and Suggestions

### 5(a) - Cross-validated Accuracy

Cross-validated accuracy is a crucial step in machine learning, that takes the training data and tests the models against a number of subsets of the data to assess the accuracy of the model. Cross validation is done for several important reasons: 1) Cross validation provides an estimate of the goodness of fit of the model against unseen data 2) Cross validation allows the comparison of different hyperparameters to select the hyperparameters which perform the best 3) Cross validation allows for the assessment of overfitting and

underfitting of the model 4) Cross validation allows for the assessment of model robustness, by testing on multiple splits of the data, this ensures that the models performance is consistent, and not due to a specific split of the training data 5) Cross validation provides a an estimate of the model's overall accuracy. This helps assess whether the model is stable or not.

Repeated cross validation was used in the hyperparameter tuning for the SVM model initially, allowing for the best cost and sigma parameters to be chosen. Cross validation was not performed for the ANN model, which is a major limitation in trusting the model on unseen data, although its performance on the testing data is somewhat of a mitigating factor

## 5(b) - Computational Time to Train Models

```
cat("Time to train SVM (seconds): ", time_svm[3], "\n")
```

```
## Time to train SVM (seconds):  0.01
```

```
cat("Time to train ANN (seconds): ", time_ann[3], "\n")
```

```
## Time to train ANN (seconds):  9.41
```

```
cat("Time to tune SVM Parameters (seconds): ", time_svm_tuning[3], "\n")
```

```
## Time to tune SVM Parameters (seconds):  255.75
```

One of the major considerations in choosing a machine learning model is the training time. Not only the true training time for the model, but the amount of time it takes to find the appropriate hyperparameters (tuning time). The tuning time is affected by multiple factors, including the number of observations, model type/complexity and validation measures (cross validation, bootstrapping etc.).

In this report, the time it took to train the SVM and ANN are negligible, 0.01 and 9.41 seconds respectively. This is in large part due to the small number of observations in the training dataset (<150).

The big problem, regarding computational time, is in the hyperparameter tuning. For the SVM this value is 255.75 seconds (~4 minutes) and no tuning was performed on the ANN model, but it is likely to be larger, with the same number of repeats

1.96 hours (~117 minutes) respectively. These values can be adjusted by changing the hyperparameters to be tested, and changing the number of cross-validations. In this report proper hyperparameter tuning has not been performed on the ANN model, although it could be done in the same way as the SVM. Proper hyperparameter tuning of the ANN model, utilising cross validation, would result in a longer time to tune.

## 5(c) - Interpretability

Neither the SVM nor ANN models are particularly interpretable. We can extract the hyperparameters (C and Sigma) used in the SVM, and the defined number of Hidden Layers and Neurons used in the ANN. One of the drawbacks in this case for the SVM is the choice of

kernel, a linear kernel would make the result more interpretable, as the hyperplane is more easily described, but the radial kernel used here makes this difficult, but not necessarily impossible. An ANN model is considered a black box in general, the process of back propagation, used to adjust the weights of the neurons, by assessing the predictions with the labelled data, is not easily understood.

## 6 - Suggestions for Prediction Improvement (Methodologies and Data)

### Methodology

One major improvement to the implemented methodologies would be to implement cross-validation of the ANN model, allowing better trusting of the ANN model on different data.

A second major improvement would be to increase the range of hyperparameters to be tested. This would allow better trusting of the hyperparameters chosen to train the model on.

Both of these suggested improvements will be significantly limited in implementation, by computational constraints, and number of observations.

### Data

The major improvement to the data would be by gathering more observations, and more predictors. Gathering more observations is self-explanatory, the more observations a model can be trained on, the more accurate the model can be. More predictors however, has the side effect of introducing more noise into the data. The choice of predictors is ultimately up to domain experts. The 7 predictors in this dataset may be the most appropriate predictors to use, or maybe one should be swapped out for something more relevant. Ideally, the number of predictors is minimised, but the importance of the predictors is maximized. Whether or not this is the case in this data is unknown.

## References

C, P. (2021, November 17). Missing Values Be Gone. Medium. https://towardsdatascience.com/missing-values-be-gone-a135c31f87c1

Chugh, A. (2020, December 8). MAE, MSE, RMSE, Coefficient of Determination, Adjusted R Squared — Which Metric Is Better? Medium. https://medium.com/analytics-vidhya/mae-mse-rmse-coefficient-of-determination-adjusted-r-squared-which-metric-is-better-cd0326a5697e

https://twitter.com/cassiasamp. (2023, April 21). Understanding SVM Hyperparameters. Stack Abuse. https://stackabuse.com/understanding-svm-hyperparameters/

Machine learning - when to use random forest over SVM and vice versa? (n.d.). Data Science Stack Exchange. Retrieved October 5, 2023, from https://datascience.stackexchange.com/questions/6838/when-to-use-random-forest-over-svm-and-vice-versa

More, P. (2020, January 27). An Introduction to Missing Value Imputation in Univariate Time series. Medium. https://medium.com/@poojamore282/an-introduction-to-missing-value-imputation-in-univariate-time-series-7739a34e87e3

neuralnet: Train and Test Neural Networks Using R. (2019, September 19). DataScience+. https://datascienceplus.com/neuralnet-train-and-test-neural-networks-using-r/

Shulga, D. (2018, September 27). 5 Reasons Why You Should Use Cross-Validation in Your Data Science Projects. Medium. https://towardsdatascience.com/5-reasons-why-you-should-use-cross-validation-in-your-data-science-project-8163311a1e79

# Appendices

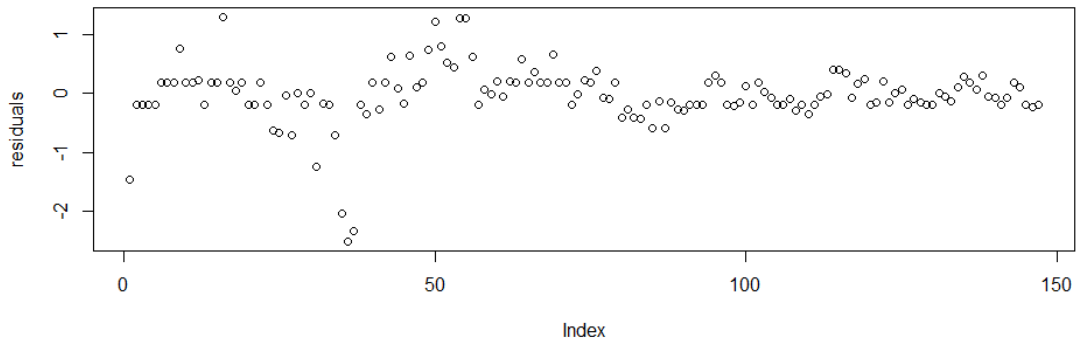
## Leave Out Of Real Report Due To Space

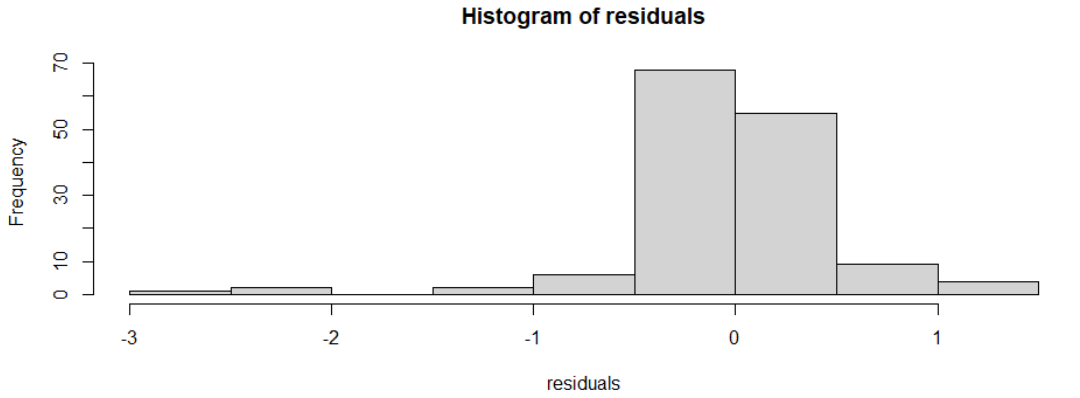
### Assess Residuals

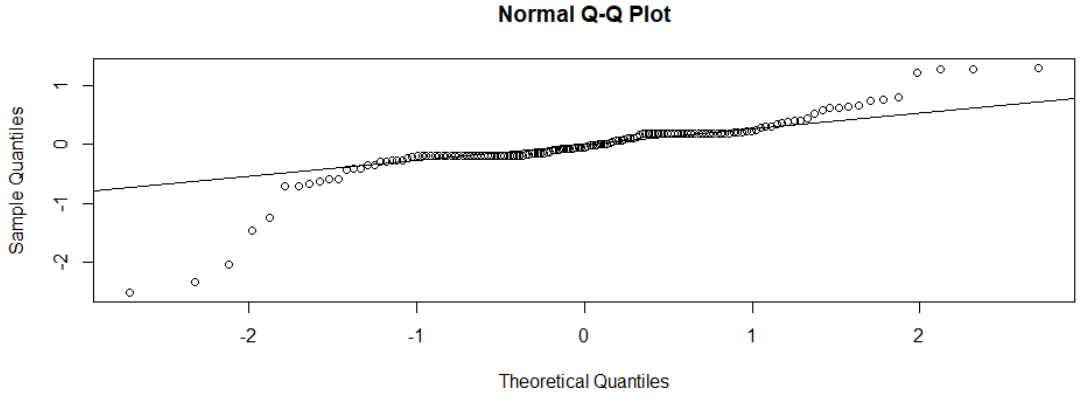
Assess Residuals There are three main assumptions for residuals in a regression model: 1) Normal Distribution - The residuals should be normally distributed 2) Independence - The residuals should be random 3) Constant Variance - The residuals should have constant variance

```
#resid() function in base R does not seem to work with ann model, therefore
manually find residuals so that results are comparable in case of
implementation difference in resid()
examine_residuals_funct <- function(model, data, target){
  x <- model
  y <- data
  z <- target
  #calculate predictions
  predictions <- predict(x, y)
  #calculate residuals
  residuals <- z - predictions
  #plots
  plot(residuals)
  hist(residuals)
  qqnorm(residuals)
  qqline(residuals)
  #statistics
  stats <- data.frame(matrix(ncol = 2, nrow = 0))
  colnames(stats) <- c("Stat", "Value")
  mean_row <- data.frame(Stat = "Mean", Value = mean(residuals))
  sd_row <- data.frame(Stat = "SD", Value = sd(residuals))
  skewness_row <- data.frame(Stat = "Skewness", Value = skewness(residuals))
  kurtosis_row <- data.frame(Stat = "Kurtosis", Value = kurtosis(residuals))
  stats <- rbind(stats, mean_row, sd_row, skewness_row, kurtosis_row)
```

```
  print(stats)
}

examine_residuals_funct(svm_model, training_data, training_data$Y)
```




```
##       Stat       Value
## 1     Mean -0.02404539
## 2       SD  0.50513418
## 3 Skewness -1.56930109
## 4 Kurtosis  7.79477867
```

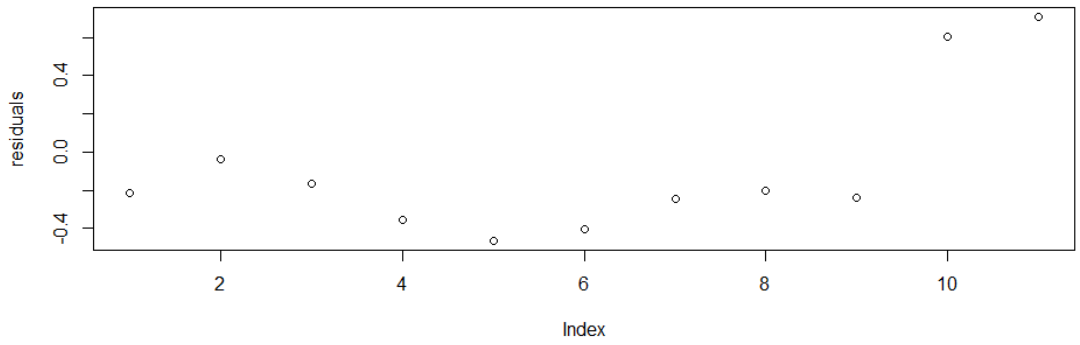
SVM Residuals (Training Data) 1) Normal Distribution Examination of the histogram show the residuals appear to be fairly normally distributed, with a couple of outliers. There might be a slight skew, although this appears negligible. The QQ plot supports this, with the majority of the data around the normal line, and a few obvious outliers below the -1 sample quantile and above the 1 sample quantile. The QQ plot also supports the idea of SOMETHING TO DO WITH KURTOSIS.

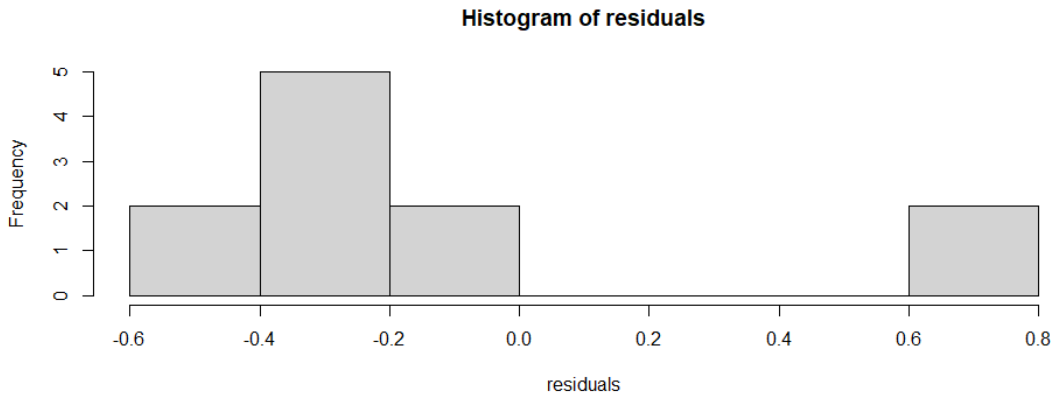
Comparing the results with the calculated skewness and kurtosis values
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

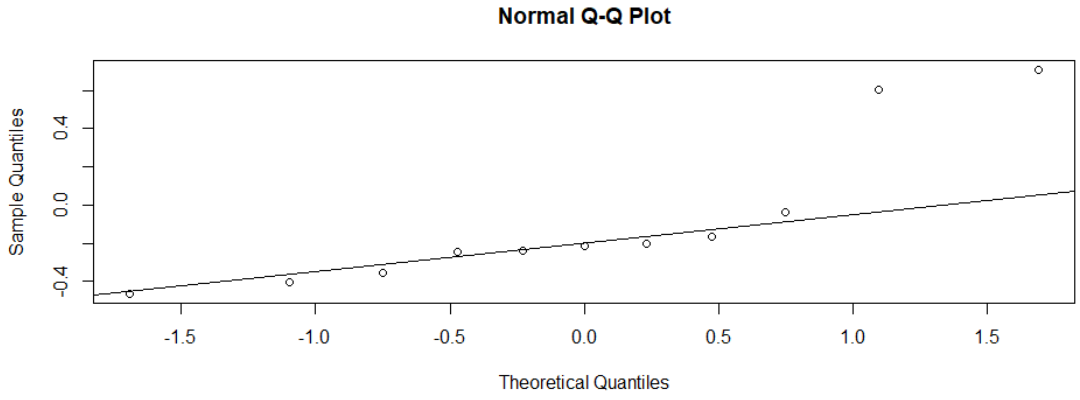
2) Independence Examining the residuals plot, there does not appear to be a relationship between consecutive residuals.
3) Constant Variance Examining the residuals plot, there does appear to be a difference in the variance, towards the beginning of the data, there are a few outliers that make the variance look like this assumption is violated. However, these appear to be outliers, and the majority of the residuals do have a clear constant variance.

The 3 assumptions appear to be met.

```
examine_residuals_funct(svm_model, testing_data, testing_data$Y)
```




```
##       Stat       Value
## 1     Mean -0.09207096
## 2       SD  0.38914862
## 3 Skewness  1.14647790
## 4 Kurtosis -0.29093524
```

SVM Residuals (Testing Data) Examining the training data must be done with extreme caution, due to the number of observations in this set being so low.

1) Normal Distribution Examination of the histogram show the residuals appear to be fairly normally distributed, with a couple of outliers. There does not appear to be any skew in the data. The QQ plot supports this, with the majority of the data around the normal line, and 2 obvious outliers towards the end. The QQ plot also supports the idea of SOMETHING TO DO WITH KURTOSIS.

Comparing the results with the calculated skewness and kurtosis values
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

2) Independence Examining the residuals plot, there does appear to be a relationship between consecutive residuals, but this is likely due to the small number of observations, and the human brains evolution to detect patterns.

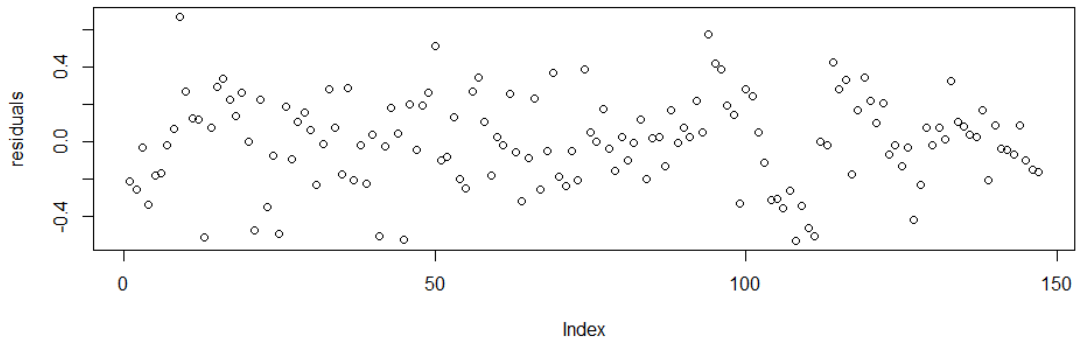
3) Constant Variance Examining the residuals plot, there does appear to be a difference in the variance, towards the beginning of the data, there are 2 obvious outliers. The scale of the residuals plot should also be noted, most of the data lies between 0.0 and -0.4, with the two outliers at 0.6.

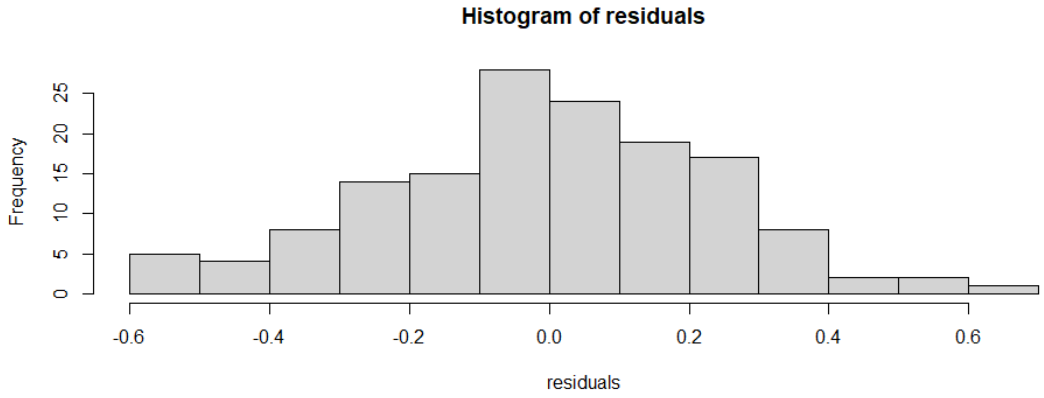
The 3 assumptions appear to be met on the testing data.

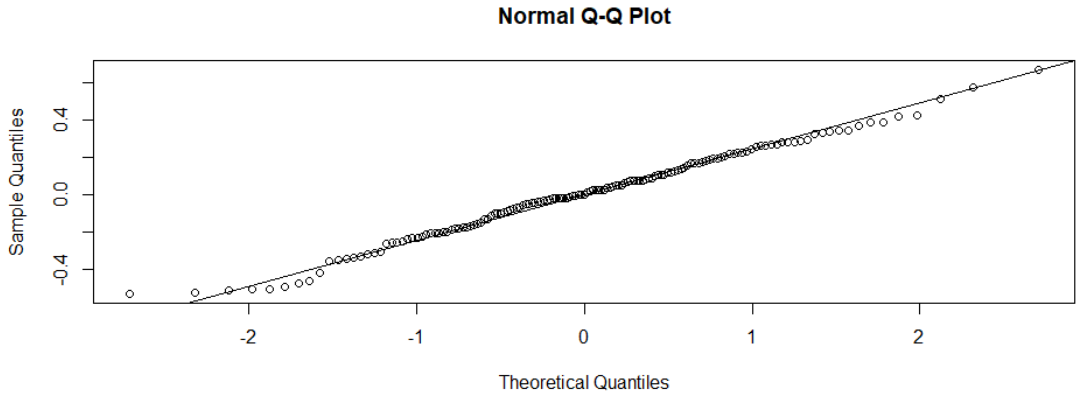
Ultimately this appears to be a good model, showing exceptional accuracy in both the training and testing data sets, with the residuals suppoting this.

Assess Residuals

```
examine_residuals_funct(initial_ann, training_data, training_data$Y)
```




```
##       Stat         Value
## 1     Mean  5.218267e-05
## 2       SD  2.381518e-01
## 3 Skewness -9.275469e-02
## 4 Kurtosis -1.209727e-01
```

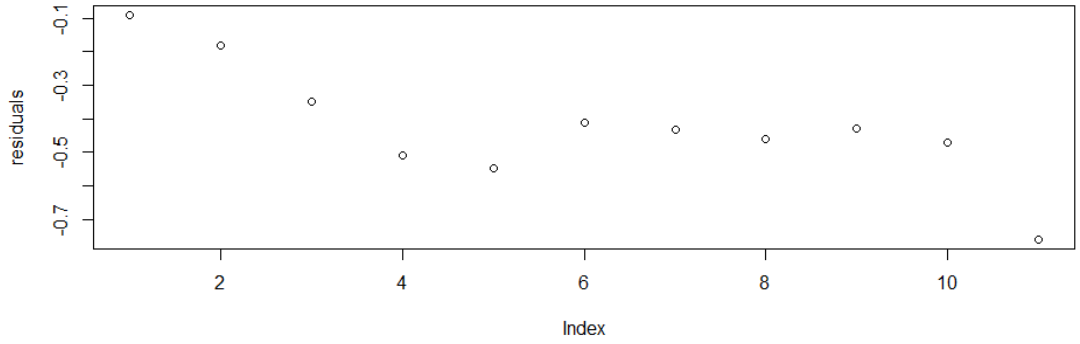
1) Normal Distribution Examining the histogram, the data appears to be normally distributed, perhaps a slight skew.

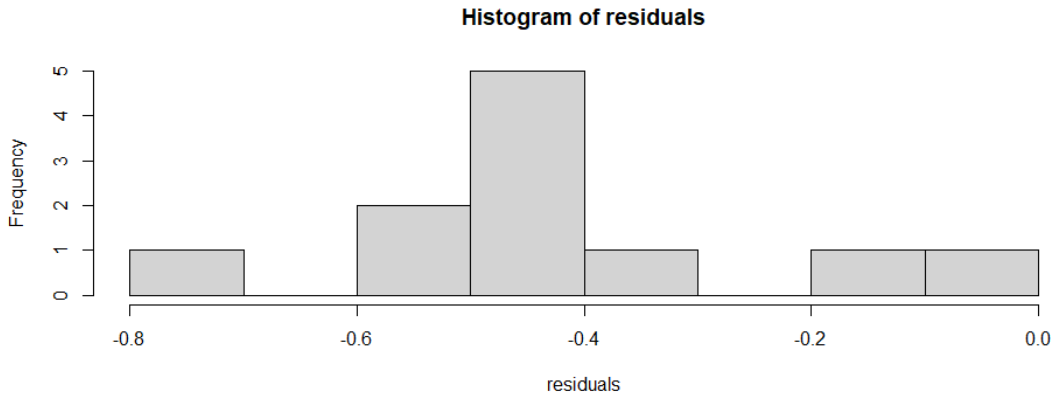
2) Independence Examining the residual plot, there does not appear to be a pattern in the residuals.

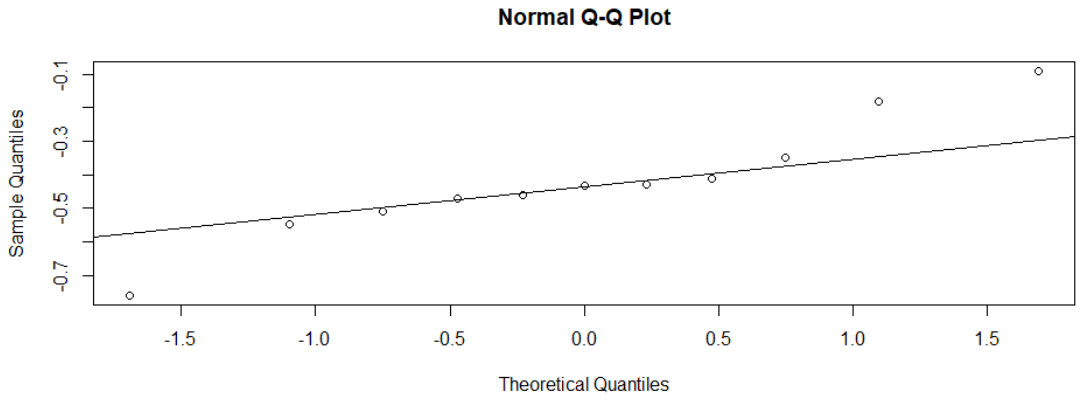
3) Constant Variance The residuals appear to have a very constant variance

The initial Artificial Neural Network very clearly meets all three residual assumptions.

```
examine_residuals_funct(initial_ann, testing_data, testing_data$Y)
```




```
##       Stat      Value
## 1     Mean -0.4220960
```

```
## 2       SD  0.1775391
## 3 Skewness  0.1560830
## 4 Kurtosis -0.3524118
```

As with the SVM model testing residuals, caution must be taken in assessing the residuals due to the small number of observations. 1) Normal Distribution Examining the histogram, the data appears to be normally distributed, perhaps a slight skew.

2) Independence Examining the residual plot, there does appear to be a pattern in the residuals.

3) Constant Variance The residuals appear to have a very constant variance, given the scale and observation count.

The initial also Artificial Neural Network appears to meet these three assumptions.