Comparing Predictions of Australian Unemployment Rate: Machine Learning vs. Neural Networks

ASSIGNMENT 3 - MA5832

Jarman Giffard [JC225731]

Built with R Version: **4.1.0**

ABSTRACT

Predicting the future macroeconomic features of a country is a prize sought by all governments, with most countries investing extensive resources into economic forecasting. Short and long-term forecasts on economic metrics such as GDP, CPI, and unemployment rate gives governments the ability to understand what drivers influence their economy, and provide opportunity to alter policy to resolve predicted problems before they arise. There are some metrics that are **particularly** crucial to an economy's health; Robert Shiller - a Nobel prize winner in Economic Science - is quoted as saying "The most important problem that we are facing now today, I think, is rising inequality" (*CHRISTOFFERSEN, J. 2013, October 14*).

One of the main drivers of inequality "is the ability to access well paid employment" (*Causes of Inequalities - Reasons Why Income and Wealth Inequality Exists - Higher Modern Studies Revision, n.d.*) - thus rising unemployment rates in a country is often a telling predictor of inequality in that country. Rising unemployment also results in higher mortality rates, health risks, and crime rates of a country (*Orecchio-Egresitz, 2020*)

The purpose of this paper is to strategically employ the use of two machine learning techniques in an effort to predict the unemployment rate in Australia, with the outputs of these two approaches then being compared for illustrative purposes. This is especially relevant today in light of the ongoing COVID-19 outbreaks in New South Wales and Victoria resulting in as much as a 50% rise in government assistance allowance for some demographics (*Rolfe, 2021*). This will be achieved through the use of the tree-based method *Random Forests*, as well as the more recent and increasingly-popular approach of artificial neural networks (aNN). The overarching goal for both approaches is to determine the performance of both methods in predicting the unemployment rate of Australia from March 2018 - December 2020 using historic data taken every 3 months from 1981 onwards - across a range of economic factors such as GDP, CPI, and population.

Being able to accurately predict components of a country such as the unemployment rate will better equip countries to act upon problems before they arise.

AUSTRALIAN UNEMPLOYMENT RATE

The unemployment rate in Australia saw historic lows prior to 2010 - with 4.1% in March 2008 - resulting from an overall decreasing trend following a high of 11.1% in December 1992.

```
ggplot(aus_data, aes(x=X, y=Y)) +
    geom_line() +
    labs(x = "YYYY-MM", y = "Unemployment Rate") +
    ggtitle("Australian Unemployment Rate: 1981 - 2020")
```

The above chart highlights the peaks and troughs recorded for Australia's unemployment rate over the past 40 years. Notable peaks in the first half of the chart highlight two recessions in 1982 and 1990-1991. The 1982-1983 recession was heavily influenced by post-war stagflation - where the inflation rate is high and the economic growth is slow. (*Department of the Parliamentary Library, 1994*). This coincided with severe drought affecting the south eastern states for years between 1979 and 1983 (*Wikipedia contributors, 2021)*. Following this saw a period of "two years of very strong growth, lifting GDP per head 6.2 per cent above its previous peak" (*Colebatch, 2012*) which drove the unemployment rate down to a low of 5.8% in December 1989.

In the early 1990s much of the Western world was impacted by a period of economic downturn, primarily influenced by restrictive monetary policies, decline in consumer spending, and a drop in construction following a period of excessive building in the late '80s. (*Carl Walsh, 1993*) Following the low from 1989, Australia's unemployment rate rose for 12 consecutive quarters to a point of 11.1% in December 1992, where it plateaued at 10.9% for all of 1993, and has seen overall year on year improvement reaching a low of 4.1% in March 2008. Between June 2008 and March 2020 the unemployment rate rose slightly to an average of 5.4% with some intermittent small dips and rises. This behaviour is considered not only 'normal' but is generally regarded as **desirable**, as once "…unemployment rate gets below 5%, the economy is very close to or at full capacity" (*The Downside of Low Unemployment, 2020*).
However following the onset of impacts from COVID-19, Australia's unemployment rate rose sharply in the final two datapoints of June and September 2020 to 7 and 7.1% respectively. This was primarily driven by a rise in unemployed across retail and hospitality businesses which were particularly vulnerable to the effects of closed borders, drop in consumer confidence, and early lockdowns. (*Lannin & Pupazzoni, 2020*).

This rise in unemployment differs to the prior recessions of the mid 80's and 90's as the workforce has been quick to rebound with most of the impacted population able to find work quickly following the partial reopening

of most business sectors in 2021 - resulting in the unemployment rate dropping “from 4.9 per cent in June to 4.6 per cent in July” (*Janda & Pupazzoni, 2021*).

This light on the horizon is softened by the recent prolonged lockdowns in Australia’s two most populous states - Victoria and New South Wales - with economists concerned on the impact of the unemployment rate in upcoming months following this extended pause in trading for many businesses and heavy government assistance (*SBS Australia, 2021*). Understanding the drivers of previous spikes in unemployment rate in Australia will give experts the ability to anticipate with accuracy the effects of the ongoing COVID pandemic in Australia in future months to lessen the consequences.

DATA

The data is sourced entirely from the Australian Bureau of Statistics (ABS) website, and provides coverage of 9 variables - outlined below.

```
# define seed for both R and keras packages (used later on)
seed <- 5555
set.seed(seed)
set_random_seed(seed)

# import data
aus_data_pre <- paste0(gen_loc,"AUS_Data.csv")
aus_data <- read.table(aus_data_pre, header=TRUE , sep = ",", dec = ".")

#drop 1st row as it has headers
aus_data <- aus_data[-1,]
#drop last row as it's empty
aus_data <- aus_data[-159,]

# drop empty column
aus_data <- subset(aus_data, select = -c(X.1))

# replace missing values for population - sourced from:
# https://www.abs.gov.au/statistics/people/population/national-state-and-territory-po
pulation/mar-2021
aus_data$X7 <- with(aus_data,   ifelse(X == "Sep-2019",254722.6,
                                ifelse(X == "Dec-2019",255579.1,
                                ifelse(X == "Mar-2020",256686.5,
                                ifelse(X == "Jun-2020",256933.4,
                                ifelse(X == "Sep-2020",256809.4,aus_data$X7))))))

# add 01 to dates as they are currently missing the day
aus_data$X <- paste0("01-",aus_data$X)
# then convert to date and format
aus_data$X <- as.Date(format(strptime(aus_data$X, format = "%d-%b-%Y"), "%d/%m/%
Y"),"%d/%m/%Y")


# convert stated variables to int
varsToINT <- c("Y","X1","X2","X3","X4","X5","X6","X7")
aus_data[,varsToINT] <- lapply(aus_data[,varsToINT],as.numeric)

#### DATA IS NOW CLEAN ####

str(aus_data)
```

```
## 'data.frame':    158 obs. of  9 variables:
##  $ X : Date, format: "1981-06-01" "1981-09-01" ...
##  $ Y : num  5.5 5.8 6 6.2 6.6 7.1 8.8 9.7 10.2 10.3 ...
##  $ X1: num  1.6 2 -0.4 -0.8 0.9 -0.7 -1.6 -1 -0.2 2.8 ...
##  $ X2: num  1.6 2.7 -4.4 -2.8 7.5 -4.6 4 2.8 -0.2 0.2 ...
##  $ X3: num  1.9 2.5 -0.9 -0.1 3.4 -1.3 1.7 0.7 -0.6 0.6 ...
##  $ X4: num  -1.5 1.9 -2.8 -2.9 2.5 -0.5 -1 0.5 1.2 -0.2 ...
##  $ X5: num  28.4 29 30.2 30.8 31.5 32.6 33.6 34.3 35 35.6 ...
##  $ X6: num  44 44.5 41.7 37.3 30.3 26.8 30.3 29.7 32.4 33.6 ...
##  $ X7: num  149233 149887 150541 151217 151843 ...
```

The column names have been left using their **X1** - **X7** reference, along with the response Unemployment Rate being left as **Y** and date of observation being left as **X**.
This was done in order to keep the variable names short and succinct (i.e. it's easier to refer to a variable called *X7* than *Percentage change in Final consumption expenditure of all industry sectors*) but comes at the expense of needing to lookup the variable glossary when analysing dataset results and relationships.

The above code first sets the seed for this exercise (5555), imports the dataset, and drops empty rows and columns. It was noted that there were five missing values in the X7 ("Estimated Resident Population") variable for the months of September 2019 - September 2020. Rather than use value imputation for these observations, the data was sourced directly from the ABS website (https://www.abs.gov.au/statistics/people/population/national-state-and-territory-population/mar-2021 (https://www.abs.gov.au/statistics/people/population/national-state-and-territory-population/mar-2021)) for this time period and then hardcoded through direct assignment via the use of the *ifelse()* function in base R. Next, the date column X was formatted to a date by concatenating a leading '01-' to the start of each value (representing a day), then converted to a date through the use of the *as.Date()* and *format()* functions in base R. Finally, the remaining variables were converted to numeric through use of the base R *lapply* function.

In addition to the 5 missing population values, it was discovered that there were 5 missing values for variable X6 (job vacancies) for the months of September 2008 and September 2009. This was a result of the Job Vacancy Survey being suspended during this time and as a result the ABS has been unable to "…produce reliable estimates by collecting this missing data retrospectively, and has not been able to fill the gap using other data sources and modelling techniques." (*6354.0.55.001 - Information Paper: Reinstatement of Job Vacancies Survey, Nov 2009, n.d.*) Consequently, in order to proceed with meaningful application of machine learning methods, value imputation should be performed to the dataset prior to any statistical measures.

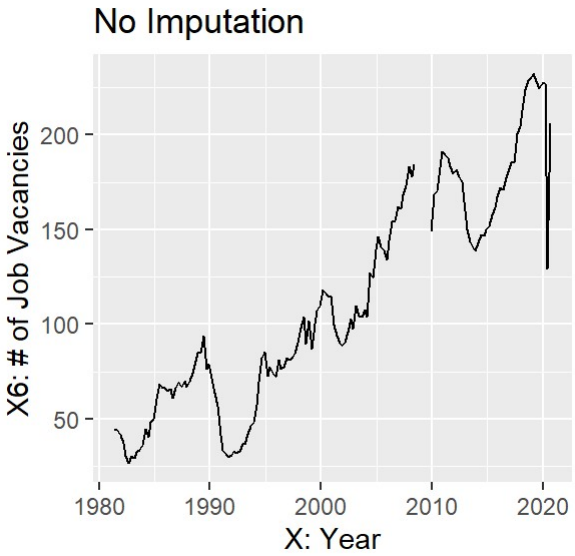
For this exercise the R package *MICE* has been used and imputation is being performed through regression by specifying the input parameter 'meth' as '*norm.predict*'.

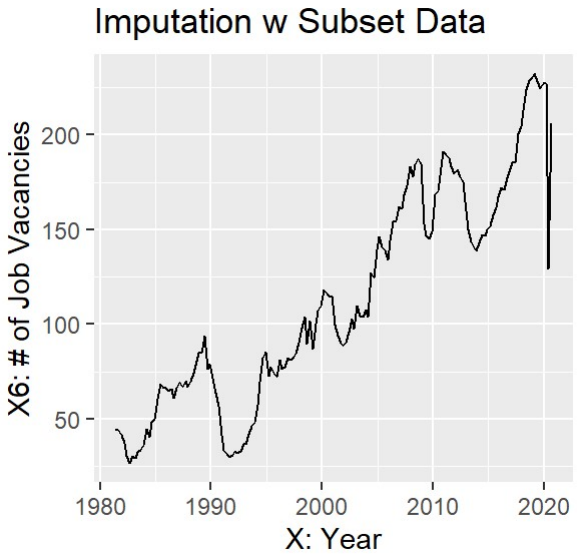
```
#============================
# start of subset imputation
#============================
# subset data so there are 12 observations prior to start of missing data, and 12 obs
ervations post end of missing data
aus_data_sub  <- subset(aus_data, aus_data$X >= "2007-03-01" & aus_data$X <= "2011-06
-01")

# take remainder of observations into separate dataframe so the two can be stacked la
ter
aus_data_sub2 <- subset(aus_data, aus_data$X < "2007-03-01" | aus_data$X > "2011-06-0
1")

# missing data in dataset needs to be imputed
aus_data_imp <- mice(aus_data_sub,maxit=50,meth='norm.predict',seed=seed)

# impute missing values
aus_data_imputed <- complete(aus_data_imp,1)

# combine both dataframes
imp_aus_data <- rbind(aus_data_sub2,aus_data_imputed)

# reorder this dataframe so rows are in chronological order
imp_aus_data <- sqldf(" SELECT  *
                        FROM       imp_aus_data
                        ORDER BY X asc")
```

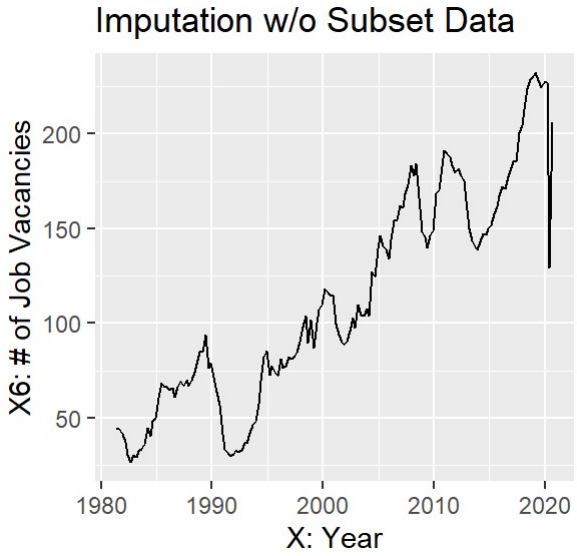
An additional technique has been applied to subset the data to 12 observations **before** the first missing value, and 12 observations **after** the last missing value. This was done so the regression would factor in only the most relevant values for this timeframe - and exclude some of the dramatic peaks and dips discussed earlier prior to 1995 in the regression calculations.

Three visualisations below demonstrate the plotting of the X6 Variables - Job Vacancies prior to imputation, using imputed values in addition to subsetting the data, and imputing the values *without* imputing the data.

```
par(mfrow=c(1,3)) # 3 charts side by side

#show comparison of applying imputation using subset data vs without subset data
par(mfrow=c(3,1)) # one chart at one time
ggplot(aus_data, aes(x=X, y=X6)) +
    geom_line() +
    ggtitle("No Imputation") +
       ylab("X6: # of Job Vacancies") +
       xlab("X: Year")

ggplot(imp_aus_data, aes(x=X, y=X6)) +
    geom_line() +
    ggtitle("Imputation w Subset Data") +
       ylab("X6: # of Job Vacancies") +
       xlab("X: Year")

ggplot(aus_data_imputed2, aes(x=X, y=X6)) +
    geom_line() +
    ggtitle("Imputation w/o Subset Data") +
       ylab("X6: # of Job Vacancies") +
       xlab("X: Year")
```







The result is subtle but the dataset that is subset and then imputed, displays a marginally more gradual descent, a smoother dip prior to the incline, and does not descend as much as the values that were imputed without prior subsetting applied.

An overview of the variables used in the dataset is printed below, containing a table for each column showing basic summary statistics as well as distribution plots for each variable:

```
summary(imp_aus_data)
```

```
##        X                     Y                 X1                 X2
##  Min.   :1981-06-01   Min.   : 4.100   Min.   :-7.0000   Min.   :-4.6000
##  1st Qu.:1991-03-24   1st Qu.: 5.500   1st Qu.: 0.4000   1st Qu.: 0.1000
##  Median :2001-01-15   Median : 6.250   Median : 0.7000   Median : 1.0000
##  Mean   :2001-01-15   Mean   : 6.855   Mean   : 0.7247   Mean   : 0.8532
##  3rd Qu.:2010-11-08   3rd Qu.: 8.275   3rd Qu.: 1.1000   3rd Qu.: 1.6000
##  Max.   :2020-09-01   Max.   :11.100   Max.   : 3.3000   Max.   : 7.5000
##        X3                X4                 X5                X6
##  Min.   :-8.300   Min.   :-8.1000   Min.   : 28.40   Min.   : 26.80
##  1st Qu.: 0.400   1st Qu.:-1.1000   1st Qu.: 59.00   1st Qu.: 67.55
##  Median : 0.800   Median : 0.3000   Median : 73.50   Median :102.50
##  Mean   : 0.762   Mean   : 0.3456   Mean   : 75.08   Mean   :113.00
##  3rd Qu.: 1.200   3rd Qu.: 1.6500   3rd Qu.: 96.80   3rd Qu.:161.47
##  Max.   : 5.900   Max.   :13.2000   Max.   :116.60   Max.   :232.30
##        X7
##  Min.   :149233
##  1st Qu.:172491
##  Median :191831
##  Mean   :196791
##  3rd Qu.:221555
##  Max.   :256933
```

RANDOM FOREST

Analysis and Investigation of a Machine learning (ML) method

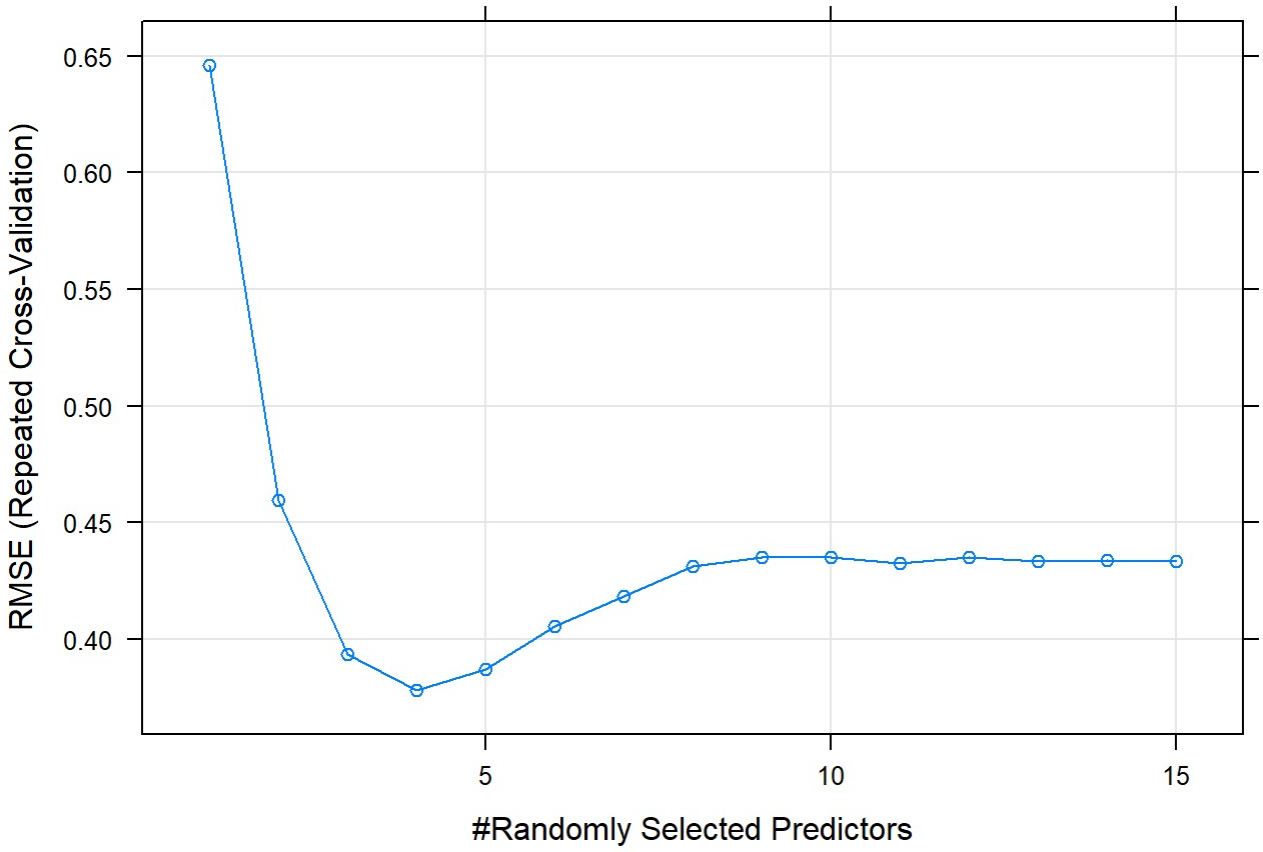
The machine learning technique chosen for this exercise is **Random Forest**.
Random Forest is "…flexible to both classification and regression problems." (*Great Learning Team, 2021*) and is not vulnerable to overfitting, making it an ideal candidate for this regression-based problem on a small dataset.
Random Forest is not as easily interpretable as Decision Trees but is generally capable of producing better accuracy, and is more than capable of outputting relevant performance metrics such as MAE, MAPE, R2, and RMSE.

The R packages randomForest, mlbench, and caret are used in the intitial step to identify and select the best value for '*mtry*' - which is the input parameter that determines the "…number of variables randomly sampled as candidates at each split." (*Brownlee, 2020*).
The below conducts a 'grid' search, running the random forest model multiple times using mtry values between 1 and 15.
It selects the most suitable mtry value from this list using the defined performance metric of RMSE (root mean square error) - a useful metric in determining "…the absolute fit of the model to the data–how close the observed data points are to the model's predicted values" (*Grace-Martin, 2020*).

```
library(randomForest)
library(mlbench)
library(caret)

control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")
set.seed(seed)
tunegrid <- expand.grid(.mtry=c(1:15))
rf_gridsearch <- caret::train(Y~., data=aus_data_train, method="rf", metric="RMSE", t
uneGrid=tunegrid, trControl=control)
print(rf_gridsearch)
```

```
## Random Forest
##
## 147 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 133, 131, 134, 132, 132, 132, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE       Rsquared   MAE
##    1    0.6461163  0.8822936  0.4578238
##    2    0.4595153  0.9407531  0.3238552
##    3    0.3935399  0.9563042  0.2788068
##    4    0.3780500  0.9592127  0.2694942
##    5    0.3871341  0.9573839  0.2742880
##    6    0.4055446  0.9520214  0.2894263
##    7    0.4182577  0.9490219  0.2992571
##    8    0.4311114  0.9455294  0.3098533
##    9    0.4351517  0.9441487  0.3112889
##   10    0.4351013  0.9442463  0.3113262
##   11    0.4326894  0.9449288  0.3101751
##   12    0.4350113  0.9443653  0.3112997
##   13    0.4332928  0.9446701  0.3111859
##   14    0.4337809  0.9445558  0.3093770
##   15    0.4335859  0.9452517  0.3106151
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 4.
```

```
plot(rf_gridsearch)
```



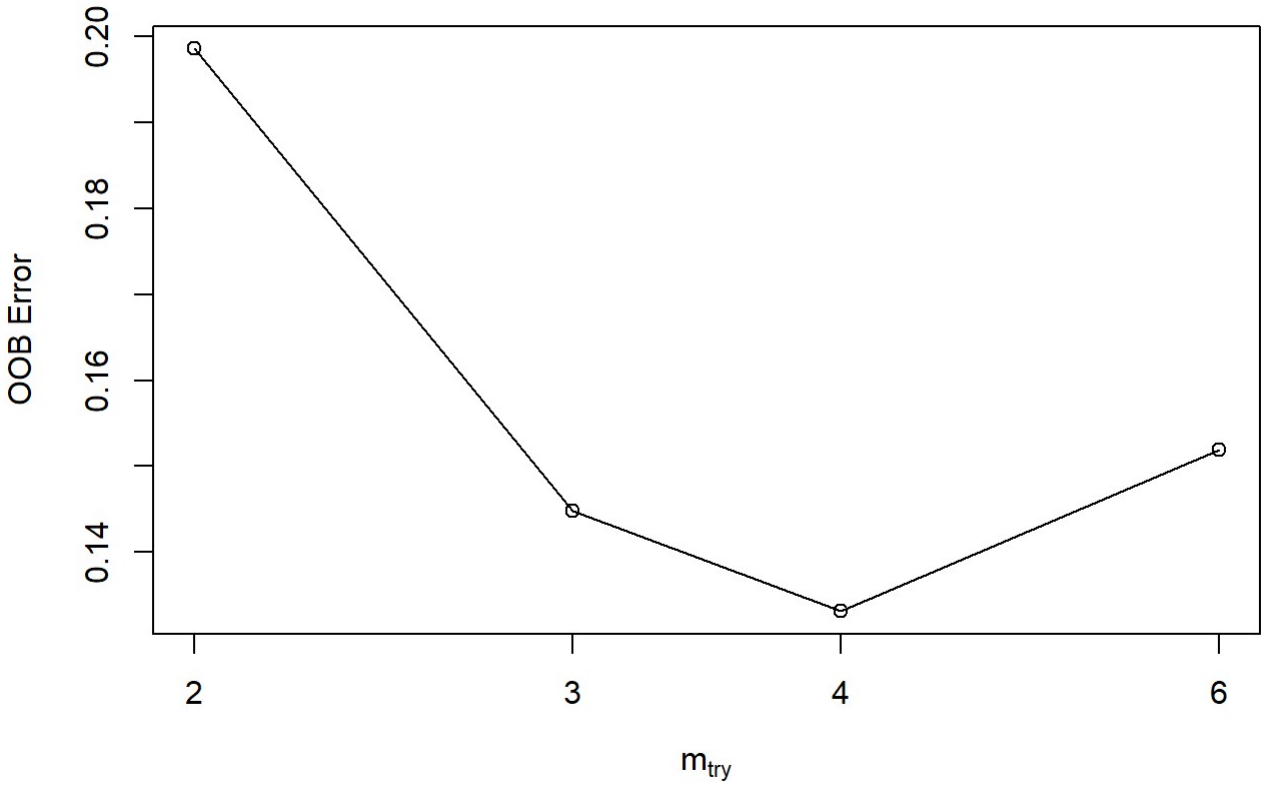
The output of this grid search determined that an mtry value of **4** produces the best fit model.
This is highlighted through the visualisation above indicating that the loss metric - RMSE - falls to its lowest point of 0.378 when using the value of 4 for the mtry parameter.

```
library(randomForest)
aus_data_rforest <-randomForest(Y ~., data=aus_data_train, mtry=8, importance=TRUE, n
tree=500)

#setting mtry = number of predictors to get bagging results
print(aus_data_rforest)
```

```
##
## Call:
##  randomForest(formula = Y ~ ., data = aus_data_train, mtry = 8,      importance =
TRUE, ntree = 500)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##           Mean of squared residuals: 0.1685849
##                     % Var explained: 94.78
```

```
# Find the optimal mtry value
mtry <- tuneRF(aus_data_train[-2],aus_data_train$Y, ntreeTry=500,
               stepFactor=1.5,improve=0.01, trace=TRUE, plot=TRUE)
```

```
## mtry = 2  OOB error = 0.1985411
## Searching left ...
## Searching right ...
## mtry = 3     OOB error = 0.1448235
## 0.2705615 0.01
## mtry = 4     OOB error = 0.133159
## 0.08054283 0.01
## mtry = 6     OOB error = 0.1518956
## -0.1407081 0.01
```




```
best_mtry <- mtry[mtry[, 2] == min(mtry[, 2]), 1]
print(mtry)
```

```
##   mtry  OOBError
## 2    2 0.1985411
## 3    3 0.1448235
## 4    4 0.1331590
## 6    6 0.1518956
```

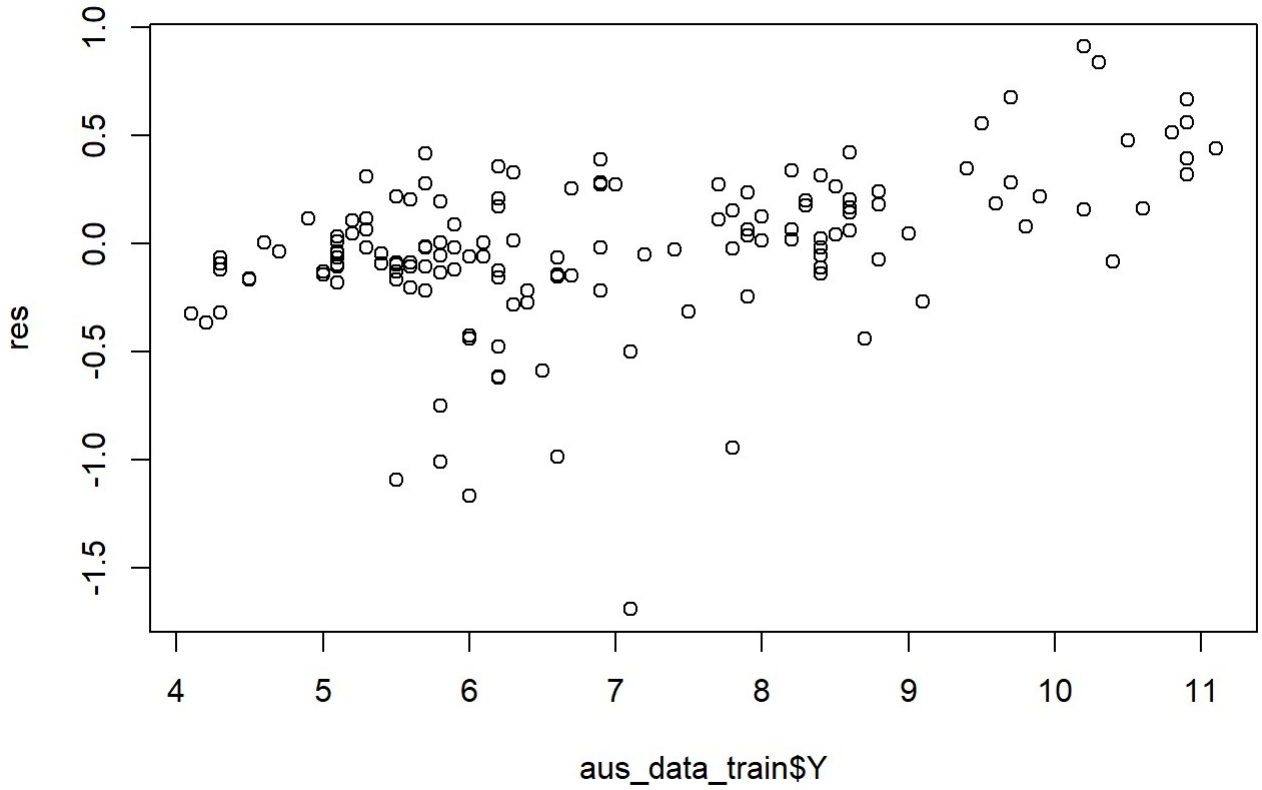
```
print(best_mtry)
```

```
## [1] 4
```

This is verified by the OOBError metric (Out-of-bag error) which is another "..method of measuring the prediction error of random forests, boosted decision trees, and other machine learning models utilizing bootstrap aggregating (bagging)" (*Bhatia, 2019*)

The R package randomForest and equivelant function is invoked and provided the *Y* response variable (unemployment rate), the mtry parameter of 4 has been assigned to the variable *best_mtry* and is provided as an input value to the model invocation below:

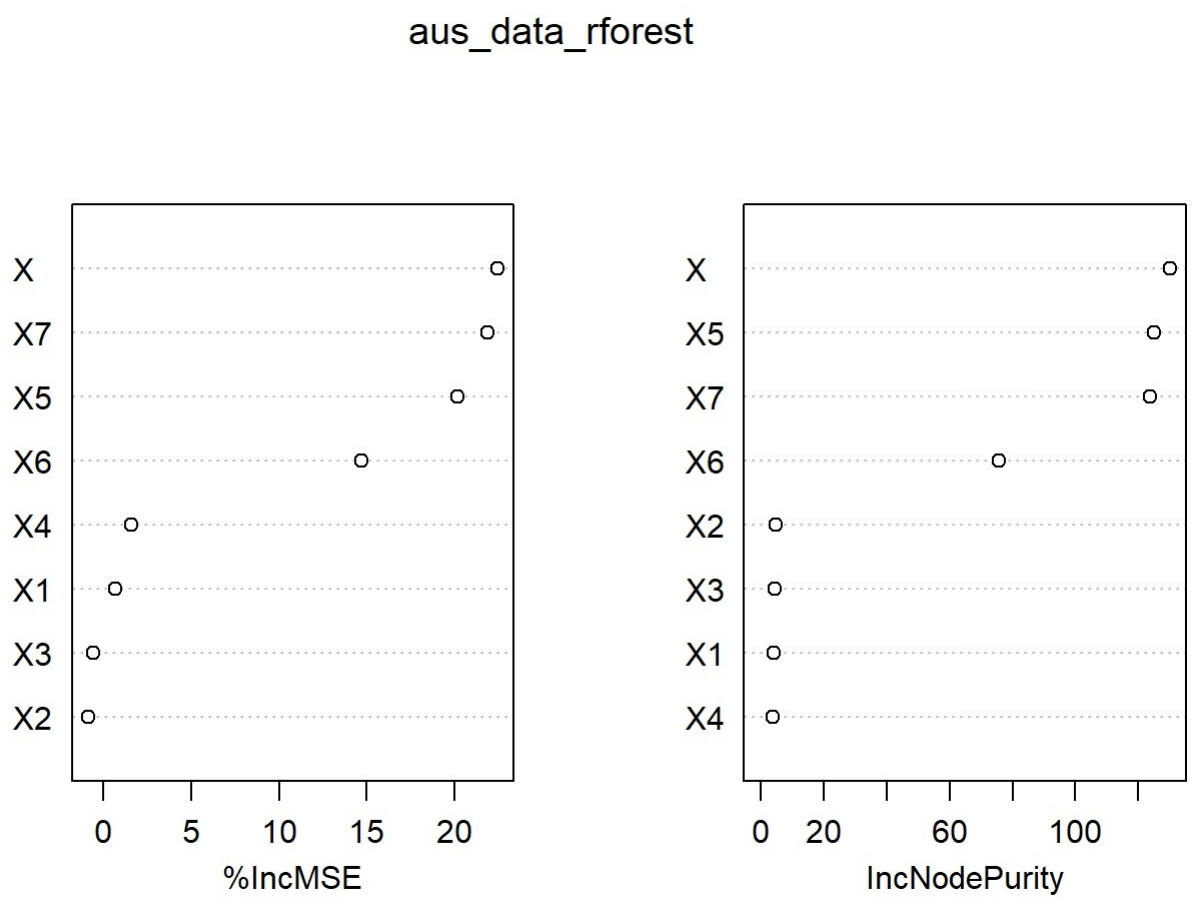
```
aus_data_rforest <-randomForest(Y ~., data=aus_data_train, mtry=best_mtry, importance
=TRUE, ntree=500)

res <- aus_data_train$Y-aus_data_rforest$predicted
plot(aus_data_train$Y, res)
```




```
#aus_data_rforest$importance
round(randomForest::importance(aus_data_rforest),2)
```

```
##    %IncMSE IncNodePurity
## X    22.42        129.96
## X1    0.69          4.04
## X2   -0.87          4.96
## X3   -0.59          4.43
## X4    1.57          3.97
## X5   20.18        125.08
## X6   14.69         75.58
## X7   21.87        123.60
```

```
varImpPlot(aus_data_rforest)
```



The above output table contains two columns; %IncMSE and IncNodePurity.

%IncMSE - or percent increase in mean-squared error - is a metric that describes "…how much our model accuracy decreases if we leave out that variable." (*Improving Your Model | R, n.d.*) Thus, the higher the scores for each variable, the higher the accuracy of the model is by leaving it in. Extending from this it can be seen that the variable X (year+month) is highly useful to the model's predictive power, followed by X7 (population) and X5 (CPI). Conversely it can be seen that X2 (% change in gov consumption) and X3 (% change in industry expenditure) have a negative influence on the model's accuracy and do little to predict the unemployment rate.

Node Purity "…is a measure of variable importance based on the Gini impurity index used for the calculating the splits in trees." (*Improving Your Model | R, n.d.-b*). Similar to IncMSE, the higher the value against each variable, the higher that variable is in terms of importance to the model. Aligned with the IncMSE results, the

variables X (date), X5 (CPI), and X7 (population) score the highest values here.

These results indicate that the key metrics available in the dataset to predict unemployment orientate towards CPI and population - which is especially useful for historic results, while the dates of historic unemployment rates is a useful measure to predict unemployment rate for that same time period but is less likely to be useful in predicting *future* unemployment rates.

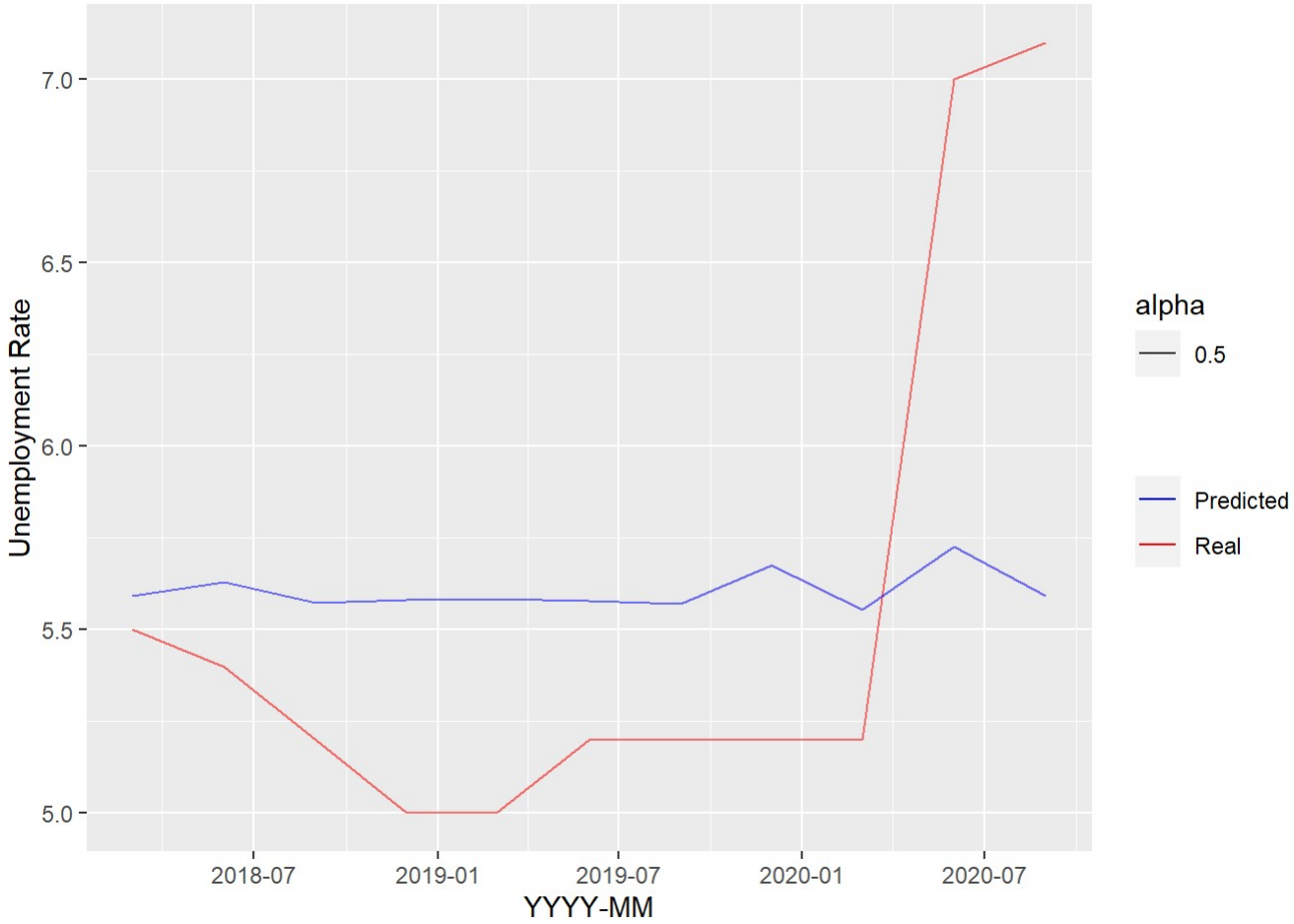
```
predictions <- predict(aus_data_rforest, aus_data_test)

result <- aus_data_test
result['Y'] <- aus_data_test$Y
result['prediction']<-  predictions
```

The above produces an output table showing the predicted values of the unemployment rate between the months of March-2018 and September-2020, compared to the actual unemployment rate values for this time period. As seen, the predictive power of the random forest is quite strong over the course of the time period with differences ranging between 0.0 and 0.5% but tapers off in the final two data points of June and September 2020 when the difference between actual vs predicted scores (i.e. MAE), increases to 1.3% and 1.6% respectively.
This is presumably due to these months being in the midst of the COVID-19 pandemic where unemployment rate rose in a way that differed to historic increases in the dataset. As machine learning methods build their predictions on historic data, when circumstances occur that have no historic reference (e.g. a global pandemic), the accuracy of these models often falls when faced with new data points that have no prior occurence.
Efforts can be made to implement rigorous stress-testing practises that adopt monte-carlo style scenarios to curtail the effects of novel events like COVID, as well as including additional variables that may help act as a ‘canary in a coal-mine’ and give predictive models the capacity to predict swift changes like this using more subtle clues.

```
result
```

```
##              X   Y   X1  X2   X3   X4    X5    X6       X7 prediction
## 148 2018-03-01 5.5  0.9 1.3  0.7  3.3 112.6 212.8 248986.3   5.592323
## 149 2018-06-01 5.4  0.9 0.0  0.6 -1.2 113.0 223.8 249826.9   5.631003
## 150 2018-09-01 5.2  0.4 1.3  0.5  1.0 113.5 228.6 250917.5   5.572393
## 151 2018-12-01 5.0  0.2 1.7  0.6  2.4 114.1 230.4 251701.8   5.580313
## 152 2019-03-01 5.0  0.5 0.9  0.5  3.3 114.1 232.3 252887.6   5.583020
## 153 2019-06-01 5.2  0.6 2.8  1.0  1.5 114.8 228.0 253643.1   5.578490
## 154 2019-09-01 5.2  0.6 0.8  0.2  0.6 115.4 224.2 254722.6   5.570933
## 155 2019-12-01 5.2  0.4 1.2  0.6 -4.4 116.2 227.0 255579.1   5.674780
## 156 2020-03-01 5.2 -0.3 1.9 -0.4  0.4 116.6 227.3 256686.5   5.553913
## 157 2020-06-01 7.0 -7.0 3.0 -8.3  0.9 114.4 129.2 256933.4   5.726863
## 158 2020-09-01 7.1  3.3 1.4  5.9  0.7 116.2 206.1 256809.4   5.592457
```

```
ggplot(  ) +
    geom_line( aes(x = aus_data_test$X, y = aus_data_test$Y, color = 'red', alpha =
0.5) ) +
    geom_line( aes(x = aus_data_test$X , y = predictions, color = 'blue',  alpha = 0.
5)) +
    labs(x = "YYYY-MM", y = "Unemployment Rate", color = "", ) +
    scale_color_manual(labels = c( "Predicted", "Real"), values = c("blue", "red"))
```



The above chart highlights the strong predictive power of the random forest model up until June 2020 at which point there is a sudden spike in actual unemployment rates to 7+%, while the model suggests a modest jump to 5.7, before maintaining a 5.5% unemployment rate.

```
library(Metrics)
print(paste0('MAE: ' , mae(aus_data_test$Y,predictions)))
## [1] "MAE: 0.565259090909089"
print(paste0('MAPE: ' , mape(aus_data_test$Y,predictions)))
## [1] "MAPE: 0.0965032648065955"
print(paste0('MSE: ' ,caret::postResample(predictions , aus_data_test$Y)['RMSE']^2 ))
## [1] "MSE: 0.491119964138383"
print(paste0('R2: ' ,caret::postResample(predictions , aus_data_test$Y)['Rsquared']
))
## [1] "R2: 0.269353976363459"
print(paste0('RMSE: ' ,caret::postResample(predictions , aus_data_test$Y)['RMSE'] ))
## [1] "RMSE: 0.70079951779263"
```

Finally, the above code outputs a range of additional metrics including MAE, MAPE, MSE, R2, and RMSE. In isolation, the R2 score of 0.26 suggests the model's predictive power based on the data is "substantial" (*Mark & Peucker, 2020*), while the MAPE score of 0.09 is regarded as "excellent" (*Is My Model Any Good, Based on the Diagnostic Metric (R^2/ AUC/ Accuracy/ RMSE Etc.) Value?, 2019*).

However a shortcoming of these metrics is that they do not reflect the reducing accuracy of the model in the final two data points, which are likely to persist with the introduction of new data points following the cutoff date. That is to say that if the model was introduced data from the cutoff date of September 2020 to now include data up until September 2021, it is likely there would continue to be a large variance between the actual vs predicted unemployment rate values due to the ongoing effects of COVID.

NEURAL NETWORK

Analysis and Investigation of a neural network (NN) method

Parameters defined for the neural network are as follows: - activation function is set as '*relu*'; the Rectified Linear Unit.
The ReLU activation function has becomg "the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance." (*Brownlee, 2020b*) This rise in popularity is also credited due to its novel feature of being able to solve for the well known machine learning problem of **Vanishing Gradient**, a problem characterised in complex neural networks where the output of the model is unable to propagate meaningful information back to the input layer(s) of the model. (*Brownlee, 2020c*) ReLU is particularly applicable for this exercise as the response variable *Y* is numeric and follows a linear behaviour in the historic data, concluding that applying a activation function that is orientated towards solving linear problems (rather than sigmoid for logistic, softmax plus for exponential etc) is preferable over the suite of alternative activation functions available via the Keras API.

- the batch number for this neural network is configured to **64**. This value was established through a grid-search pattern trial-and-error approach, where the batch value was set to 1, 2, 4, 8 and so on - doubling until the neural network was run with 2048 observations. This final number is excessive and would generally be regarded as impractical for most implementations of a model of this type (i.e. <10 variables and < 1000 observations), but was conducted as a point of comparison. The input parameter for this variable was defined above the neural network and assigned to 'batchSize'.

- The number of hidden layers used in this neural network is **10**.
  Again, a grid-search was applied in increments of 2 - from 2 to 20. 10 was found to be a suitable stopping point based on the returned performance metrics of MAPE (mean absolute percentage error)

and MAE (mean absolute error).

- The Validation split for this neural network is set to *0.2*. This translates to 20% of the training dataset being 'iteratively 'held-out' for the process of each iteration, and then compared to the results of the trained 80% training data at the end of each epoch. This value was attained by moving in increments of 0.05 (5%) from 0.05 to 0.3, with 0.2 offering the best MAPE scores.

The optimiser for this neural network is set using the **Stochastic gradient descent** (SGD) optimiser as paret of the Keras package, this is invoked using the *optimizer_sgd()* function as part of the *compile()* function. SGD is a commonly-used optimiser in machine learning problems and was selected for this exercise following some trial and error with other popular alternatives, such as adam and rmsprop which were surpassed by SGD in MAE and R2 scores.
SGD differs to gradient descent in that the latter computes the gradient of the dataset at each iteration, while SGD does this by instead referring to random observations of the "…training data at each step and then computes the gradient making it much faster as there is much fewer data to manipulate at a single time, unlike Batch GD." (*GeeksforGeeks, 2020*)

In an effort to reduce over-fitting, the model is designed to employ a strategy dubbed 'early-stopping'. This common approach results in a model that will "…stop training at the point when performance on a validation dataset starts to degrade." (*Brownlee, 2019*) This is implemented by defining a performance metric to monitor, and once this metric begins to degrade, the model is prematurely stopped after a particular defined threshold. In this instance the defined loss metric was set to track *MSE* (mean-squared error) and the input parameter of **patience** was set to a value of *4*. This parameter defines the "…number of epochs with no improvement after which training will be stopped." (*Keras Team, n.d.*)

This number was achieved through iterative searching, by running the model on the parameters described above, and trying values from 1-10. This value will impact the number of epochs that are actually run in the neural network - discussed further below.

- The number of epochs assigned for this neural network was set to **50**. This value describes the number of times a neural network will 'loop' through the input dataset as part of the training process. However - although this number is set to 50, because of the early-stopping approach adopted and discussed above, the *actual* number of epochs run for this neural network is **11**.
  This is highlighted below through use of visualisations that record the changes to metrics such as MAE, MAPE, and particularly the MSE (which is the defined loss metric) across the number of epochs for the neural network as it runs.
  As noted in the MSE section, the distance between the training and validation error nears to 0 at 11 epochs.
  This has been left as 50 for the time being to illustrate the application and usefulness of the early-stopping methodology in action.

The assignment of these values can be viewed below:

```
#======================
#    ======================
#        ======================
# NEURAL NETWORK
#        ======================
#    ======================
#======================

# get current time to check duration of the neural network run time at the end
start_time <- Sys.time()

# set the seeds
set.seed(seed)
set_random_seed(seed)

# define parameters used in nn
act = "relu"    # activation function
batchSize = 64  # batch size for iterations
unit = 1        # number of rows for output layer at one time
verb = 0        # 1 for verbose notes, 0 for nonverbose notes
valSplit = 0.2  # validation split (% of dataset to be withheld for validation)
epoch_n = 20  # epoch number (total iterations of dataset)
patnce = 4      # patience for early stopping
```

Numeric variables are scaled prior to being passed through to the model, to reduce the impact of outliers on the neural network.

```
# scale numeric columns using python
spec <- feature_spec(aus_train_nn, Y ~ . ) %>%
    step_numeric_column(all_numeric(), normalizer_fn = scaler_standard()) %>%
    fit()

spec
```

```
## -- Feature Spec ----------------------------------------------------------
## A feature_spec with 7 steps.
## Fitted: TRUE
## -- Steps -----------------------------------------------------------------
## The feature_spec has 1 dense features.
## StepNumericColumn: X1, X2, X3, X4, X5, X6, X7
## -- Dense features --------------------------------------------------------
```

Define the neural network structure; 10 layers deep for this exercise. This value is larger than the what may typically be expected for neural networks created for a dataset of this size, however an iterative grid-search between the number of layers of 1-15 found that 10 produced the most accurate results, while maintaining the risk of overfitting the model. Additional parameters such as the activation function (ReLU), the batch size (64), and the final output row count (1) are also provided here.

A summary of the model is printed below:

```
output <- input %>%
    layer_dense_features(dense_features(spec))   %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
    layer_dense(units = unit)

model <- keras_model(input, output)

summary(model)
```

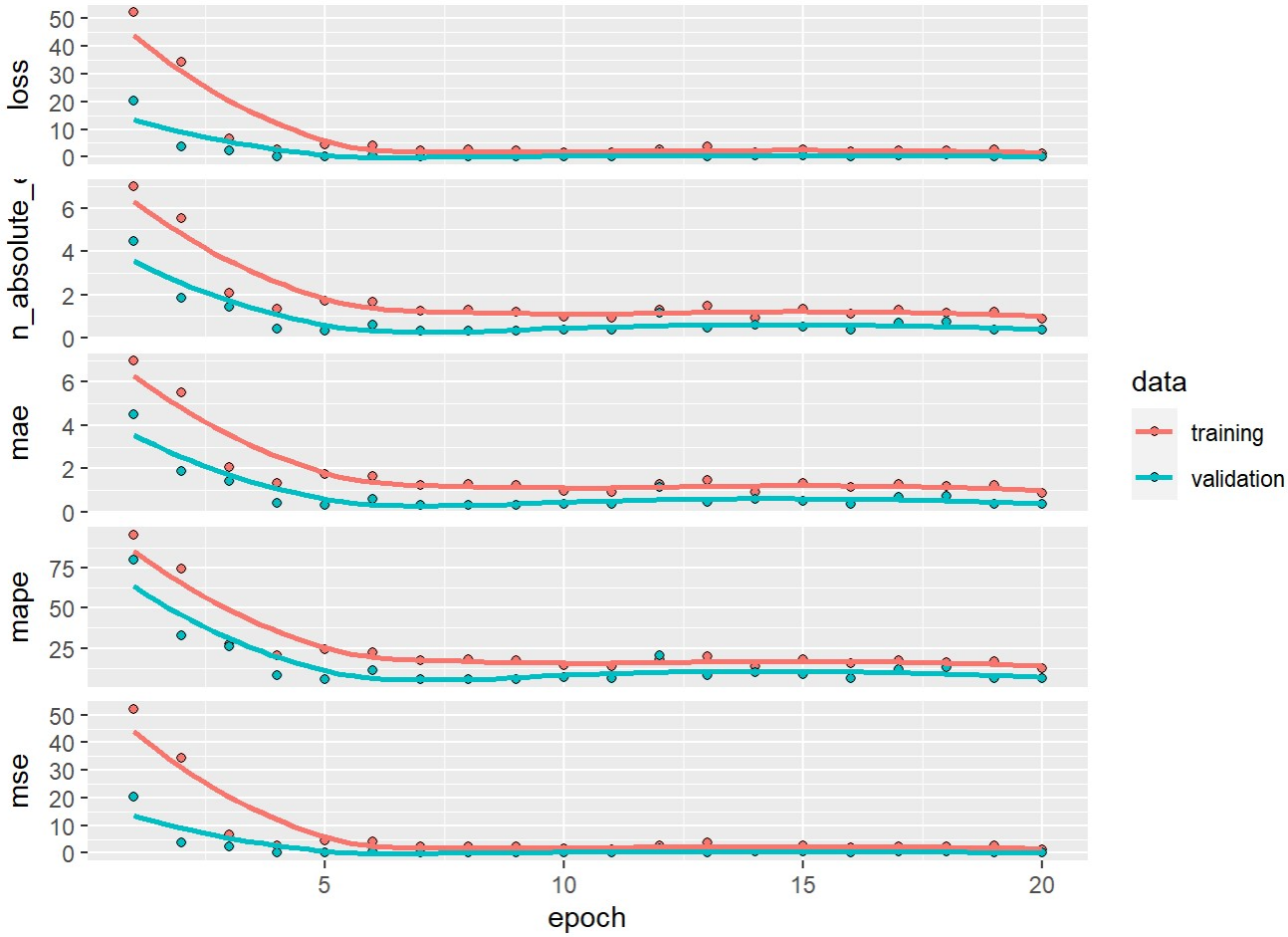
Now that the neural network structure is configured, define the optimisation method of SGD as well as provide a list of the key performance metrics to be output: MAE, MAPE, and MSE.

```
model %>%
    compile(
        loss = "mse",
        optimizer = optimizer_sgd(),
        metrics = list("mean_absolute_error","mae","mape","mse")
    )
```

The model is then run, passing through the training dataset minus the response variable *Y* for argument x, providing the response variable *Y* (unemployment rate) for argument y, stating the validation split, epochs, and then printing the performance metrics using a line chart.

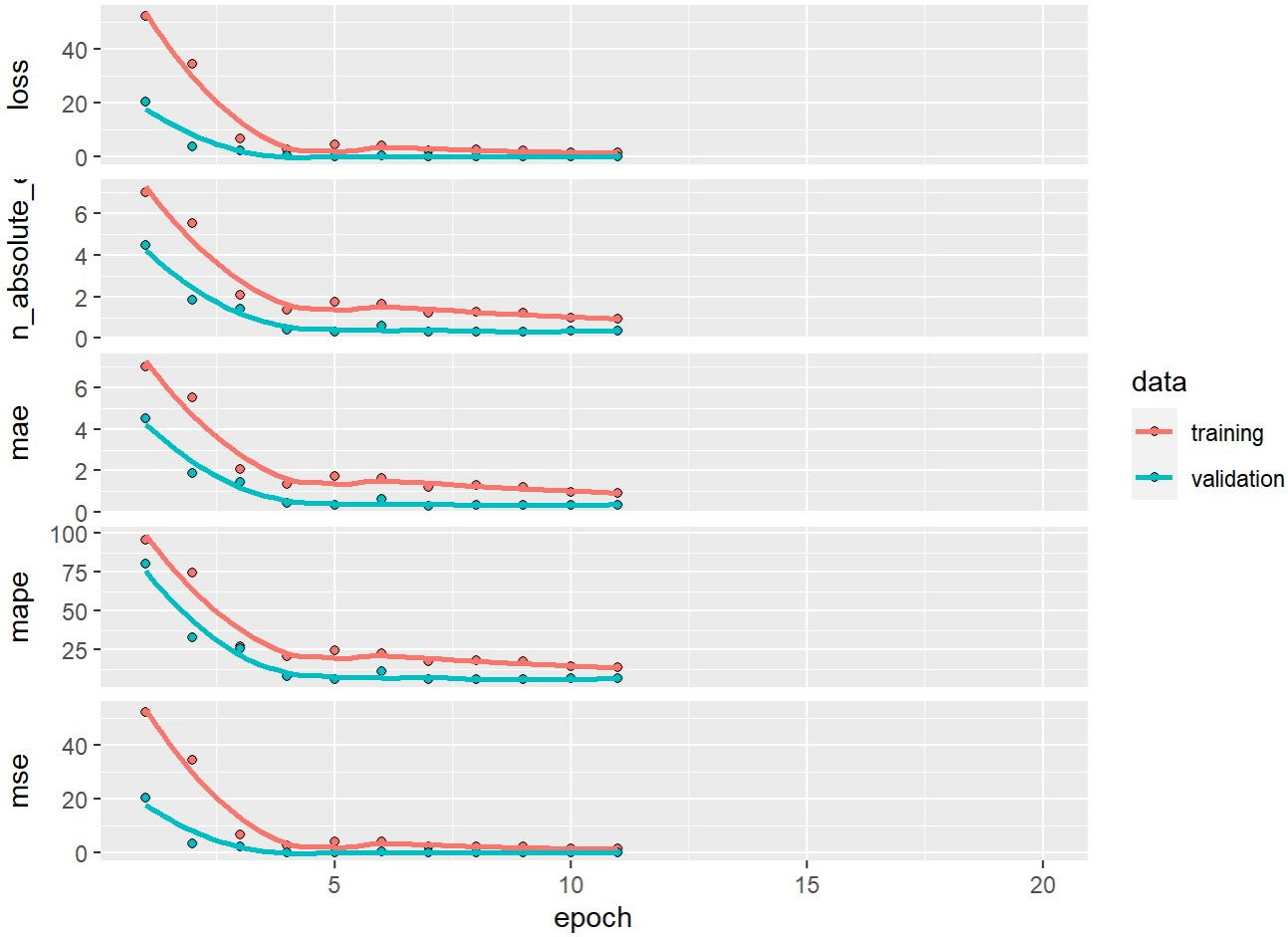
```
history <- model %>% fit(x = aus_train_nn[-1],
                         y = aus_train_nn$Y,
                    epochs = epoch_n,
          validation_split = valSplit,
                   verbose = verb,
                 callbacks = list(print_dot_callback))
set.seed(seed)
set_random_seed(seed)
```

```
plot(history)
```



Now run the model but apply the early-stopping procedure and plot the output.

```
early_stop <- callback_early_stopping(monitor = "val_loss", patience = patnce)

model <- build_model()

history <- model %>% fit(
    x = aus_train_nn[-1],
    y = aus_train_nn$Y,
    epochs = epoch_n,
    validation_split = valSplit,
    verbose = verb,
    callbacks = list(early_stop)
)

set.seed(seed)
set_random_seed(seed)

plot(history)
```




```
set.seed(seed)
set_random_seed(seed)
```

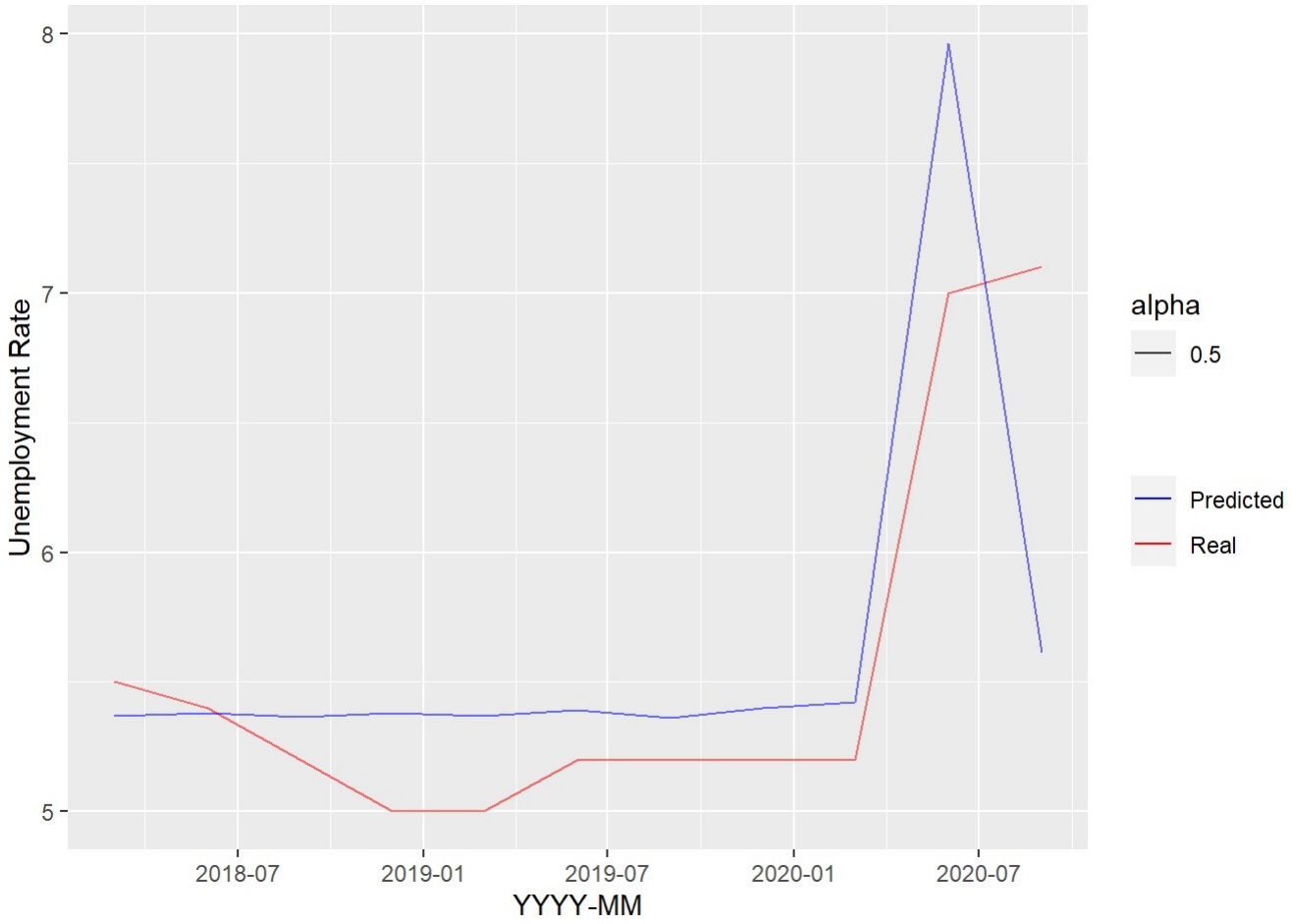
Performance metrics and run time of the model are then assigned to variables and then stated below:

```
paste0("Mean absolute error on test set: ", round(mae,2))
## [1] "Mean absolute error on test set: 0.39"
paste0("Mean absolute percentage error on test set: ", round(mape,2),"%")
## [1] "Mean absolute percentage error on test set: 6.41%"
paste0("Mean squared error on test set: ", round(mse,2))
## [1] "Mean squared error on test set: 0.33"
paste0("Inferred accuracy: ", round(1-(mape/100),4)*100,"%")
## [1] "Inferred accuracy: 93.59%"
print("Predicted Values for Test set: ")
## [1] "Predicted Values for Test set: "
test_predictions[,1]
##  [1] 5.367697 5.379242 5.363659 5.381693 5.368178 5.390708 5.361416 5.399473
##  [9] 5.421508 7.961507 5.611669


end_time <- Sys.time()
end_time - start_time
## Time difference of 9.641283 secs
```

As noted from the above output, the mean absolute error (MAE) score for this test is 0.39, the mean absolute percentage error (MAPE) score is 0.064, and the MSE score is 0.33. These scores are promising, indicating that there is on average a 0.39 difference in actual vs predicted unemployment rates (e.g. if actual unemployment rate is 10%, the prediction is returning 10.39%), while the 6.4% relative difference indicated by MAPE indicates that there is on average a 6.4% difference between actual vs predicted unemployment rates (e.g. if actual unemployment is 10% the predicted is 10.64% - 6.4% higher).

The predicted values for unemployment rate are printed near the bottom of this window, indicating that the first few months of the test dataset are predicted with reasonable accuracy, while the June 2020 prediction overshoots by almost 1%, before underestimating the September 2020 unemployment rate by 1.5%.

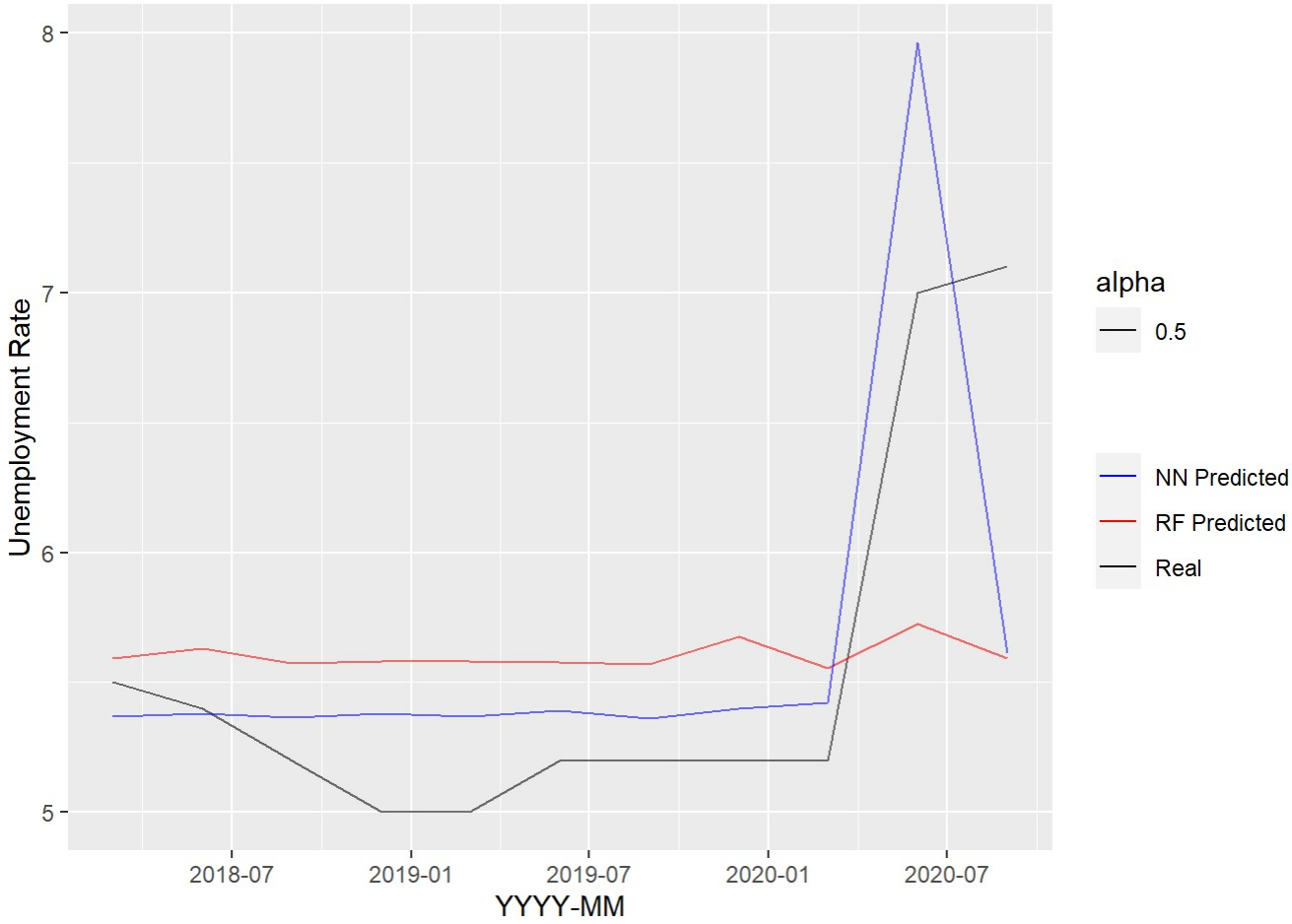
```
ggplot(  ) +
    geom_line( aes(x = aus_data_test$X, y = aus_data_test$Y, color = 'red', alpha =
0.5) ) +
    geom_line( aes(x = aus_data_test$X , y = test_predictions[,1], color = 'blue',  a
lpha = 0.5)) +
    labs(x = "YYYY-MM", y = "Unemployment Rate", color = "", ) +
    scale_color_manual(labels = c( "Predicted", "Real"), values = c("blue", "red"))
```



Comparison and Contrasting ML and NN models

A quick side by side comparison of the Random Forest and Neural Network Predictions vs the *actual* values is provided below:

```
ggplot(  ) +
    geom_line( aes(x = aus_data_test$X, y = aus_data_test$Y, color = 'red', alpha =
0.5) ) +
    geom_line( aes(x = aus_data_test$X , y = predictions, color = 'blue',  alpha = 0.
5)) +
    geom_line( aes(x = aus_data_test$X , y = test_predictions[,1], color = 'black',
alpha = 0.5)) +
    labs(x = "YYYY-MM", y = "Unemployment Rate", color = "", ) +
    scale_color_manual(labels = c( "NN Predicted","RF Predicted", "Real"), values = c
("blue", "red","black"))
```



As seen, for the first 9 out of 11 datapoints the Neural Network and Random Forest predictions follow a roughly identical trend, maintaining a fixed point of approximately ~5.55 for RF and ~5.36. This is somewhat higher than the actual values recorded for unemployment rate during this period, which starts off at 5.5% in March 2018, dips to 5% in December 2019 before moving back up to 5.2% in March 2020. June 2020 is the first month where the effects of COVID in Australia is detectable in the unemployment rate, where a sudden spike is seen going to 7% and then 7.1% in September 2020. The Random forest prediction for these two months demonstrates a very small spike of 5.7% in June 2020, before going back down to 5.5% in September. This prediction is entirely at odds with the actual values recorded for these two months, in terms of both the magnitude of the values, and the direction of the trendline. This highlights the nature of the Random Forest approach and indeed almost all supervised machine learning techniques that have their predictive algorithyms based on historic data; where these predictions are likely to be true and mostly accurate as long as the new data is somewhat similar to the data the model was fed in the training dataset. As seen in this instance, the

spike in unemployment due to COVID is not a scenario that the training data contained, thus the model was unable to account for the sudden spike in unemployment due to this happening in the test dataset.

These results are contrasted by the Neural Network predictions, which as stated initially follow a similar trajectory to those predicted by the Random Forest (albeit closer to the actual values), however at the point of June 2020 the similarities between the Random Forest and Neural Network results cease.
As seen from the chart, **in June 2020 the Neural Network correctly predicted that a very sudden, dramatic spike would occur that would result in the unemployment rate rising to levels not seen in 20+ years**.
The NN overestimated this jump by a large margin of almost 1% too high - however the trajectory and direction of the Neural Network's climb against the actual results can be recognised as startlingly similar, and shows that the Neural Network's predictive power is not simply 'better' than those of the Random Forest, but highlights it is much more capable of predicting unusual events such as COVID.

To further cement this, a comparison of the metrics for the Random Forest and Neural Network models above shows that the NN returned a MAE score of **0.39** - compared to the RF MAE score of **0.56**.
The MAPE score for the Neural Network was **0.064** - compared to **0.096** for the Random Forest. And the loss metric MSE was returned for the Neural Network was **0.33**, while the Random Forest returned a value of **0.49**.

From these three performance measures it is apparent that the Neural Network is performing roughly 40-50% better in terms of the difference between predicted vs actual unemployment rates, and in terms of finding the line of best fit from the MSE metric. (*Mean Squared Error: Definition and Example, 2021*)

Interpretability was a shortfall of the Neural Network, however this may be more symptomatic of the platform used rather than a critique of the information that *can* be extracted from an NN. It was discovered during the development of this analysis that the API to Keras via python for R did not provide the same exhaustive list of metrics available for extraction - such as R-squared score and RMSE.
Both Random Forest and Neural Networks suffer from a degree of difficult interpretability, especially in comparison to Random Forest's counterpart - Decision Trees - which offer highly interpretable criteria and logic used in the model computation in explaining how the model determined its output.

In terms of overall runtime the effect was negligible, with a difference of just several seconds favouring the Random Forest.
However it is worth noting that as this dataset is small, similar analysis on a much larger dataset will result in a compounding time difference where the Neural Network may take *hours* longer for datasets that fall under the big-data category.

Conclusions

As demonstrated, for this exercise the Neural Network demonstrated higher accuracy than the Random Forest machine learning method at predicting the Australian unemployment rate. The Neural Network's predictions against the test dataset were closer to the real values prior to COVID, and despite overshooting the actual values - the Neural Network displayed surprising precision at foreseeing the sudden rise in unemployment during the outbreak of COVID in Australia would occur based on the values populated in the other variables in the dataset.
Following the onset of COVID, many financial institutions and governments in Australia and around the world found that their credit risk predictive models were unable to offer much predictive accuracy for metrics such as rising default rates, unemployment, and shifts in the retail sectors (*Machine Learning: Challenges and Opportunities in Credit Risk Modeling, n.d.*). These models are often built using a linear (or logistic) regression-style approach (*Introduction to Credit Risk Modeling, 2020*) and are inflexible in rapidly changing environments and unable to accurately predict relevant metrics when the *new* data is too dissimilar to the *historic* data.

As evidenced by the above comparison, it would seem that there is opportunity to adopt aNNs as a credit-risk modelling approach that offers many distinct advantages, such as overall superior accuracy, flexibility in changing circumstances, and the ability to more accurately predict the results of previously-unencountered macroeconomic events such as COVID.

Future study in this area and on this dataset specifically may find improvement by doubling the size of the dataset, by simply repeating the rows. This will curtail one of the shortcomings of this dataset which is too few rows.
Another opportunity for analysis is to investigate the effects of dropping the X (date) column from the models. It was left in for this exercise as the Random Forest in particular benefited from its presence but there may be alternate methods such as SVM which will produce better results without it, than Random Forest with it.

Bibliography

6354.0.55.001 - Information Paper: Reinstatement of Job Vacancies Survey, Nov 2009. (n.d.). Australian Bureau of Statistics. Retrieved October 13, 2021, from https://www.abs.gov.au/ausstats/abs@.nsf/mf/6354.0.55.001 (https://www.abs.gov.au/ausstats/abs@.nsf/mf/6354.0.55.001)

Brownlee, J. (2019, August 6). A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks. Machine Learning Mastery. https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/ (https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/)

Brownlee, J. (2020a, July 31). Tune Machine Learning Algorithms in R (random forest case study). Machine Learning Mastery. https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/ (https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/)

Brownlee, J. (2020b, August 20). A Gentle Introduction to the Rectified Linear Unit (ReLU). Machine Learning Mastery. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ (https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/)

Brownlee, J. (2020c, August 25). How to Fix the Vanishing Gradients Problem Using the ReLU. Machine Learning Mastery. https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/ (https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/)

Causes of Inequalities - Reasons why income and wealth inequality exists - Higher Modern Studies Revision. (n.d.). BBC Bitesize. Retrieved October 13, 2021, from https://www.bbc.co.uk/bitesize/guides/zspttfr/revision/5 (https://www.bbc.co.uk/bitesize/guides/zspttfr/revision/5)

CHRISTOFFERSEN, J. (2013, October 14). Rising inequality “most important problem.” Stltoday. https://www.stltoday.com/business/local/rising-inequality-most-important-problem-says-nobel-winning-economist/article_a5065957-05c3-5ac0-ba5b-dab91c22973a.html (https://www.stltoday.com/business/local/rising-inequality-most-important-problem-says-nobel-winning-economist/article_a5065957-05c3-5ac0-ba5b-dab91c22973a.html)

Colebatch, T. (2012, September 26). Australia dodged recession but growth estimates haunt. The Sydney Morning Herald. https://www.smh.com.au/national/australia-dodged-recession-but-growth-estimates-haunt-20120926-26lih.html (https://www.smh.com.au/national/australia-dodged-recession-but-growth-estimates-haunt-20120926-26lih.html)

The Downside of Low Unemployment. (2020, September 29). Investopedia. https://www.investopedia.com/insights/downside-low-unemployment/ (https://www.investopedia.com/insights/downside-low-unemployment/)

GeeksforGeeks. (2020, July 2). Difference between Batch Gradient Descent and Stochastic Gradient Descent. https://www.geeksforgeeks.org/difference-between-batch-gradient-descent-and-stochastic-gradient-descent/ (https://www.geeksforgeeks.org/difference-between-batch-gradient-descent-and-stochastic-gradient-descent/)

Grace-Martin, K. (2020, November 2). Assessing the Fit of Regression Models. The Analysis Factor. https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/ (https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/)

Great Learning Team. (2021, May 20). Random Forest Algorithm- An Overview. GreatLearning Blog: Free Resources What Matters to Shape Your Career! https://www.mygreatlearning.com/blog/random-forest-algorithm/ (https://www.mygreatlearning.com/blog/random-forest-algorithm/)

Improving Your Model | R. (n.d.-a). DataCamp. Retrieved October 13, 2021, from https://campus.datacamp.com/courses/introduction-to-machine-learning-in-r/how-much-will-i-earn?ex=6 (https://campus.datacamp.com/courses/introduction-to-machine-learning-in-r/how-much-will-i-earn?ex=6)

Improving Your Model | R. (n.d.-b). DataCamp. Retrieved October 13, 2021, from https://campus.datacamp.com/courses/introduction-to-machine-learning-in-r/how-much-will-i-earn?ex=6 (https://campus.datacamp.com/courses/introduction-to-machine-learning-in-r/how-much-will-i-earn?ex=6)

Introduction to Credit Risk Modeling. (2020, August 31). IMSL by Perforce. https://www.imsl.com/blog/credit-risk-modeling (https://www.imsl.com/blog/credit-risk-modeling)

Is my model any good, based on the diagnostic metric (R^2/ AUC/ accuracy/ RMSE etc.) value? (2019, June 23). Cross Validated. https://stats.stackexchange.com/questions/414349/is-my-model-any-good-based-on-the-diagnostic-metric-r2-auc-accuracy-rmse (https://stats.stackexchange.com/questions/414349/is-my-model-any-good-based-on-the-diagnostic-metric-r2-auc-accuracy-rmse)

Janda, M., & Pupazzoni, R. (2021, August 19). Unemployment rate dropped early in lockdown as people gave up looking for work, data reveals. ABC News. https://www.abc.net.au/news/2021-08-19/unemployment-jobs-covid-lockdowns-abs-july-2021/100389960 (https://www.abc.net.au/news/2021-08-19/unemployment-jobs-covid-lockdowns-abs-july-2021/100389960)

Lannin, S., & Pupazzoni, R. (2020, November 19). Unemployment rate rises to 7 per cent as more people look for work, but employment jumps by 179,000. ABC News. https://www.abc.net.au/news/2020-11-19/jobs-unemployment-coronavirus-economy-abs/12899560 (https://www.abc.net.au/news/2020-11-19/jobs-unemployment-coronavirus-economy-abs/12899560)

Machine Learning: Challenges and Opportunities in Credit Risk Modeling. (n.d.). Moodys Analytics. Retrieved October 13, 2021, from https://www.moodysanalytics.com/risk-perspectives-magazine/managing-disruption/spotlight/machine-learning-challenges-lessons-and-opportunities-in-credit-risk-modeling (https://www.moodysanalytics.com/risk-perspectives-magazine/managing-disruption/spotlight/machine-learning-challenges-lessons-and-opportunities-in-credit-risk-modeling)

Mark, D. M., & Peucker, T. K. (2020, November 27). Regression Analysis and Geographic Models. ResearchGate. https://www.researchgate.net/post/what_is_a_good_r_square_value_in_regression_analysis (https://www.researchgate.net/post/what_is_a_good_r_square_value_in_regression_analysis)

Mean Squared Error: Definition and Example. (2021, June 26). Statistics How To. https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/mean-squared-error/

(https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/mean-squared-error/)

Orecchio-Egresitz, H. (2020, October 28). With limited access to resources, high unemployment, and the mental health nightmare of 2020, it's no surprise crime is up in New York City. Business Insider Australia. https://www.businessinsider.com.au/unemployment-mental-health-burdens-2020-crime-2020-10 (https://www.businessinsider.com.au/unemployment-mental-health-burdens-2020-crime-2020-10)

Rolfe, B. (2021, October 5). Figure reveals Victorian welfare spike. News. https://www.news.com.au/national/victoria/news/number-of-victorians-receiving-jobseeker-and-youth-allowance-has-increased-by-nearly-50-per-cent-during-covid/news-story/b916c509220539eb3a1f31c6f7164282 (https://www.news.com.au/national/victoria/news/number-of-victorians-receiving-jobseeker-and-youth-allowance-has-increased-by-nearly-50-per-cent-during-covid/news-story/b916c509220539eb3a1f31c6f7164282)

Team, K. (n.d.). Keras documentation: Callbacks API. Keras. Retrieved October 13, 2021, from https://keras.io/api/callbacks/#earlystopping (https://keras.io/api/callbacks/#earlystopping)

Wikipedia contributors. (2021, May 25). 1979–1983 Eastern Australian drought. Wikipedia. https://en.wikipedia.org/wiki/1979%E2%80%931983_Eastern_Australian_drought (https://en.wikipedia.org/wiki/1979%E2%80%931983_Eastern_Australian_drought)

Parliamentary Research Service. (1994, June). From There to Back Again?: Australian Inflation and Unemployment 1964 to 1993 (No. 9). Department of the Parliamentary Library. https://www.aph.gov.au/binaries/library/pubs/bp/1994-95/94bp09.pdf (https://www.aph.gov.au/binaries/library/pubs/bp/1994-95/94bp09.pdf)

Carl Walsh. (1993, February). What Caused the 1990–1991 Recession? University of California, Santa Cruz. https://www.researchgate.net/publication/5033434_What_Caused_the_1990-1991_Recession (https://www.researchgate.net/publication/5033434_What_Caused_the_1990-1991_Recession)

SBS Australia. (2021, July 16). Lockdowns in Victoria, NSW expected to drive unemployment and stall economic recovery. SBS News. https://www.sbs.com.au/news/lockdowns-in-victoria-nsw-expected-to-drive-unemployment-and-stall-economic-recovery/f7f0d51d-688f-422d-8466-db38f1303f23 (https://www.sbs.com.au/news/lockdowns-in-victoria-nsw-expected-to-drive-unemployment-and-stall-economic-recovery/f7f0d51d-688f-422d-8466-db38f1303f23)

Bhatia, N. (2019, June 27). What is Out of Bag (OOB) score in Random Forest? - Towards Data Science. Medium. https://towardsdatascience.com/what-is-out-of-bag-oob-score-in-random-forest-a7fa23d710 (https://towardsdatascience.com/what-is-out-of-bag-oob-score-in-random-forest-a7fa23d710)

Appendix

R code used to produce all of the above:

```
gen_loc = "D:/Uni/MA5832_AdvMachineLearning/Assignment 3/"
usRp <-"https://cran.csiro.au/"


if(!require(ggplot2))   install.packages("ggplot2" ,repos = usRp)
if(!require(caret))     install.packages("caret"   ,repos = usRp)
if(!require(dplyr))     install.packages("dplyr"   ,repos = usRp)
if(!require(MASS ))     install.packages("MASS"    ,repos = usRp)
if(!require(sqldf))     install.packages("sqldf"   ,repos = usRp)
if(!require(GGally))    install.packages("GGally"  ,repos = usRp)
if(!require(qqplotr))   install.packages("qqplotr" ,repos = usRp)
if(!require(corrplot))  install.packages("corrplot",repos = usRp)
if(!require(randomForest))  install.packages("randomForest",repos = usRp)
if(!require(rpart))  install.packages("rpart",repos = usRp)
if(!require(rpart.plot))  install.packages("rpart.plot",repos = usRp)
if(!require(e1071))  install.packages("e1071",repos = usRp)
if(!require(rminer))  install.packages("rminer",repos = usRp)
if(!require(rattle))  install.packages("rattle",repos = usRp)
if(!require(RColorBrewer))  install.packages("RColorBrewer",repos = usRp)
if(!require(quadprog)) install.packages("quadprog")
if(!require(data.table)) install.packages("data.table")
if(!require(forcats)) install.packages("forcats")
if(!require(psych)) install.packages("psych")
if(!require(ggpubr)) install.packages("ggpubr")
if(!require(mice)) install.packages("mice")
if(!require(Metrics)) install.packages("Metrics")
if(!require(tfdatasets)) install.packages("tfdatasets")




#elemstatlrnLOC <- paste0(gen_loc,"ElemStatLearn_2015.6.26.tar.gz")         # specify
file name for import
#install.packages(elemstatlrnLOC, repos=NULL, type="source" )
library(Metrics)
library(caret)
library(forcats)
library(dplyr)
library(MASS)
library(sqldf)
library(GGally)
library(qqplotr)
library(corrplot)
library(randomForest)
library(rpart)
library(rpart.plot)
library(e1071)
#library(ElemStatLearn)
library(data.table)
library(rminer)
library(rattle)
```

```
library(RColorBrewer)
library(quadprog)
library(psych)
library(ggpubr)
library(mice)
library(tfdatasets)
library(keras)
library(tensorflow)
ggplot(aus_data, aes(x=X, y=Y)) +
    geom_line() +
    labs(x = "YYYY-MM", y = "Unemployment Rate") +
    ggtitle("Australian Unemployment Rate: 1981 - 2020")
# define seed for both R and keras packages (used later on)
seed <- 5555
set.seed(seed)
set_random_seed(seed)

# import data
aus_data_pre <- paste0(gen_loc,"AUS_Data.csv")
aus_data <- read.table(aus_data_pre, header=TRUE , sep = ",", dec = ".")

#drop 1st row as it has headers
aus_data <- aus_data[-1,]
#drop last row as it's empty
aus_data <- aus_data[-159,]

# drop empty column
aus_data <- subset(aus_data, select = -c(X.1))

# replace missing values for population - sourced from:
# https://www.abs.gov.au/statistics/people/population/national-state-and-territory-po
pulation/mar-2021
aus_data$X7 <- with(aus_data,   ifelse(X == "Sep-2019",254722.6,
                                ifelse(X == "Dec-2019",255579.1,
                                ifelse(X == "Mar-2020",256686.5,
                                ifelse(X == "Jun-2020",256933.4,
                                ifelse(X == "Sep-2020",256809.4,aus_data$X7))))))

# add 01 to dates as they are currently missing the day
aus_data$X <- paste0("01-",aus_data$X)
# then convert to date and format
aus_data$X <- as.Date(format(strptime(aus_data$X, format = "%d-%b-%Y"), "%d/%m/%
Y"),"%d/%m/%Y")


# convert stated variables to int
varsToINT <- c("Y","X1","X2","X3","X4","X5","X6","X7")
aus_data[,varsToINT] <- lapply(aus_data[,varsToINT],as.numeric)

#### DATA IS NOW CLEAN ####
```

```
str(aus_data)

#============================
# start of subset imputation
#============================
# subset data so there are 12 observations prior to start of missing data, and 12 obs
ervations post end of missing data
aus_data_sub  <- subset(aus_data, aus_data$X >= "2007-03-01" & aus_data$X <= "2011-06
-01")

# take remainder of observations into separate dataframe so the two can be stacked la
ter
aus_data_sub2 <- subset(aus_data, aus_data$X < "2007-03-01" | aus_data$X > "2011-06-0
1")

# missing data in dataset needs to be imputed
aus_data_imp <- mice(aus_data_sub,maxit=50,meth='norm.predict',seed=seed)

# impute missing values
aus_data_imputed <- complete(aus_data_imp,1)

# combine both dataframes
imp_aus_data <- rbind(aus_data_sub2,aus_data_imputed)

# reorder this dataframe so rows are in chronological order
imp_aus_data <- sqldf(" SELECT  *
                        FROM       imp_aus_data
                        ORDER BY X asc")

#============================================================================
================================
# start of none-subset imputation
aus_data_imp2 <- mice(aus_data,maxit=50,meth='norm.predict',seed=500)
aus_data_imputed2 <- complete(aus_data_imp2,1)
# end of none-subset imputation
#============================================================================
================================

# now subset the data so it contains observations
aus_data_train <- subset(imp_aus_data, imp_aus_data$X <  "2018-03-01")
aus_data_test  <- subset(imp_aus_data, imp_aus_data$X >= "2018-03-01")

par(mfrow=c(1,3)) # 3 charts side by side

#show comparison of applying imputation using subset data vs without subset data
par(mfrow=c(3,1)) # one chart at one time
ggplot(aus_data, aes(x=X, y=X6)) +
    geom_line() +
    ggtitle("No Imputation") +
       ylab("X6: # of Job Vacancies") +
       xlab("X: Year")
```

```
ggplot(imp_aus_data, aes(x=X, y=X6)) +
    geom_line() +
    ggtitle("Imputation w Subset Data") +
       ylab("X6: # of Job Vacancies") +
       xlab("X: Year")

ggplot(aus_data_imputed2, aes(x=X, y=X6)) +
    geom_line() +
    ggtitle("Imputation w/o Subset Data") +
       ylab("X6: # of Job Vacancies") +
       xlab("X: Year")

summary(imp_aus_data)
library(randomForest)
library(mlbench)
library(caret)

control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")
set.seed(seed)
tunegrid <- expand.grid(.mtry=c(1:15))
rf_gridsearch <- caret::train(Y~., data=aus_data_train, method="rf", metric="RMSE", t
uneGrid=tunegrid, trControl=control)
print(rf_gridsearch)
plot(rf_gridsearch)

library(randomForest)
aus_data_rforest <-randomForest(Y ~., data=aus_data_train, mtry=8, importance=TRUE, n
tree=500)

#setting mtry = number of predictors to get bagging results
print(aus_data_rforest)
# Find the optimal mtry value
mtry <- tuneRF(aus_data_train[-2],aus_data_train$Y, ntreeTry=500,
               stepFactor=1.5,improve=0.01, trace=TRUE, plot=TRUE)
best_mtry <- mtry[mtry[, 2] == min(mtry[, 2]), 1]
print(mtry)
print(best_mtry)
aus_data_rforest <-randomForest(Y ~., data=aus_data_train, mtry=best_mtry, importance
=TRUE, ntree=500)

res <- aus_data_train$Y-aus_data_rforest$predicted
plot(aus_data_train$Y, res)

#aus_data_rforest$importance
round(randomForest::importance(aus_data_rforest),2)
varImpPlot(aus_data_rforest)

plot(aus_data_rforest)
print(aus_data_rforest)
## "x" can be a matrix instead of a data frame:
```

```
x <- matrix(runif(5e2), 100)
y <- gl(2, 50)
(myrf <- randomForest(x, y))
(predict(aus_data_rforest, aus_data_test))

predictions <- predict(aus_data_rforest, aus_data_test)

result <- aus_data_test
result['Y'] <- aus_data_test$Y
result['prediction']<-  predictions
result

ggplot(  ) +
    geom_line( aes(x = aus_data_test$X, y = aus_data_test$Y, color = 'red', alpha =
0.5) ) +
    geom_line( aes(x = aus_data_test$X , y = predictions, color = 'blue',  alpha = 0.
5)) +
    labs(x = "YYYY-MM", y = "Unemployment Rate", color = "", ) +
    scale_color_manual(labels = c( "Predicted", "Real"), values = c("blue", "red"))
library(Metrics)
print(paste0('MAE: ' , mae(aus_data_test$Y,predictions)))
print(paste0('MAPE: ' , mape(aus_data_test$Y,predictions)))
print(paste0('MSE: ' ,caret::postResample(predictions , aus_data_test$Y)['RMSE']^2 ))
print(paste0('R2: ' ,caret::postResample(predictions , aus_data_test$Y)['Rsquared']
))
print(paste0('RMSE: ' ,caret::postResample(predictions , aus_data_test$Y)['RMSE'] ))

#======================
#    ======================
#        ======================
# NEURAL NETWORK
#        ======================
#    ======================
#======================

# get current time to check duration of the neural network run time at the end
start_time <- Sys.time()

# set the seeds
set.seed(seed)
set_random_seed(seed)

# define parameters used in nn
act = "relu"    # activation function
batchSize = 64  # batch size for iterations
unit = 1        # number of rows for output layer at one time
verb = 0        # 1 for verbose notes, 0 for nonverbose notes
valSplit = 0.2  # validation split (% of dataset to be withheld for validation)
epoch_n = 20  # epoch number (total iterations of dataset)
patnce = 4      # patience for early stopping
```

```
#drop the date
aus_train_nn <- aus_data_train[-1]
aus_test_nn <- aus_data_test[-1]

set.seed(seed)
set_random_seed(seed)

# scale numeric columns using python
spec <- feature_spec(aus_train_nn, Y ~ . ) %>%
    step_numeric_column(all_numeric(), normalizer_fn = scaler_standard()) %>%
    fit()

spec
library(keras)

set.seed(seed)
set_random_seed(seed)

# Create a list of Keras input layers
input <- layer_input_from_dataset(aus_train_nn)
output <- input %>%
    layer_dense_features(dense_features(spec))   %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
    layer_dense(units = unit)

model <- keras_model(input, output)

summary(model)

model %>%
    compile(
        loss = "mse",
        optimizer = optimizer_sgd(),
        metrics = list("mean_absolute_error","mae","mape","mse")
    )

set.seed(seed)
set_random_seed(seed)


build_model <- function() {
    input <- layer_input_from_dataset(aus_train_nn[-1])
```

```
        output <- input %>%
            layer_dense_features(dense_features(spec))   %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = batchSize, activation = act) %>%
            layer_dense(units = unit)

    model <- keras_model(input, output)

    model %>%
        compile(
            loss = "mse",
            optimizer = optimizer_sgd(),
            metrics = list("mean_absolute_error","mae","mape","mse"))

    model
}

set.seed(seed)
set_random_seed(seed)

print_dot_callback <- callback_lambda(  on_epoch_end = function(epoch, logs) {
                                        if (epoch %% 80 == 0) cat("\n")
                                        cat(".") } )

set.seed(seed)
set_random_seed(seed)

model <- build_model()
history <- model %>% fit(x = aus_train_nn[-1],
                         y = aus_train_nn$Y,
                    epochs = epoch_n,
          validation_split = valSplit,
                   verbose = verb,
                 callbacks = list(print_dot_callback))
set.seed(seed)
set_random_seed(seed)
plot(history)

early_stop <- callback_early_stopping(monitor = "val_loss", patience = patnce)

model <- build_model()

history <- model %>% fit(
```

```
    x = aus_train_nn[-1],
    y = aus_train_nn$Y,
    epochs = epoch_n,
    validation_split = valSplit,
    verbose = verb,
    callbacks = list(early_stop)
)

set.seed(seed)
set_random_seed(seed)

plot(history)

set.seed(seed)
set_random_seed(seed)

set.seed(seed)
set_random_seed(seed)

# assign performance metrics
c(loss, mean_absolute_error,mae,mape,mse) %<-% (model %>% evaluate(aus_test_nn , aus_
data_test$Y, verbose = 0))

xxx <- (model %>% evaluate(aus_test_nn , aus_data_test$Y, verbose = 0))

# xxx
# loss
# mean_absolute_error
# mae
# mape
# mse

test_predictions <- model %>% predict(aus_test_nn[-1])
test_predictions[,1]

paste0("Mean absolute error on test set: ", round(mae,2))
paste0("Mean absolute percentage error on test set: ", round(mape,2),"%")
paste0("Mean squared error on test set: ", round(mse,2))
paste0("Inferred accuracy: ", round(1-(mape/100),4)*100,"%")
print("Predicted Values for Test set: ")
test_predictions[,1]


end_time <- Sys.time()
end_time - start_time

ggplot(  ) +
    geom_line( aes(x = aus_data_test$X, y = aus_data_test$Y, color = 'red', alpha =
0.5) ) +
    geom_line( aes(x = aus_data_test$X , y = test_predictions[,1], color = 'blue',  a
lpha = 0.5)) +
```

```
    labs(x = "YYYY-MM", y = "Unemployment Rate", color = "", ) +
    scale_color_manual(labels = c( "Predicted", "Real"), values = c("blue", "red"))
ggplot(  ) +
    geom_line( aes(x = aus_data_test$X, y = aus_data_test$Y, color = 'red', alpha =
0.5) ) +
    geom_line( aes(x = aus_data_test$X , y = predictions, color = 'blue',  alpha = 0.
5)) +
    geom_line( aes(x = aus_data_test$X , y = test_predictions[,1], color = 'black',
alpha = 0.5)) +
    labs(x = "YYYY-MM", y = "Unemployment Rate", color = "", ) +
    scale_color_manual(labels = c( "NN Predicted","RF Predicted", "Real"), values = c
("blue", "red","black"))
```