Forecasting Volatility

Hendrik Adriaan Nieuwenhuizen

1. Introduction

The dataset that I will be using for this project is the US Dollar versus the South African Rand exchange rate from January 2000 to November 2020, I exported the data from a reputable source and saved in my Github account.

The goal of the project is to try and see if I can forecast the next 7 day volatility of the USDZAR exchange rate with the 30 day rolling volatility of this exchange rate as the predictor.

“Volatility is a statistical measure of the dispersion of returns for a given security or market index. In most cases, the higher the volatility, the riskier the security. Volatility is often measured as either the standard deviation or variance between returns from that same security or market index.” (Investopedia)

I work in the financial services industry and forecasted volatility can be used in options pricing using the Black and Scholes pricing formula to hedge currency risk.

The outcome and the predicor will be continuous and in percentage format. We use SSE and RMSE as loss functions where applicable.

I will start off by looking at linear regression as our baseline, then we will try to see which machine learning model has the best forecasting ability. For this project we will use K-nearest neighbours, regression trees and neural networks. For the machine learning we will split the data into a training and test set. We will use RMSE and SSE as loss functions to measure the efficacy for the continuous outcomes.

2. Method

2.1 Dataset

Currency dataset is the US Dollar versus the South African Rand exchange rate from January 2000 to November 2020. Dataset has 5 columns with 5426 rows.

Train set has 3796 lines and test set has 1630 lines and both have the same headings as the currency dataset from which they are derived.

For the train and test datasets I chose a 70/30 split. From what I could gather online most data scientists recommend the 70/30 split to be sufficient.

```
head(currency)   #tibble of currency dataset
```

```
## # A tibble: 6 x 7
##   Date       USDZAR.Curncy LOG.RETURNS ROLLING.30.DAY.~ ROLLING.7.DAY.V~
##   <date>             <dbl>       <dbl>            <dbl>            <dbl>
## 1 2000-02-04          6.26    0.000480           0.0899           0.0829
## 2 2000-02-07          6.28    0.00431            0.0867           0.0914
## 3 2000-02-08          6.29    0.000477           0.0852           0.0649
## 4 2000-02-09          6.33    0.00713            0.0871           0.0742
## 5 2000-02-10          6.32   -0.00198           0.0863           0.0573
## 6 2000-02-11          6.33    0.00229           0.0862           0.0555
## # ... with 2 more variables: DAILY.VOL <dbl>, NEXT.7.DAY.VOL <dbl>
```

```
dim(currency)    #total lines in currency dataset
```

```
## [1] 5426    7
```

```
dim(train_set)   #total lines in train dataset
```

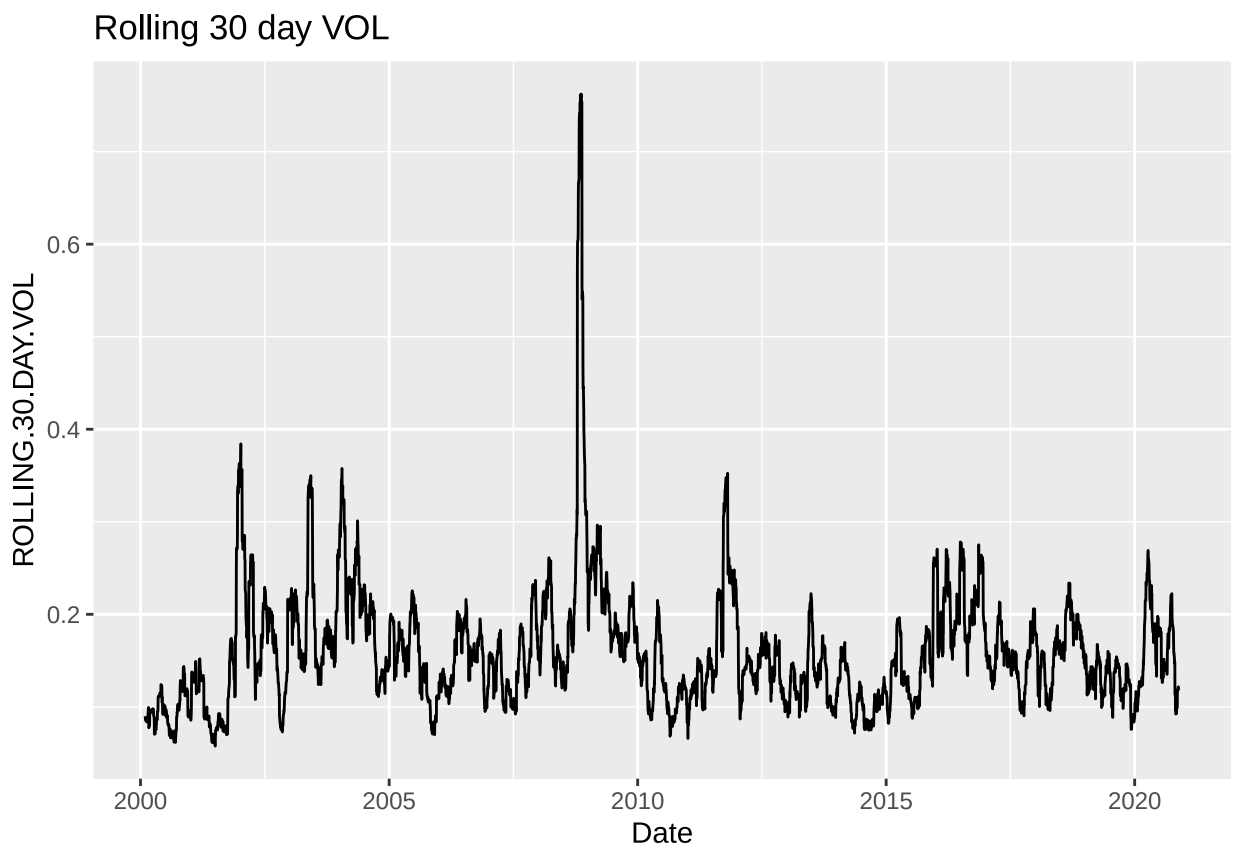
```
## [1] 3796    7
```

```
dim(test_set)    #total lines in test dataset
```

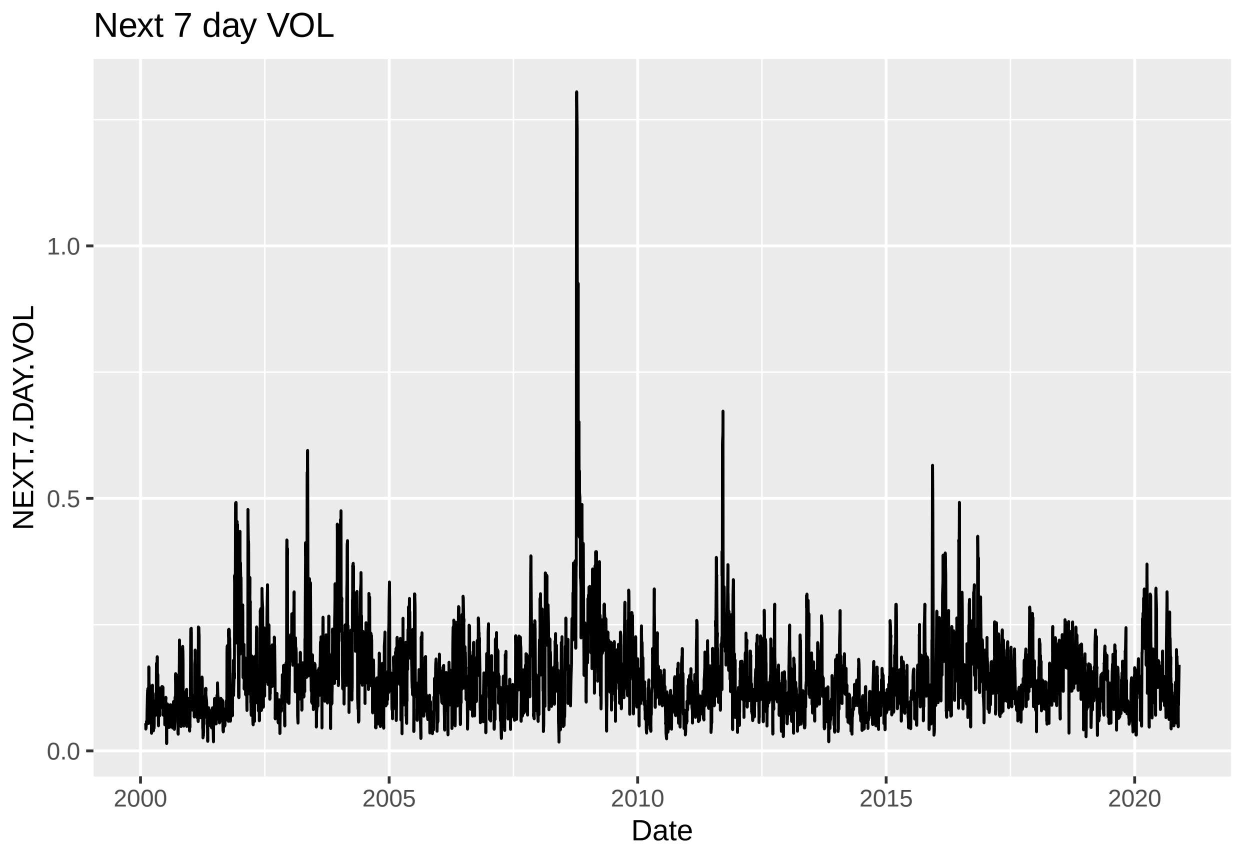
```
## [1] 1630    7
```

Below we show the 30 day rolling volatility (calculated on the last 24 business days historical data) and the 7 day forecasted volatility (based on historical data but forward looking for the next 5 business days) over time. We can clearly see the volatility spiked during the 2008 global financial crisis as know as the "GFC".

```
currency %>% ggplot(aes(Date, ROLLING.30.DAY.VOL)) +
  ggtitle("Rolling 30 day VOL") +
  geom_line()
```



```r
currency %>% ggplot(aes(Date, NEXT.7.DAY.VOL)) +
  ggtitle("Next 7 day VOL") +
  geom_line()
```



2.2 Linear regression

For the first model I try to explain the outcome with 1 predictor using a linear regression model. Linear regression is seen as too rigid but it will serve as our baseline. We also compare this against simply guessing the outcome to see if the linear regression is actually explaining the outcome.

2.3 KNN

For the first machine learning model I use K-nearest neighbours to see if we can beat linear regression. KNN works with continuous outcomes and should be able to provide a better forecast than linear regression. I will also use cross validation to optimise the k which is the parameter that represents the number of neighbours to include.

2.4 Regression trees

For the second machine learning model I use regression trees which are used when the dependent variable is continuous. In continuous, a value obtained is a mean response of observation. We use cross validation to optimise the complexity parameter(cp).

2.5 Neural Net

For the third machine learning model I use neural networks, I tried a number of different parameter settings with over 5 layers, 5 neurons and 5 repititions. I only show the most relevant results that made sense. Overfitting is also a frequent problem in machine learning, the overfitted model doesn't generalise well. Neural networks can prevent overfitting.

3. Results

3.1 Linear regression and guessing

We first start with guessing the outcome as a way to see if the linear regression makes sense. The guess has a RMSE of 8.83%.

```
#guessing
guess <- mean(train_set$NEXT.7.DAY.VOL)
guess
```

```
## [1] 0.1507467
```

```
mean((guess - test_set$NEXT.7.DAY.VOL)^2)   #squared loss of the guess
```

```
## [1] 0.007793653
```

```
RMSE(test_set$NEXT.7.DAY.VOL, guess)     #RMSE 8.83%
```

```
## [1] 0.08828167
```

The linear regression has a RMSE of 7.63% so the linear regression is more accurate in forecasting than simple guessing.

I use the root mean squared error(RMSE) as loss function for a continuous variable.

```
#linear regression

fit <- lm(NEXT.7.DAY.VOL ~ ROLLING.30.DAY.VOL, data=train_set)
fit$coef
```

```
##        (Intercept) ROLLING.30.DAY.VOL
##         0.05279729         0.61173783
```

```
y_hat <- fit$coef[1] + fit$coef[2]*test_set$ROLLING.30.DAY.VOL
mean((y_hat - test_set$NEXT.7.DAY.VOL)^2)                 #squared loss of linear regression is lower t
```

```
## [1] 0.005817543
```

```
y_hat1 <- predict(fit, test_set)
mean((y_hat1 - test_set$NEXT.7.DAY.VOL)^2)  #using predict to calculate outcome
```

```
## [1] 0.005817543
```

```
LinearReg_RMSE <- RMSE(test_set$NEXT.7.DAY.VOL, y_hat)      #RMSE 7.63%
LinearReg_RMSE
```

```
## [1] 0.07627282
```

The correlation between the outcome and predictor has a correlation of 0.49 indicating that there is a moderate positive correlation.

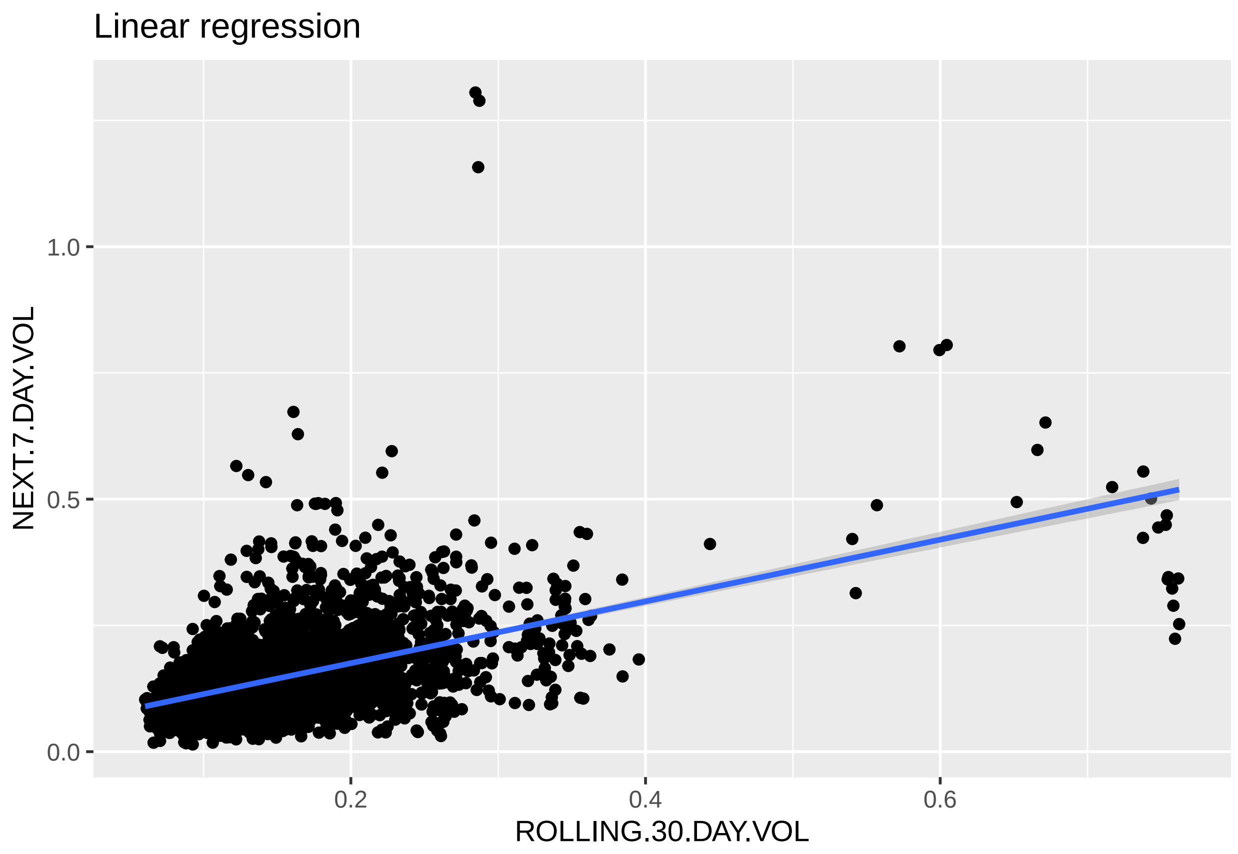
```r
cor(train_set$NEXT.7.DAY.VOL , train_set$ROLLING.30.DAY.VOL)
```

```
## [1] 0.4881428
```

From the graph below we can surmise that forecasting the next 7 day vol with the rolling 30 day vol might be a challenge. The data appears to be non-linear.

```r
data <- data.frame(train_set)
ggplot(data, aes(ROLLING.30.DAY.VOL, NEXT.7.DAY.VOL)) +
  geom_point() +
  geom_smooth(method='lm') +
  ggtitle("Linear regression") +
  xlab("ROLLING.30.DAY.VOL") +
  ylab("NEXT.7.DAY.VOL")
```

```
## 'geom_smooth()' using formula 'y ~ x'
```

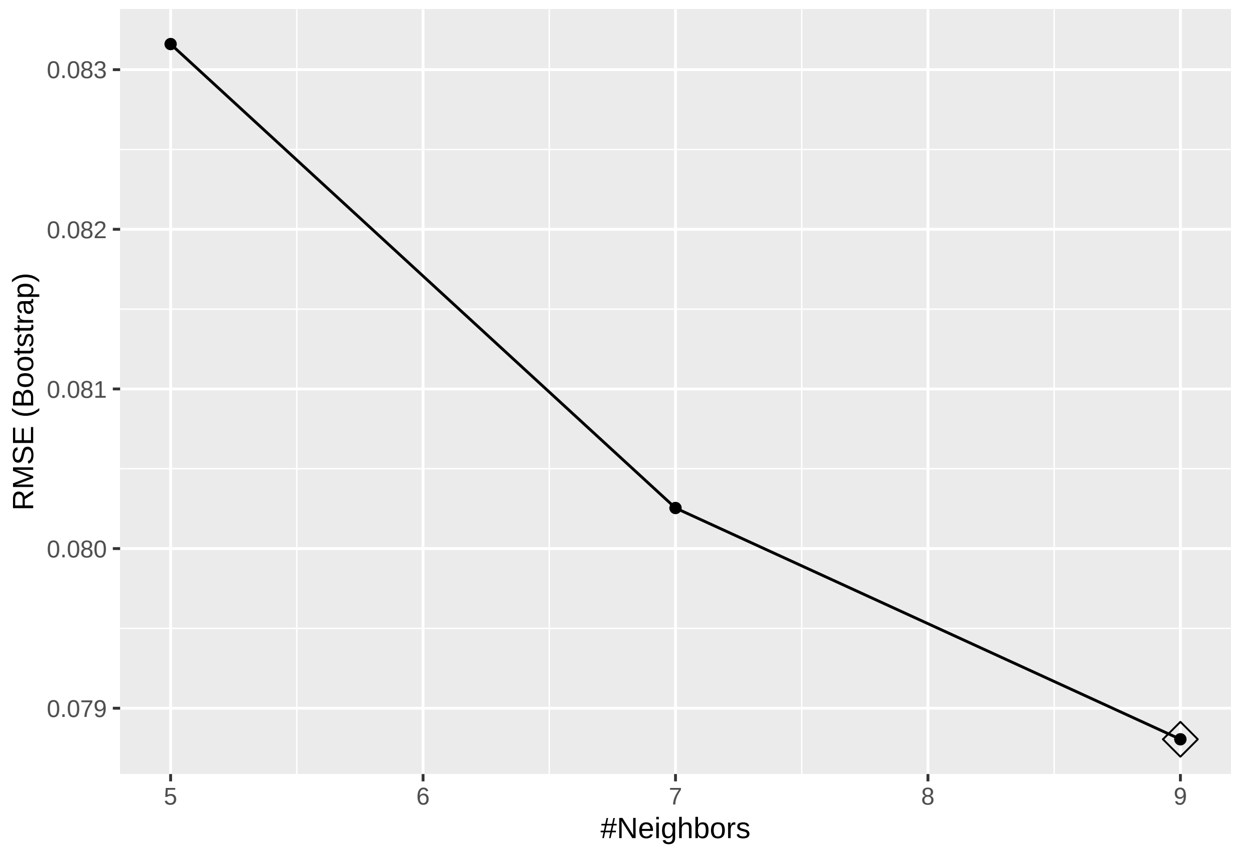


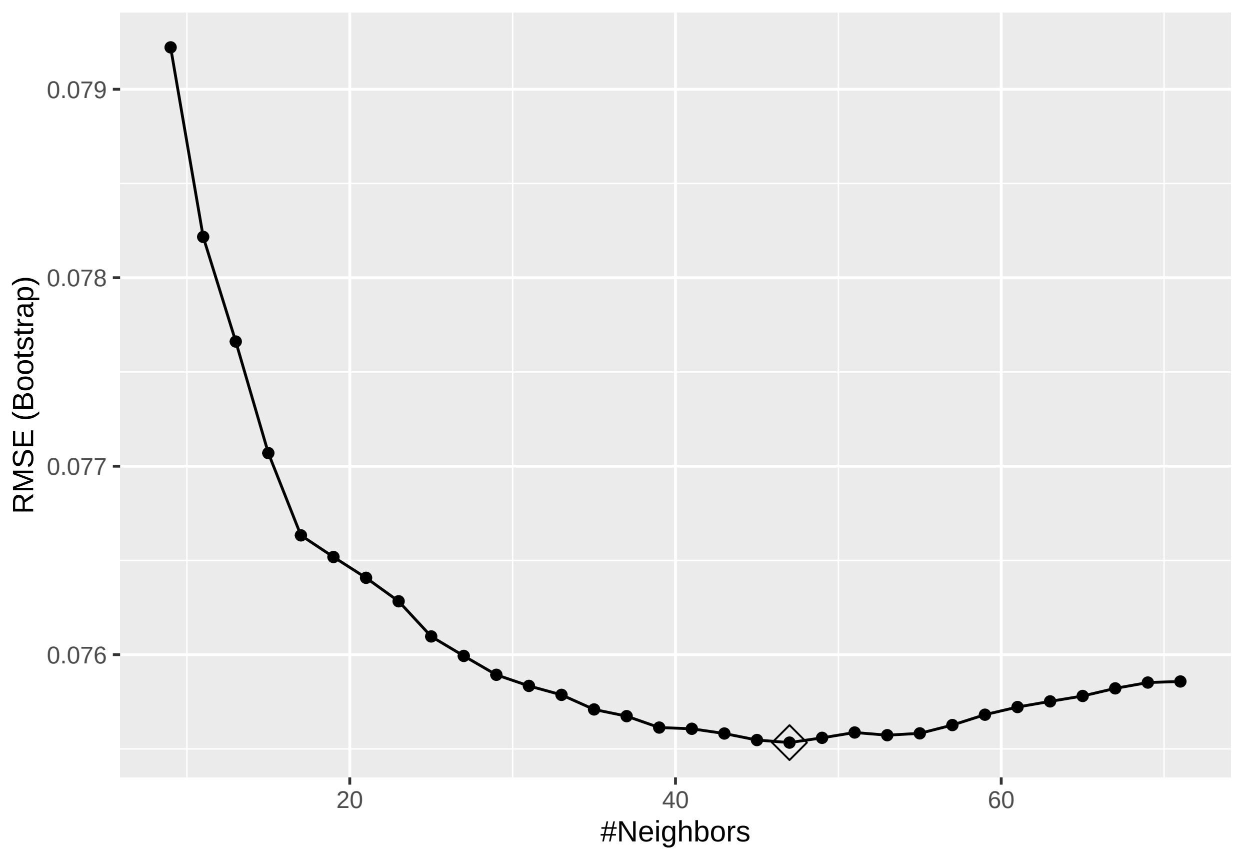
3.2 KNN

For the K-nearest neighbours model we will train the model with the train and test set.

The below graph shows the results of the cross validation function. The default parameter recommends K = 9 to get the lowest RMSE.

```r
train_knn <- train(  NEXT.7.DAY.VOL ~ ROLLING.30.DAY.VOL, method = "knn", data = train_set)  #by defaul

ggplot(train_knn, highlight = TRUE)        #see results of cross validation and use highlight to show p
```





Now we add the tuneGrid function to try more options for k. The parameter that maximises K's accuracy is 47.

```
train_knn$bestTune #this is the best parameter that maximizes accuracy
```

```
##     k
## 20 47
```

```
train_knn$finalModel         #best performing model,  all up to now was on training set, not test set.
```

```
## 47-nearest neighbor regression model
```

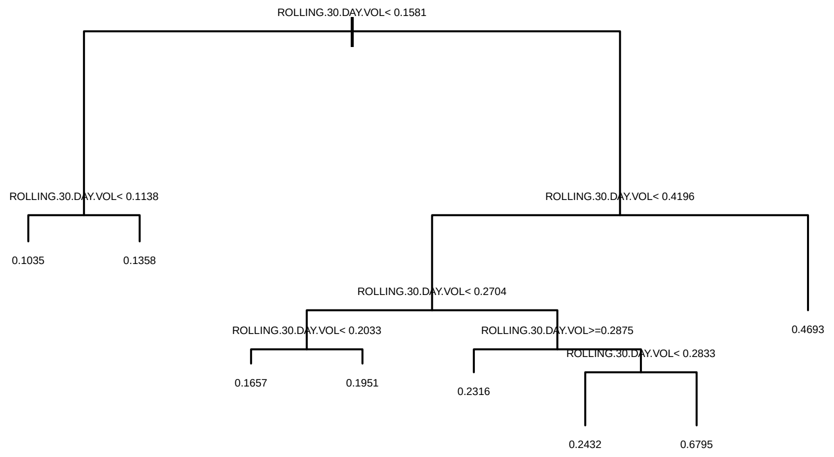
Next we run the knn model that was optimised and predict the outcome and compare the results to the test set. The RMSE for this exercise is 17.49%. This result is very disappointing since this RMSE is higher than the RMSE for the linear regression of 7.63%.

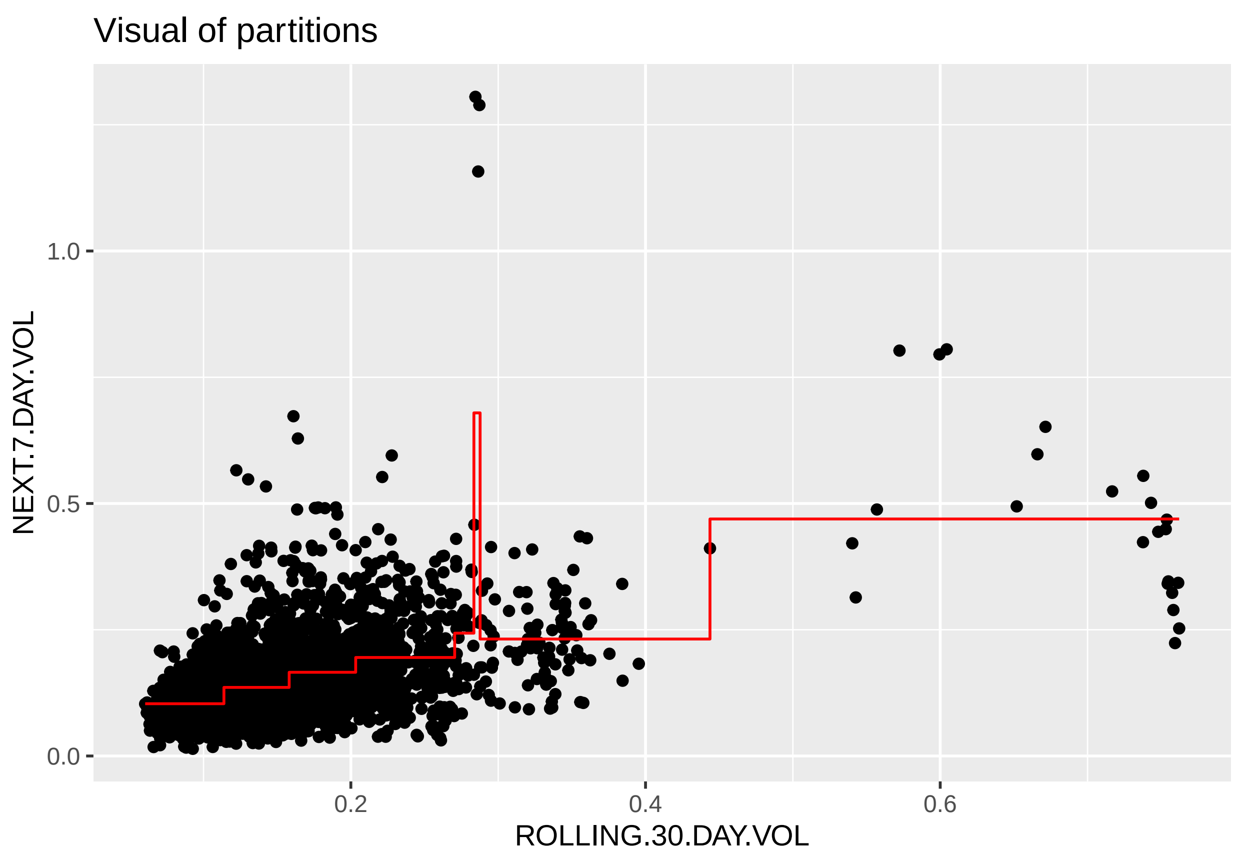
```
knn_fit <- knn3( NEXT.7.DAY.VOL ~ ROLLING.30.DAY.VOL, data = train_set, k=47)  #number of neighbours to

y_hat_knn <- predict(knn_fit, test_set)

KNN_RMSE <- RMSE(test_set$NEXT.7.DAY.VOL, y_hat_knn)    #RMSE 17.49%
KNN_RMSE
```

```
## [1] 0.1748989
```

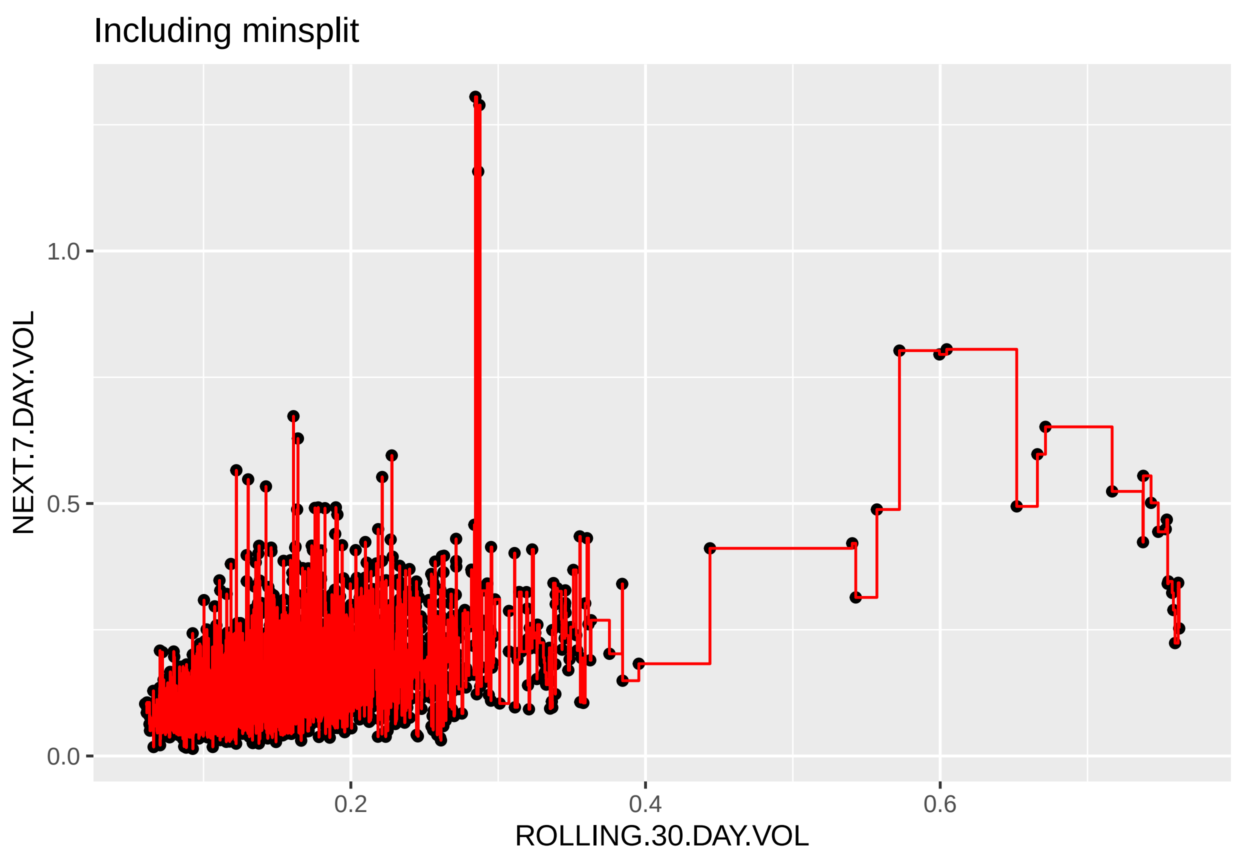
3.3 Regression tree

In this section we look at regression trees. To start we run rpart and we get a plot showing that the first split happens at 15.81% with 8 partitions.

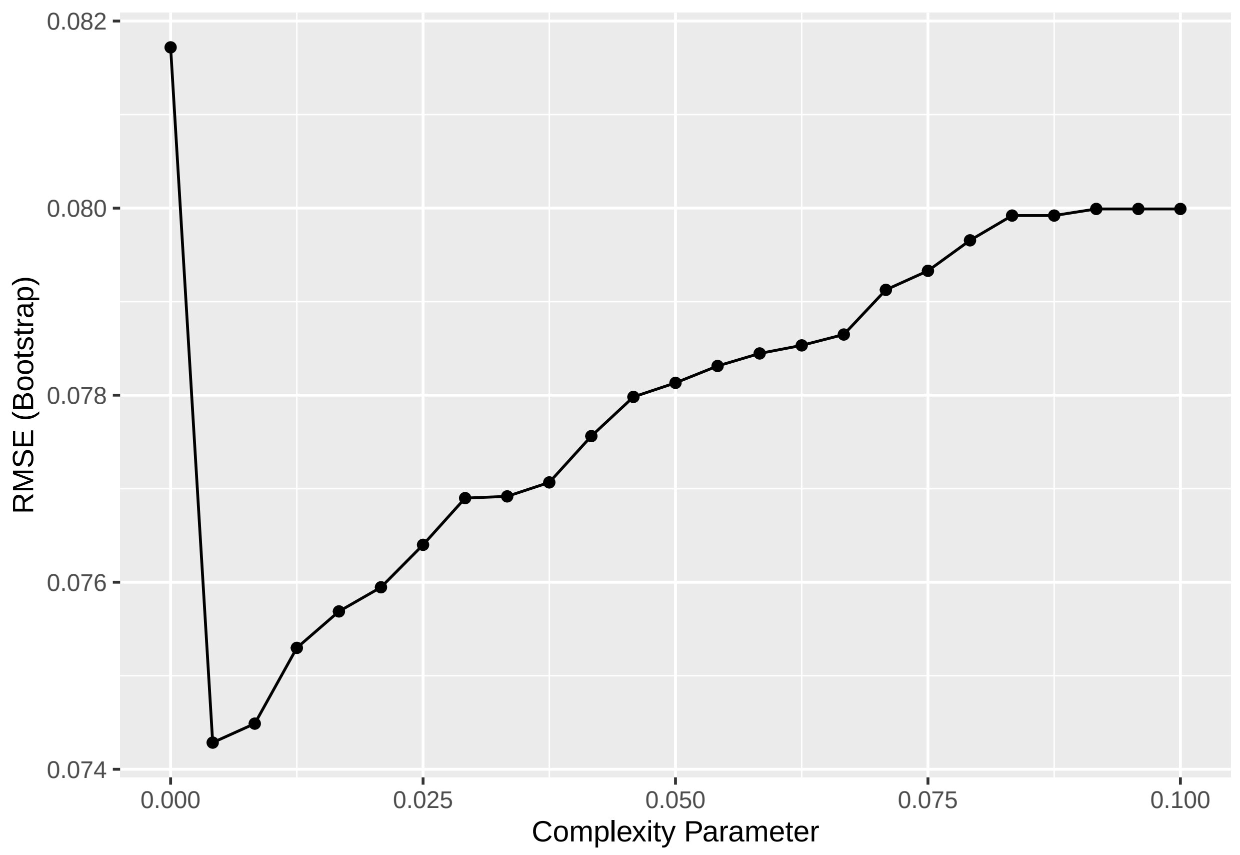

ROLLING.30.DAY.VOL< 0.1581
ROLLING.30.DAY.VOL< 0.1138
ROLLING.30.DAY.VOL< 0.4196
0.1035
0.1358
ROLLING.30.DAY.VOL< 0.2704
ROLLING.30.DAY.VOL< 0.2033
ROLLING.30.DAY.VOL>=0.2875
0.4693
ROLLING.30.DAY.VOL< 0.2833
0.1657
0.1951
0.2316
0.2432
0.6795



The final estimate f_hat of x. Every time we split RSS decreases. With more partitions our model has more flexibility to adapt to the training data.



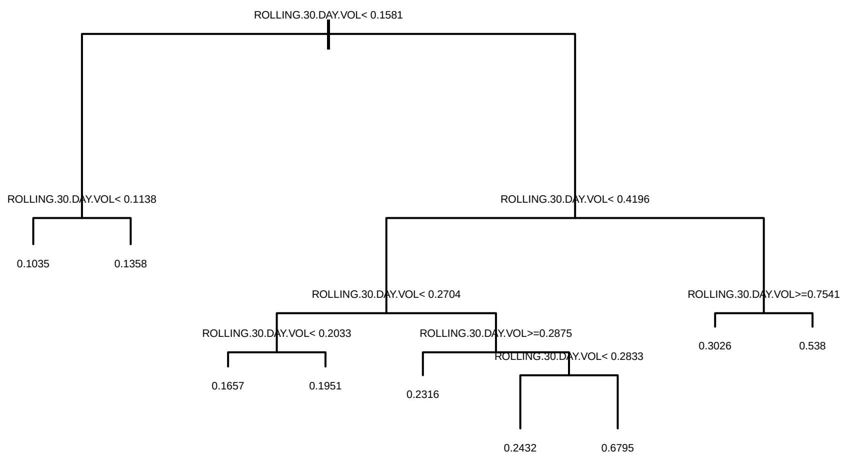
A way to decide if the current partition should partition further is to use minsplit in the rpart function. This parameter is based on the minimum number of observations required to continue to partition further. This could lead to over training.



The complexity parameter (cp) that is most effective is 0.004166 as shown below. You can continue splitting until RSS drops to zero but with the cp parameter we set the minimum for the RSS to add another partition. We use cross validation to optimise the cp.

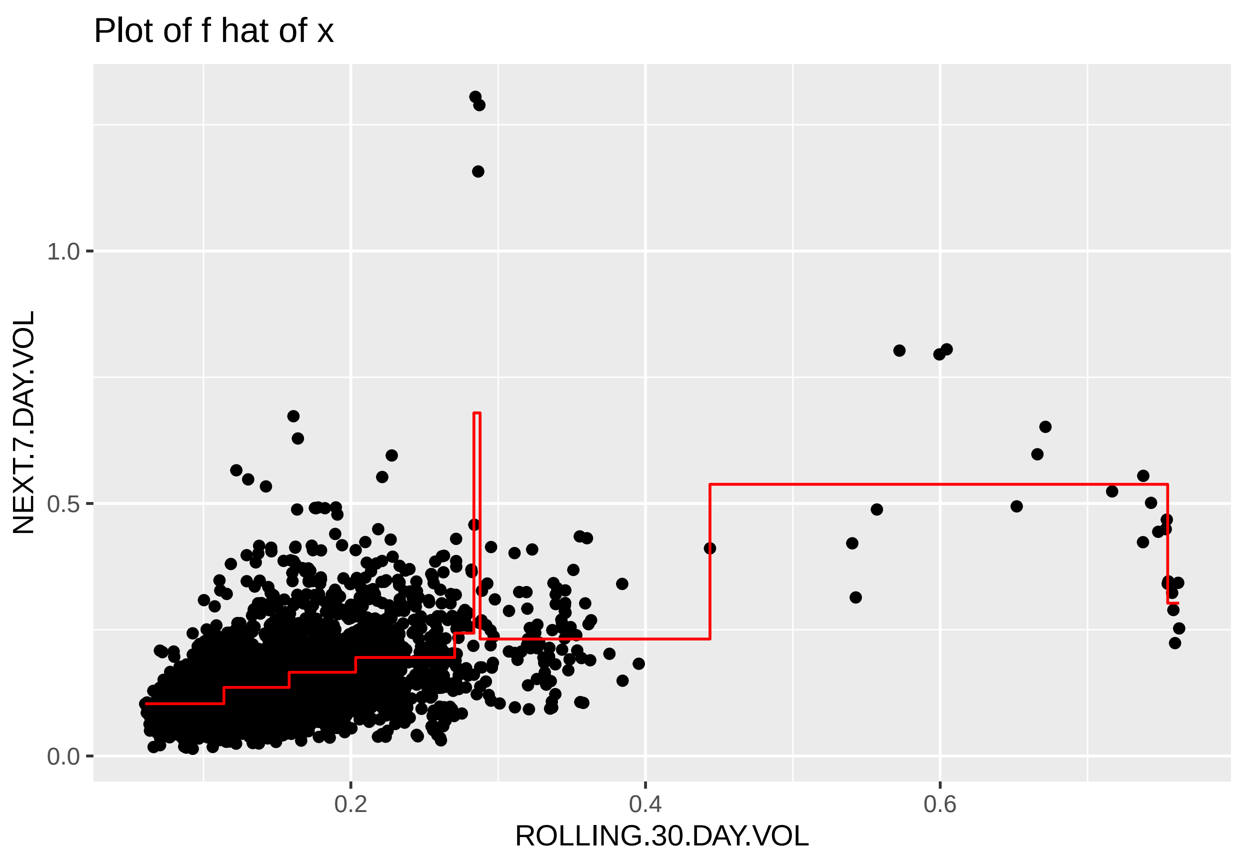
The final tree with the optimised k minimizes the mean squared error.

```r
# access the final model and plot it
plot(train_rpart$finalModel, margin = 0.1)
text(train_rpart$finalModel, cex = 0.3)
```



ROLLING.30.DAY.VOL< 0.1581
ROLLING.30.DAY.VOL< 0.1138
ROLLING.30.DAY.VOL< 0.4196
0.1035
0.1358
ROLLING.30.DAY.VOL< 0.2704
ROLLING.30.DAY.VOL>=0.7541
ROLLING.30.DAY.VOL< 0.2033
ROLLING.30.DAY.VOL>=0.2875
0.3026
0.538
ROLLING.30.DAY.VOL< 0.2833
0.1657
0.1951
0.2316
0.2432
0.6795

The regression tree has now been optimised compared to the initial tree. Using a regression tree approach doesn't provide specific outcomes but useful to use as a broad tool to forecast volatility between specified levels like if volatility > x then expect high volatility in the next few days.

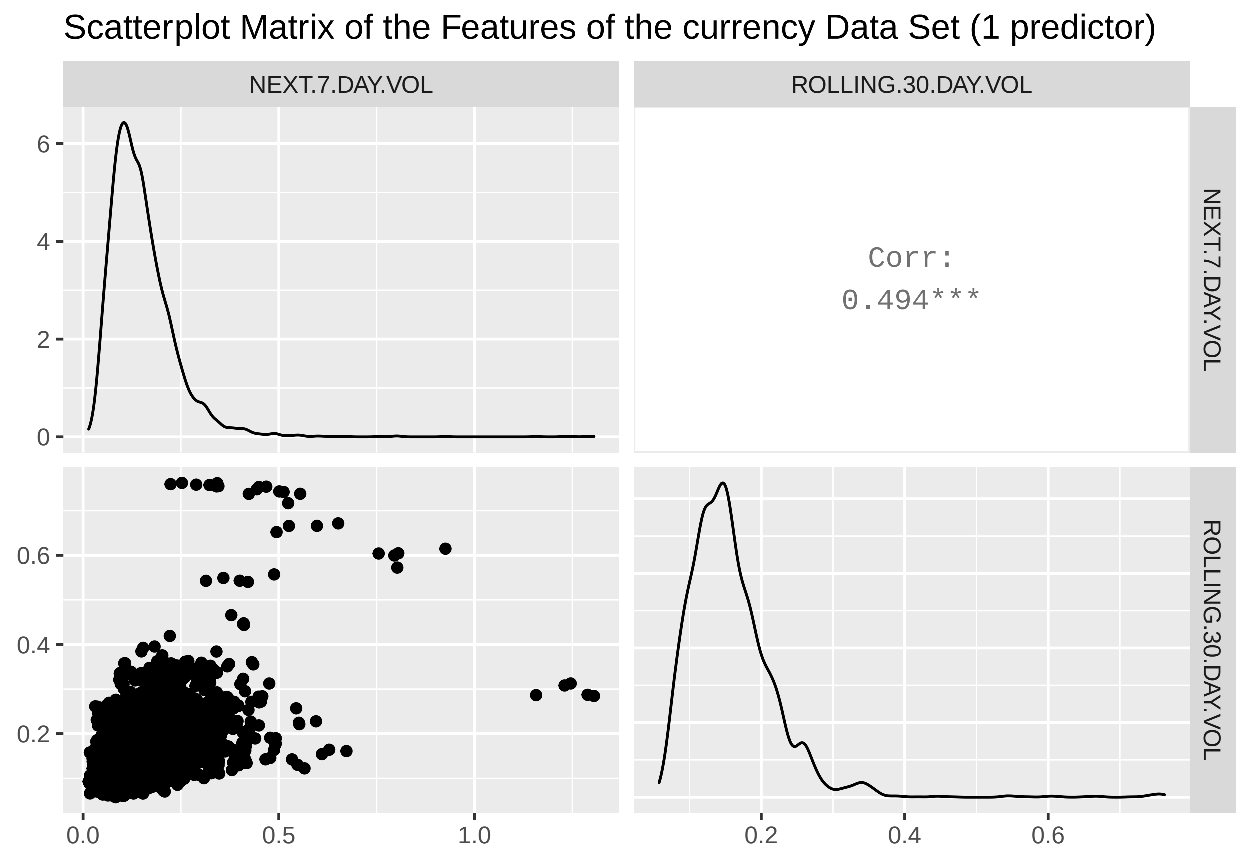


This is the final graph with the optimsed k parameter of 0.004166.

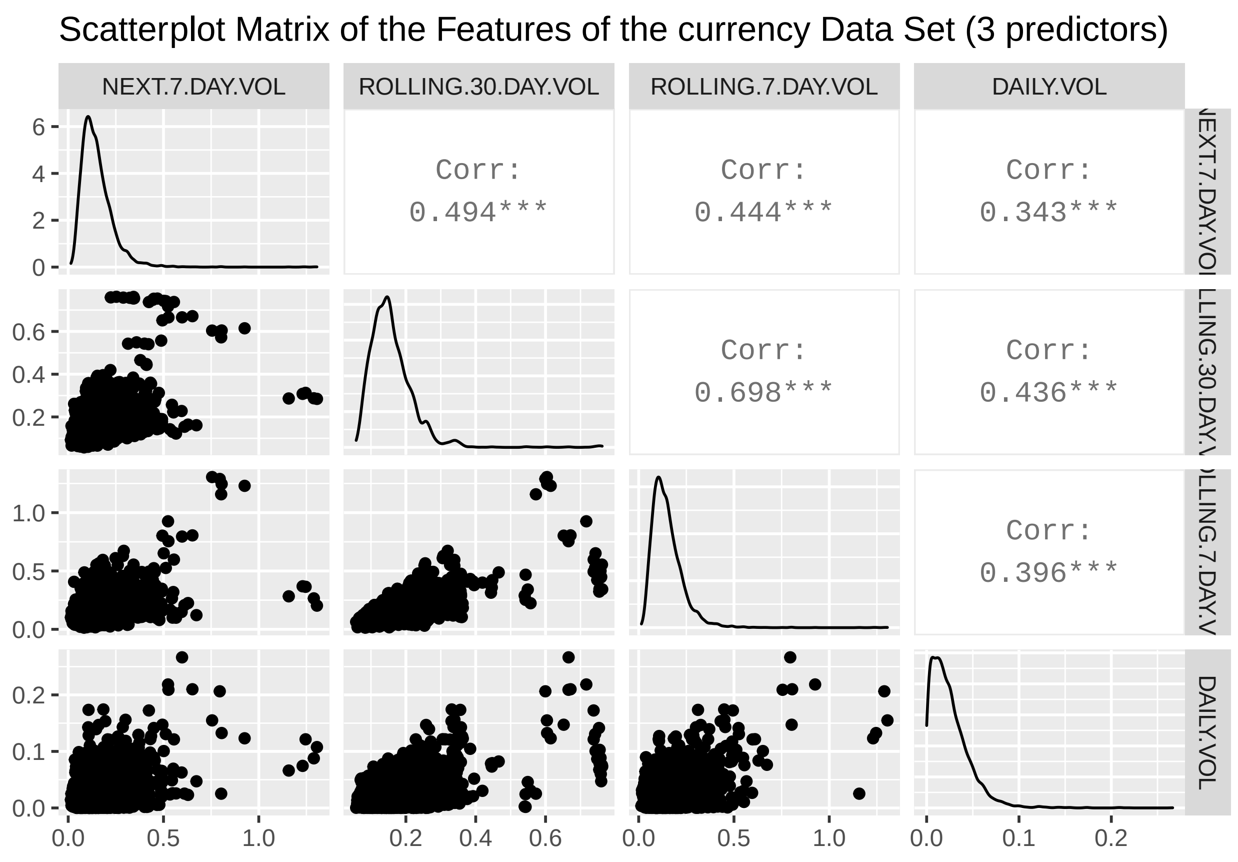
3.4 Neural network

Since the above results for the K-nearest neighbours and regression tree was so disappointing I decided to try something radical and investigate the option of using a neural network.

```r
#data visual
ggpairs(currency1, title = "Scatterplot Matrix of the Features of the currency Data Set (1 predictor)")
```

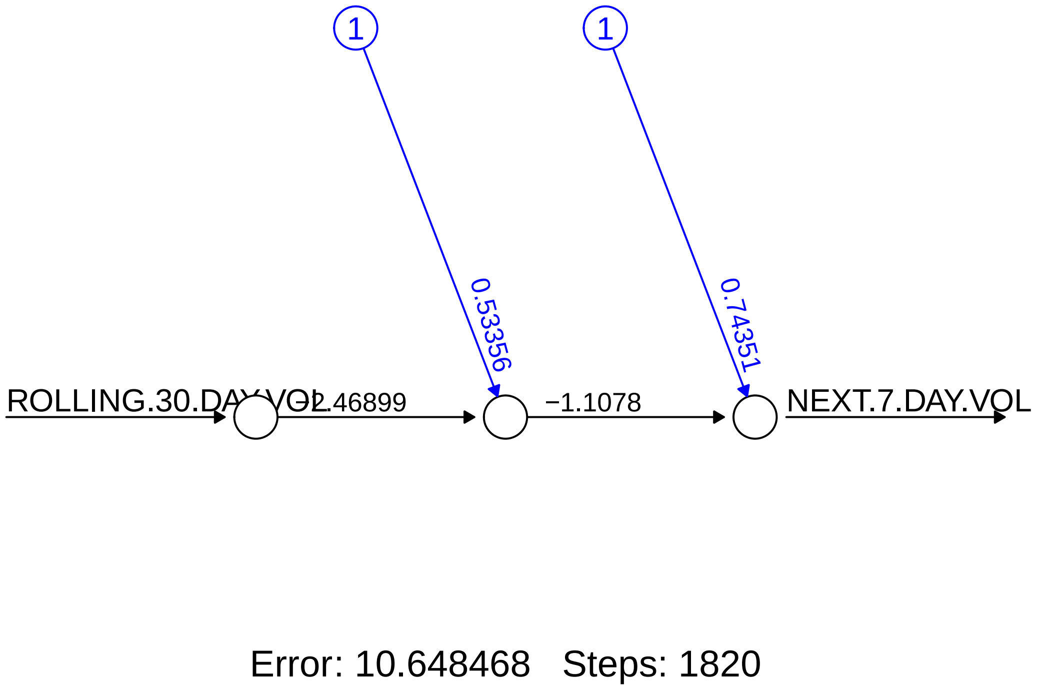


```
ggpairs(currency, title = "Scatterplot Matrix of the Features of the currency Data Set (3 predictors)")
```



```r
#1st Regression neural network
#To begin we construct a 1-hidden layer ANN with 1 neuron, the simplest of all neural networks.

set.seed(12321)
train_NN1 <- neuralnet( NEXT.7.DAY.VOL ~ ROLLING.30.DAY.VOL, data = train_set)
```

```r
plot(train_NN1, rep = 'best')   #all parameters of regression and results of neural network on train se
```



The plot for the 1st regression shows the weights learned by neural network and number of iterations before convergence and sum of squared errors (SSE) on the train set.

SSE for the 1st regression on the training set.

```
## [1] "SSE:  10.6485"
```

SSE of the NN1 training model against the test set.

```
## [1] "SSE:  4.7135"
```

Now I run 4 more variations of the neural network, below is a summary of all models run. I also included 2 smoothing techniques namely logistic and tanh.

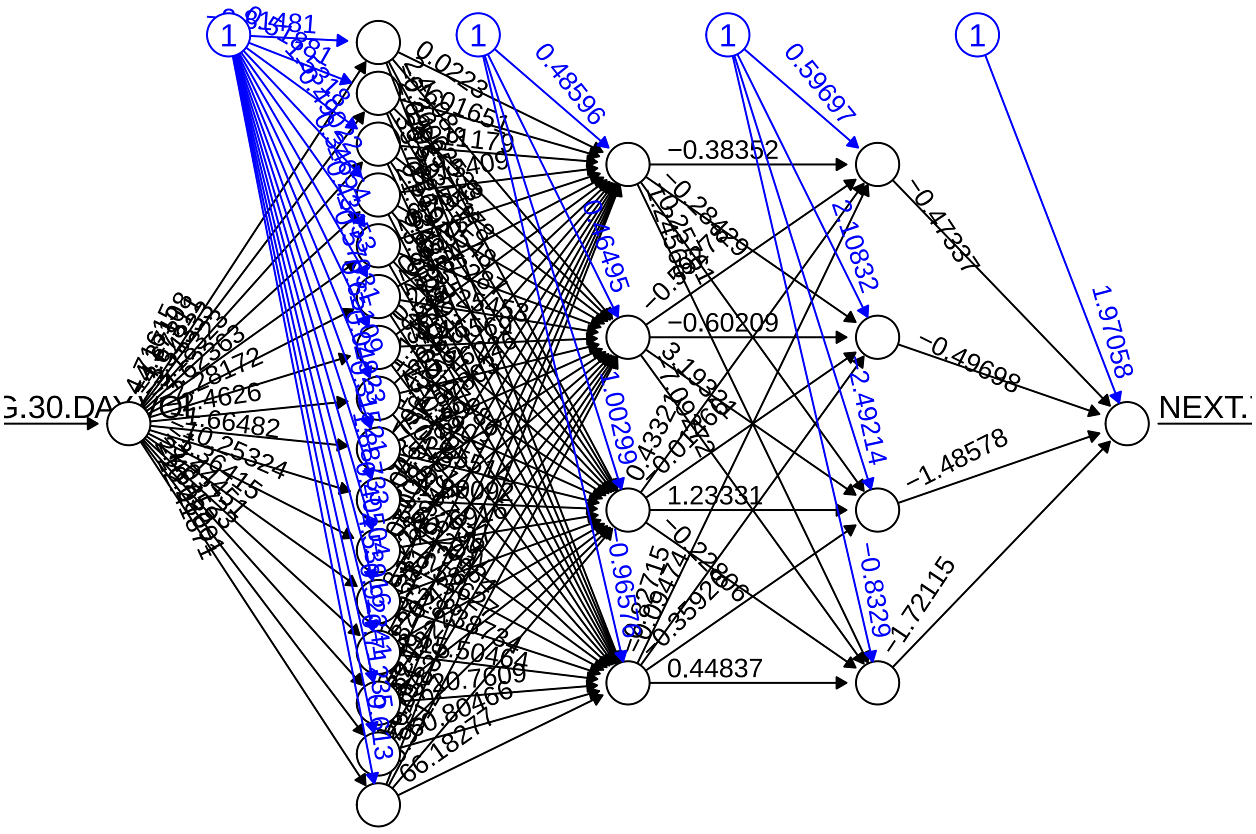
NN1 - 1-hidden layer ANN with 1 neuron.

NN2 - 3-Hidden Layers, Layer-1 16-neurons, Layer-2, 4-neurons, Layer-3, 4 neurons, logistic activation.

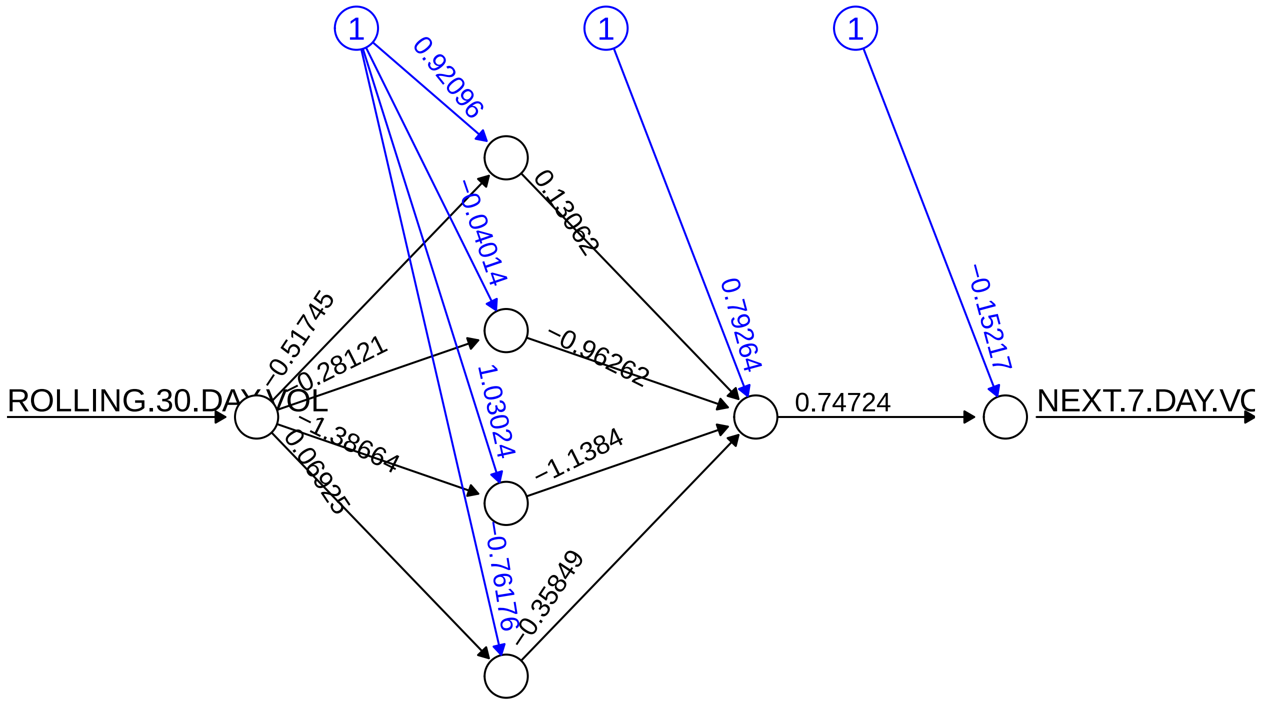
NN3 - 2-Hidden Layers, Layer-1 4-neurons, Layer-2, 1-neuron, tanh activation.

NN4 - 1-Hidden Layer, 1-neuron, tanh activation function.

NN5 - 3-Hidden Layers, Layer-1 16-neurons, Layer-2, 4-neurons, Layer-3, 4 neurons, logistic activation. (3 predictors)

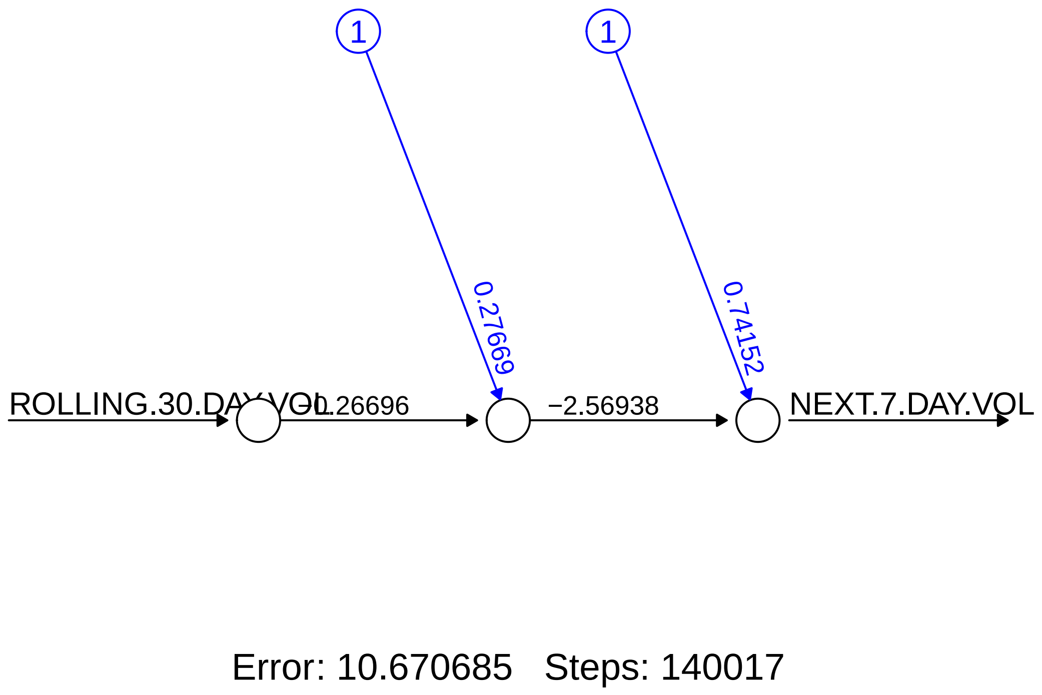


Plot of the NN2 - 3-Hidden Layers, Layer-1 16-neurons, Layer-2, 4-neurons, Layer-3, 4 neurons, logistic activation.



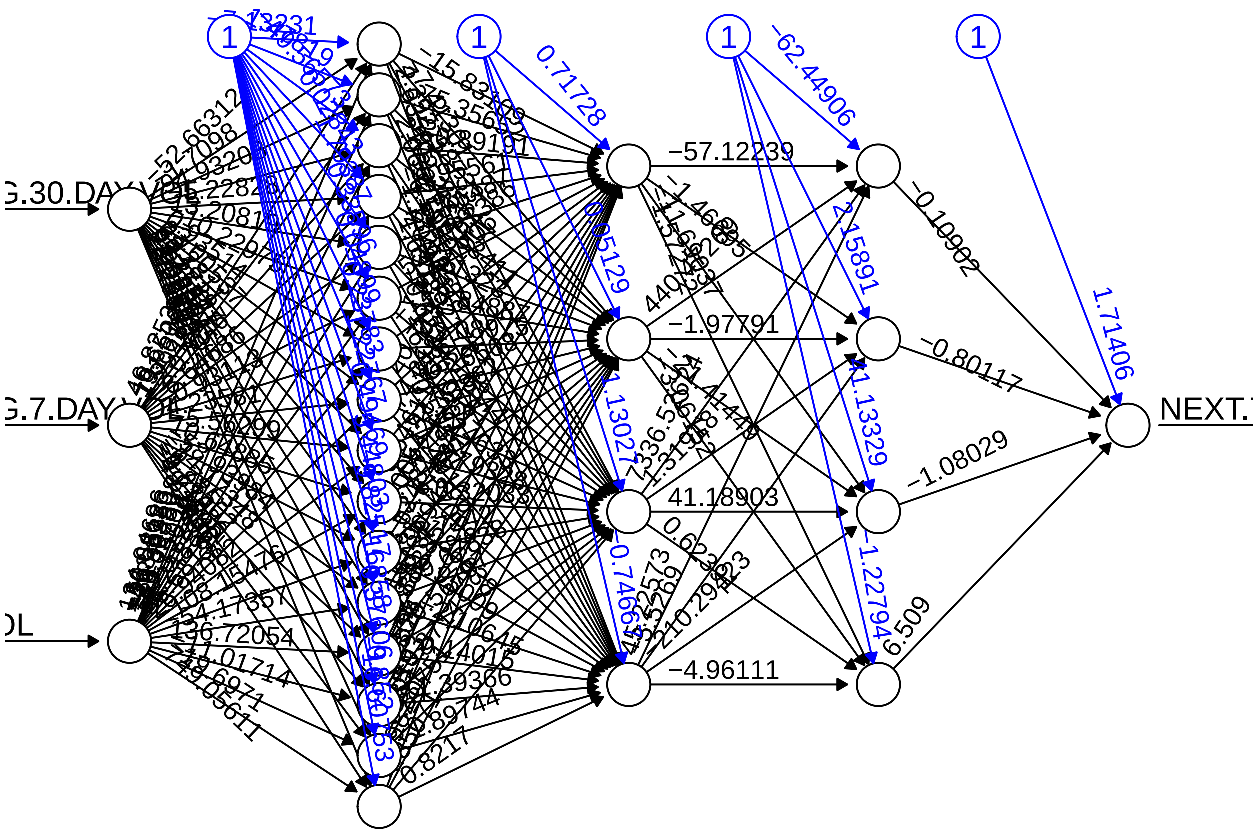
Error: 10.643387 Steps: 1197

Plot of the NN3 - 2-Hidden Layers, Layer-1 4-neurons, Layer-2, 1-neuron, tanh activation.



Plot of the NN4 - 1-Hidden Layer, 1-neuron, tanh activation function.

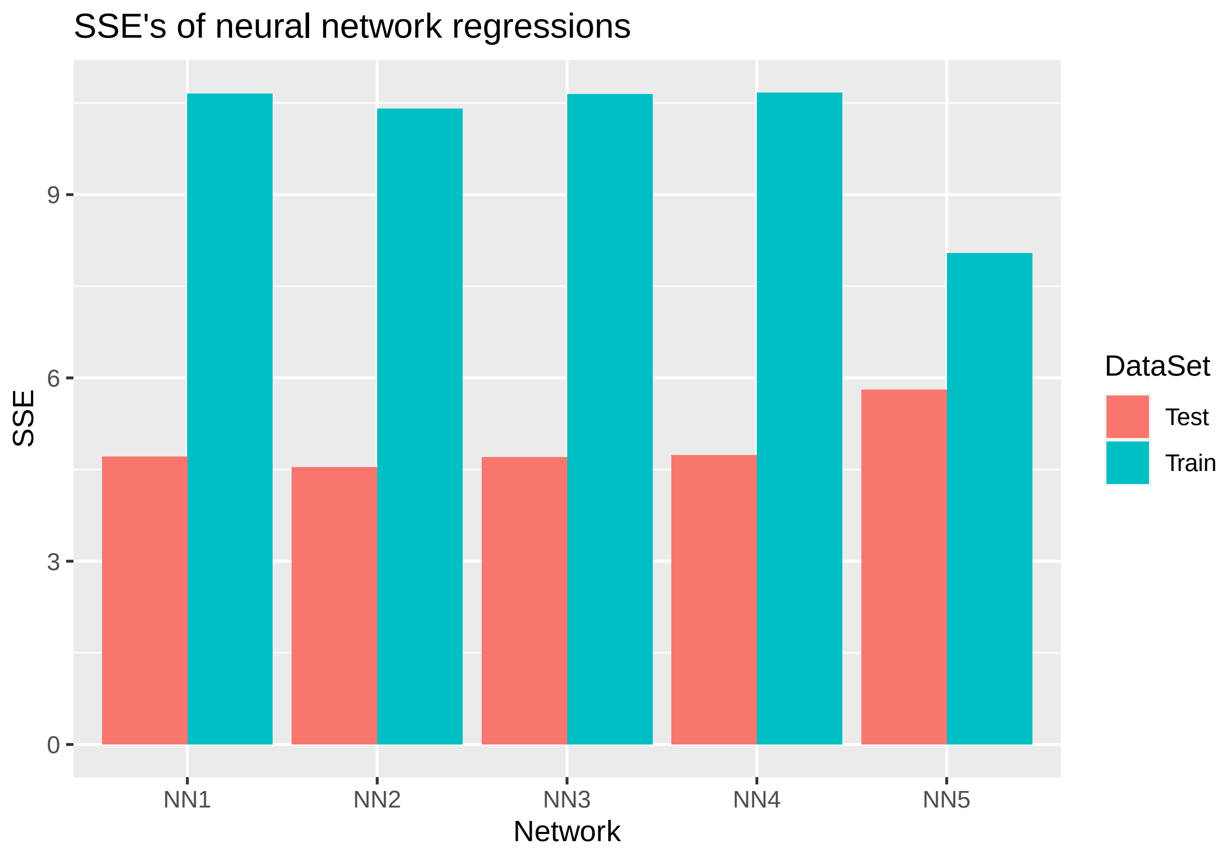
```r
plot(train_NN5, rep = 'best')  #all parameters of regression and results of neural network on train set
```



NN5 - 3-Hidden Layers, Layer-1 16-neurons, Layer-2, 4-neurons, Layer-3, 4 neurons, logistic activation (3 predictors).

I decided to add one model with 3 predictors since the results from the first 4 neural networks still had very high RMSE's which shows they aren't very good at forecasting. Adding the additional predictors didn't make a big difference.

The graph shows a summary of the SSE's for all 5 neural networks for the train and test set.



RMSE of the 5 neural networks performed for the test set. Overall the NN2 had the lowest SSE and RMSE.

```
NN1_Test_RMSE
```

```
## [1] 0.07609566
```

```
NN2_Test_RMSE
```

```
## [1] 0.07469249
```

```
NN3_Test_RMSE
```

```
## [1] 0.07602969
```

```
NN4_Test_RMSE
```

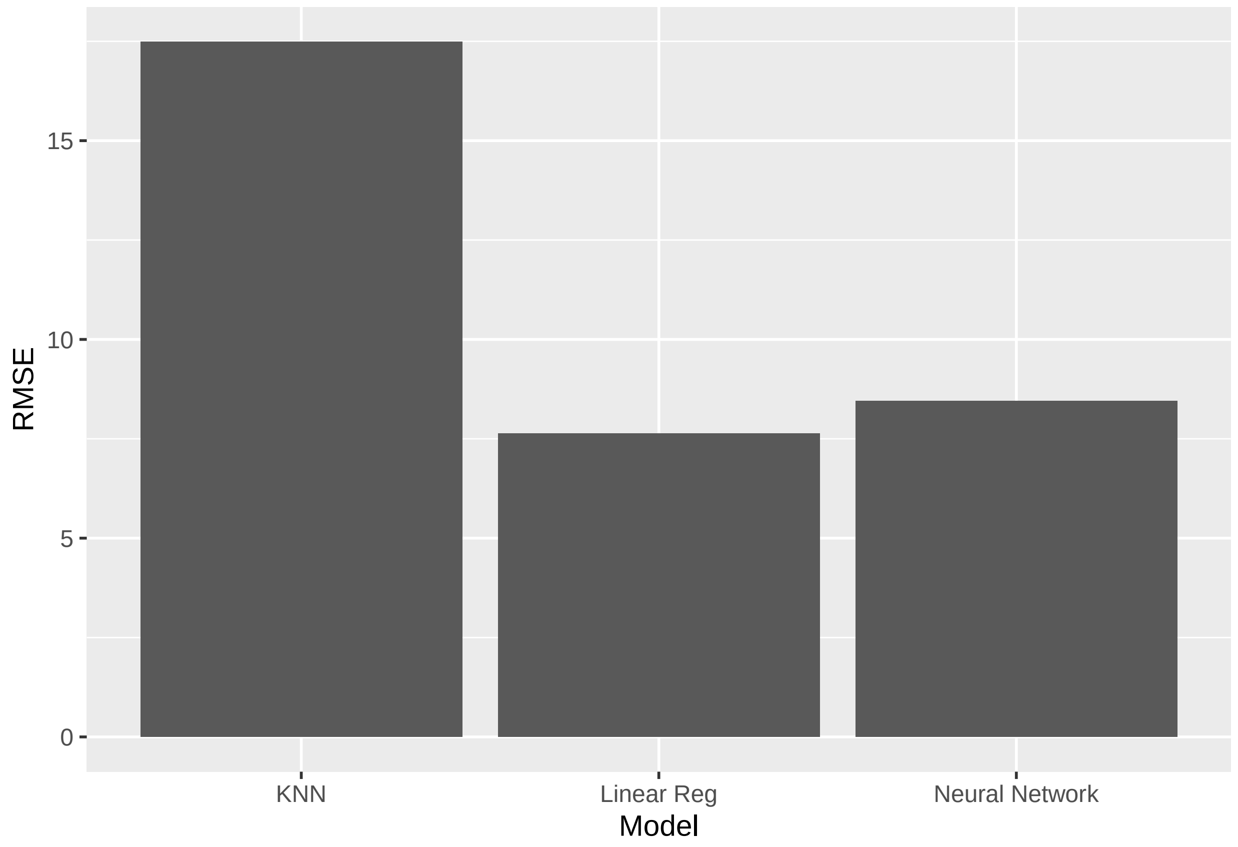
```
## [1] 0.07624229
```

```
NN5_Test_RMSE
```

```
## [1] 0.08447037
```

NN2 - 3-Hidden Layers, Layer-1 16-neurons, Layer-2, 4-neurons, Layer-3, 4 neurons, logistic activation. were the most accurate from all the variations I attempted.

4. Conclusion



I looked at various methods of forecasting the 7 day volatility from linear regression to 3 machine learning techniques including K-nearest neighbours, regression trees and neural networks.

I wasn't able to successfully construct a model that could reliably use the 30 day rolling volatility to predict the 7 day volatility, the RMSE's were just too high. The linear regression method was the most accurate at forecasting the next 7 day volatility from the models I tried. The regression tree could be useful to classify volatility into different regimes such as low or high volatility expecations.

One limitation in this project was the use of one predictor namely the 30 day rolling volatility. But in the neural network section I did introduce 2 additional predictors just to see if it can improve the model but it didn't.

Forecasting volatility is very difficult however, one of my work colleagues did a similar exercise in Python using machine learning and his results were very similar to mine.

Future work can include looking at other predictors to forecast the 7 day volatility such as different periods for the rolling volatility or a different predictor like the daily returns of the currency. Another possible model to explore is the GARCH model that supports changes in time dependent volatility.