DTI 5126: Fundamentals for Applied Data Science

Summer 2021 - Assignment 2

Name: Yomna Jehad Abdelsattar

Part A: Decision Trees

a) Missing Values Handling.

To know which method is better to handle the missing values, we first need to know how many records contain missing values in the dataset.
After investigating, we find out there's 769 records with missing values.
In my opinion that's a large number to drop from the 3772 records dataset.
So I decided to impute the numeric values with their **mean values** and the categorical values with their **mode values**.

```
library(modeest)
sex_mode = mfv(hypo$sex) #mode
hypo$sex[hypo$sex == "??"] <- sex_mode #impute categorical
sum(hypo$sex == "?") #double check that the imputation worked
```

```
# First give a unique numerical value to be able to calculate the mean with no errors
hypo$age[hypo$age=="??"] <- "000"
hypo$age[hypo$age=="000"] <- mean(as.numeric(as.character( hypo$age) ) ) #impute
hypo$age <- as.numeric(hypo$age)
```

We also note that the column "TBG" is completely empty so we'd better **drop it**.

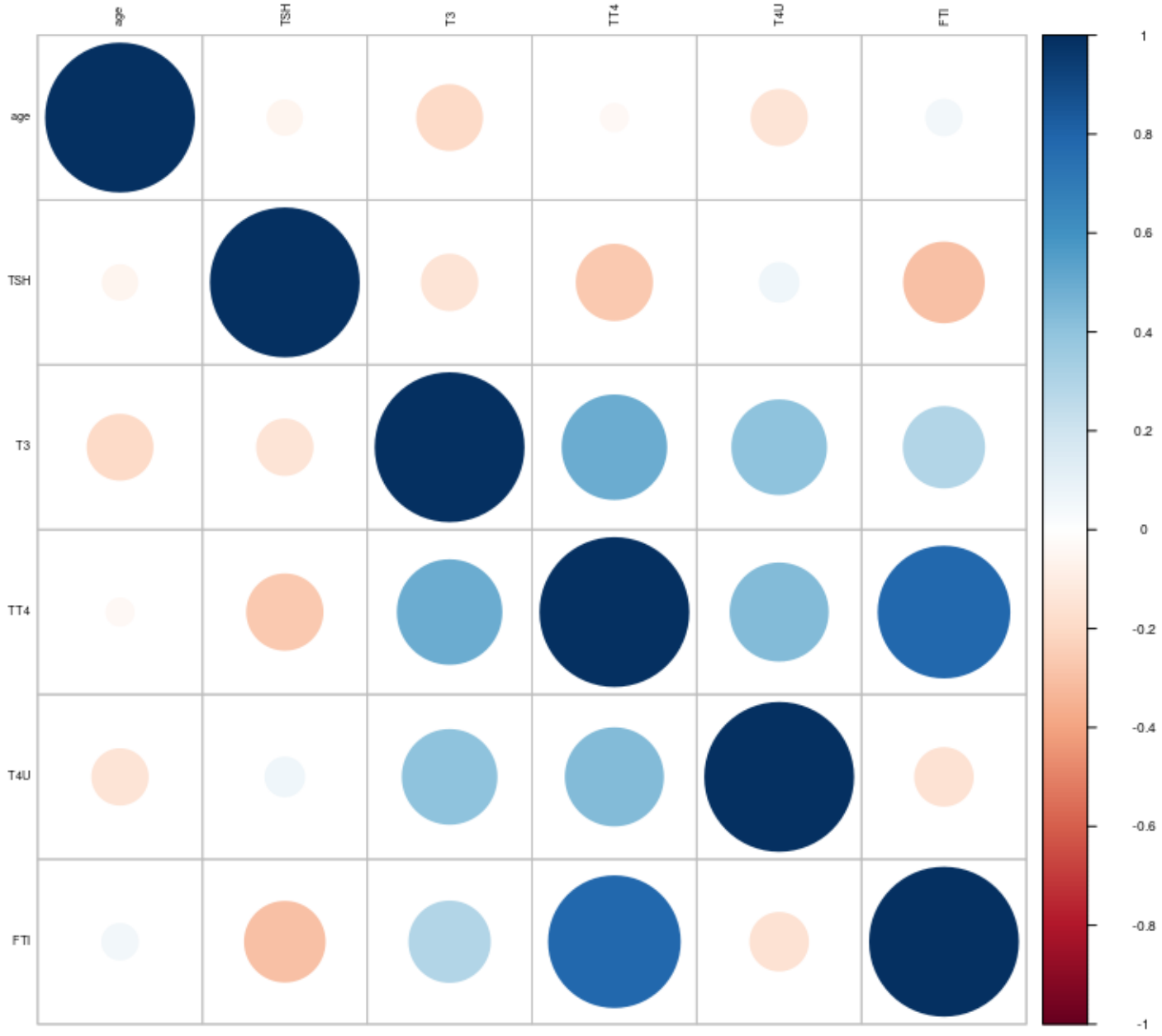
```
drop <- c("TBG")
hypo = hypo[,!(names(hypo) %in% drop)]
```

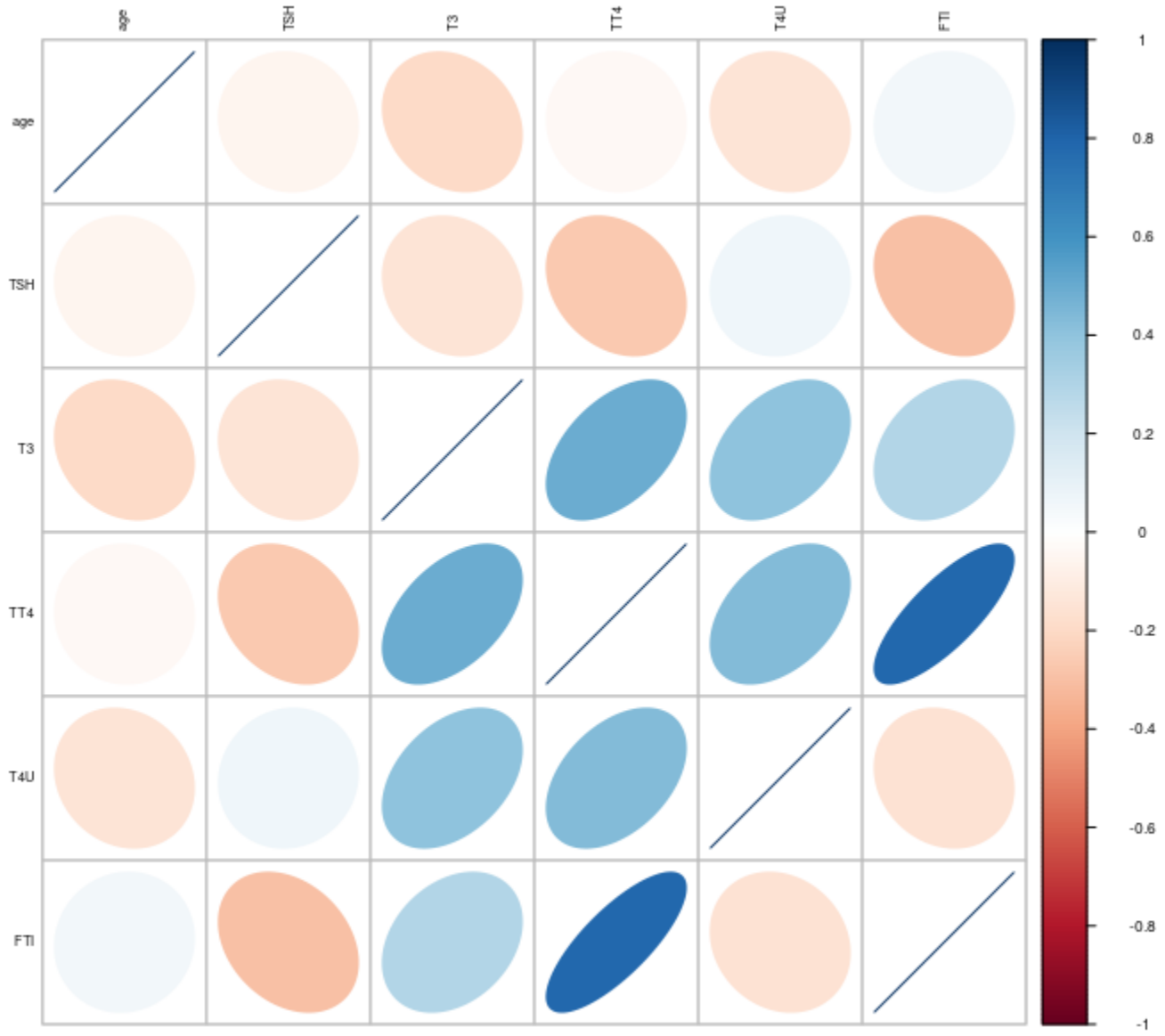
b) Attribute Selection.

Attribute selection is a very important step as it may either reduce the computational cost of modeling (By applying the model on fewer features which have the most amount information we need) or, in some cases, improve the performance of the model (maybe also by removing features which could be noisy or irrelevant business wise), or both. There are several methods to be used to determine which attributes to keep and which to drop.

- The first method I used is **Correlation Matrix**. Which shows me if there are columns with a correlation that shows they have a similar effect on the output, in this case we can drop one of them.

```
> print(correlationMatrix)
            age         TSH         T3         TT4         T4U         FTI
age  1.00000000 -0.05549798 -0.1937422 -0.03445986 -0.14127632  0.05887338
TSH -0.05549798  1.00000000 -0.1417587 -0.26084055  0.06967186 -0.29162792
T3  -0.19374219 -0.14175871  1.0000000  0.49152214  0.40317921  0.29684711
TT4 -0.03445986 -0.26084055  0.4915221  1.00000000  0.43276775  0.78173241
T4U -0.14127632  0.06967186  0.4031792  0.43276775  1.00000000 -0.15269986
FTI  0.05887338 -0.29162792  0.2968471  0.78173241 -0.15269986  1.00000000
```


age
TSH
T3
TT4
T4U
FTI
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1



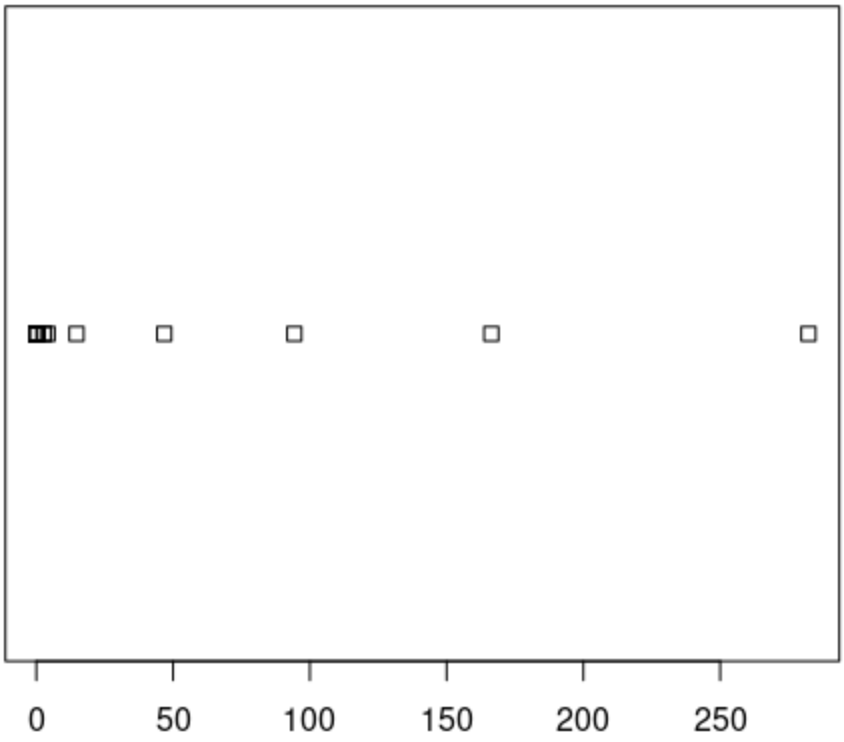
By looking at these numbers and graphs we can conclude that **TT4** and **FTI** are **highly correlated** so we **drop TT4** for example.

- Another thing I noticed is that some columns contain almost only one single value, which probably has a very **weak** effect on the model, so i decided to **drop** them.

```
# Check if any column has only one value

for (i in 1:ncol(hypo)){
  print(names(hypo[i]))
  print(unique(hypo[i]))
}
# Drop TBG_measured
drop <- c("TBG_measured", "hypopituitary")
hypo = hypo[,!(names(hypo) %in% drop)]
```

- The last thing I did was measuring the **importance** of the remaining **features**

```
> importances %>% arrange(desc(Overall))
                             Overall
TSH                       282.277688
FTI                       166.333463
on_thyroxine               94.322425
T3                         46.768557
thyroid_surgery            14.697349
referral_source             4.150914
TSH_measured                2.838044
query_hypothyroid           2.643877
age                         0.000000
sex                         0.000000
query_on_thyroxine          0.000000
on_antithyroid_medication   0.000000
sick                        0.000000
pregnant                    0.000000
I131_treatment              0.000000
query_hyperthyroid          0.000000
lithium                     0.000000
goitre                      0.000000
tumor                       0.000000
psych                       0.000000
T3_measured                 0.000000
TT4_measured                0.000000
T4U_measured                0.000000
T4U                         0.000000
FTI_measured                0.000000
```

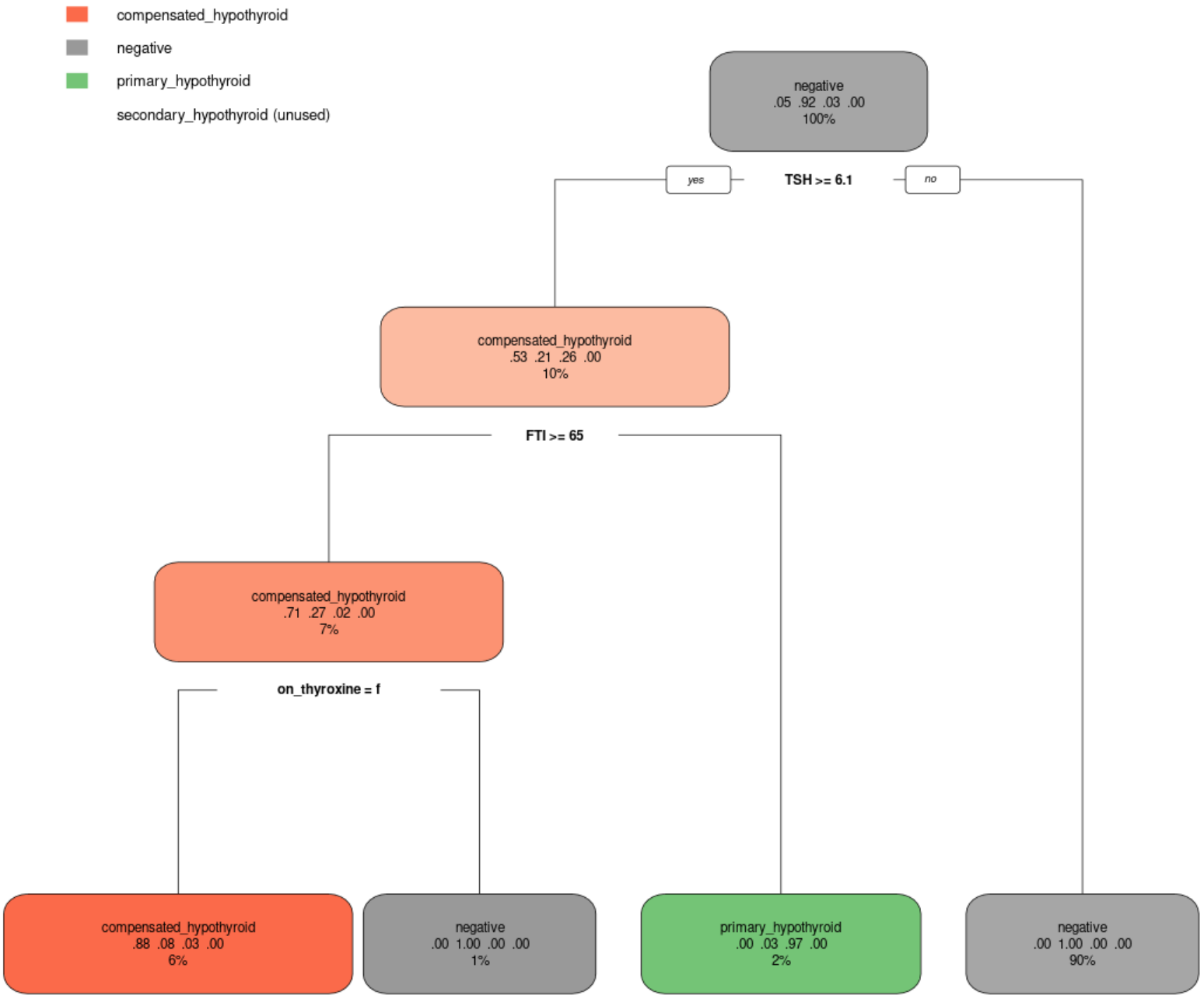


I **dropped** all the features whose **importance** is **0**. (I think I should have dropped the **1<importance<5** features too, but then i would have been left with **too few features** which may affect the training so i decided not to) .

At the end of this stage I was left with these dimensions, this will definitely **improve performance** of **training at least.**

hypo	3772 obs. of 26 variables
$ age	: num 41 23 46 70 70 18 59 80 66 68 ...
$ sex	: chr "F" "F" "M" "F" ...
$ on_thyroxine	: chr "f" "f" "f" "t" ...
$ query_on_thyroxine	: chr "f" "f" "f" "f" ...
$ on_antithyroid_medication	: chr "f" "f" "f" "f" ...
$ sick	: chr "f" "f" "f" "f" ...
$ pregnant	: chr "f" "f" "f" "f" ...
$ thyroid_surgery	: chr "f" "f" "f" "f" ...
$ I131_treatment	: chr "f" "f" "f" "f" ...
$ query_hypothyroid	: chr "f" "f" "f" "f" ...
$ query_hyperthyroid	: chr "f" "f" "f" "f" ...
$ lithium	: chr "f" "f" "f" "f" ...
$ goitre	: chr "f" "f" "f" "f" ...
$ tumor	: chr "f" "f" "f" "f" ...
$ psych	: chr "f" "f" "f" "f" ...
$ TSH_measured	: chr "t" "t" "t" "t" ...
$ TSH	: num 1.3 4.1 0.98 0.16 0.72 ...
$ T3_measured	: chr "t" "t" "f" "t" ...
$ T3	: num 2.5 2 1.6 1.9 1.2 ...
$ TT4_measured	: chr "t" "t" "t" "t" ...
$ T4U_measured	: chr "t" "f" "t" "f" ...
$ T4U	: num 1.14 0.893 0.91 0.893 0.87 ...
$ FTI_measured	: chr "t" "f" "t" "f" ...
$ FTI	: num 109 99.2 120 99.2 70 ...
$ referral_source	: chr "SVHC" "other" "other" "other" ...
$ Class	: chr "negative" "negative" "negative" "ne…

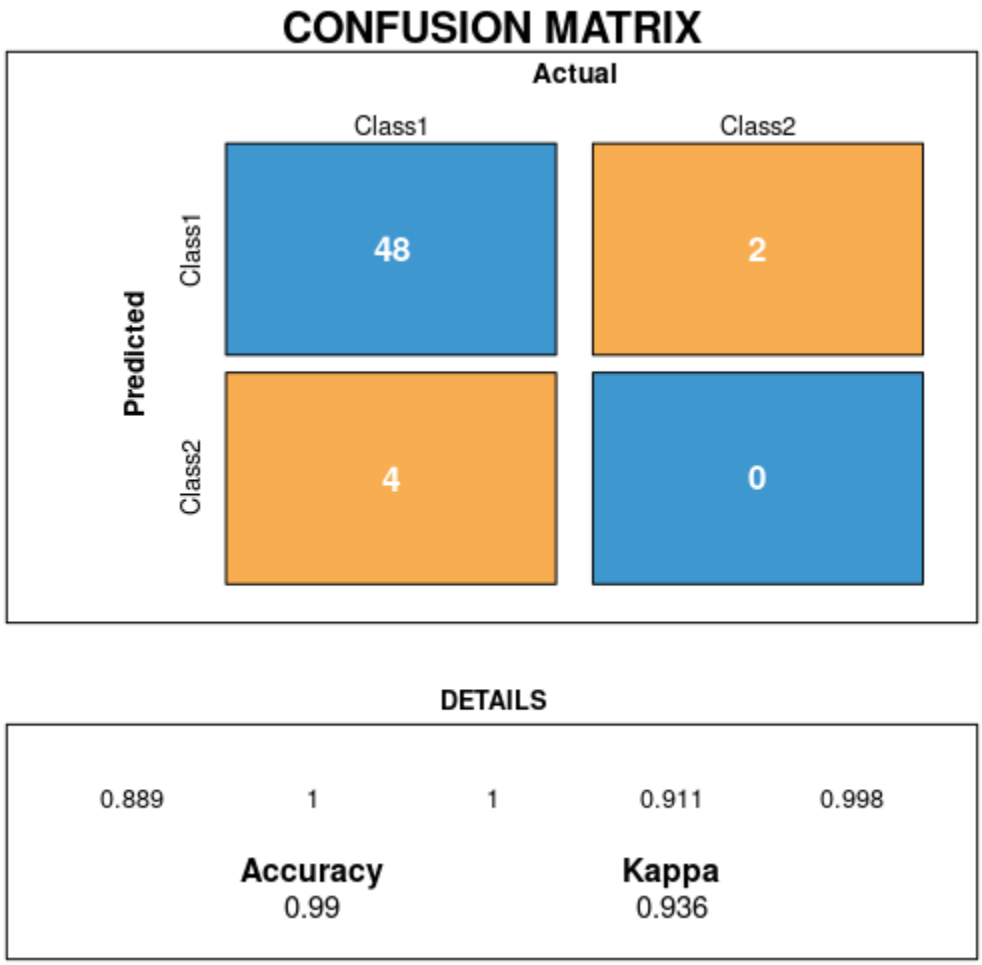
c) Tree and accuracy before K-fold cross validation



The percentages at the bottom of each box represent how accurately the tree can predict this class at this split.

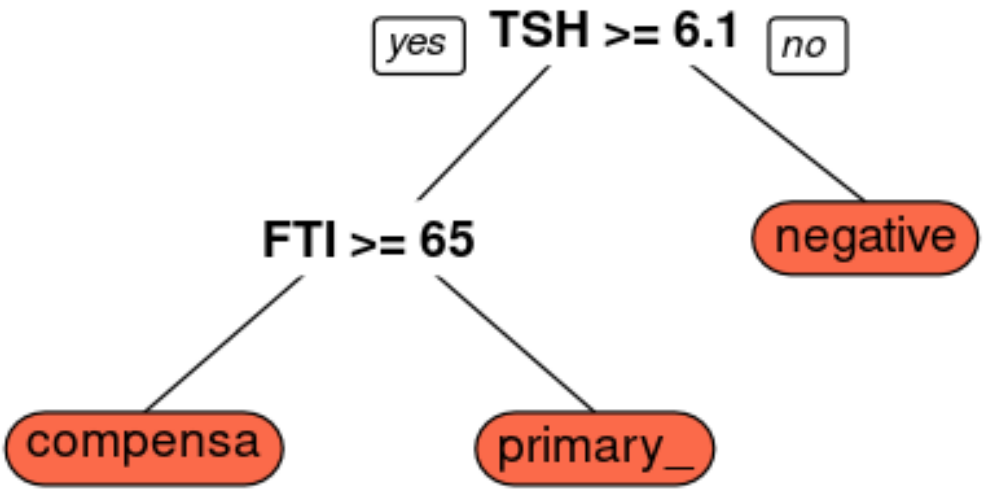
By looking at the tree we can conclude rules like: we rely on 3 features:

- **If** TSH<6.1 then the class is **NEGATIVE.**
- **Else** then **if** FTI<65 then the class is **PRIMARY_HYPOTHYROID.**
- **Else** then **if** on_thyroxide = f then the class is **COMPENSATED_HYPOTHYROID** **else** it's **NEGATIVE.**



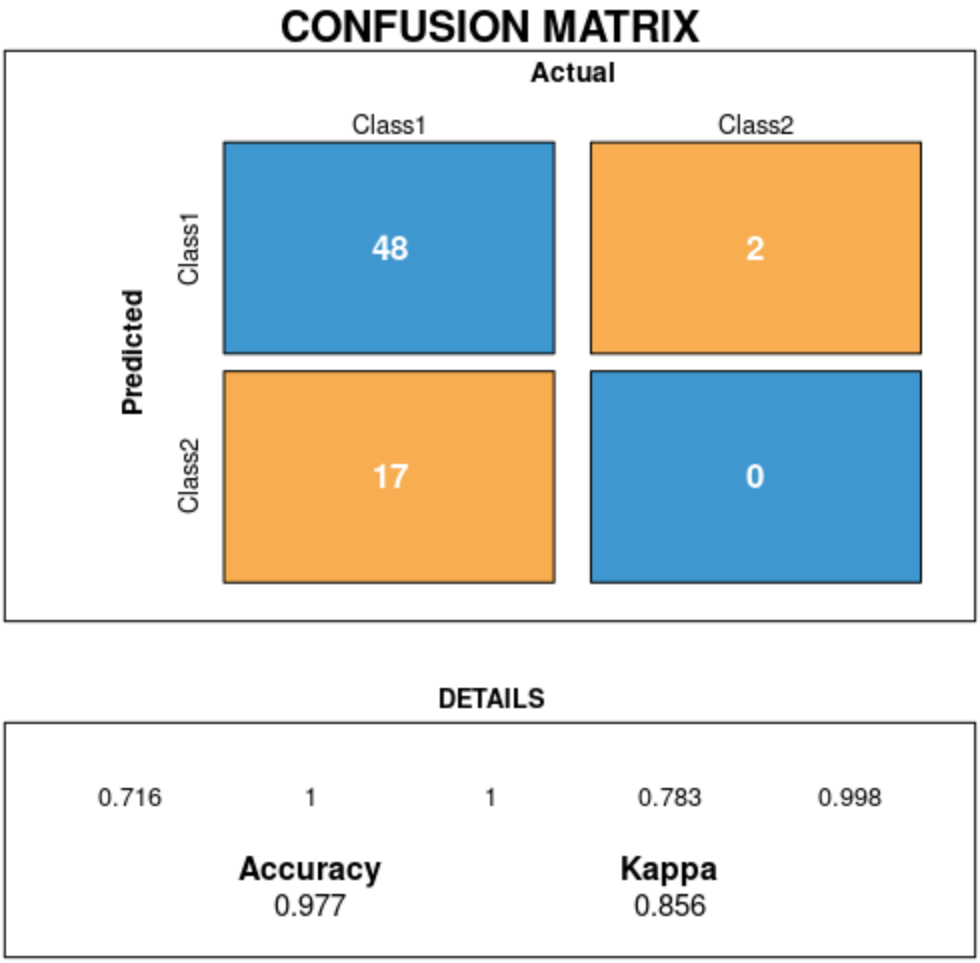
As shown in the plot the Accuracy is 0.99 (Let's keep it in mind to compare it to the accuracy after k folds cross validation, it looks like a very high score for a base model which could imply overfitting)

d) Decision Tree after K-fold cross validation



By looking at the tree we can conclude rules like: we rely on 2 features:

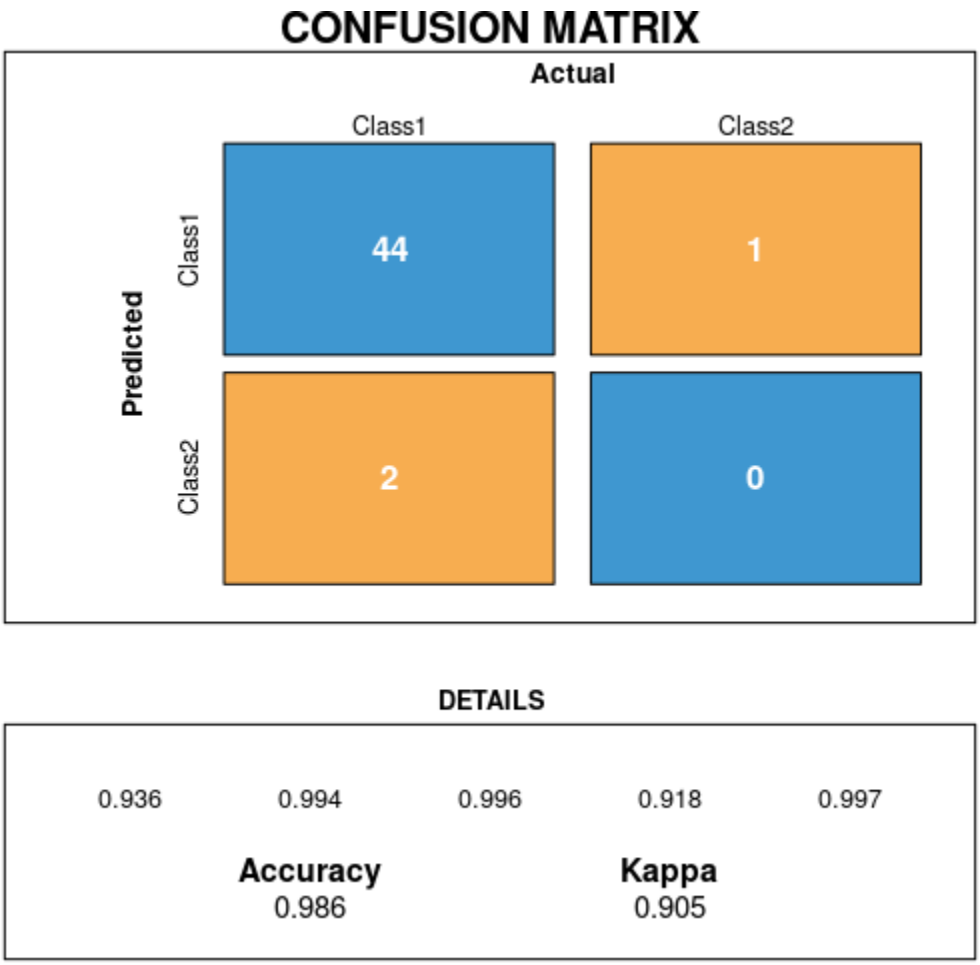
- **If** TSH<6.1 then the class is **NEGATIVE.**
- **Else** then **if** FTI < 65 then the class is **PRIMARY_HYPOTHYROID.**
- **Else** the class is **COMPENSATED_HYPOTHYROID.**



The accuracy now has become **worse**, it became 0.977.
Which could mean that at the beginning there was **overfitting**, and now it gives a better **approximation** for that accuracy.

e) Model improvements: Pruning - Gini

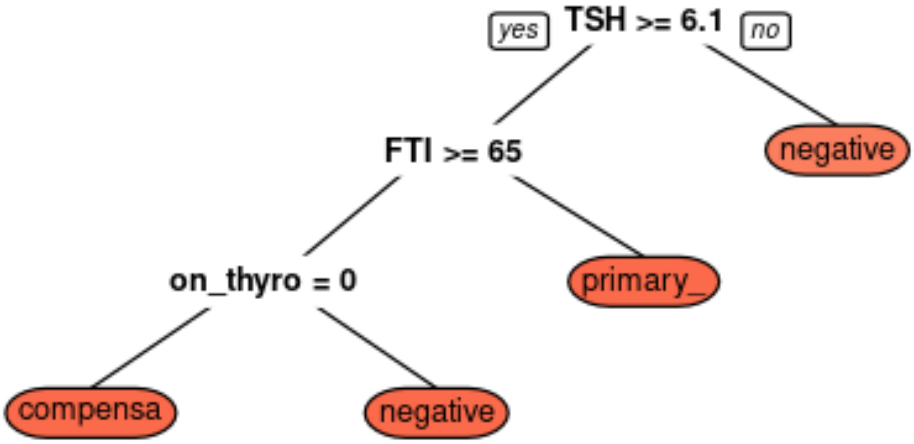
- **Pruning (minsplit=2, cp=0.001)**



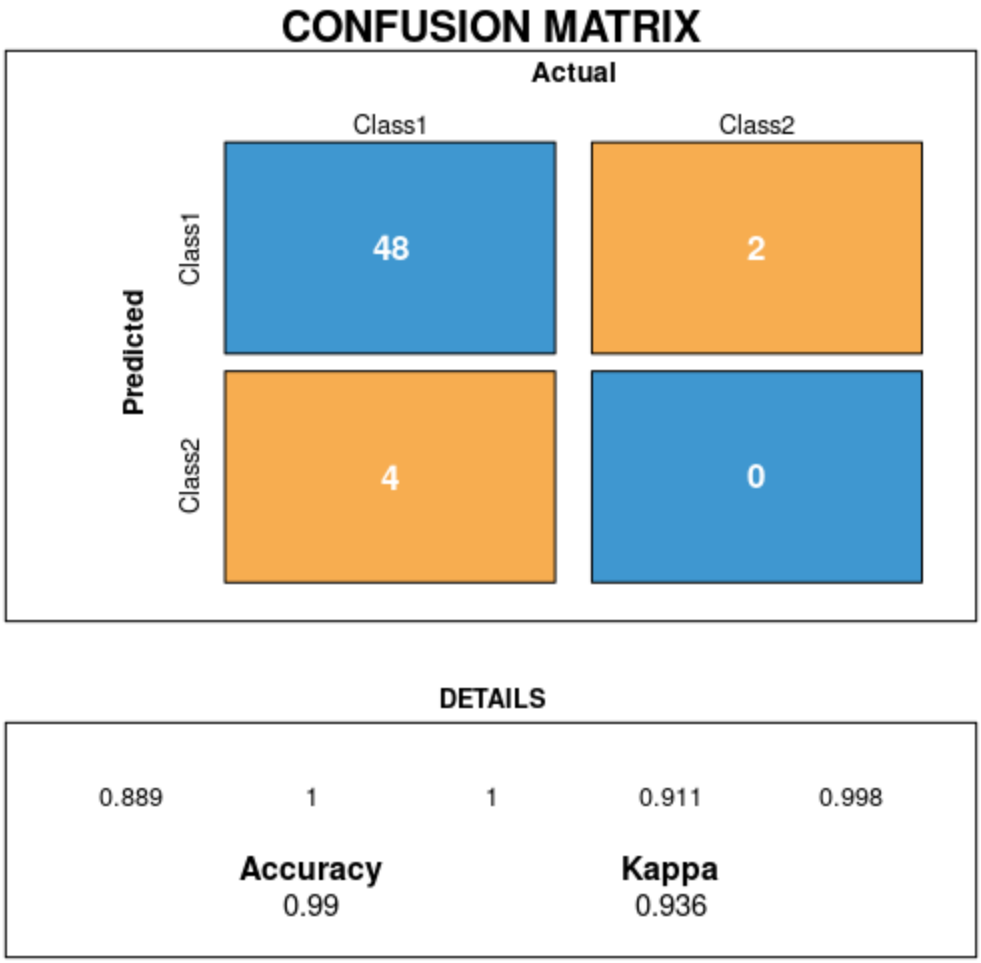
We notice that the **accuracy** is also **less** than the base model which could mean that pruning prevented some of the **overfitting**.

- **Gini (tuneLength = 10)**

I applied this Gini with the 10-k folds cross validation.



Now it relies on 3 features instead of only 2 in the case of k-folds cross validation only.

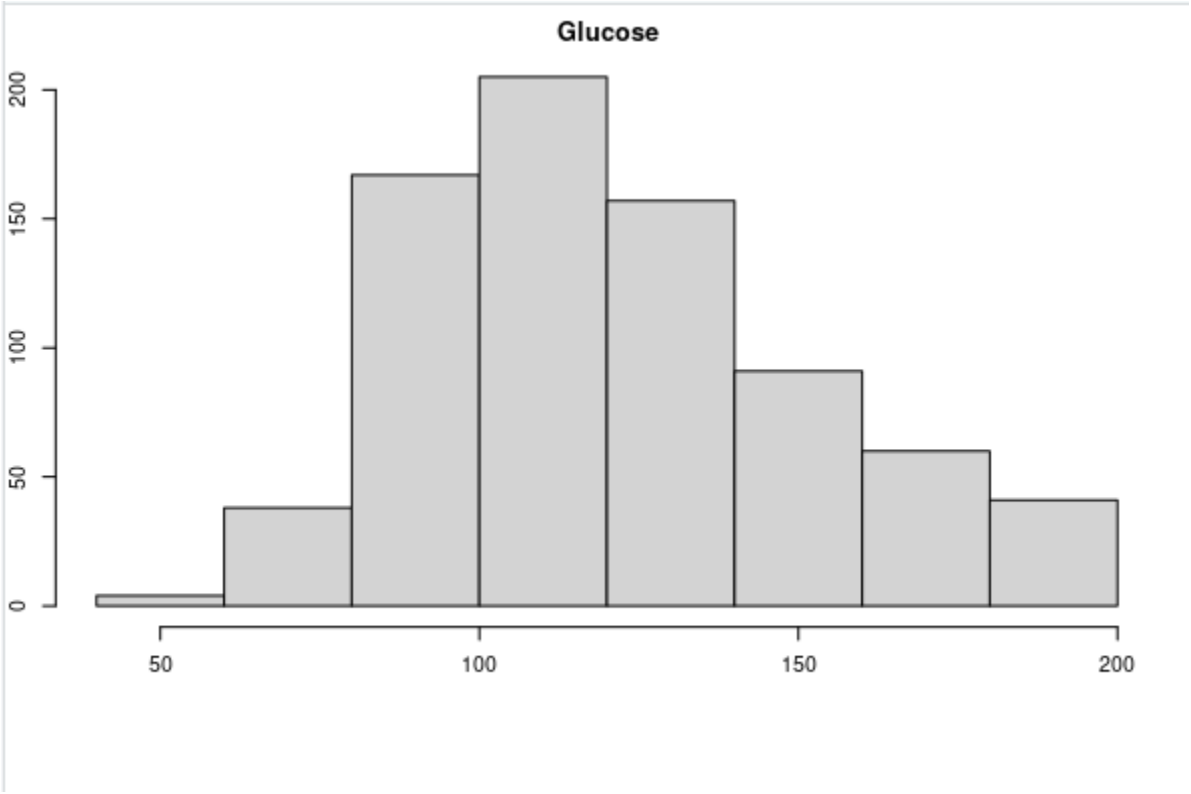


I notice no apparent difference in accuracy.

Part B: Neural Network

a) Missing Values Handling by applying a central measure of tendency.

We check the distribution (through **histogram** plot) of e



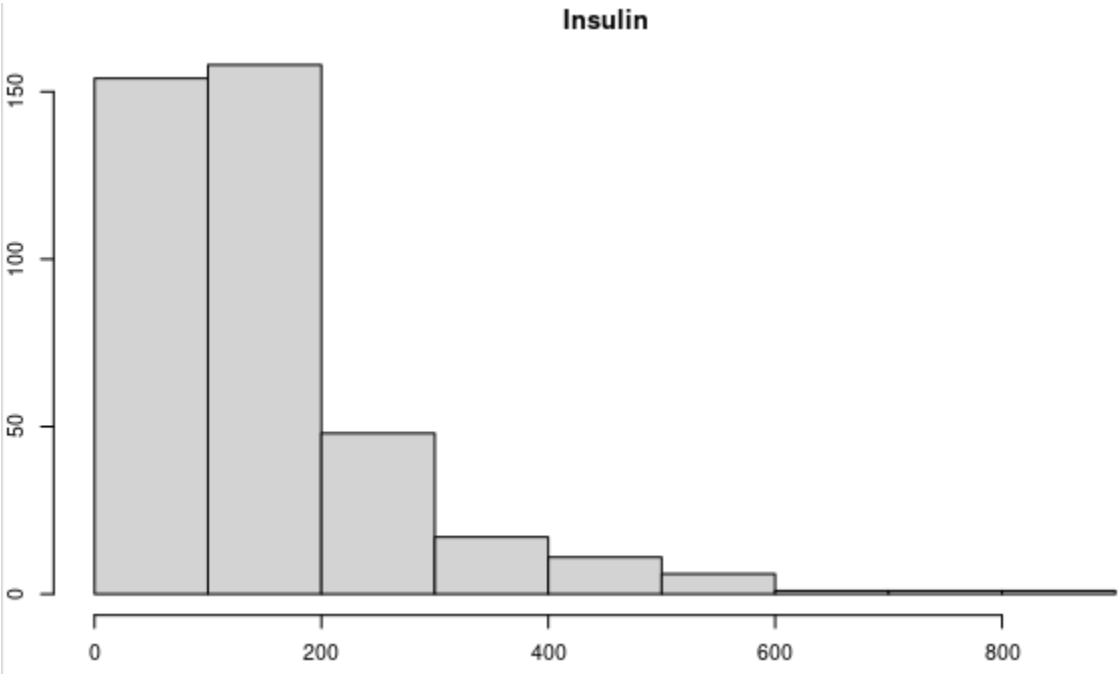
ach feature.

For features with a **symmetric normal distribution** we impute the missing values with its **mean**.

(Similarly: **BloodPressure**, **SkinThickness**, **BMI**)

```
# Impute numeric
hist(dia$Glucose, main = "Glucose", xlab = "Density") #Normal distribution: impute with mean
dia$Glucose[is.na(dia$Glucose)] <- mean( as.numeric(dia$Glucose), na.rm = TRUE) #impute
```

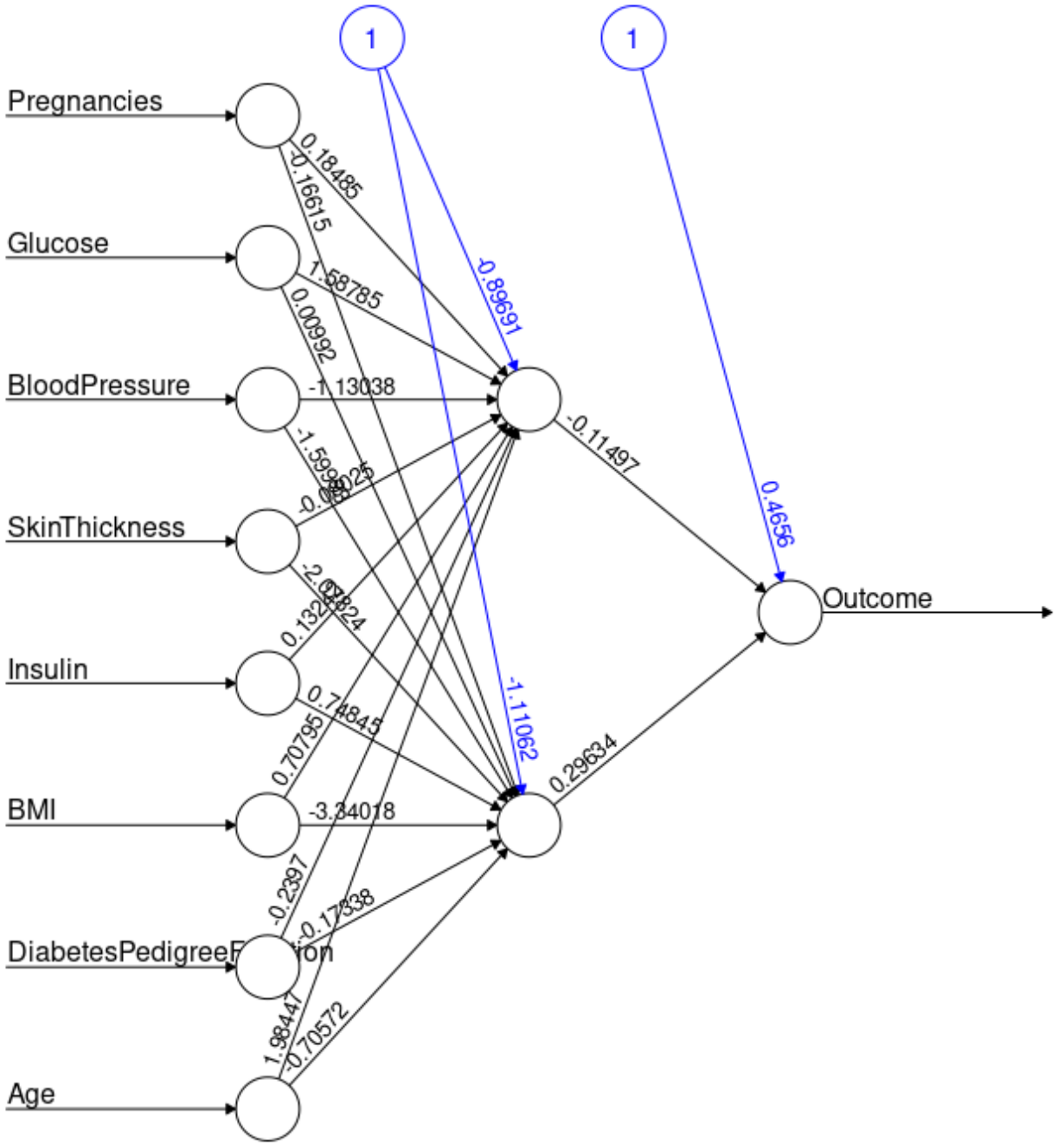
For features with a **positively skewed distribution** we impute with its **median**.




```
hist(dia$Insulin, main = "Insulin", xlab = "Density") #Positively Skewed: impute with median
dia$Insulin[is.na(dia$Insulin)] <- median(dia$Insulin, na.rm = TRUE ) #impute
```

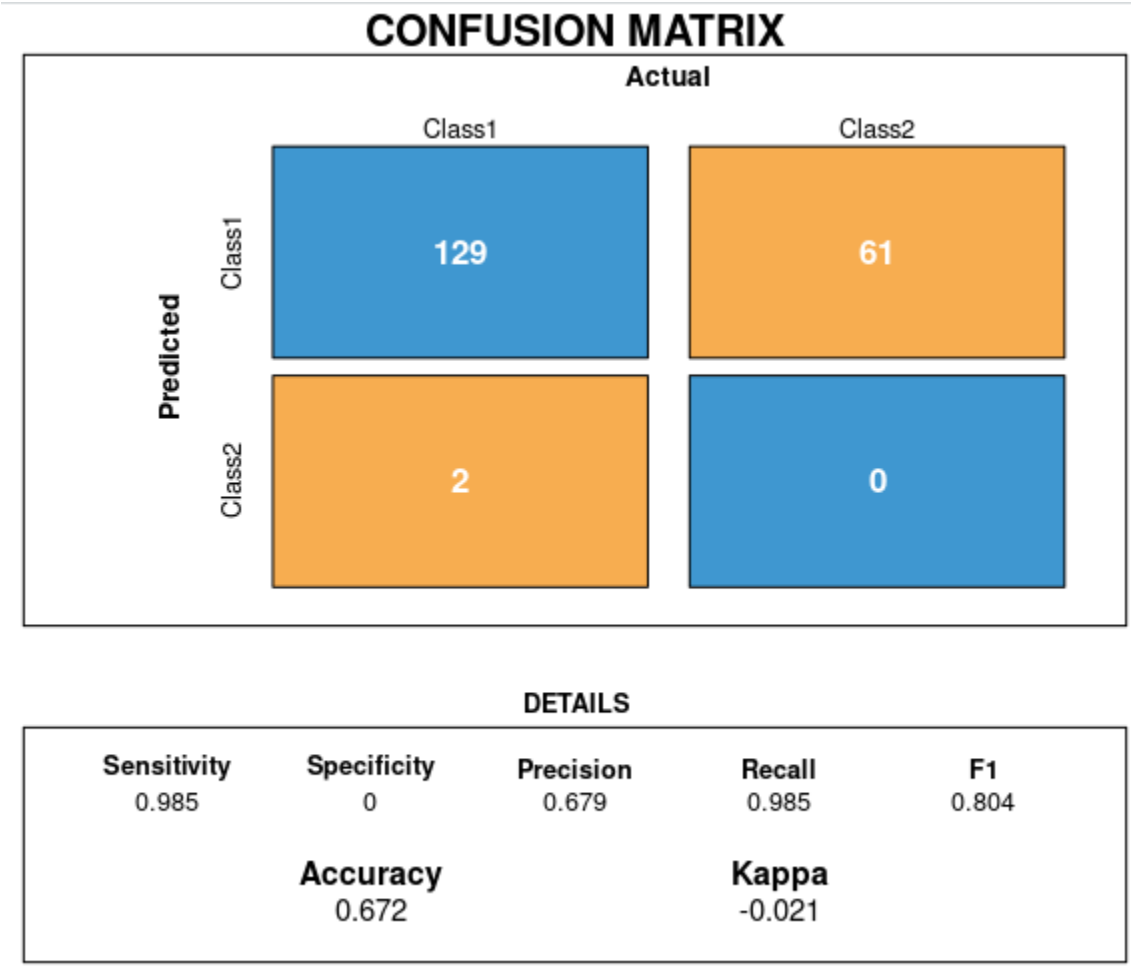
b) Partition Dataset, build, train, test Neural Network.

Partitioned **75% training set** and **25% test set.**
Used them to train a basic Neural Network model.



Error: 65.579987 Steps: 66

Let’s also take a look at the accuracy of this model



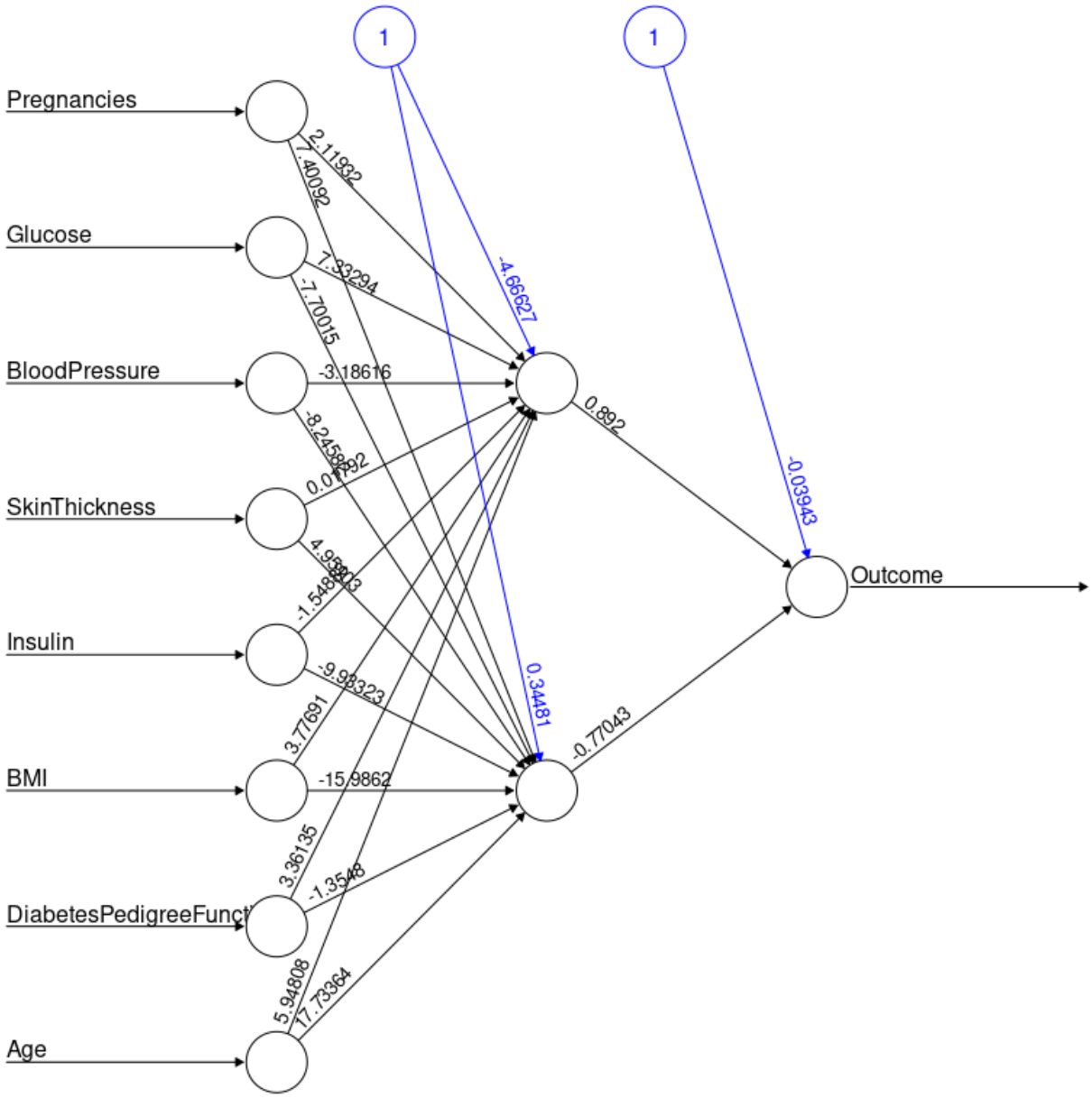
Quite poor accuracy on the **test** set.

c) Scale Dataset, build, train, test Neural Network.

To **improve** this accuracy we're going to scale the dataset points by **min-max normalization.**

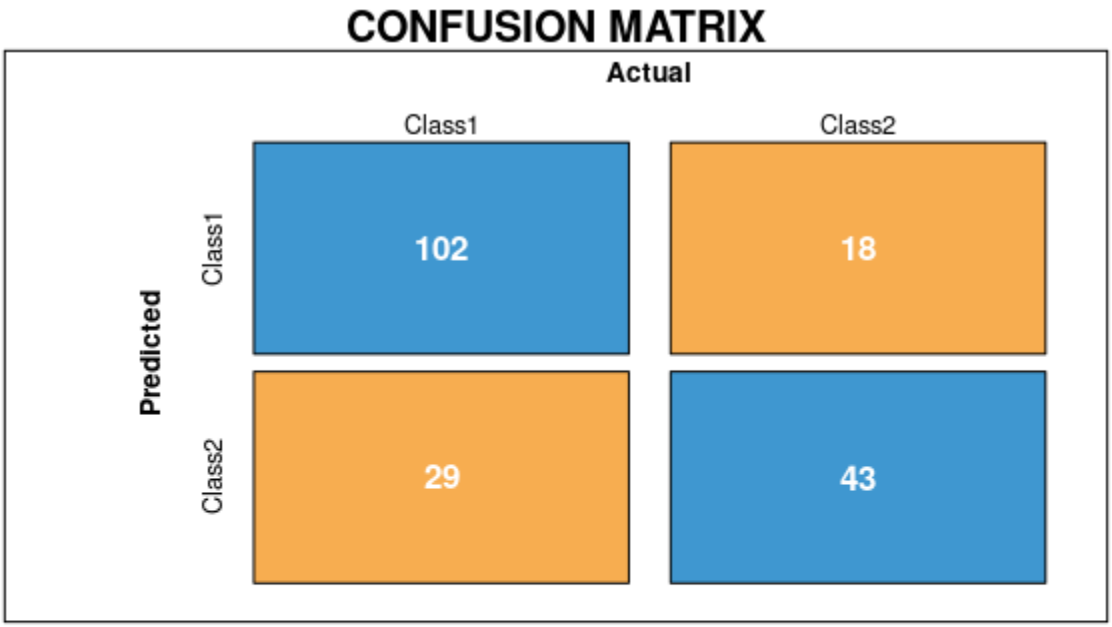
```
# Scaling
## Scale data for neural network - important to prevent a variable from having a large impact on the prediction variable
max = apply(dia , 2 , max)
min = apply(dia, 2 , min)
scaled = as.data.frame(scale(dia, center = min, scale = max - min))

## Fit neural network
```

After **training** and **testing** a **2 hidden nodes Neural Network** with this scaled data



Error: 40.096652 Steps: 4028

By looking at the **confusion matrix** plot



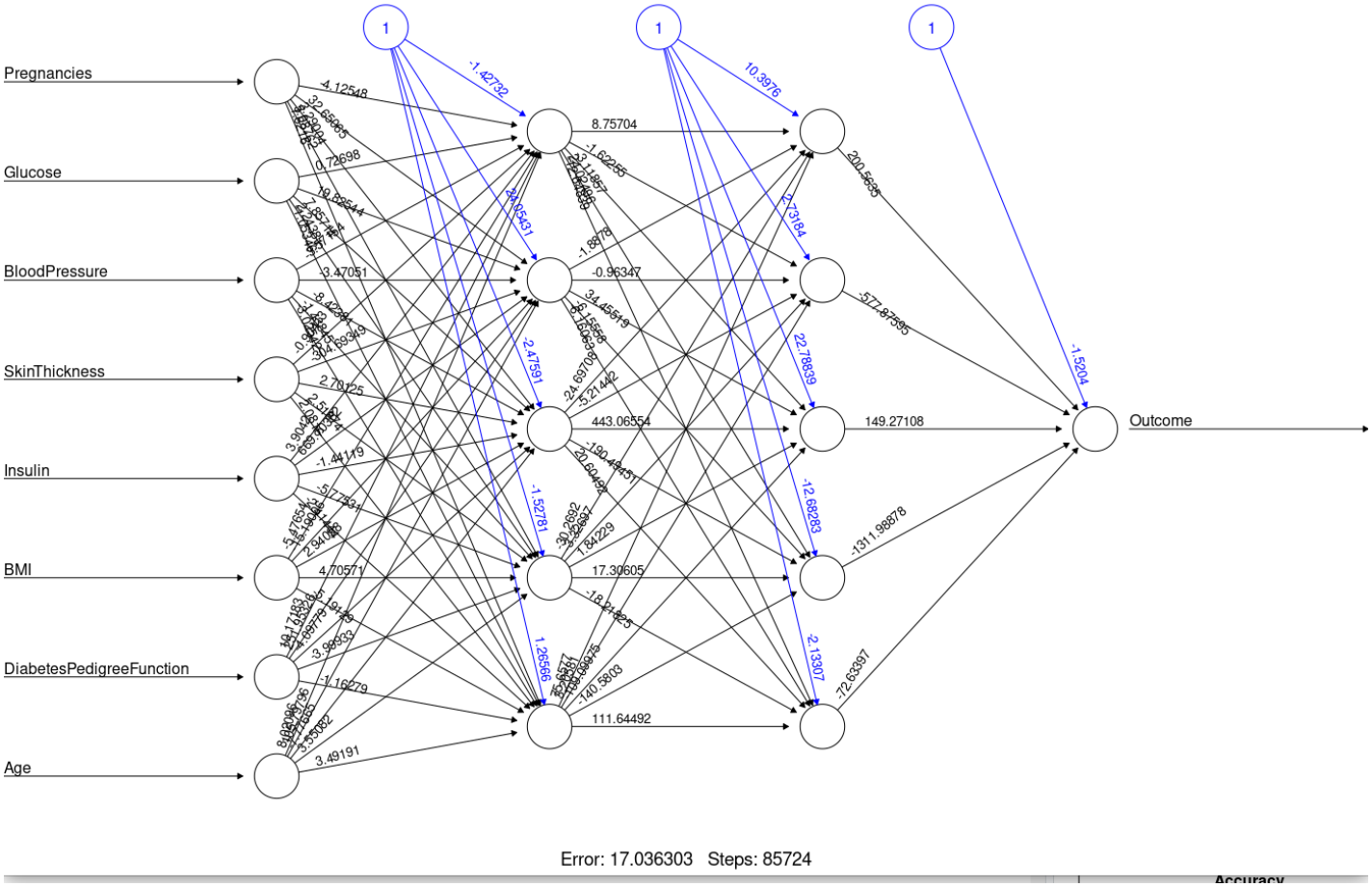
DETAILS

Sensitivity	Specificity	Precision	Recall	F1
0.779	0.705	0.85	0.779	0.813

Accuracy	Kappa
0.755	0.461

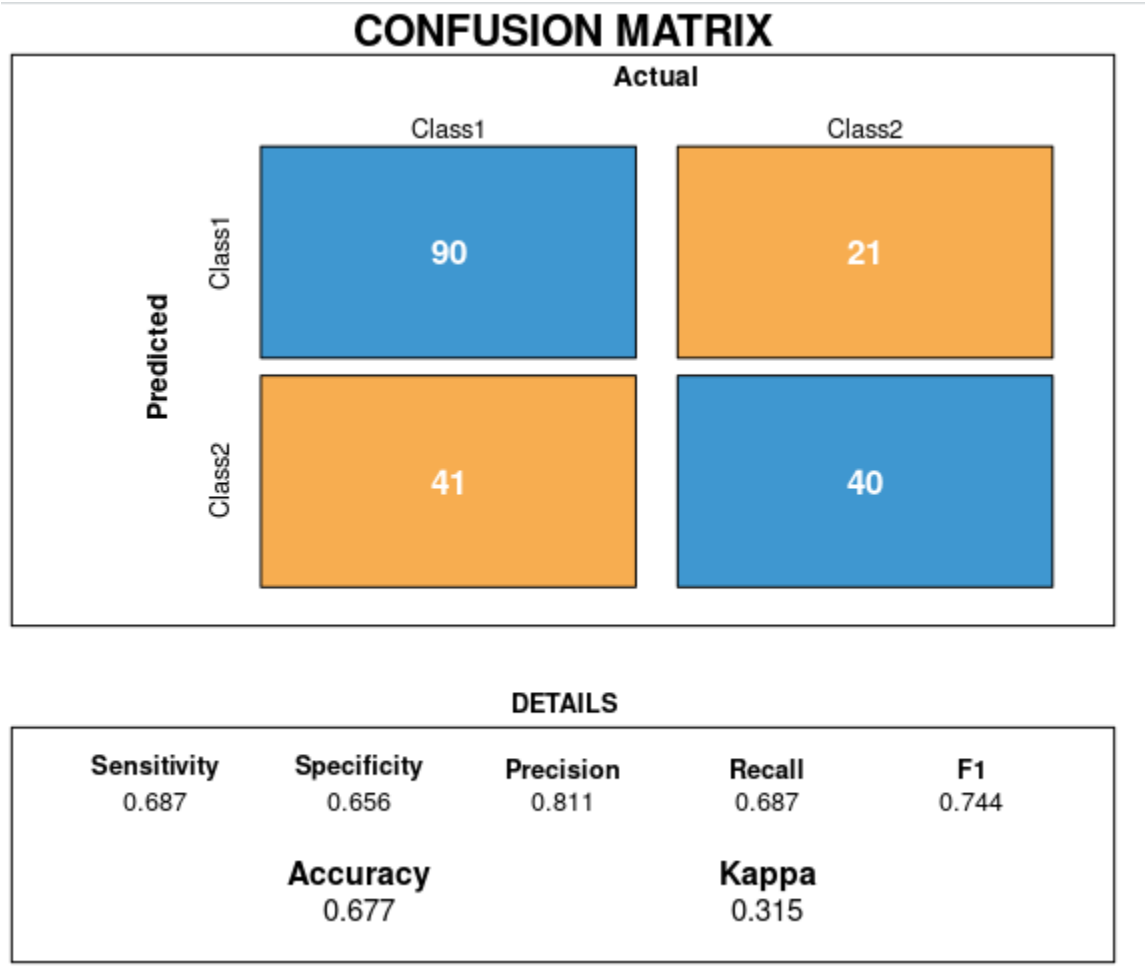
We notice that the accuracy has **improved** quite well. Which proves how the **unscaled** data can **badly** affect a NN model.

d) 2 layers & 5 Nodes NN.



A **bigger** NN with much more **weights** and **details** to catch.

And by looking at the confusion matrix plot

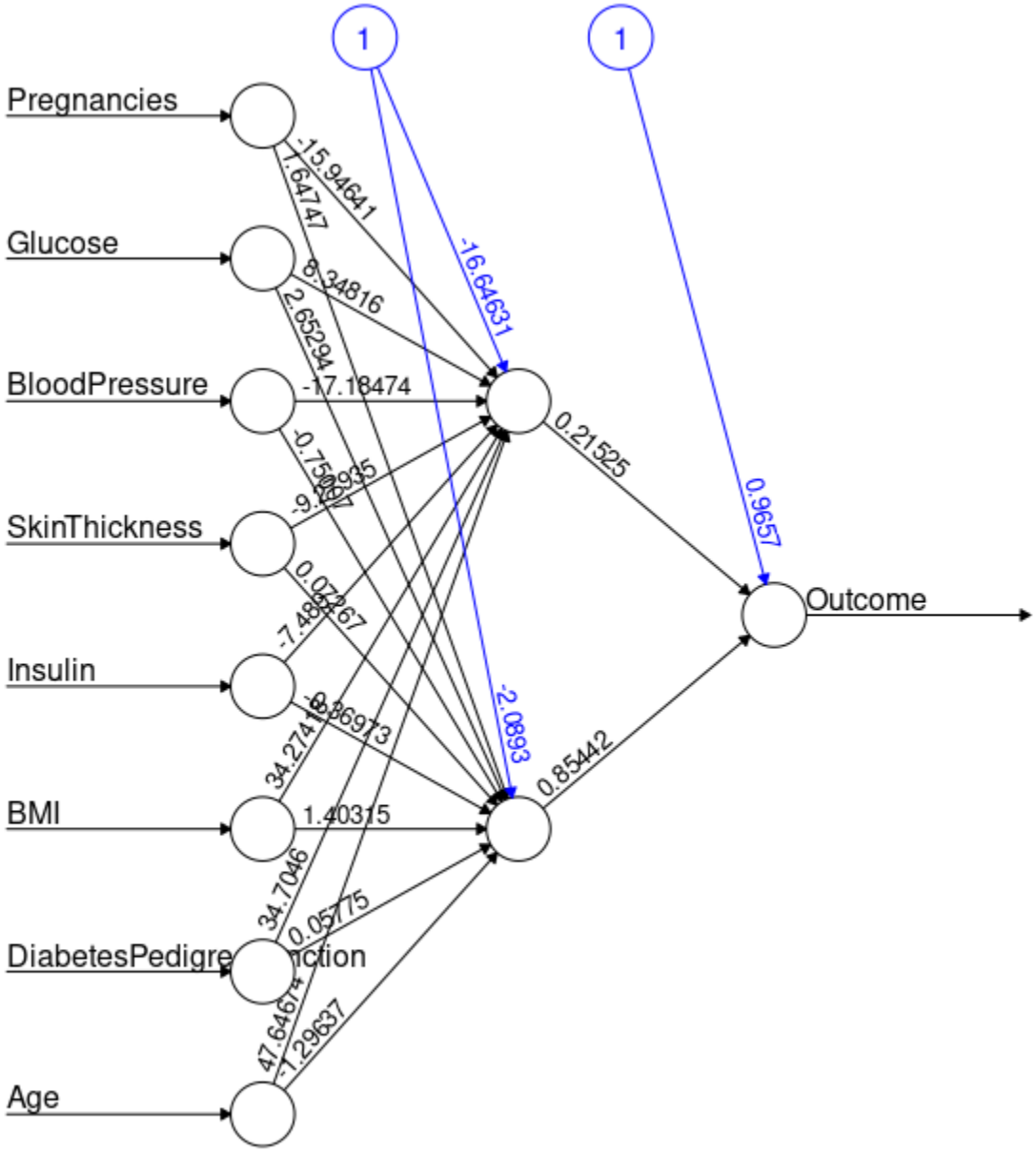


The accuracy got **worse**. This could be because making the NN bigger caught **more details** in the training set and thus it did **not** do a good job **generalizing** the model on the test set. Which means that it's not always a good idea to just make the NN bigger, sometimes the bigger it gets the more prone to overfitting it becomes.

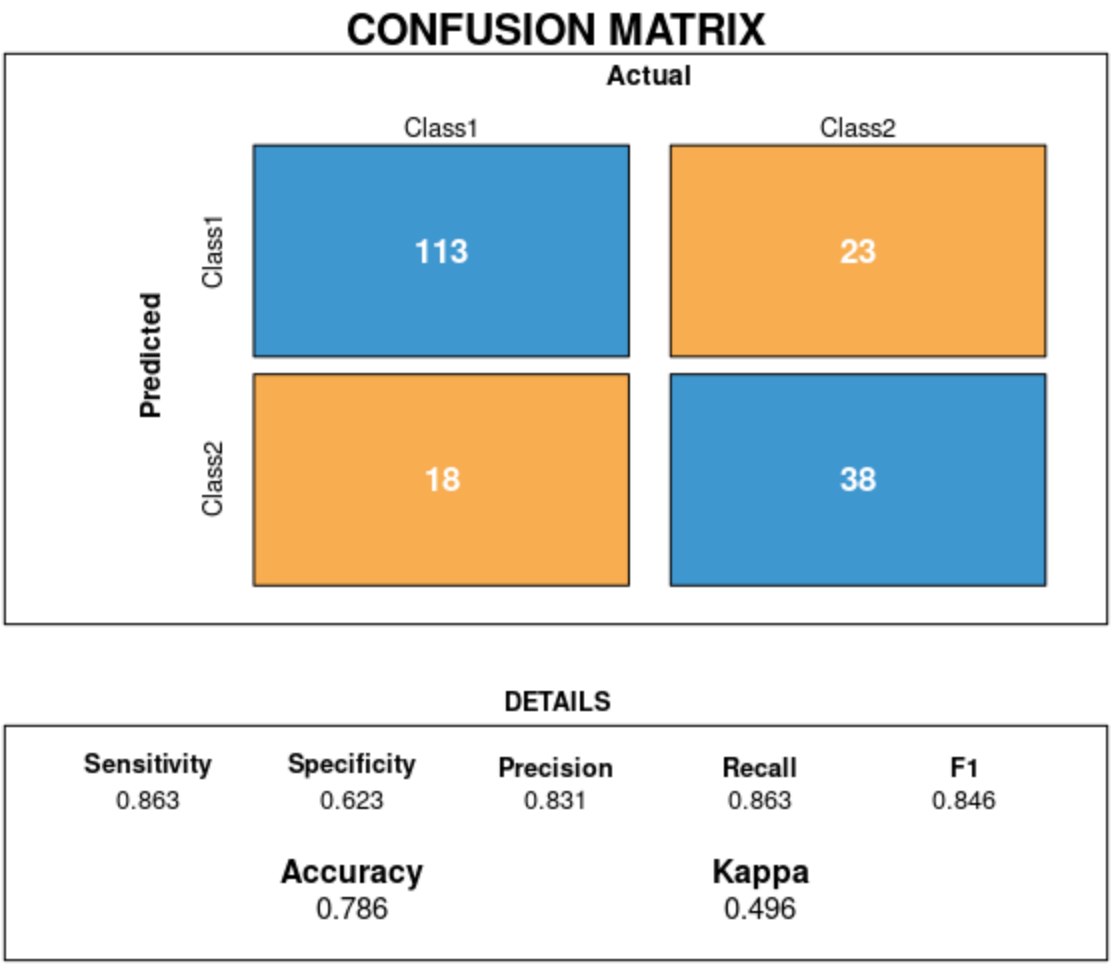
e) Tuning Activation Function, Learning Rate, Epochs.

Now trying out other methods to try to improve the accuracy of the 2 hidden nodes NN.

- **First change the activation function (tanh).**



Error: 41.319869 Steps: 5917

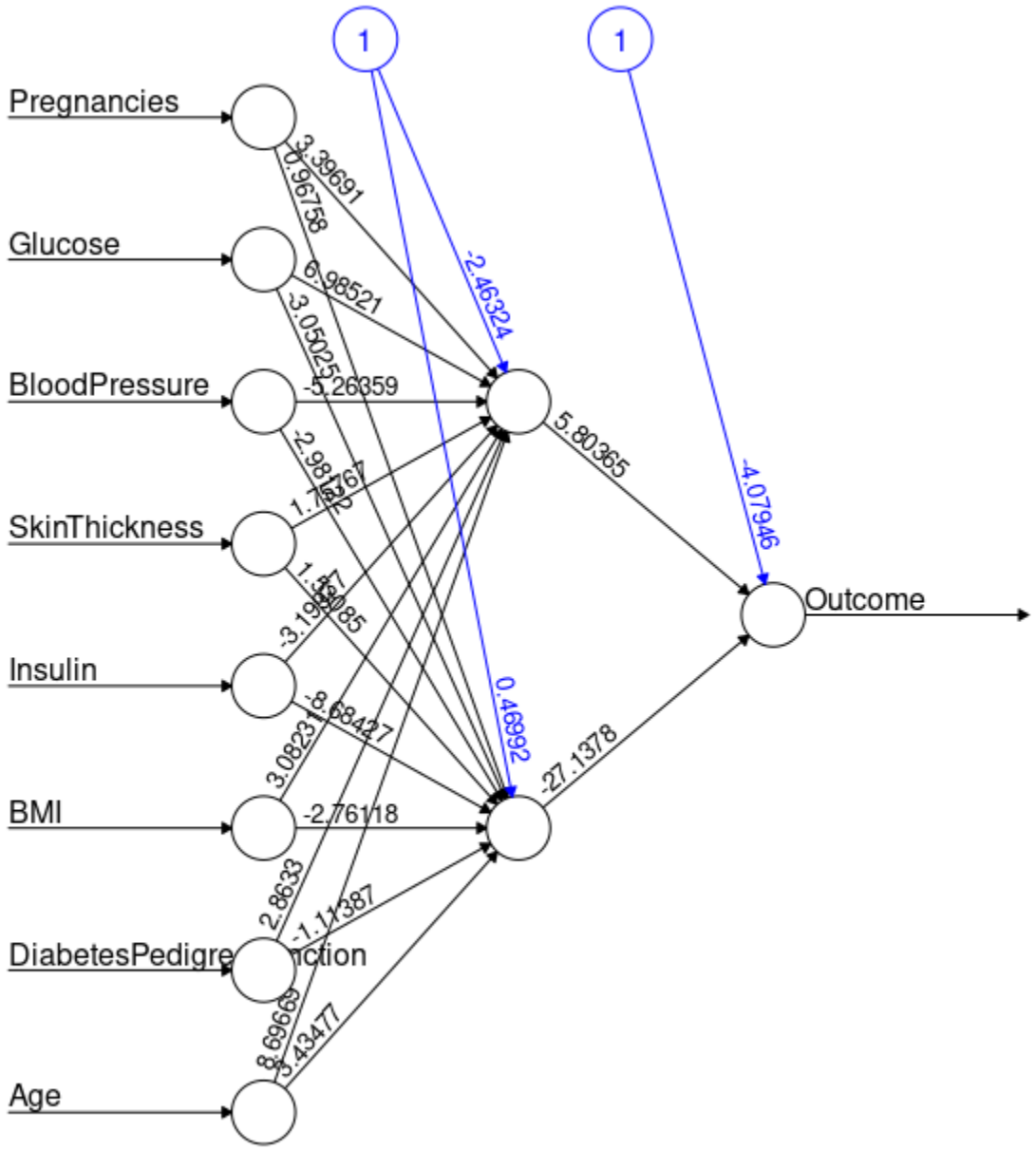


Accuracy got **better**.
I tried to search why tanh improved the performance and I found this:

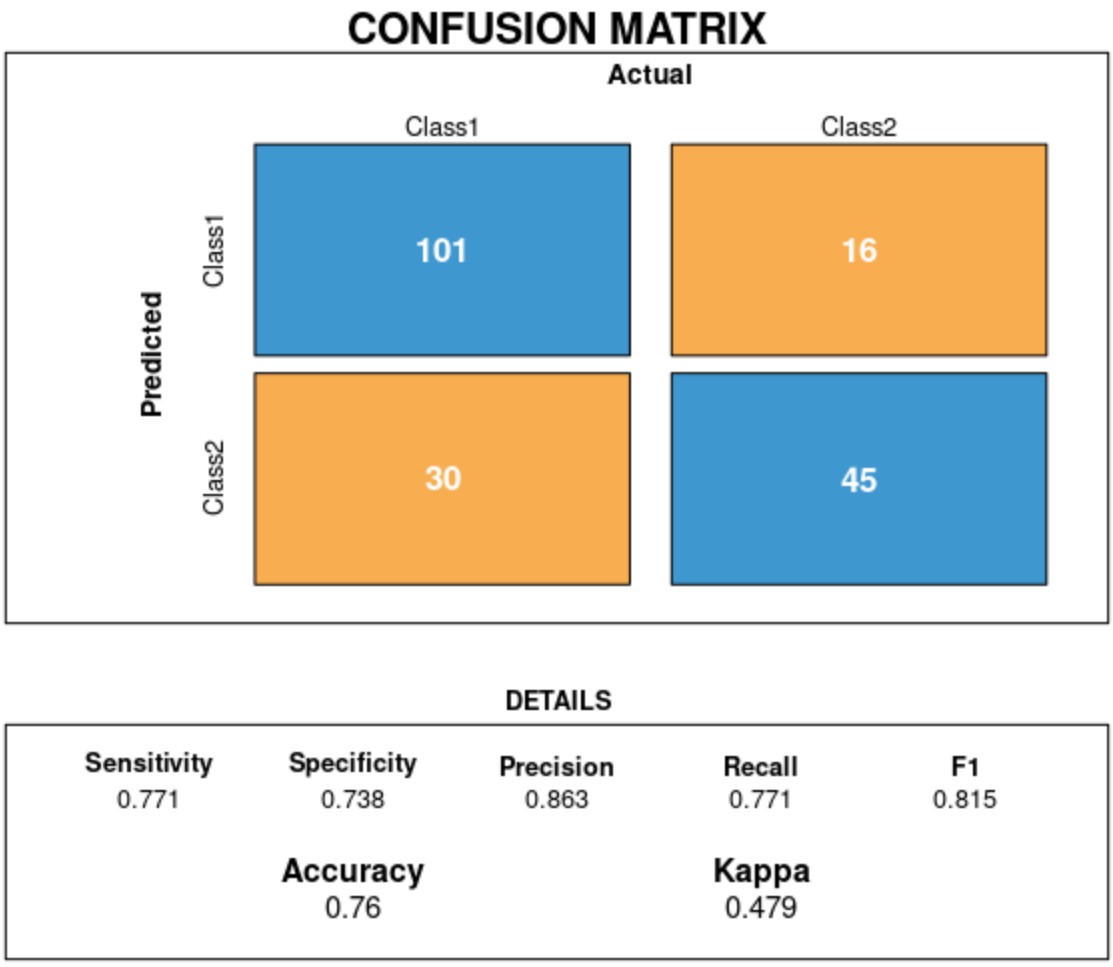
“tanh function is symmetric about the origin, where the inputs would be normalized and they are more likely to produce outputs (which are inputs to the next layer)and also, they are on an average close to zero.
It can also be said that data is centered around zero for tanh (centered around zero is nothing but the mean of the input data is around zero.”
Ref: link

In the case of our data here, it’s not centered around zero but rather somewhere around 0.4 which is very close to zero, so this could be the reason.

- **Changing the learning rate (0.01).**

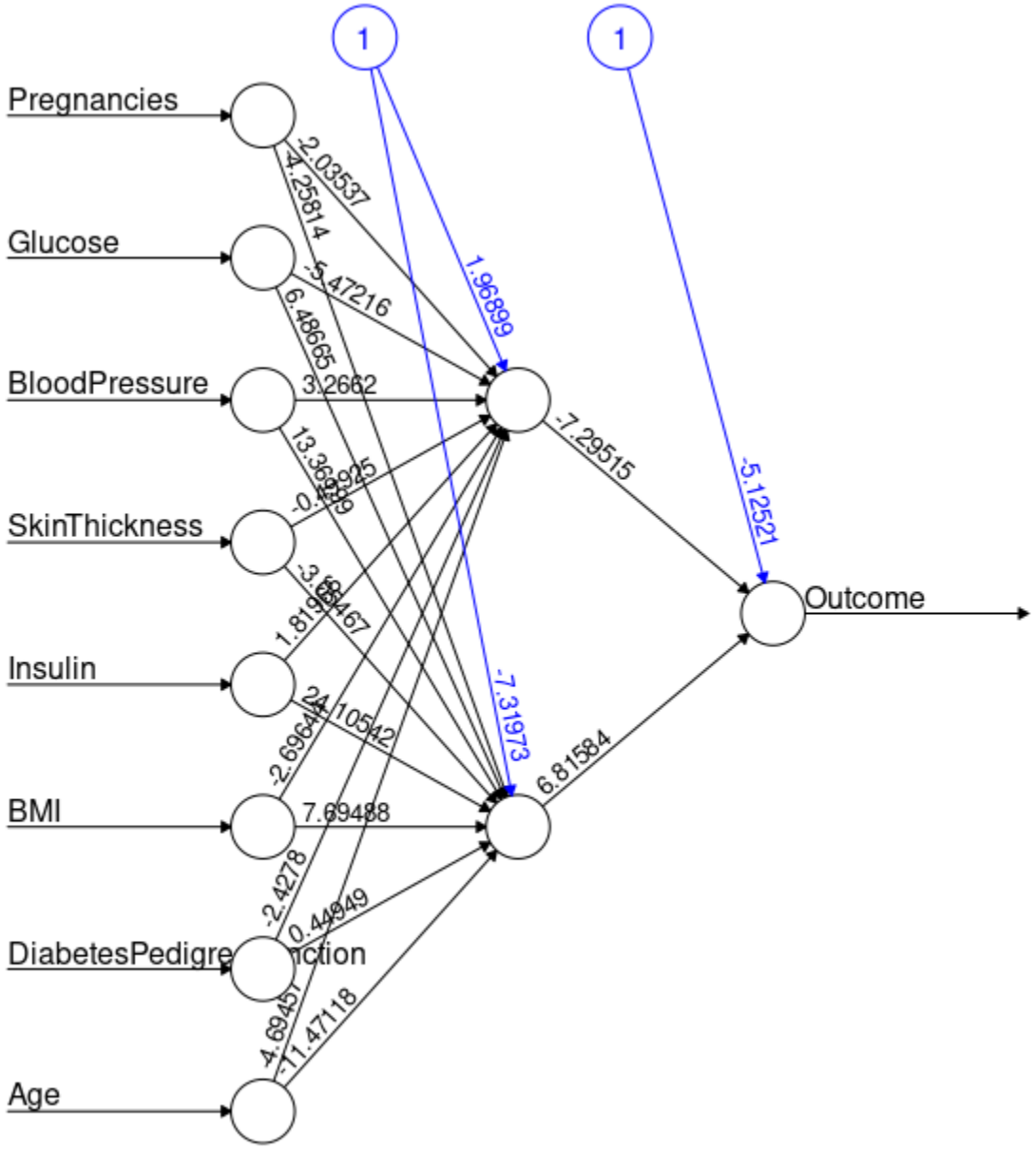


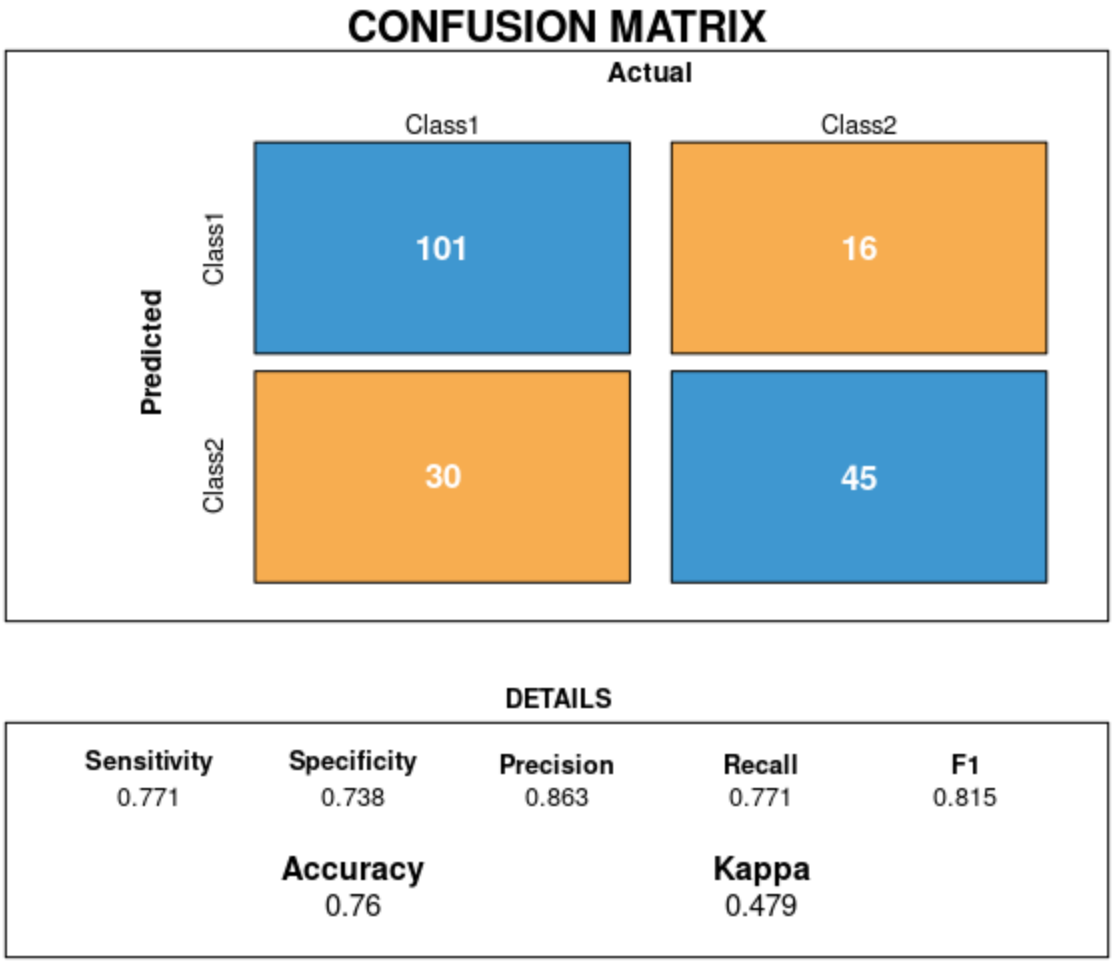
Error: 39.680781 Steps: 2590



We notice that it did in fact improve the accuracy but not as effectively as changing the activation function.
That's because decreasing the learning rate to 0.01, helps the algorithm converge better to the local minima. (R's neuralnet() function default learning rate is 0.1),

- **Changing the epochs number (10).**





As expected, after increasing the number of epochs, the accuracy only slightly improved. Because the more epochs we have, the more the model gets exposed to the training data only and thus the more prone to overfitting it becomes (i.e. decrease in model test accuracy). And here since the accuracy slightly improved it could mean that we're on the verge of causing overfitting.