DTI 5126[EG]: Fundamentals for Applied Data Science

Summer 2021
Assignment 2 Decision tree and Neural network Using R

By
Omar Sorour

Part A

a. There are some missing values in the dataset. Several strategies can be used to handle them, e.g., remove cases with unknowns. Apply one of these methods to address the missing values.

The Hypothyroid dataset contains **3772** records **1130** record of them have unknown values. I first tried to remove all unknown values and see how it would go so the dataset I worked on had **2642** records.

This has been implemented in below figure:

```
#rempving all values that has unkown value
Hypothyroid<-Hypothyroid[!(Hypothyroid$sex=="?"|Hypothyroid$TSH=="?"|Hypothyroid$T3=="?"|
                           Hypothyroid$TT4=="?"|Hypothyroid$T4U=="?"|
                           Hypothyroid$FTI=="?"|Hypothyroid$age=="?"),]
```

The unknown values were represented by a question mark so I just removed all records with the question mark in one of its field.

b. Perform attribute selection on the dataset and state briefly why attribute selection is sometimes important.

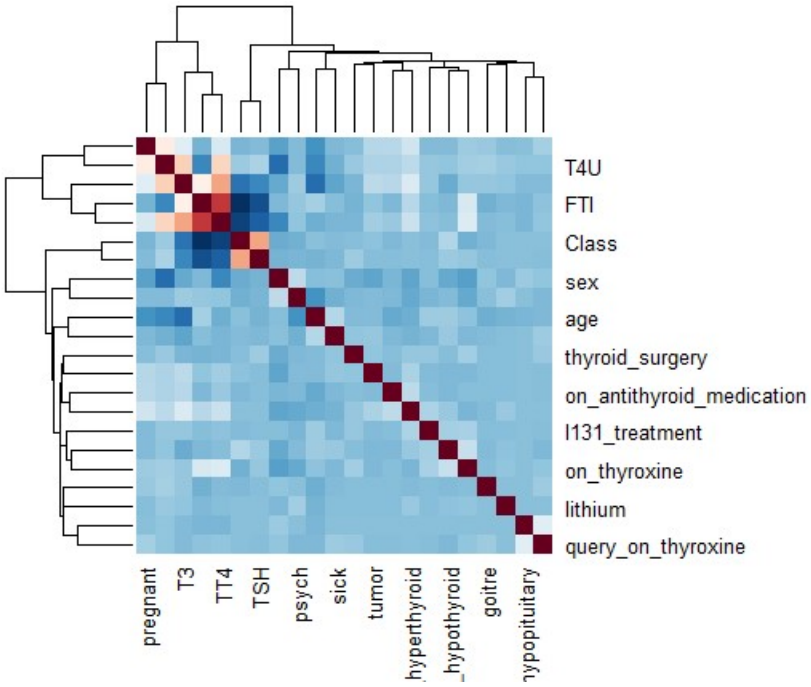
I noticed that the **TBG** column does not have any value so it was the first column to be removed. Also, all columns (**TBG_measured, FTI_measured, T4U_measured, TT4_measured, T3_measured, TSH_measured)** are just indicators whether the values have been recorded or not. For example, if we have a value for TSH for specific record the corresponding value of the TSH_measured, will be true. If not, the value would be false. Therefore, we can do without these columns.

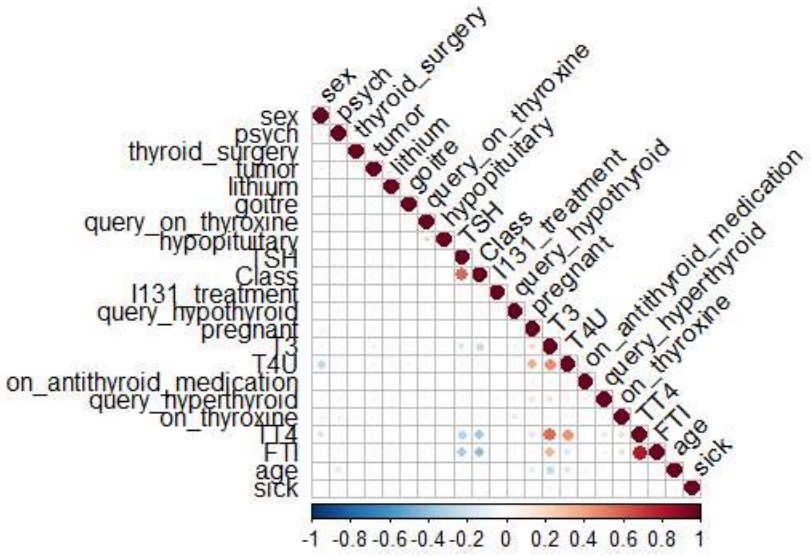
Also, the column **(referral_source)** indicates the hospital or the doctor who asked for these tests, so it does NOT have any effect on whether the patient has the disease or not hence, I removed it as well.

Code:

```
#Dropping un important columns from the dataset

Hypothyroid <- subset(Hypothyroid, select=-c(referral_source,
                                             TBG,
                                             TBG_measured,
                                             FTI_measured,
                                             T4U_measured,
                                             TT4_measured,
                                             T3_measured,
                                             TSH_measured))
```

I used also the correlation matrix to get an overview on the correlation between the columns we have in the dataset





As we can see there was a strong correlation between columns TT4 and the FTI column. We can do without any one of them but I first wanted to test how it would go with keeping both of them.

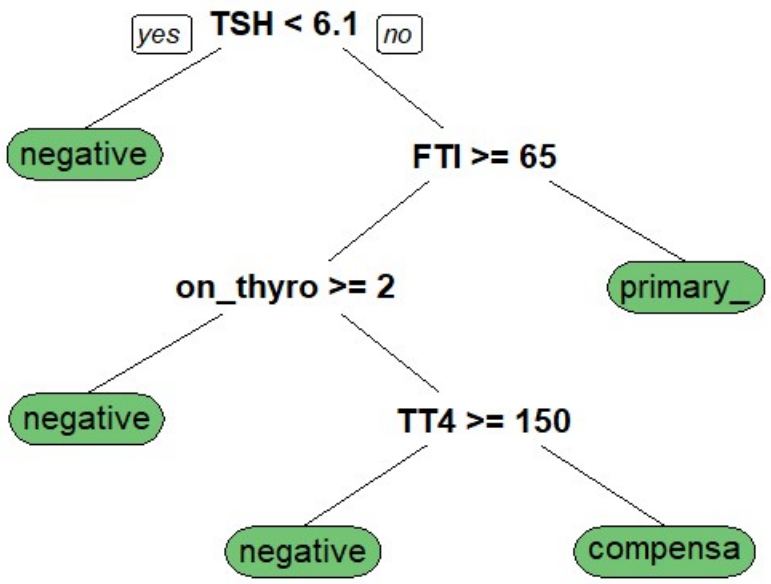
Also the class attribute firstly had four classes ("negative","compensated_hypothyroid", "primary_hypothyroid", "secondary_hypothyroid"). However, the class secondary_hypothyroid only had two records, even one of them has been removed because it had unknown values so we ended with only one column for this class. We cannot do any oversampling to increase the weight of this class so I had to remove this class from the dataset and had my model predict only the other three classes.

Removing that class by code:

```
#Removing the class secondary_hypothyroid from the dataset
Hypothyroid <- Hypothyroid[!(Hypothyroid$Class=="secondary_hypothyroid"),]
```

c. Split the dataset into a train and test set using k-fold cross-validation (k= 10). Create a decision tree model using the selected attributes from your dataset that can predict the type of thyroid disease a patient has.

Here was the resulted tree created after the training with 10 folds cross validation.



This model achieved 99.64% accuracy on the testing set that I set aside before doing the 10-fold cross validation.

```
                          Reference
Prediction                 negative compensated_hypothyroid primary_hypothyroid
  negative                      760                       0                   0
  compensated_hypothyroid         0                      48                   0
  primary_hypothyroid             3                       0                  29

Overall Statistics

               Accuracy : 0.9964
                 95% CI : (0.9896, 0.9993)
    No Information Rate : 0.9083
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9794

 Mcnemar's Test P-Value : NA
```

Code: splitting dataset to training and test data

```
Hypothyroid_final$Class<-factor(Hypothyroid_final$Class, levels = c(1, 2, 3),
                                labels = c("negative","compensated_hypothyroid",
                                           "primary_hypothyroid"))
set.seed(123)
sample_data = sample.split(Hypothyroid_final, SplitRatio = 0.7)
train_data <- subset(Hypothyroid_final, sample_data == TRUE)
test_data <- subset(Hypothyroid_final, sample_data == FALSE)
#
```

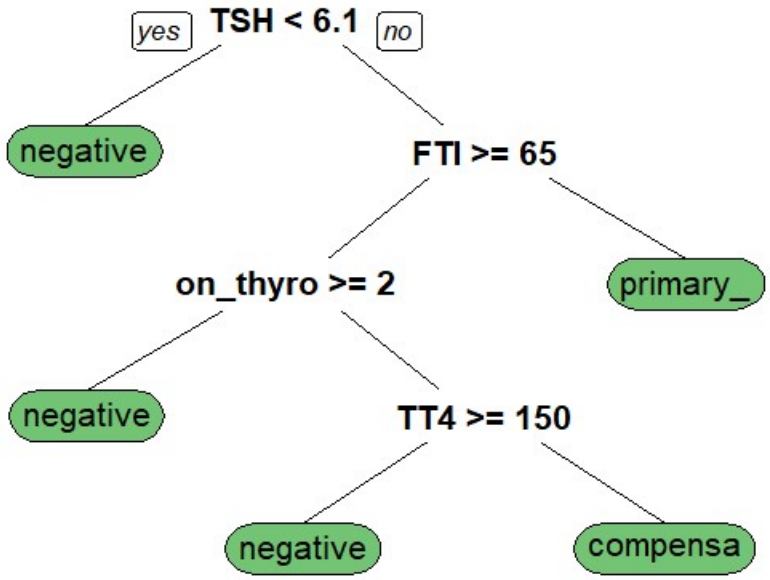
Code: using cross validation and training the decision tree model:

```
set.seed(123) #generates a reproducible random sampling

#specify the cross-validation method
ctrl <- trainControl(method = "cv", number = 10)

#fit a decision tree model and use k-fold CV to evaluate performance
dtree_fit_gini <- train(Class~., data = Hypothyroid_final,
                        method = "rpart", parms = list(split = "gini")
                        , trControl = ctrl, tuneLength = 10)

#Step 5: Evaluate - view summary of k-fold CV
print(dtree_fit_gini) #metrics give us an idea of how well the model performed on previously u

#view final model
dtree_fit_gini$finalModel
prp(dtree_fit_gini$finalModel, box.palette = "Greens", tweak = 1.3) #view the tree using prop(

#view predictions for each fold
dtree_fit_gini$resample
```

Code: measuring the accuracy on the test dataset

```
#view predictions for each fold
dtree_fit_gini$resample

#Check accuracy
test_pred_gini <- predict(dtree_fit_gini, newdata = test_data)
confusionMatrix(test_pred_gini, test_data$Class )  #check accuracy
```

d. Visualize and describe the first few splits in the decision tree. Extract some rules.



The model first checks the TSH value, if the TSH is less than 6.1 then the patient is normal (negative).

If the TSH is more than 6.1 the model will check the FTL value. If it's equal or less than 65 then the patient is diagnosed as primary hypothyroid case.

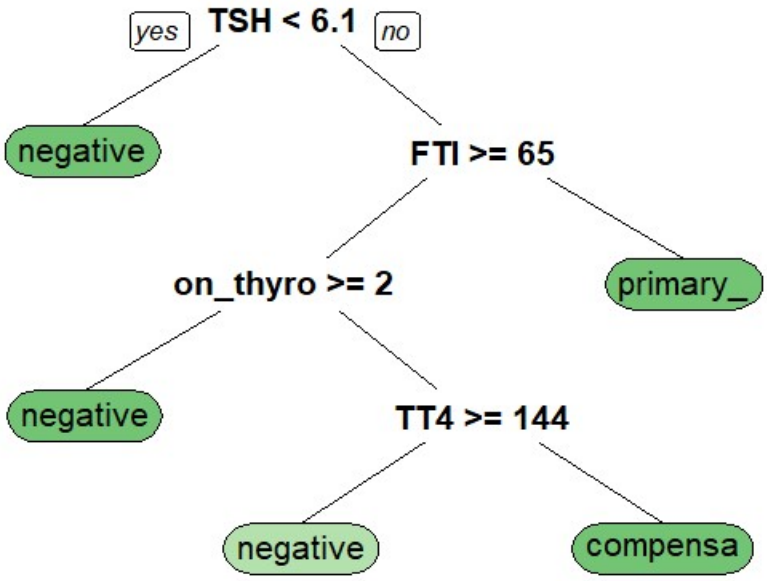
If the FTI is greater than 65, the models check the on_thyro value an so on.

e. Try different ways to improve your decision tree algorithm, e.g., use different splitting strategies, prune tree after splitting.

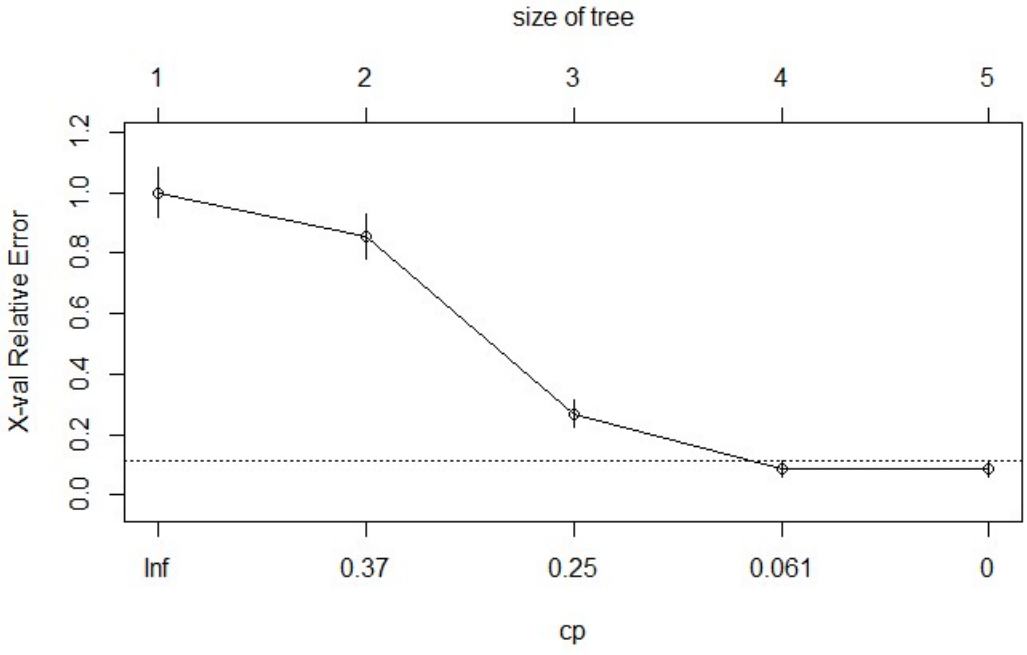
There are different strategies of splitting like gini, gain ratio, information gain, entropy, and chi-square. I tried each one of them and calculated the complexity parameter for each one of them.

And I used that complexity parameter CP for pruning the tree therefore minimizing the risk of overfitting and maximizing the generalization for all unseen data.

The figure below shows the results for training the decision tree with the CHI-square splitting strategy.

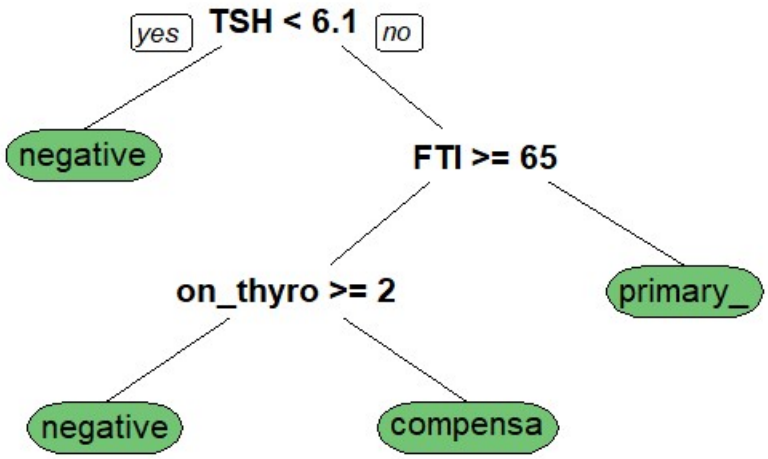


I also plotted the complexity parameter versus the relative error along with the size of the tree in this case here was the results:



As we can see the cp of 0.061 achieves a very good comprise between the size of the tree and the generalization that this model can have on the unseen data.

Therefore, I pruned the tree with this value of CP. This was the pruned tree:



As we can see here the last node of the tree was pruned. I also applied that pruned tree on the testing set that I left aside and calculated the accuracy and it was 99.29%.

Code:

```
DT_newmodel2 <- rpart(Class ~ ., data = train_data, method = "class",
                      parms=list(split = "chi"),
                     control = rpart.control(cp = 0,maxdepth = 5, minsplit =20))

summary(DT_newmodel2)
#Plot Decision Tree
prp(DT_newmodel2, box.palette = "Greens", tweak = 1.3)
# Examine the complexity plot
printcp(DT_newmodel2)
plotcp(DT_newmodel2)
pred2 <-(predict(DT_newmodel2, newdata = test_data,type="class"))
confusionMatrix(pred2, test_data$Class)  #check accuracy on the training set
mean(pred2==test_data$Class)
opt <- which.min(DT_newmodel2$cptable[,"xerror"]) #Caculating the CP
cp <- DT_newmodel2$cptable[opt, "CP"]
pruned_model2 <- prune(DT_newmodel2,cp) # pruning the tree
prp(pruned_model2, box.palette = "Greens", tweak = 1.3)
pred2_pruned <-(predict(pruned_model2, newdata = test_data,type="class"))
confusionMatrix(pred2_pruned, test_data$Class)  #check accuracy on the tesing set
mean(pred2_pruned==test_data$Class)
#_____________________________________________________________
```

I also did the same last three steps using five different splitting strategy and got the same trees. In the attached R script “R Part one.R” we will find 6 different models DT_newmodel1:6 along with their pruned models “pruned_model1:6”

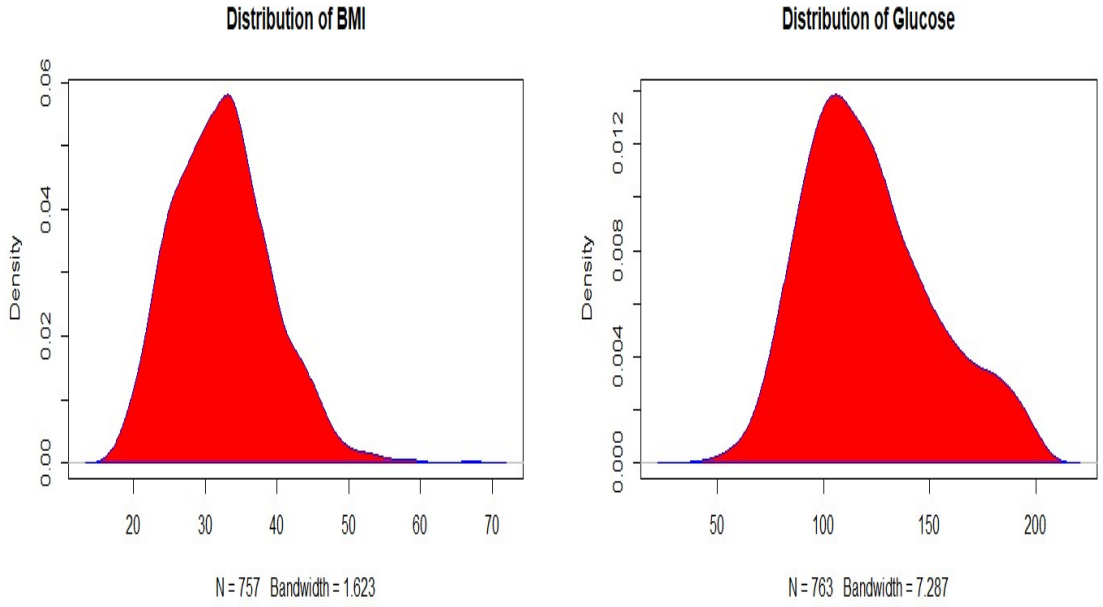
The Gini splitting strategy achieved relatively the best accuracy that was **99.52**%. but overall, all the algorithms did well on that dataset after the feature selection criteria that we set at the beginning.

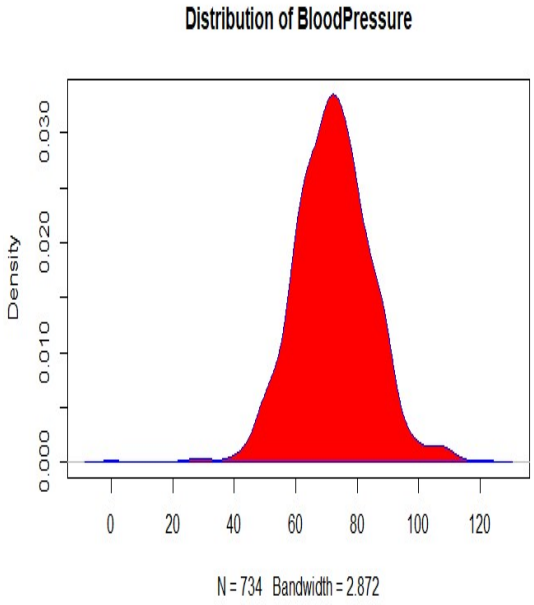
Part B

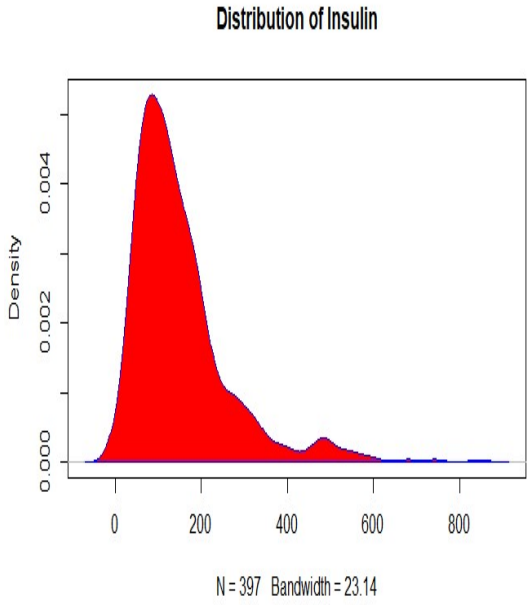
a. Some data points are not available, handle the missing data by applying central measure of tendency to derive the missing value.

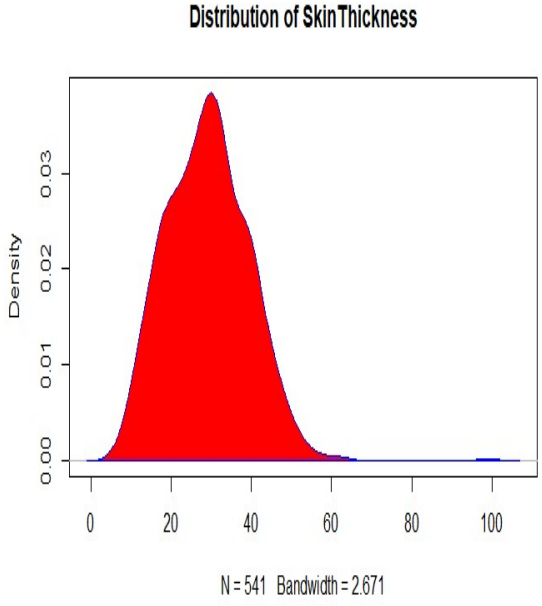
First of all, the unknown values were represented by a question mark. So, we can easily convert them to NA by just change the type of these column from “chr” to “numeric” using as.numeric function.

For each column I calculated the mean, the median and plotted the probability distribution to figure out how we are going to fill in the NA values.




Distribution of BloodPressure
Density
0.000
0.010
0.020
0.030
0
20
40
60
80
100
120
N = 734 Bandwidth = 2.872


Distribution of Insulin
Density
0.000
0.002
0.004
0
200
400
600
800
N = 397 Bandwidth = 23.14


Distribution of SkinThickness
Density
0.00
0.01
0.02
0.03
0
20
40
60
80
100
N = 541 Bandwidth = 2.671

Here was the summary of the above figures:

Measure\Column	BMI	Glucose	SkinThickness	BloodPressure	Insulin
Mean	32.45	121.6867	29.1534	72.3065	154.3727
Median	32.3	117	29	72	125
Null values count	11	5	227	34	371

AS we can see that nearly all of the columns have near normal distribution except the insulin column there was a big difference between the mean and median values and the Skin thickness column there was a difference as well to some extent.

So, for all columns except these two columns, we can replace the NAs with mean value of the column.

For the insulin and SkinThickness column (because they have a big number of NAs) It's better to calculate the mean and the median per each class than way we can have better representation of the unknown value.

Measure\ Class	Class 0 (Insulin)	Class1 (Insulin)	Class 0 (ST)	Class 1 (ST)
Mean	129.7962	203.7121	27.235	33
Median	100	168	27	32

So according to the previous table we are going to replace all NAs values of insulin columns with median of class 0 if they belonged to that class and the same for all NA values of class 1.

We also did the same for SkinThickness column ST.

Code:

```
# updating the na values for BMI, Glucose, and Bloodpressure columns with means calculated
diabetes$BMI[is.na(diabetes$BMI)] = BMI_mean
diabetes$Glucose[is.na(diabetes$Glucose) ]=Glucose_mean
diabetes$BloodPressure[is.na(diabetes$BloodPressure)]= BloodPressure_mean
# Updating the Na values of insulin for with the median each class

diabetes$Insulin[is.na(diabetes$Insulin) & diabetes$Outcome==0]<- median_insulin_class0

diabetes$Insulin[is.na(diabetes$Insulin) & diabetes$Outcome==1]<- median_insulin_class1

# Updating the Na values of SkinThickness for with the median each class
diabetes$SkinThickness[is.na(diabetes$SkinThickness) & diabetes$Outcome==0]<- median_SkinThickness_class0

diabetes$SkinThickness[is.na(diabetes$SkinThickness) & diabetes$Outcome==1]<- median_SkinThickness_class1
```

b. Partition the dataset into a train dataset (75%) and test dataset (25%). Use the train dataset to build the Neural Network and the test dataset to evaluate how well the model generalizes to future results.

Code:

```
# splitting the dataset into training and testing data
set.seed(123)
sample_data = sample.split(diabetes, SplitRatio = 0.7)
train_data <- subset(diabetes, sample_data == TRUE)
test_data <- subset(diabetes, sample_data == FALSE)
```

c. Neural networks work best when the input data are scaled to a narrow range around zero. Rescale the data with a normalizing (e.g., min_max normalization) or standardization (e.g., z_score standardization) function.

I tried to train the model using ZScore method. There is pre-built function in R called scale that we can use to obtained this.

```
#standardization of all of the columns' values using zscore
diabetes[c(1: 8)] <- scale(diabetes[c(1: 8)])
```

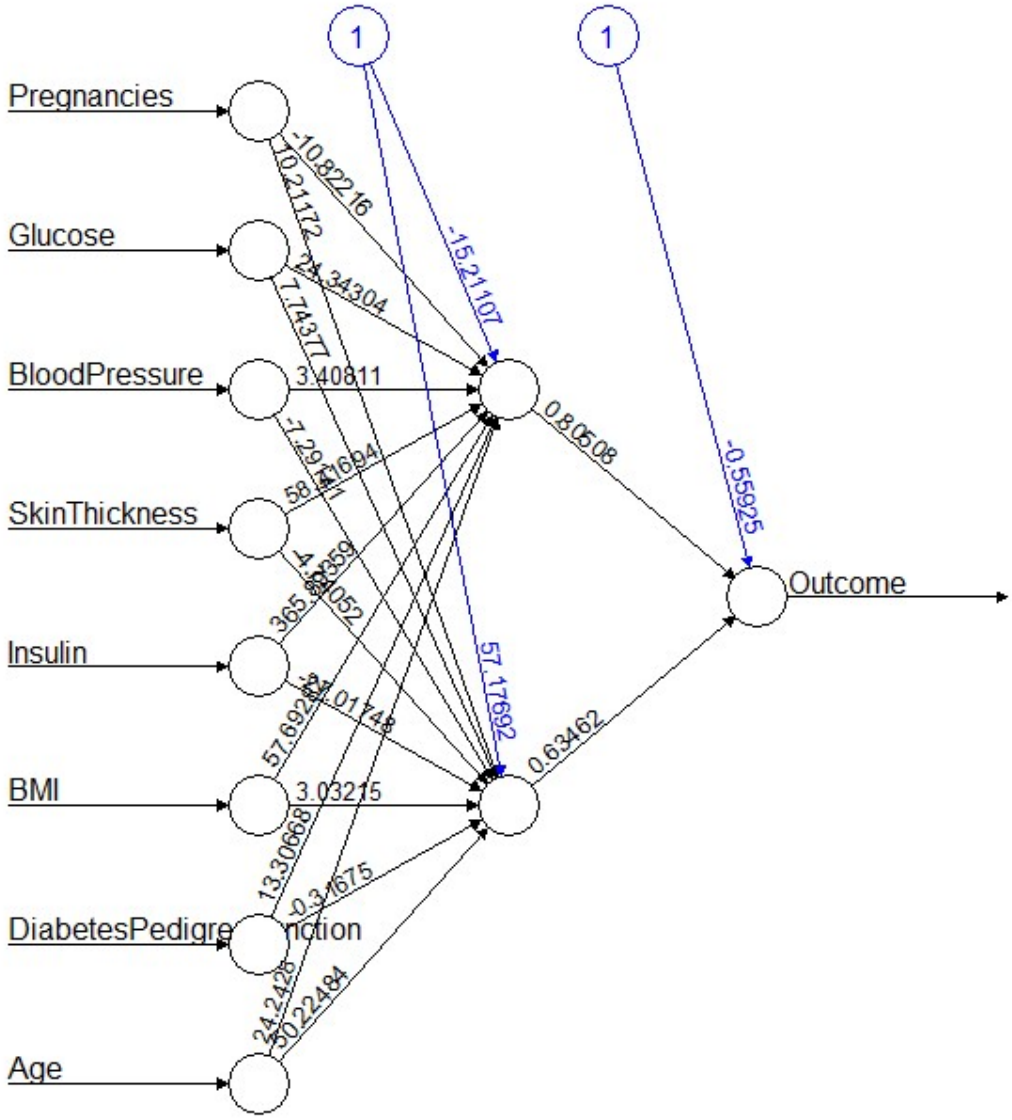
Before standardization:

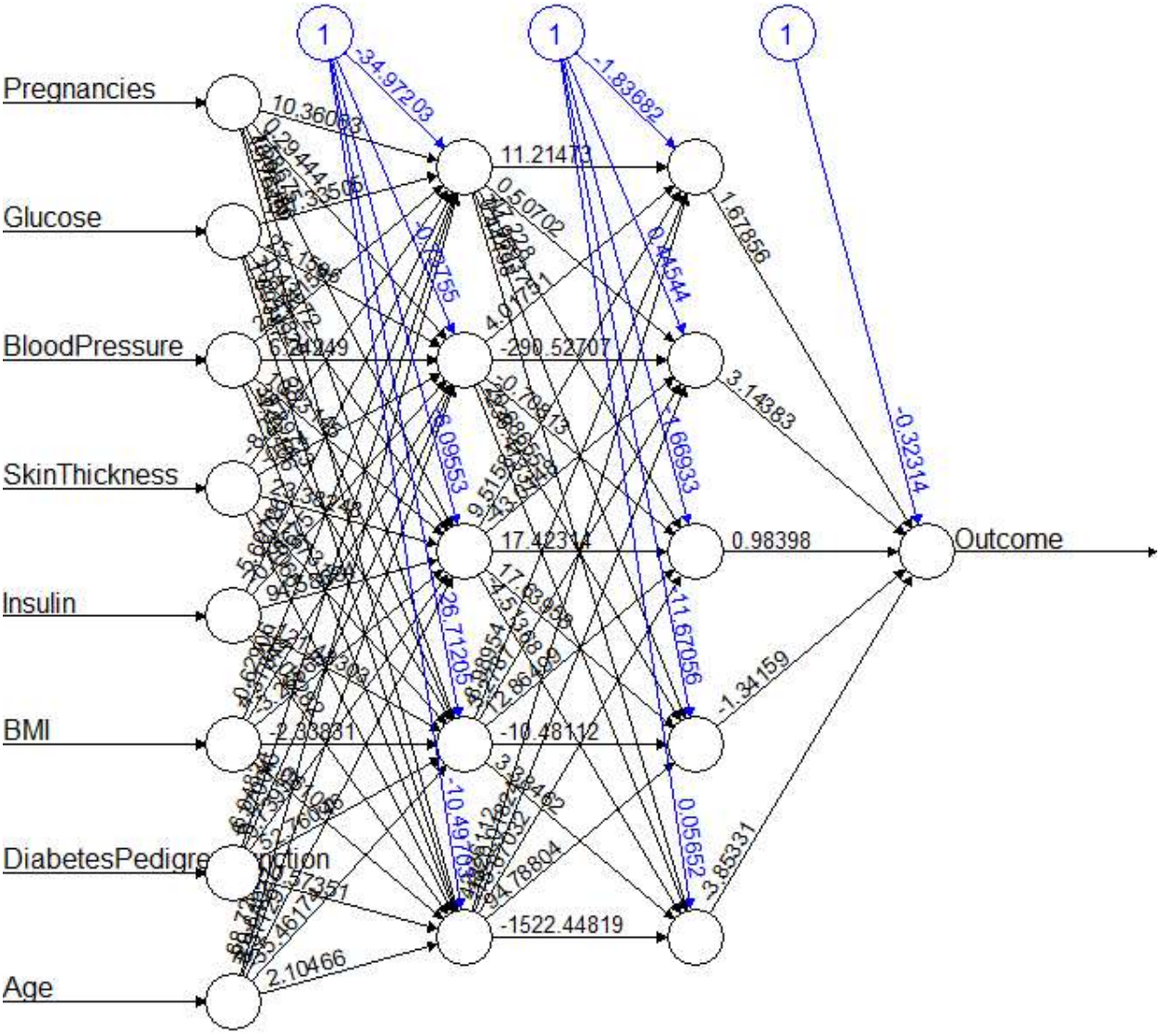
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	6	148.0000	72.00000	35	168	33.60000	0.627	50	1
2	1	85.0000	66.00000	29	100	26.60000	0.351	31	0
3	8	183.0000	64.00000	32	168	23.30000	0.672	32	1
4	1	89.0000	66.00000	23	94	28.10000	0.167	21	0
5	0	137.0000	40.00000	35	168	43.10000	2.288	33	1
6	5	116.0000	74.00000	27	100	25.60000	0.201	30	0
7	3	78.0000	50.00000	32	88	31.00000	0.248	26	1
8	10	115.0000	72.30654	27	100	35.30000	0.134	29	0
9	2	197.0000	70.00000	45	543	30.50000	0.158	53	1
10	8	125.0000	96.00000	32	168	32.45746	0.232	54	1
11	4	110.0000	92.00000	27	100	37.60000	0.191	30	0
12	10	168.0000	74.00000	32	168	38.00000	0.537	34	1
13	10	139.0000	80.00000	27	100	27.10000	1.441	57	0
14	1	189.0000	60.00000	23	846	30.10000	0.398	59	1
15	5	166.0000	72.00000	19	175	25.80000	0.587	51	1
16	7	100.0000	72.30654	32	168	30.00000	0.484	32	1
17	0	118.0000	84.00000	47	230	45.80000	0.551	31	1
18	7	107.0000	74.00000	32	168	29.60000	0.254	31	1

After standardization:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	0.63953049	0.86454467	-0.02477032	0.66474818	0.310254771	0.166183444	0.46818687	1.42506672	1
2	-0.84433482	-1.20537602	-0.50960814	-0.01010523	-0.447236822	-0.851975963	-0.36482303	-0.19054773	0
3	1.23307662	2.01450060	-0.67122075	0.32732148	0.310254771	-1.331965398	0.60400370	-0.10551539	1
4	-0.84433482	-1.07395248	-0.50960814	-0.68495863	-0.514074315	-0.633798947	-0.92016296	-1.04087112	0
5	-1.14110788	0.50312994	-2.61057201	0.66474818	0.310254771	1.547971211	5.48133703	-0.02048305	1
6	0.34275743	-0.18684362	0.13684228	-0.23505636	-0.447236822	-0.997427307	-0.81754580	-0.27558007	0
7	-0.25078869	-1.43536720	-1.80250898	0.32732148	-0.580911809	-0.211990050	-0.67569267	-0.61570943	1
8	1.82662274	-0.21969950	0.00000000	-0.23505636	-0.447236822	0.413450729	-1.01976197	-0.36061241	0
9	-0.54756176	2.47448297	-0.18638293	1.78950385	4.487598117	-0.284715722	-0.94732633	1.68016374	1
0	1.23307662	0.10885934	1.91458094	0.32732148	0.310254771	0.000000000	-0.72398310	1.76519608	1
1	0.04598437	-0.38397892	1.59135573	-0.23505636	-0.447236822	0.747988820	-0.84772732	-0.27558007	0
2	1.82662274	1.52166234	0.13684228	0.32732148	0.310254771	0.806169357	0.19655321	0.06454929	1
3	1.82662274	0.56884171	0.62168010	-0.23505636	-0.447236822	-0.779250291	2.92496245	2.02029310	0
4	-0.84433482	2.21163590	-0.99444596	-0.68495863	7.862891540	-0.342896259	-0.22296990	2.19035777	1
5	0.34275743	1.45595057	-0.02477032	-1.13486090	0.388231847	-0.968337038	0.34746080	1.51009906	1
6	0.93630355	-0.71253776	0.00000000	0.32732148	0.310254771	-0.357441394	0.03659116	-0.10551539	1
7	-1.14110788	-0.12113185	0.94490531	2.01445499	1.000908871	1.940689840	0.23880733	-0.19054773	1

d. Train & plot a simple Neural Network with only 2 hidden nodes (not layer). Then, train & plot a multilayer perceptron with 2 layers & 5 nodes. What impact does the change in the number of layers & nodes have on the accuracy of your model?





Error: 10.721224 Steps: 27220

To measure how well the two networks did, I calculated the accuracy on both the training and on the testing set as well. The following table gives a quick insight about the performance of the both models.

Neural network	2 nodes	2 layers 5 nodes
acc. Training set	90.06%	93.57%
acc. Testing set	86.27%	79.61%

We can see that the accuracy of the model **increased on the training set but at the same time it decreased on the testing dataset**. That means that increasing the number of hidden layers reduces the bias but on the other hand increases the variance and causes overfitting. So, we have to carefully choose the number of the hidden layers that achieves the best bias and variance.

e. Try changing the activation function, varying the learning rate, epochs or removing the bias. What effects does any of these have on the result?

- Changing the activation function from “logistic” to “tanh” reduced the accuracy of the model.
- Changing the learning rate has no significance effect on the accuracy of the model.
- Tried many algorithms such as “the resilient backpropagation”, “back propagation” and “the resilient backpropagation” achieved the best accuracy.