

Diabetes Risk Prediction

Madibaa

2024-04-02

Introduction

Diabetes is a chronic, metabolic disease characterised by elevated blood sugar levels, which leads over time to damage to the organs¹. The incidence of diabetes has rapidly risen in association with the global rise in obesity, sedentary lifestyles, consumption of ultra-processed foods, and rapidly ageing populations. It is estimated that there are 537 million individuals living with diabetes as of 2021². As a healthcare researcher, I'm deeply interested in understanding the risk factors associated with diabetes, in hopes of making a small contribution towards humanity's efforts in ridding us of this disease.

About the dataset

This dataset was provided by the National Institute of Diabetes and Digestive and Kidney Diseases. The data are from a population of females of Pima Indian heritage, aged 21 years and above, with 8 medical predictors of diabetes outcome:

- number of times pregnant
- plasma glucose concentration a 2 hours in an oral glucose tolerance test
- diastolic blood pressure readings (mm Hg)
- triceps skinfold thickness (mm)
- 2-Hour serum insulin (μ U/ml)
- body mass index ($\text{weight in kg}/(\text{height in m})^2$)
- diabetes pedigree function (DPF)
- age (years)

To access the dataset or view others attempts to build classification models, visit <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database/data> or <https://www.kaggle.com/datasets/mathchi/diabetes-data-set/data>

Objective

The aim of this project is to develop train and evaluate several machine learning models and ensembles of models to optimise accuracy and area under the receiver operating characteristic curve (AUC). For this project, we will consider the following models: general linear model (GLM), k-nearest neighbours (KNN), locally estimated scatterplot smoothing (LOESS), recursive partitioning and regression trees (rpart), random forest (RF), extreme gradient boosting (XGB), and naive Bayes (NB).

Let's dive right in!

Methods and Results

Exploratory Analyses

Let us begin by importing the diabetes dataset.

```
# Set the working directory accordingly  
setwd("C:/Users/cyc_c/Desktop/Data Science/projects/diabetes_risk_prediction")  
diabetes <- read.csv("diabetes.csv")
```

Next, we will load some required packages.

```
library(tidyverse) # for it's awesomeness
```

```
## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --  
## v dplyr      1.1.2      v readr      2.1.4  
## v forcats    1.0.0      v stringr   1.5.0  
## v ggplot2     3.4.2      v tibble     3.2.1  
## v lubridate  1.9.2      v tidyr      1.3.0  
## v purrr       1.0.1  
## -- Conflicts ----- tidyverse_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag()     masks stats::lag()  
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(caret) # for training our models
```

```
## Loading required package: lattice  
##  
## Attaching package: 'caret'  
##  
## The following object is masked from 'package:purrr':  
##  
## lift
```

```
library(corrplot) # for visualising correlations
```

```
## corrplot 0.92 loaded
```

```
library(gam) # for building a Loess model using the caret package
```

```
## Loading required package: splines  
## Loading required package: foreach  
##  
## Attaching package: 'foreach'  
##  
## The following objects are masked from 'package:purrr':  
##  
## accumulate, when  
##  
## Loaded gam 1.22-3
```

```
library(rpart) # for building a classification tree model using the caret package
library(randomForest) # for building a random forest model using the caret package
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##   combine
##
## The following object is masked from 'package:ggplot2':
##
##   margin
```

```
library(xgboost) # for building an extreme gradient boosting model using the caret package
```

```
##
## Attaching package: 'xgboost'
##
## The following object is masked from 'package:dplyr':
##
##   slice
```

```
library(pROC) # for calculating AUC values
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##   cov, smooth, var
```

Let us now conduct some preliminary exploration of the data.

```
head(diabetes)
```

```
##   Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI
## 1           6     148           72           35         0 33.6
## 2           1      85           66           29         0 26.6
## 3           8     183           64            0         0 23.3
## 4           1      89           66           23        94 28.1
## 5           0     137           40           35       168 43.1
## 6           5     116           74            0         0 25.6
##   DiabetesPedigreeFunction Age Outcome
## 1                0.627   50         1
## 2                0.351   31         0
## 3                0.672   32         1
## 4                0.167   21         0
## 5                2.288   33         1
## 6                0.201   30         0
```

```
dim(diabetes)
```

```
## [1] 768 9
```

```
str(diabetes)
```

```
## 'data.frame': 768 obs. of 9 variables:
## $ Pregnancies : int 6 1 8 1 0 5 3 10 2 8 ...
## $ Glucose : int 148 85 183 89 137 116 78 115 197 125 ...
## $ BloodPressure : int 72 66 64 66 40 74 50 0 70 96 ...
## $ SkinThickness : int 35 29 0 23 35 0 32 0 45 0 ...
## $ Insulin : int 0 0 0 94 168 0 88 0 543 0 ...
## $ BMI : num 33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
## $ DiabetesPedigreeFunction: num 0.627 0.351 0.672 0.167 2.288 ...
## $ Age : int 50 31 32 21 33 30 26 29 53 54 ...
## $ Outcome : int 1 0 1 0 1 0 1 0 1 1 ...
```

```
summary(diabetes)
```

```
## Pregnancies      Glucose      BloodPressure      SkinThickness
## Min.   : 0.000    Min.   : 0.0    Min.   : 0.00    Min.   : 0.00
## 1st Qu.: 1.000    1st Qu.: 99.0    1st Qu.: 62.00    1st Qu.: 0.00
## Median : 3.000    Median :117.0    Median : 72.00    Median :23.00
## Mean   : 3.845    Mean   :120.9    Mean   : 69.11    Mean   :20.54
## 3rd Qu.: 6.000    3rd Qu.:140.2    3rd Qu.: 80.00    3rd Qu.:32.00
## Max.   :17.000    Max.   :199.0    Max.   :122.00    Max.   :99.00
## Insulin          BMI          DiabetesPedigreeFunction      Age
## Min.   : 0.0    Min.   : 0.00    Min.   :0.0780    Min.   :21.00
## 1st Qu.: 0.0    1st Qu.:27.30    1st Qu.:0.2437    1st Qu.:24.00
## Median : 30.5    Median :32.00    Median :0.3725    Median :29.00
## Mean   : 79.8    Mean   :31.99    Mean   :0.4719    Mean   :33.24
## 3rd Qu.:127.2    3rd Qu.:36.60    3rd Qu.:0.6262    3rd Qu.:41.00
## Max.   :846.0    Max.   :67.10    Max.   :2.4200    Max.   :81.00
## Outcome
## Min.   :0.000
## 1st Qu.:0.000
## Median :0.000
## Mean   :0.349
## 3rd Qu.:1.000
## Max.   :1.000
```

We see that the dataset comprises 768 observations of 9 variables, including 8 predictors and an Outcome variable. BMI and DiabetesPedigreeFunction are numerics, while the rest of the variables are integers.

We can also note that certain clinical values are off - there are values of 0 for glucose, blood pressure, skin thickness, insulin, and BMI which is impossible. The possible explanation is that the 0s in these cases represent missing values. We can confirm this with some visual analyses shortly.

Let us check if there are any missing values and duplicates in the data.

```
any(is.na(diabetes)) # to check for missing values
```

```
## [1] FALSE
```

```
diabetes[duplicated(diabetes), ] # to check for duplicates
```

```
## [1] Pregnancies      Glucose      BloodPressure
## [4] SkinThickness      Insulin      BMI
## [7] DiabetesPedigreeFunction Age      Outcome
## <0 rows> (or 0-length row.names)
```

The data looks good, with no missing values or duplicates.

Let us look at the overall prevalence of diabetes in this population.

```
mean(diabetes$Outcome)
```

```
## [1] 0.3489583
```

We can see that 34.9%, or slightly over a third of this population present with diabetes. It is time to move on to some visual analyses.

Visual Analyses

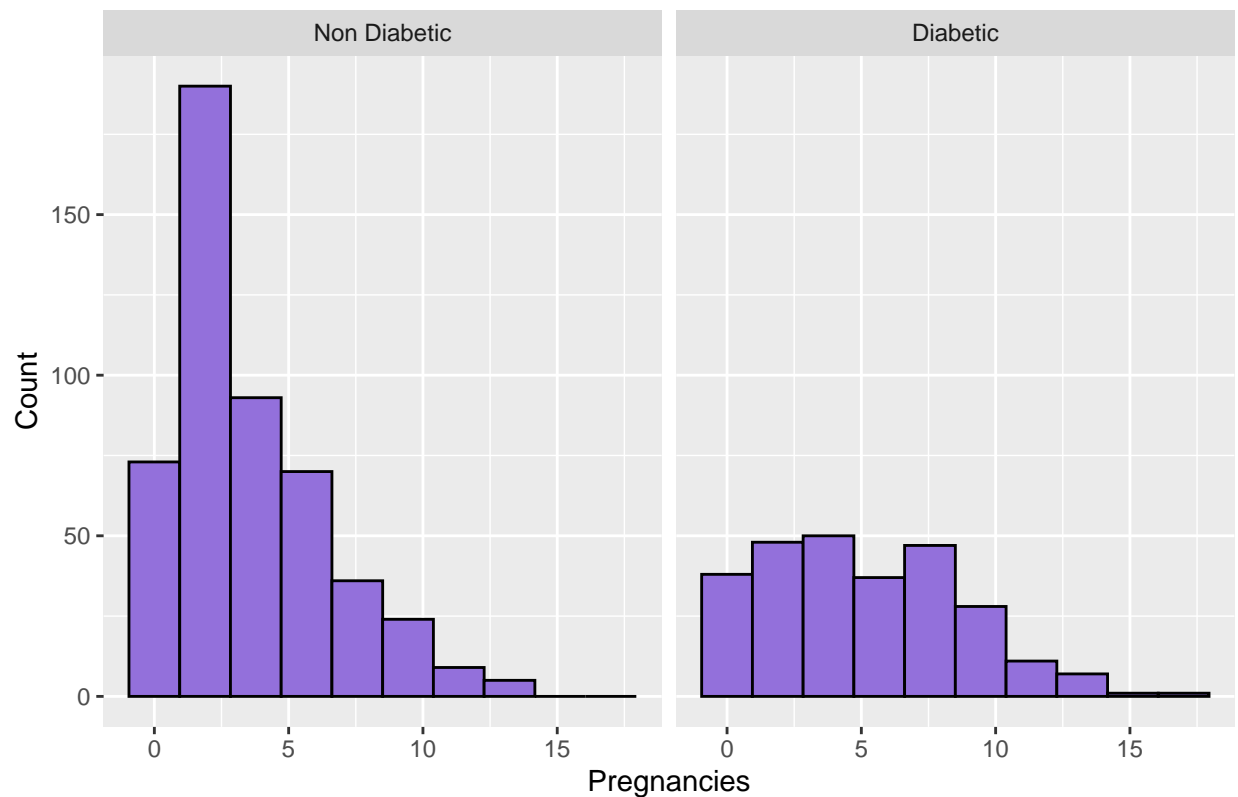
We are going to plot all the variables for a quick visual inspection. First, let's convert Outcome variable to factor with labels for plotting.

```
diabetes_plot <- diabetes %>%
  mutate(Outcome = factor(Outcome, labels = c("Non Diabetic", "Diabetic")))
```

Let us plot the number of pregnancies by diabetes status.

```
diabetes_plot %>%
  ggplot(aes(x = Pregnancies, fill = factor(Outcome))) + geom_histogram(position = "identity",
  bins = 10, colour = "black", fill = "mediumpurple") + facet_wrap(~Outcome) +
  labs(title = "Distribution of Pregnancies by Diabetes Status", x = "Pregnancies",
  y = "Count", fill = "Diabetes Status")
```

Distribution of Pregnancies by Diabetes Status

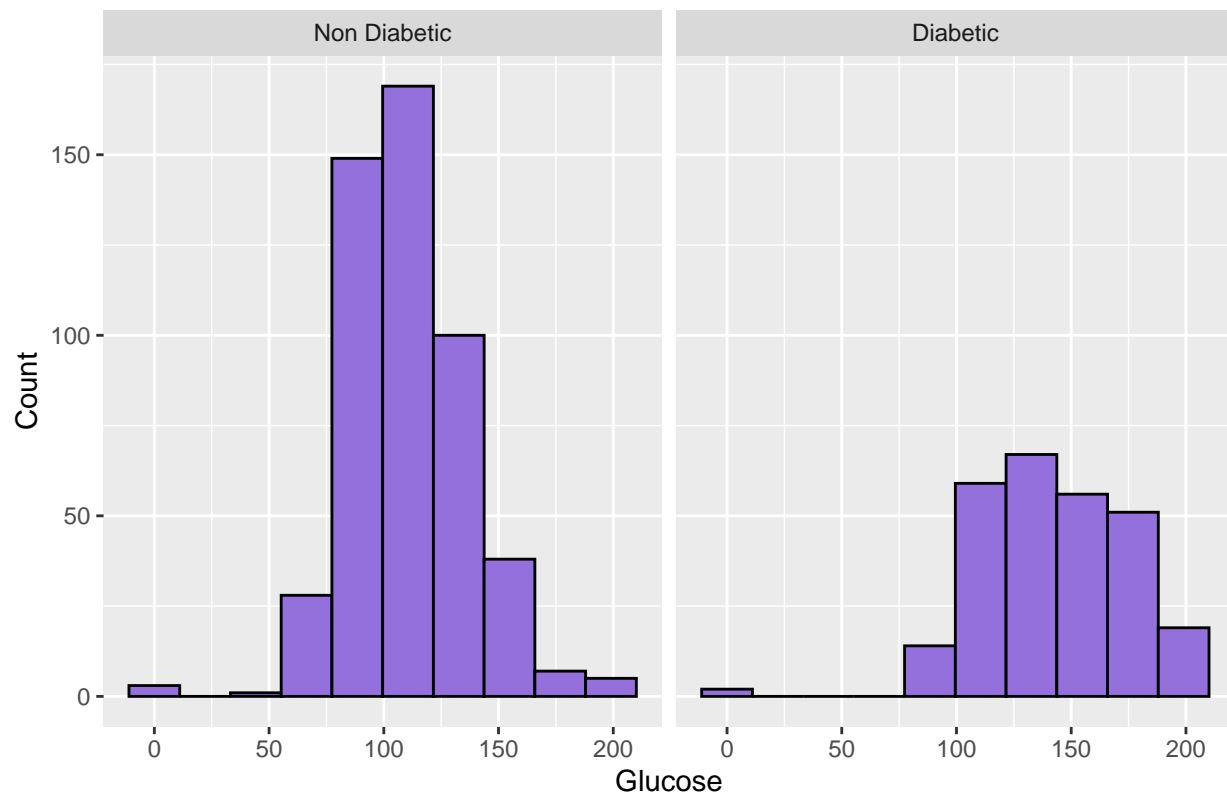


We see a non-normal distribution of values, and the number of pregnancies seems to be correlated with diabetes status.

Now, let us plot blood glucose by diabetes status.

```
diabetes_plot %>%  
  ggplot(aes(x = Glucose, fill = factor(Outcome))) + geom_histogram(position = "identity",  
    bins = 10, colour = "black", fill = "mediumpurple") + facet_wrap(~Outcome) +  
  labs(title = "Distribution of Blood Glucose by Diabetes Status", x = "Glucose",  
    y = "Count", fill = "Diabetes Status")
```

Distribution of Blood Glucose by Diabetes Status

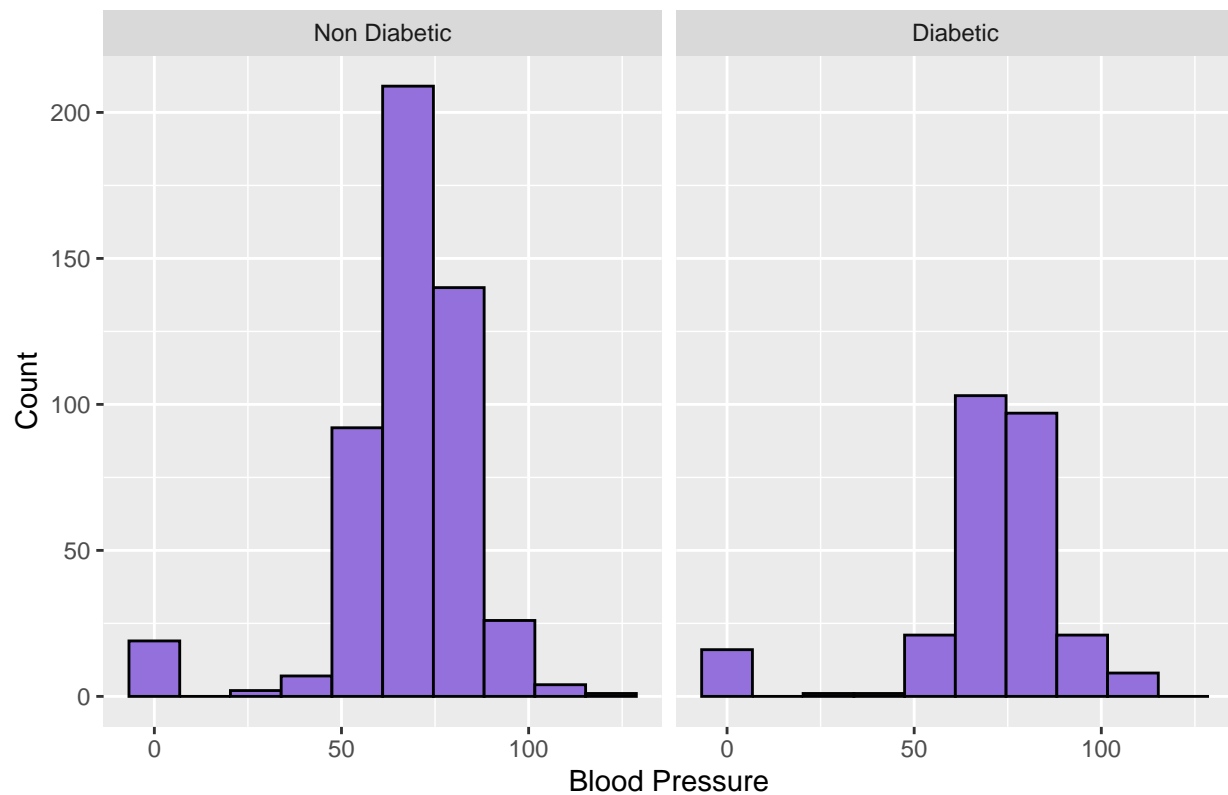


The glucose readings of individuals with diabetes seem to be skewed to the right. We also see a small sample of 0 readings, which we will need to account for later.

Now, let us plot blood pressure by diabetes status.

```
diabetes_plot %>%  
  ggplot(aes(x = BloodPressure, fill = factor(Outcome))) + geom_histogram(position = "identity",  
    bins = 10, colour = "black", fill = "mediumpurple") + facet_wrap(~Outcome) +  
  labs(title = "Distribution of Blood Pressure by Diabetes Status", x = "Blood Pressure",  
    y = "Count", fill = "Diabetes Status")
```

Distribution of Blood Pressure by Diabetes Status

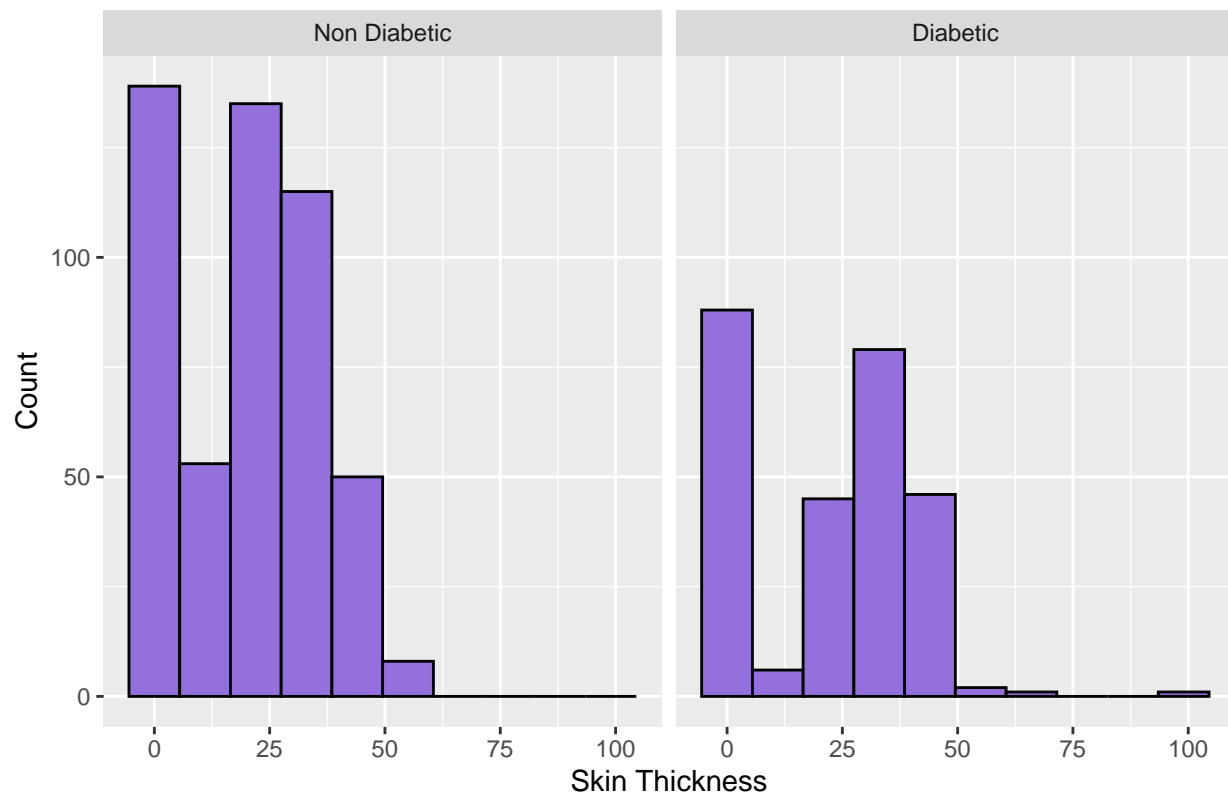


Again, the readings of individuals with diabetes seem to be skewed slightly to the right. We again see a small sample of 0 readings.

Now, let us plot skin thickness by diabetes status.

```
diabetes_plot %>%  
  ggplot(aes(x = SkinThickness, fill = factor(Outcome))) + geom_histogram(position = "identity",  
    bins = 10, colour = "black", fill = "mediumpurple") + facet_wrap(~Outcome) +  
  labs(title = "Distribution of Skin Thickness by Diabetes Status", x = "Skin Thickness",  
    y = "Count", fill = "Diabetes Status")
```


Distribution of Skin Thickness by Diabetes Status

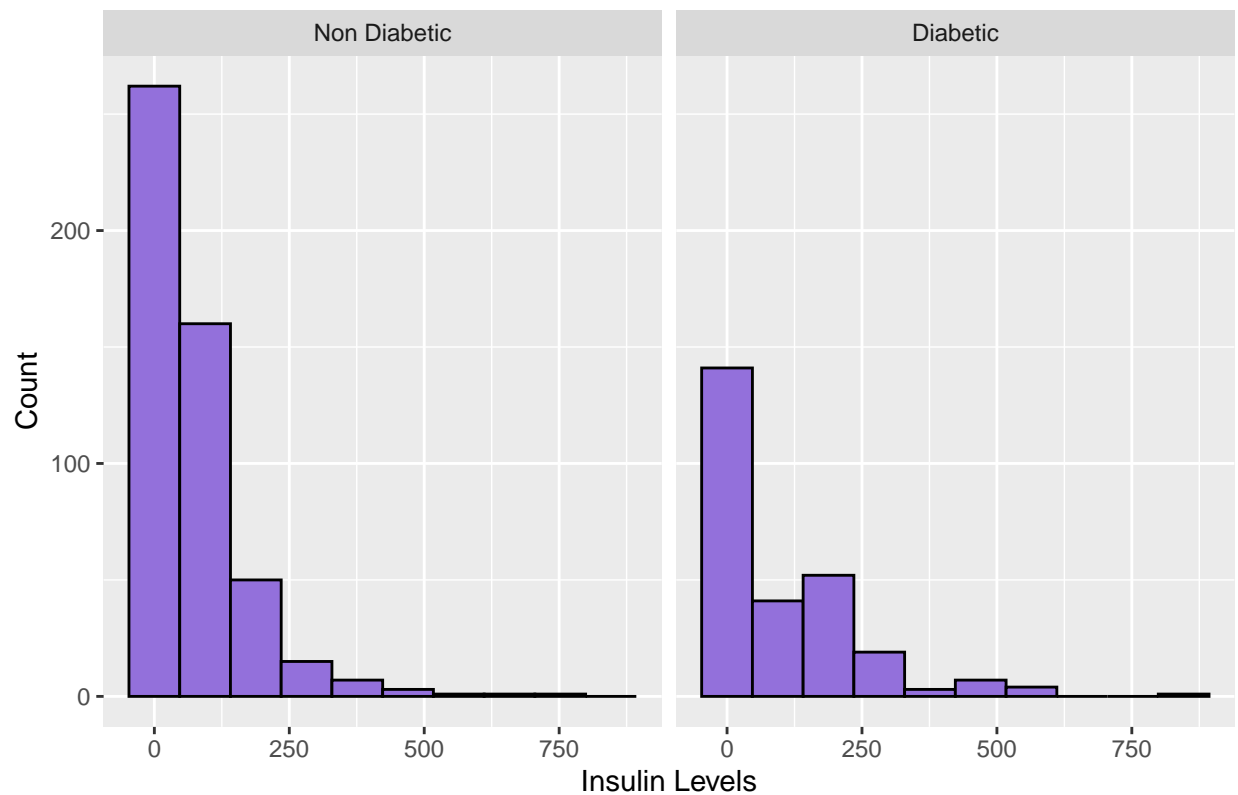


We see a large sample of 0 readings, which potentially affects the utility of skin thickness as a predictor of diabetes status.

Now, let us plot insulin levels by diabetes status.

```
diabetes_plot %>%
  ggplot(aes(x = Insulin, fill = factor(Outcome))) + geom_histogram(position = "identity",
    bins = 10, colour = "black", fill = "mediumpurple") + facet_wrap(~Outcome) +
  labs(title = "Distribution of Insulin Levels by Diabetes Status", x = "Insulin Levels",
    y = "Count", fill = "Diabetes Status")
```

Distribution of Insulin Levels by Diabetes Status

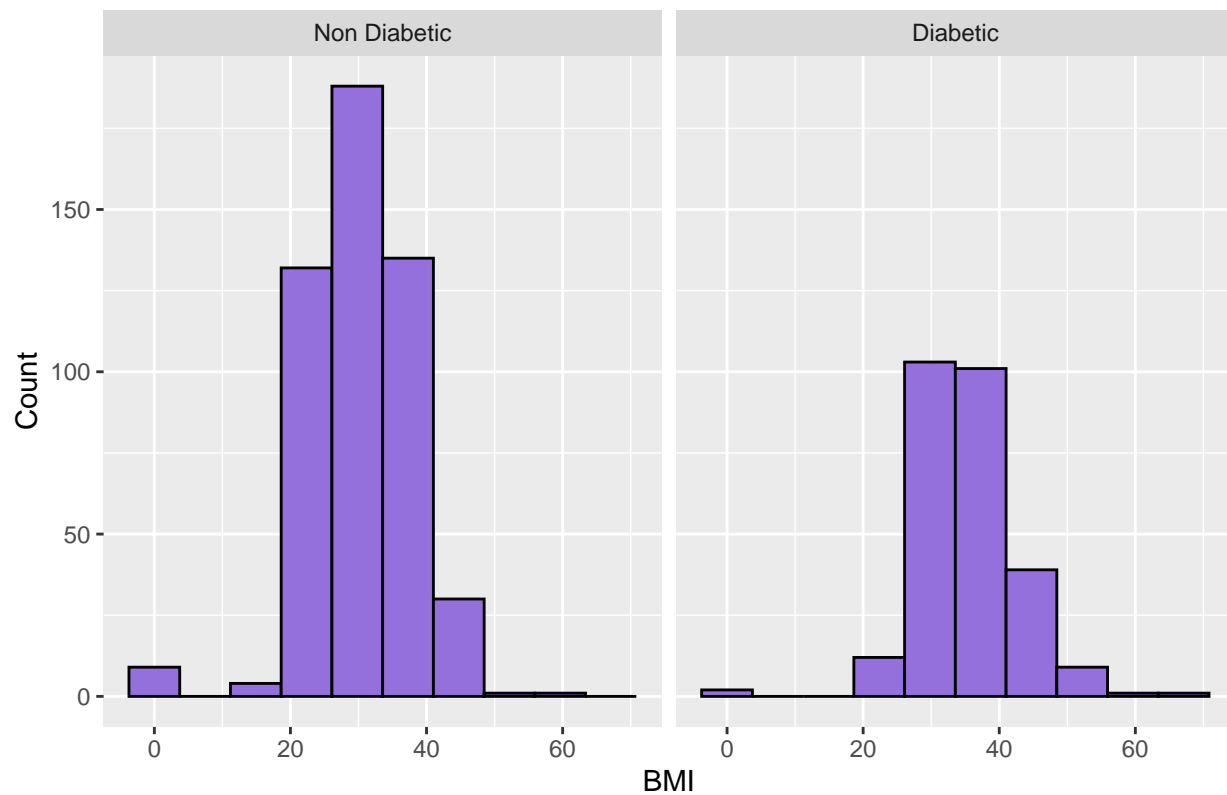


Similar to skin thickness, we see a large sample of 0 readings. The readings also appear to be non-normally distributed.

Now, let us plot BMI by diabetes status.

```
diabetes_plot %>%  
  ggplot(aes(x = BMI, fill = factor(Outcome))) + geom_histogram(position = "identity",  
    bins = 10, colour = "black", fill = "mediumpurple") + facet_wrap(~Outcome) +  
  labs(title = "Distribution of BMI by Diabetes Status", x = "BMI", y = "Count",  
    fill = "Diabetes Status")
```

Distribution of BMI by Diabetes Status

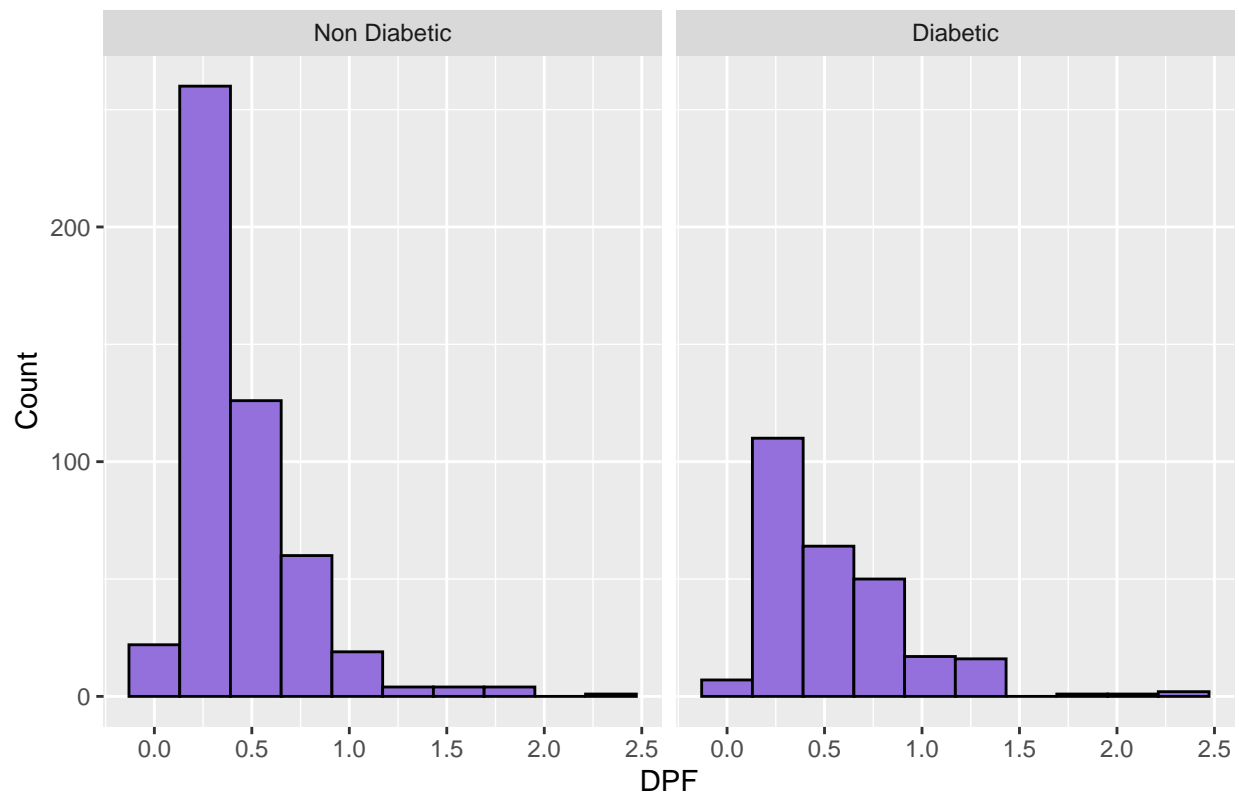


The readings of individuals with diabetes seem to be skewed to the right, but both graphs appear fairly normal, with the exception of a small sample of 0 readings.

Now, let us plot DPF by diabetes status.

```
diabetes_plot %>%
  ggplot(aes(x = DiabetesPedigreeFunction, fill = factor(Outcome))) + geom_histogram(position = "identity",
  bins = 10, colour = "black", fill = "mediumpurple") + facet_wrap(~Outcome) +
  labs(title = "Distribution of DPF by Diabetes Status", x = "DPF", y = "Count",
  fill = "Diabetes Status")
```

Distribution of DPF by Diabetes Status

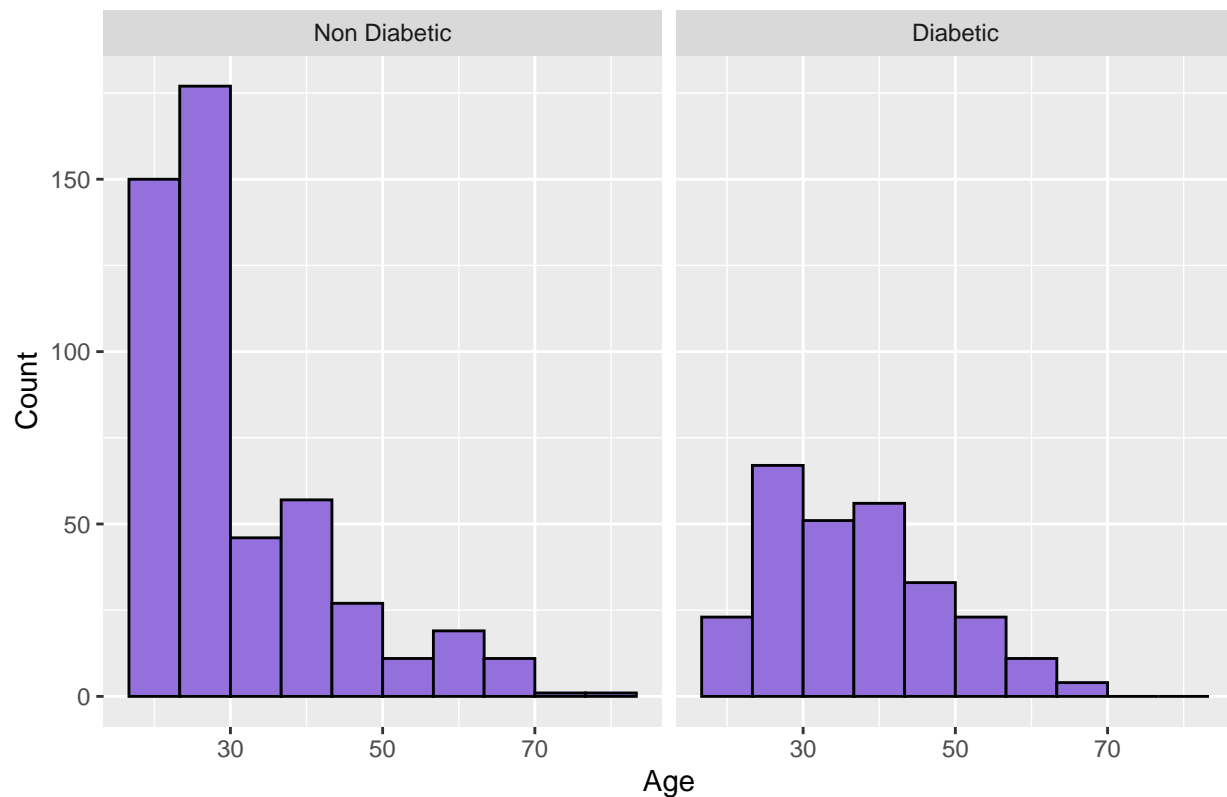


The readings of individuals with diabetes seem to be skewed slightly to the right.

Now, let us plot age by diabetes status.

```
diabetes_plot %>%
  ggplot(aes(x = Age, fill = factor(Outcome))) + geom_histogram(position = "identity",
    bins = 10, colour = "black", fill = "mediumpurple") + facet_wrap(~Outcome) +
  labs(title = "Distribution of Age by Diabetes Status", x = "Age", y = "Count",
    fill = "Diabetes Status")
```

Distribution of Age by Diabetes Status



It looks like being under 30 puts one at a lower risk of diabetes.

Finally, let us check for correlations between predictors. Remember, we need to first deal with the missing values (coded as 0) for some of the clinical data. Two common ways are to replace the missing values with the mean and the median. Given that some of our variables are not normally distributed, it is more reasonable to use the median. While there are 0 values for pregnancies, we cannot tell which are true 0s, and which are missing values. Therefore, we will leave the pregnancies variable untouched.

```
diabetes$BloodPressure[diabetes$BloodPressure == 0] <- median(diabetes$BloodPressure[diabetes$BloodPressure != 0])
diabetes$Insulin[diabetes$Insulin == 0] <- median(diabetes$Insulin[diabetes$Insulin != 0])
diabetes$Glucose[diabetes$Glucose == 0] <- median(diabetes$Glucose[diabetes$Glucose != 0])
diabetes$BMI[diabetes$BMI == 0] <- median(diabetes$BMI[diabetes$BMI != 0])
diabetes$SkinThickness[diabetes$SkinThickness == 0] <- median(diabetes$SkinThickness[diabetes$SkinThickness != 0])

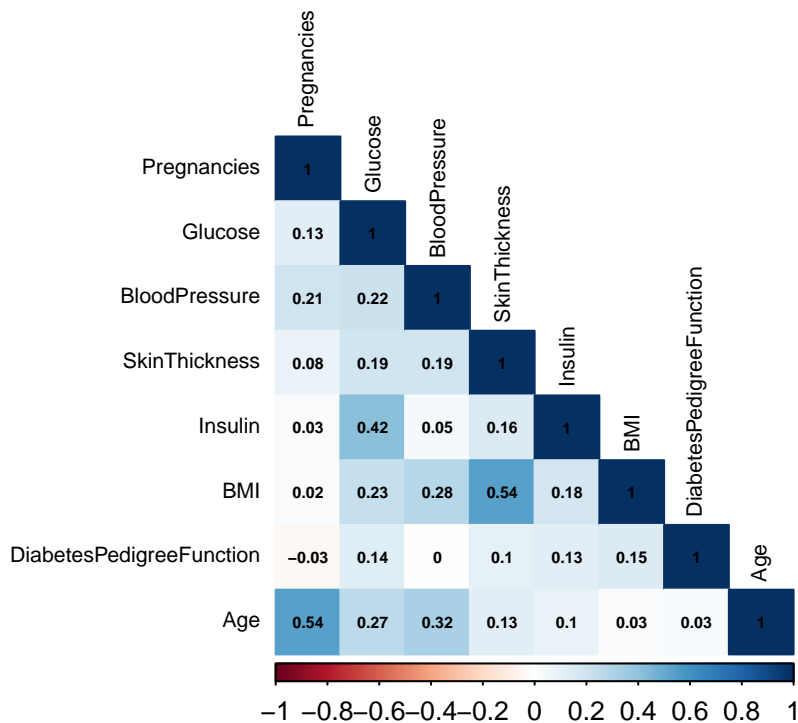
summary(diabetes) # To confirm that we have successfully imputed the missing values with median.
```

	Pregnancies	Glucose	BloodPressure	SkinThickness
## Min. :	0.000	Min. : 44.00	Min. : 24.00	Min. : 7.00
## 1st Qu. :	1.000	1st Qu.: 99.75	1st Qu.: 64.00	1st Qu.:25.00
## Median :	3.000	Median :117.00	Median : 72.00	Median :29.00
## Mean :	3.845	Mean :121.66	Mean : 72.39	Mean :29.11
## 3rd Qu. :	6.000	3rd Qu.:140.25	3rd Qu.: 80.00	3rd Qu.:32.00

```
## Max. :17.000 Max. :199.00 Max. :122.00 Max. :99.00
## Insulin BMI DiabetesPedigreeFunction Age
## Min. : 14.0 Min. :18.20 Min. :0.0780 Min. :21.00
## 1st Qu.:121.5 1st Qu.:27.50 1st Qu.:0.2437 1st Qu.:24.00
## Median :125.0 Median :32.30 Median :0.3725 Median :29.00
## Mean :140.7 Mean :32.46 Mean :0.4719 Mean :33.24
## 3rd Qu.:127.2 3rd Qu.:36.60 3rd Qu.:0.6262 3rd Qu.:41.00
## Max. :846.0 Max. :67.10 Max. :2.4200 Max. :81.00
## Outcome
## Min. :0.000
## 1st Qu.:0.000
## Median :0.000
## Mean :0.349
## 3rd Qu.:1.000
## Max. :1.000
```

We can now check for correlations between predictors.

```
diabetes_correlation <- diabetes[-9] # Remove Outcome
diabetes_correlation <- cor(diabetes_correlation)
corrplot(diabetes_correlation, method = "color", type = "lower", addCoef.col = "black",
  number.cex = 0.5, tl.cex = 0.7, tl.col = "black")
```



We can observe that BMI and skin thickness are strongly correlated, as are age and number of pregnancies. We also observe that insulin levels and blood glucose levels are moderately correlated, as are age and blood pressure. There are also several weaker correlations.

Constructing and Evaluating Models

It is time to construct our models. We will split our data into a validation and a final test set, and further split the validation set into a training and test set. We will then train our models using the caret package, and evaluate their accuracy and AUC scores. Finally, we will construct and evaluate ensemble models through a voting approach. We will then evaluate our chosen model's performance on the final test set.

Let's create our training and test sets. But first, let's code the outcome as a factor before splitting the data to allow us to build our model.

```
diabetes$Outcome <- factor(diabetes$Outcome)
```

We create a final test set using 20% of the diabetes dataset. Then, we will create training (80%) and test (20%) sets from the remaining data.

```
set.seed(123, sample.kind = "Rounding") # if using R 3.6 or later
test_index <- createDataPartition(y = diabetes$Outcome, times = 1, p = 0.2, list = FALSE)
validation_index <- diabetes[-test_index, ]
final_test_set <- diabetes[test_index, ]

validation_index <- createDataPartition(y = diabetes$Outcome, p = 0.2, list = FALSE)
train_set <- diabetes[-validation_index, ]
test_set <- diabetes[validation_index, ]

rm(test_index, validation_index) # Removing the unneeded dataset
```

For all models, we will apply cross validation and tune parameters for optimisation where applicable. Let us construct and evaluate our first model: GLM.

```
train_glm <- train(Outcome ~ .,
  method = "glm",
  trControl = trainControl(method = "cv", number = 10, p = .9), # To apply cross validation
  data = train_set)

summary(train_glm) # This is for visualising the importance of each predictor in the model
```

```
##
## Call:
## NULL
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -8.946858   0.890268 -10.050 < 2e-16 ***
## Pregnancies    0.104972   0.034632   3.031  0.00244 **
## Glucose        0.036503   0.004372   8.350 < 2e-16 ***
## BloodPressure -0.010950   0.009372  -1.168  0.24264
## SkinThickness  0.006898   0.014587   0.473  0.63628
## Insulin       -0.002013   0.001332  -1.512  0.13053
## BMI            0.095384   0.019545   4.880 1.06e-06 ***
## DiabetesPedigreeFunction 0.924466   0.332968   2.776  0.00550 **
## Age           0.018464   0.010481   1.762  0.07813 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 793.94 on 613 degrees of freedom
## Residual deviance: 580.39 on 605 degrees of freedom
## AIC: 598.39
##
## Number of Fisher Scoring iterations: 5
```

```
predict_glm <- predict(train_glm, newdata = test_set, type = "raw")

results <- tibble(method = "GLM",
                  accuracy = confusionMatrix(predict_glm, test_set$Outcome)$overall[["Accuracy"]],
                  AUC = as.numeric(roc(test_set$Outcome, as.numeric(predict_glm))$auc) # To evaluate
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

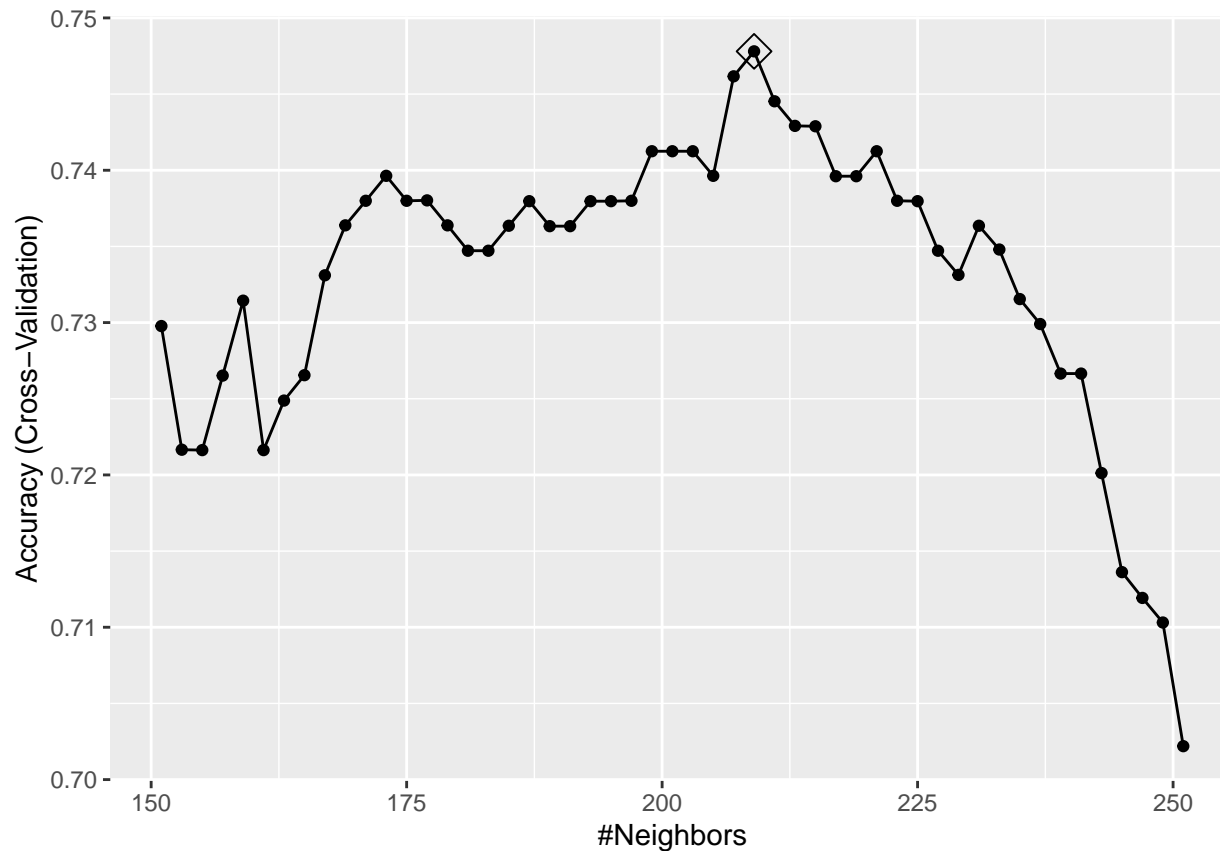
```
results
```

```
## # A tibble: 1 x 3
##   method accuracy  AUC
##   <chr>      <dbl> <dbl>
## 1 GLM        0.818 0.783
```

We see that our GLM model has an accuracy of 0.818, and AUC of 0.783. Not too bad! Can we do better?
Let us move on to our next model: KNN.

```
set.seed(123)
train_knn <- train(Outcome ~ ., method = "knn", data = train_set, tuneGrid = data.frame(k = seq(151,
  251, 2)), trControl = trainControl(method = "cv", number = 10, p = 0.9))

ggplot(train_knn, highlight = TRUE) # to visualise the relationship between k and accuracy
```

```
varImp(train_knn)
```

```
## ROC curve variable importance
##
##               Importance
## Glucose          100.000
## BMI              52.784
## Age              47.516
## Insulin          29.094
## SkinThickness    20.787
## Pregnancies       7.368
## BloodPressure     1.221
## DiabetesPedigreeFunction 0.000
```

```
train_knn$bestTune # to find the best k parameter
```

```
##      k
## 30 209
```

```
train_knn$finalModel
```

```
## 209-nearest neighbor model
## Training set outcome distribution:
##
```

```
##    0    1
## 400 214
```

```
predict_knn <- predict(train_knn, newdata = test_set, type = "raw")

results <- bind_rows(results, tibble(method = "KNN", accuracy = confusionMatrix(predict_knn,
  test_set$Outcome)$overall[["Accuracy"]], AUC = as.numeric(roc(test_set$Outcome,
  as.numeric(predict_knn))$auc)))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

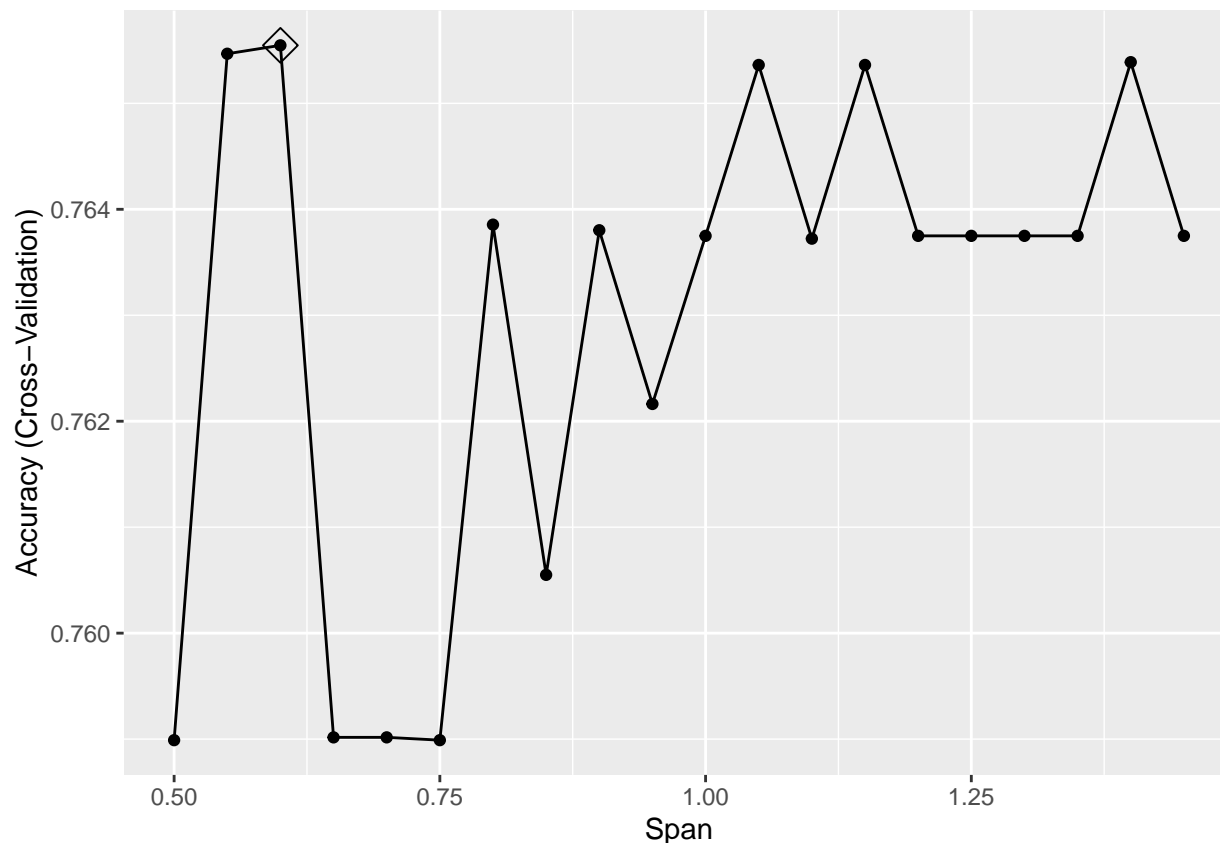
```
results
```

```
## # A tibble: 2 x 3
##   method accuracy  AUC
##   <chr>      <dbl> <dbl>
## 1 GLM        0.818 0.783
## 2 KNN        0.812 0.770
```

Our KNN model has an accuracy of 0.812, and AUC of 0.770, performing slightly worse than our GLM model. For our next model: LOESS

```
train_loess <- train(Outcome ~ ., method = "gamLoess", tuneGrid = expand.grid(span = seq(0.5,
  1.45, len = 20), degree = 1), trControl = trainControl(method = "cv", number = 10,
  p = 0.9), data = train_set)

ggplot(train_loess, highlight = TRUE)
```



```
predict_loess <- predict(train_loess, newdata = test_set, type = "raw")

results <- bind_rows(results, tibble(method = "LOESS", accuracy = confusionMatrix(predict_loess,
  test_set$Outcome)$overall[["Accuracy"]], AUC = as.numeric(roc(test_set$Outcome,
  as.numeric(predict_loess))$auc)))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

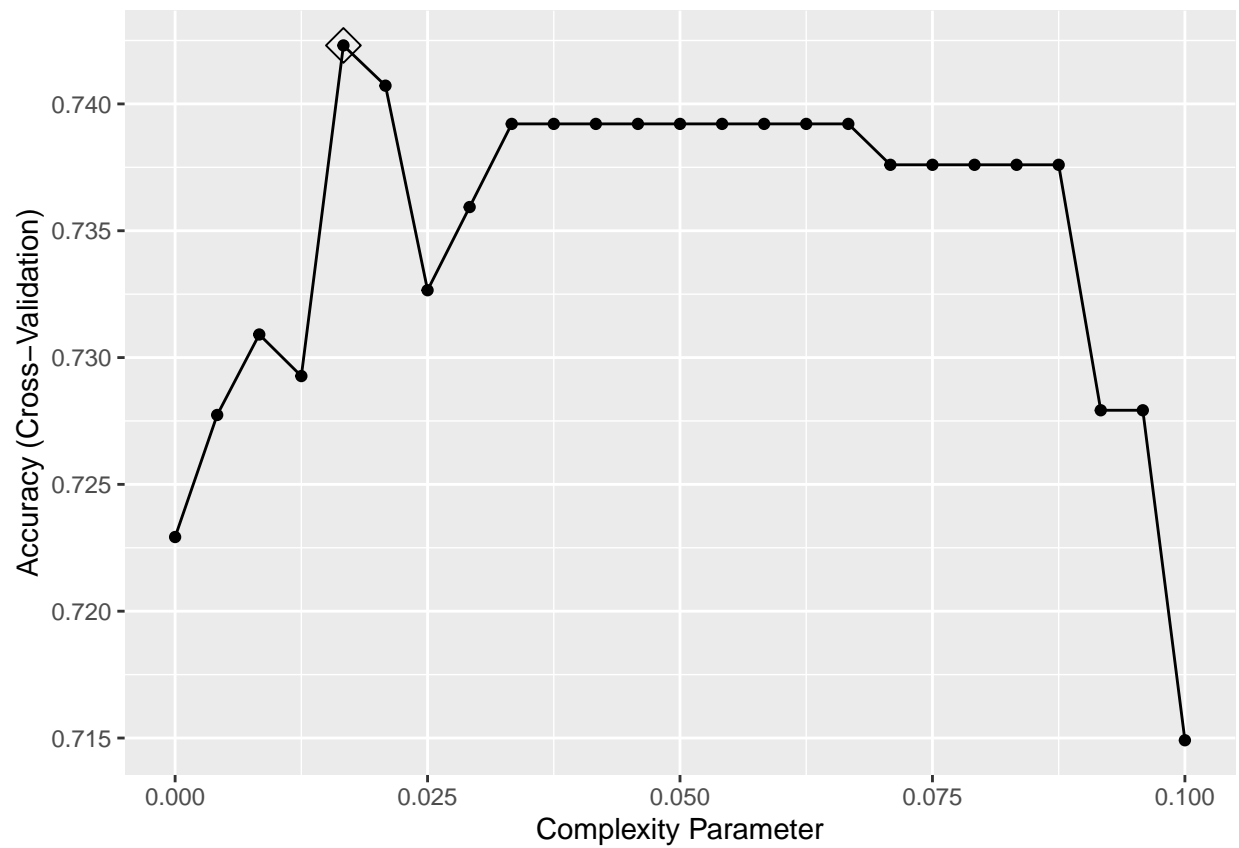
```
results
```

```
## # A tibble: 3 x 3
##   method accuracy  AUC
##   <chr>      <dbl> <dbl>
## 1 GLM        0.818 0.783
## 2 KNN        0.812 0.770
## 3 LOESS      0.786 0.763
```

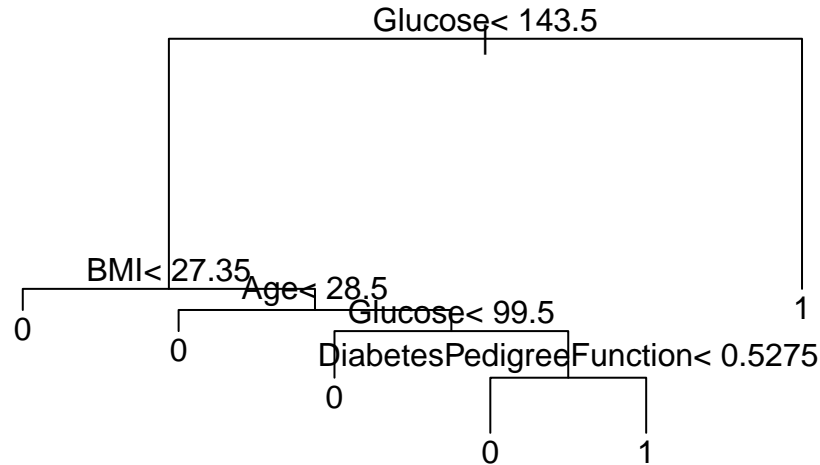
Our LOESS model has an accuracy of 0.786, and AUC of 0.763. On to our next model: rpart

```
train_rpart <- train(Outcome ~ ., method = "rpart", tuneGrid = data.frame(cp = seq(0,
  0.1, len = 25)), trControl = trainControl(method = "cv", number = 10, p = 0.9),
  data = train_set)
```

```
# display the decision tree  
ggplot(train_rpart, highlight = TRUE)
```



```
# view the final decision tree  
plot(train_rpart$finalModel, margin = 0.1) # to plot tree structure  
text(train_rpart$finalModel) # add text labels
```



```

predict_rpart <- predict(train_rpart, newdata = test_set, type = "raw")

results <- bind_rows(results, tibble(method = "Decision Tree", accuracy = confusionMatrix(predict_rpart,
  test_set$Outcome)$overall[["Accuracy"]], AUC = as.numeric(roc(test_set$Outcome,
    as.numeric(predict_rpart))$auc)))

```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
results
```

```

## # A tibble: 4 x 3
##   method      accuracy  AUC
##   <chr>         <dbl> <dbl>
## 1 GLM           0.818 0.783
## 2 KNN           0.812 0.770
## 3 LOESS         0.786 0.763
## 4 Decision Tree 0.779 0.745

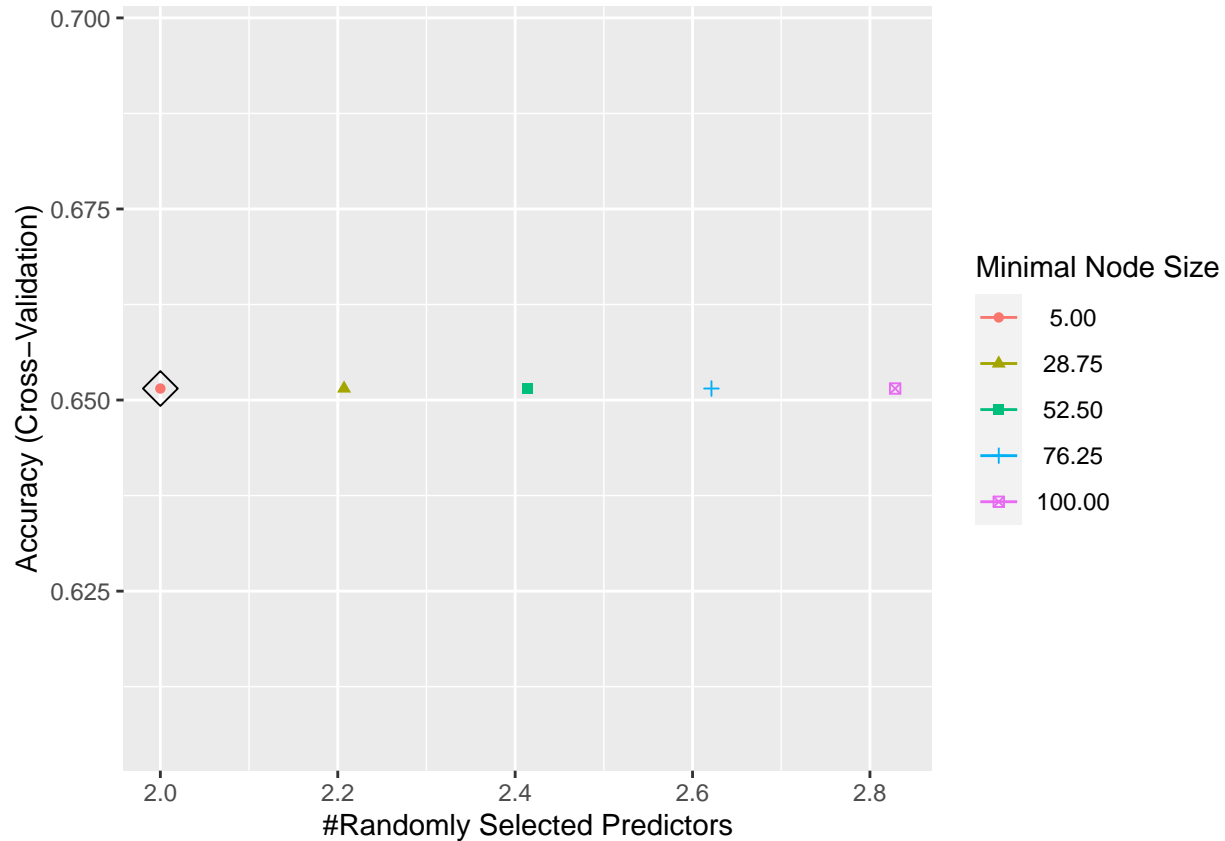
```

Our rpart model has an accuracy of 0.779, and AUC of 0.745. Let's look at the next model: RF

```
train_rf <- train(Outcome ~ ., method = "Rborist", tuneGrid = data.frame(predFixed = seq(2,
  sqrt(ncol(train_set) - 1), length = 5), minNode = seq(5, 100, length = 5)), trControl = trainControl(
  number = 10, p = , 9), data = train_set)

ggplot(train_rf, highlight = TRUE)
```

```
## 'geom_line()': Each group consists of only one observation.
## i Do you need to adjust the group aesthetic?
```



```
predict_rf <- predict(train_rf, newdata = test_set, type = "raw")

results <- bind_rows(results, tibble(method = "Random Forest", accuracy = confusionMatrix(predict_rf,
  test_set$Outcome)$overall[["Accuracy"]], AUC = as.numeric(roc(test_set$Outcome,
  as.numeric(predict_rf))$auc)))
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
results
```

```
## # A tibble: 5 x 3
##   method      accuracy  AUC
##   <chr>          <dbl> <dbl>
```

```
## 1 GLM          0.818 0.783
## 2 KNN          0.812 0.770
## 3 LOESS        0.786 0.763
## 4 Decision Tree 0.779 0.745
## 5 Random Forest 0.649 0.5
```

Our rf model has an accuracy of 0.649, and AUC of 0.5, making it the worst performer so far, particularly in AUC. For our next model: XGB

```
train_xgb <- train(Outcome ~ .,
  method = "xgbTree",
  tuneGrid = expand.grid( # Creating a tuning parameter
    nrounds = seq(25, 250, length = 10),
    max_depth = c(2, 4, 6, 8),
    eta = c(0.05, 0.1, 0.2),
    gamma = 0,
    colsample_bytree = 1,
    min_child_weight = 1,
    subsample = 1),
  trControl = trainControl(method = "cv", number = 10, p = .9),
  objective = "binary:logistic",
  data = train_set) # for binary classification
```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

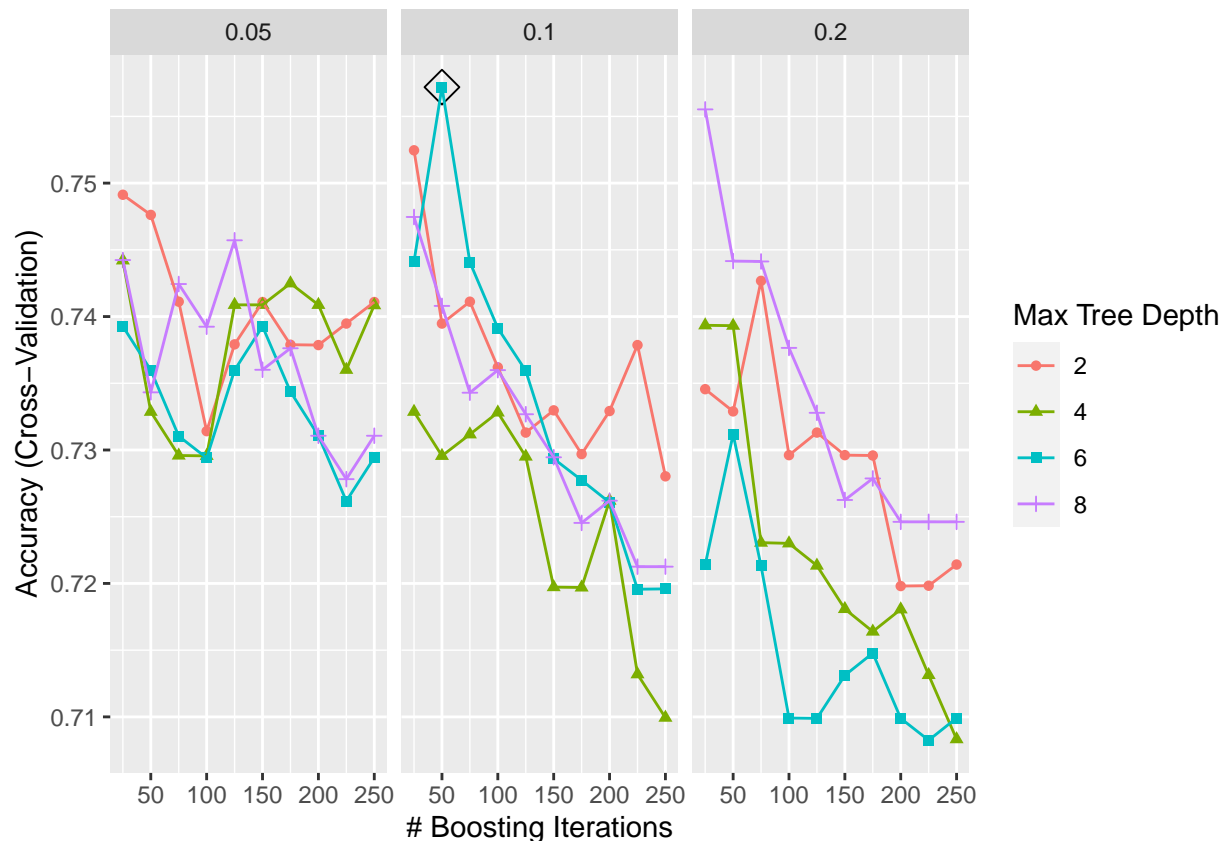
[illegible]


```

## [17:25:46] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:46] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:46] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:46] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:46] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:46] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:47] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:47] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:47] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:47] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:47] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:47] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:47] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:47] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:47] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:48] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:48] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:48] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:48] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:48] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:48] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:48] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:48] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:48] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [17:25:48] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead

```

```
ggplot(train_xgb, highlight = TRUE)
```



```

predict_xgb <- predict(train_xgb, newdata = test_set, type = "raw")

results <- bind_rows(results,
  tibble(method = "Extreme Gradient Boosting",
    accuracy = confusionMatrix(predict_xgb, test_set$Outcome)$overall[["Accuracy"]],
    AUC = as.numeric(roc(test_set$Outcome, as.numeric(predict_xgb))$auc))
)

```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
results
```

```
## # A tibble: 6 x 3
##   method          accuracy  AUC
##   <chr>          <dbl> <dbl>
## 1 GLM            0.818 0.783
## 2 KNN            0.812 0.770
## 3 LOESS          0.786 0.763
## 4 Decision Tree  0.779 0.745
## 5 Random Forest  0.649 0.5
## 6 Extreme Gradient Boosting 0.792 0.768
```

Our XGB model has an accuracy of 0.792, and AUC of 0.768. For our final standalone model: NB.

```

train_nb <- train(Outcome ~ ., method = "nb", trControl = trainControl(method = "cv",
  number = 10, p = 0.9), data = train_set)

predict_nb <- predict(train_nb, newdata = test_set, type = "raw")

results <- bind_rows(results, tibble(method = "Naive Bayes", accuracy = confusionMatrix(predict_nb,
  test_set$Outcome)$overall[["Accuracy"]], AUC = as.numeric(roc(test_set$Outcome,
  as.numeric(predict_nb))$auc)))

```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
results
```

```
## # A tibble: 7 x 3
##   method          accuracy  AUC
##   <chr>          <dbl> <dbl>
## 1 GLM            0.818 0.783
## 2 KNN            0.812 0.770
## 3 LOESS          0.786 0.763
## 4 Decision Tree  0.779 0.745
## 5 Random Forest  0.649 0.5
## 6 Extreme Gradient Boosting 0.792 0.768
## 7 Naive Bayes    0.766 0.739
```

Our NB model has an accuracy of 0.766, and AUC of 0.739.

As it stands, our GLM model has the best performance, followed by KNN, XGB, and LOESS, in that order. Let us create 3 ensemble models by majority voting, using the top 4, best 3, and best 2 models.

Let us start by creating a Top 4 ensemble model.

```
# We first combine the predictions of the top 4 models into a data frame.
top4_votes <- data.frame(predict_glm, predict_knn, predict_loess, predict_xgb)
top4_votes <- sapply(top4_votes, as.numeric) # Converting to numeric values for analysis in next section

# Apply majority voting to get final ensemble prediction
predict_top4_ensemble <- ifelse(rowMeans(top4_votes) > 1, "1", "0") # Converting factor into numeric values

results <- bind_rows(results, tibble(method = "GLM + KNN + XGB + LOESS Ensemble",
  accuracy = mean(predict_top4_ensemble == test_set$Outcome), AUC = as.numeric(roc(test_set$Outcome,
    as.numeric(predict_top4_ensemble))$auc)))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
results
```

```
## # A tibble: 8 x 3
##   method          accuracy    AUC
##   <chr>          <dbl> <dbl>
## 1 GLM            0.818 0.783
## 2 KNN            0.812 0.770
## 3 LOESS          0.786 0.763
## 4 Decision Tree  0.779 0.745
## 5 Random Forest  0.649 0.5
## 6 Extreme Gradient Boosting 0.792 0.768
## 7 Naive Bayes    0.766 0.739
## 8 GLM + KNN + XGB + LOESS Ensemble 0.779 0.783
```

Our Top 4 ensemble model has an accuracy of 0.779, and AUC of 0.783. Let's apply the same method to create a Top 3 ensemble model.

```
# Combine predictions into a data frame
top3_votes <- data.frame(predict_glm, predict_knn, predict_xgb)
top3_votes <- sapply(top3_votes, as.numeric) # Converting to numeric values for analysis in next section

# Apply majority voting to get final ensemble prediction
predict_top3_ensemble <- ifelse(rowMeans(top3_votes) > 1, "1", "0")

results <- bind_rows(results, tibble(method = "GLM + KNN + XGB Ensemble", accuracy = mean(predict_top3_ensemble == test_set$Outcome), AUC = as.numeric(roc(test_set$Outcome, as.numeric(predict_top3_ensemble))$auc)))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
results
```

```
## # A tibble: 9 x 3
##   method          accuracy  AUC
##   <chr>          <dbl> <dbl>
## 1 GLM            0.818 0.783
## 2 KNN            0.812 0.770
## 3 LOESS          0.786 0.763
## 4 Decision Tree  0.779 0.745
## 5 Random Forest  0.649 0.5
## 6 Extreme Gradient Boosting 0.792 0.768
## 7 Naive Bayes    0.766 0.739
## 8 GLM + KNN + XGB + LOESS Ensemble 0.779 0.783
## 9 GLM + KNN + XGB Ensemble    0.792 0.785
```

Our Top 3 ensemble model has an accuracy of 0.792, and AUC of 0.785. Let us create our Top 2 ensemble model.

```
# Combine predictions into a data frame
top2_votes <- data.frame(predict_glm, predict_knn)
top2_votes <- sapply(top2_votes, as.numeric) # Converting to numeric values for analysis in next section

# Apply majority voting to get final ensemble prediction
predict_top2_ensemble <- ifelse(rowMeans(top2_votes) > 1, "1", "0") # Converting to numeric in the previous section

results <- bind_rows(results, tibble(method = "Top 2 Ensemble", accuracy = mean(predict_top2_ensemble ==
  test_set$Outcome), AUC = as.numeric(roc(test_set$Outcome, as.numeric(predict_top2_ensemble))$auc)))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
results
```

```
## # A tibble: 10 x 3
##   method          accuracy  AUC
##   <chr>          <dbl> <dbl>
## 1 GLM            0.818 0.783
## 2 KNN            0.812 0.770
## 3 LOESS          0.786 0.763
## 4 Decision Tree  0.779 0.745
## 5 Random Forest  0.649 0.5
## 6 Extreme Gradient Boosting 0.792 0.768
## 7 Naive Bayes    0.766 0.739
## 8 GLM + KNN + XGB + LOESS Ensemble 0.779 0.783
## 9 GLM + KNN + XGB Ensemble    0.792 0.785
## 10 Top 2 Ensemble    0.818 0.792
```

Our Top 2 ensemble model has an accuracy of 0.818, and AUC of 0.792.

Final Evaluation

We have constructed and evaluated several standalone and ensemble models. The three best performing models, GLM, KNN, and GLM + KNN ensemble are pretty evenly matched, with KNN being slightly inferior. Let us now evaluate all 3 models on our final test set.

```
predict_glm_final <- predict(train_glm, newdata = final_test_set, type = "raw")
predict_knn_final <- predict(train_knn, newdata = final_test_set, type = "raw")

final_votes <- data.frame(predict_glm_final, predict_knn_final)
final_votes <- sapply(final_votes, as.numeric)

predict_final_ensemble <- ifelse(rowMeans(final_votes) > 1, "1", "0")

final_results <- tibble(method = c("GLM", "KNN", "Top 2 Ensemble"), accuracy = c(confusionMatrix(predict_glm_final,
  final_test_set$Outcome)$overall[["Accuracy"]], confusionMatrix(predict_knn_final,
  final_test_set$Outcome)$overall[["Accuracy"]], mean(predict_final_ensemble ==
  final_test_set$Outcome)), AUC = c(as.numeric(roc(final_test_set$Outcome, as.numeric(predict_glm_final))$auc),
  as.numeric(roc(final_test_set$Outcome, as.numeric(predict_knn_final))$auc), as.numeric(roc(final_test_set$Outcome,
  as.numeric(predict_final_ensemble))$auc)))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
final_results
```

```
## # A tibble: 3 x 3
##   method      accuracy  AUC
##   <chr>          <dbl> <dbl>
## 1 GLM            0.805 0.773
## 2 KNN            0.773 0.727
## 3 Top 2 Ensemble 0.786 0.767
```

We see that the GLM model is the most robust, with an accuracy of 0.805, and AUC of 0.773.

Conclusion

In this project, we attempted to create a diabetes risk prediction model that effectively predicts diabetes status in a population of females of Pima Indian heritage, aged 21 years and above, using 8 medical predictors.

We evaluated several standalone models and ensemble models on accuracy and AUC scores, and found the humble GLM model to be most effective in predicting diabetes status with accuracy and AUC scores of 0.805, and 0.773 respectively. For this project, we worked with a small dataset, with relatively few number of predictors. Nonetheless, the same basic principles can apply with more complex datasets. Further optimisations can also be obtained with other techniques, such as feature engineering and oversampling.

I hope you found this little exercise useful.

References

1. World Health Organisation (n.d.). *Diabetes*. https://www.who.int/health-topics/diabetes#tab=tab_1
2. International Diabetes Atlas (n.d.). *Diabetes around the world in 2021*. <https://diabetesatlas.org/>