Atividade Avaliativa

Big Data Anallytics

Ana Maria Alves da Silva

2024-12-17

1. Definir do Problema

O objetivo principal do problema é prever a ocorrência de diabetes (coluna Outcome) utilizando um modelo baseado em Árvore de Decisão.

2. Análise Exploratória de Dados (EDA)

Realize a Análise Exploratória de Dados (EDA). Apresente os seguintes itens:

Solução: Antes de apresentar a análise exploratória de dador, vamos carregar na memória o conjunto de dados em formato dataframe que trabalharemos para que possamos realizar as manipulações e análises desejadas. Para isso, usaremos o código abaixo:

```
df <- read.csv("/Users/anamaria/especializacao/modulo_8/atividade/diabetes.csv")
```

(i) Número de linhas e colunas do dataset;

Solução: Para vermos o tamanho do conjunto de dados, isso é, quantidade de linhas e colunas, basta usarmos a função dim.

```
print(dim(df))
```

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

Nessa caso, nosso dataframe possui 768 linhas e 9 colunas.

(ii) Tipos de variáveis do dataset;

```

str(df)
```

Solução:

```
## '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 ...
```

O dataframe possui 7 colunas do tipo inteiro e 2 colunas do tipo float (reais e inteiros).

(iii) Presença e número de valores ausentes;

Solução: Para verificar se há valores ausentes, podemos usar a função colSums e is.na no dataframe, conforme o código abaixo.

```
print(colSums(is.na(df)))
```

```
##              Pregnancies                  Glucose            BloodPressure
##                        0                        0                        0
##            SkinThickness                  Insulin                      BMI
##                        0                        0                        0
## DiabetesPedigreeFunction                      Age                  Outcome
##                        0                        0                        0
```

Ou ainda:

```
print(colnames(df)[colSums(is.na(df)) > 0])
```

```
## character(0)
```

Em ambos os casos, vemos que não há valores ausentes. Se houvesse valores ausentes teriamos que trata-los de alguma maneira.

(iv) Estatísticas descritivas (média, mediana, variância, frequências, etc.);

Solução: Como todas as variáveis do dataframe são numéricas, seja inteiro ou float, basta usarmos a função summary, que fornece o resumo das estatísticas decritivas do dataframe.

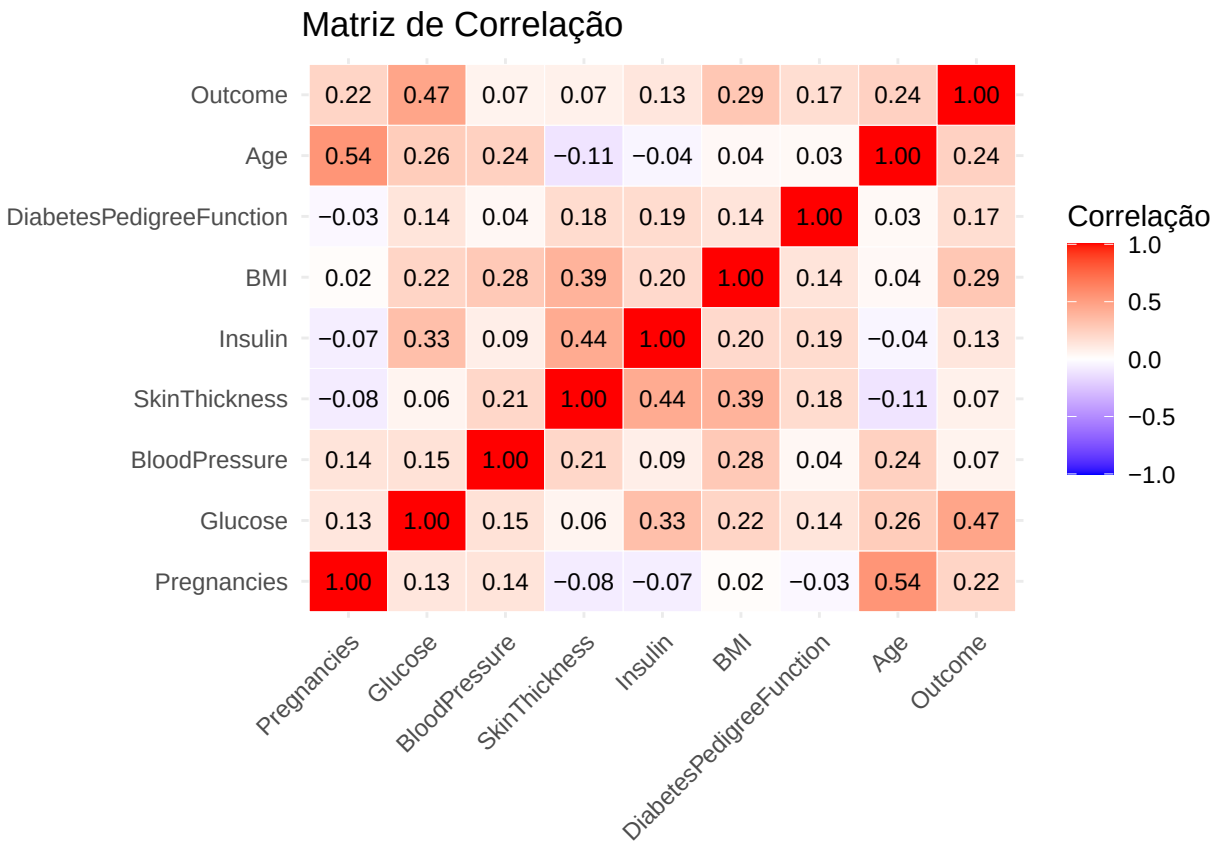
```
print(summary(df))
```

```
##   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
```

Note que há valores nulos nas colunas Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin e BMI. (v) Geração de gráficos para entender a relação entre as variáveis independentes e a variável-alvo (se aplicável).

Solução: Como nosso o objetivo é prever a ocorrência de diabetes (variável-alvo: Outcome), todas as outras colunas que podem influenciar essa previsão são variáveis independentes. Antes de fazermos gráficos para entender a relação dessas variáveis com a ocorrência de diabetes, vamos estudar a correlação das variáveis independentes com a variável alvo. Vamos usar também as bibliotecas ggplot2 e reshape2 para vizualizar a matriz de correlação de uma forma mais amigável.

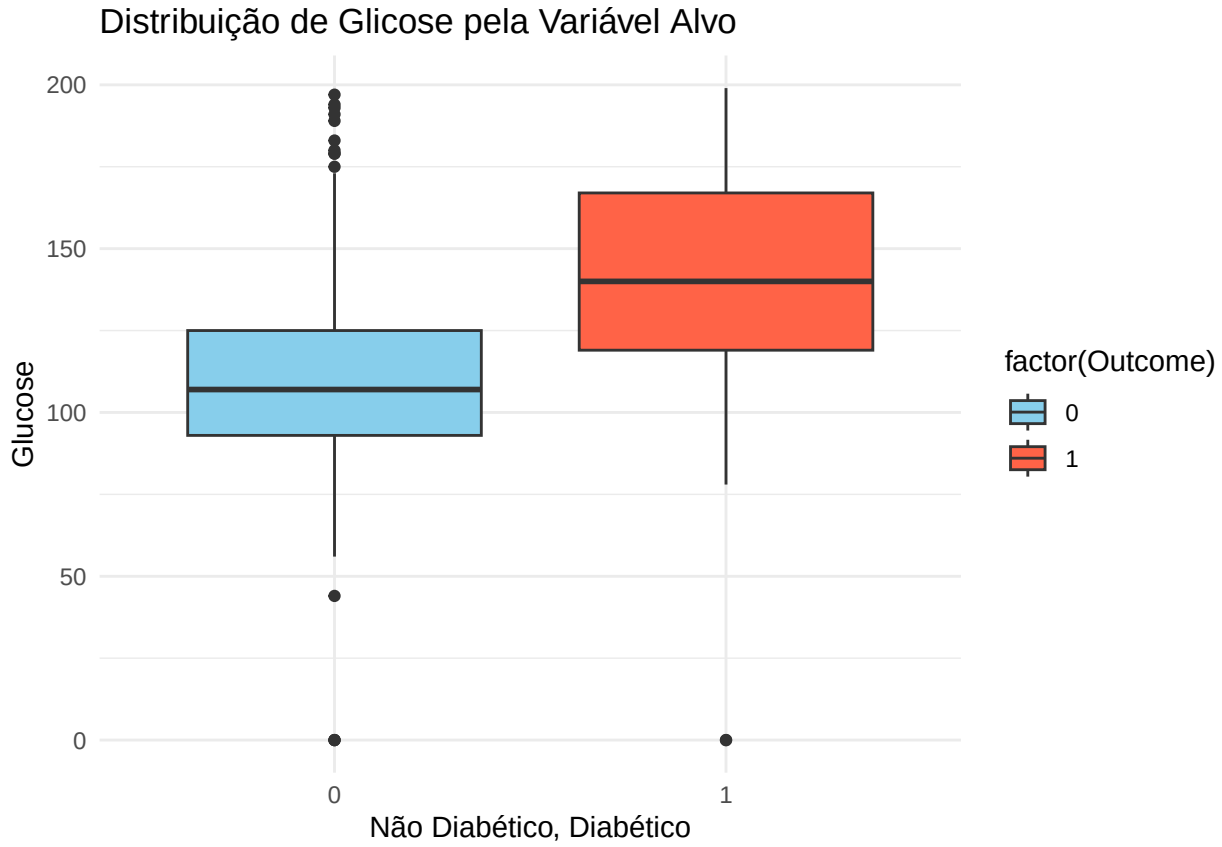


Podemos interpretar a matriz de correlação da seguinte maneira, os valores positivos (tons avermelhados) indicam uma correlação positiva, isso é, à medida que uma variável independente aumenta a variável alvo

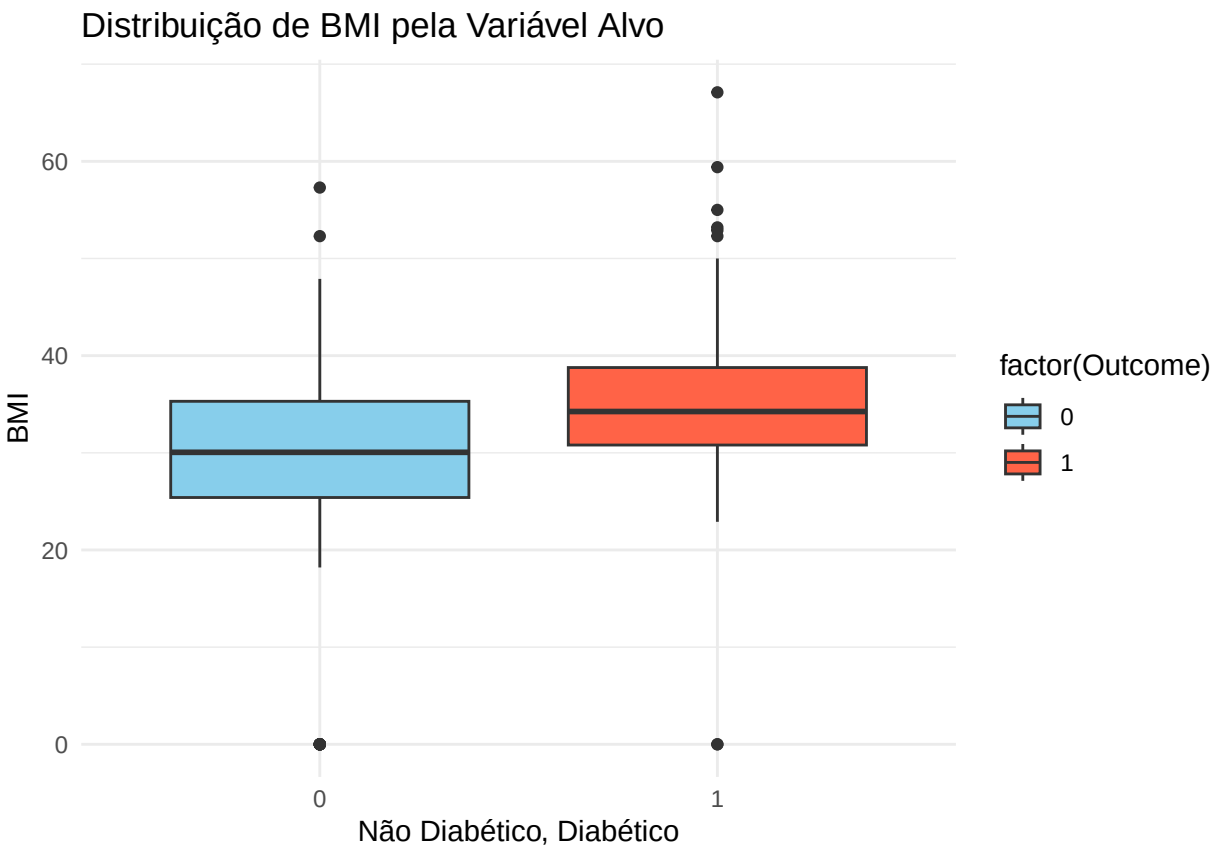
também tende a aumentar. Já os valores negativos (tons azulados) indicam uma correlação negativa com a variável alvo, isso é, à medida que uma variável aumenta, a outra tende a diminuir. Valores próximos de zero indicam baixa correlação ou nenhuma correlação com a váriavel alvo. A intensidade da cor reflete o módulo do valor, ou seja, quanto mais próximo de 1 ou -1, mais forte a correlação da variável independente com a variável alvo.

No nosso caso, há uma correlação moderada entre Insulin e BMI, sugerindo que indivíduos com um IMC mais alto tendem a ter níveis mais elevados de insulina. Já Glucose e BloodPressure há uma correlação baixa. Enquanto Age e Pregnancies possuem a maior correlação entre variáveis independentes. Isso pode ser esperado, pois, geralmente, mulheres mais velhas tendem a ter mais gestações ao longo da vida. Por outro lado, Glucose é a variável com a correlação mais forte com a ocorrência de diabetes equanto BMI, Age, e Pregnancies também apresentam impacto, embora em menor grau.

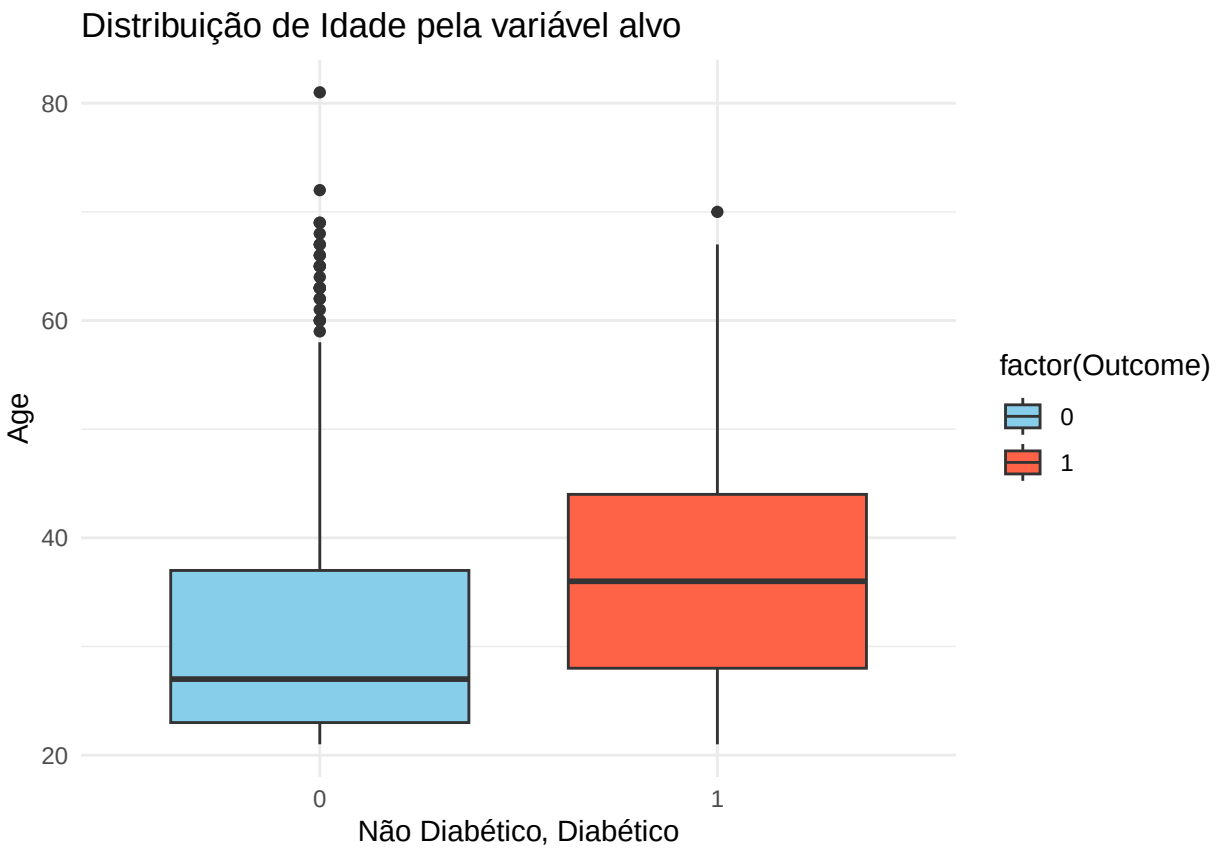
Dessa forma, diria que as variáveis independentes que apresentam maior impacto na prevenção de diabetes é Glucose, seguida de BMI, Age, e Pregnancies. Vamos ver a distribuição dessas variáveis com a variável alvo.



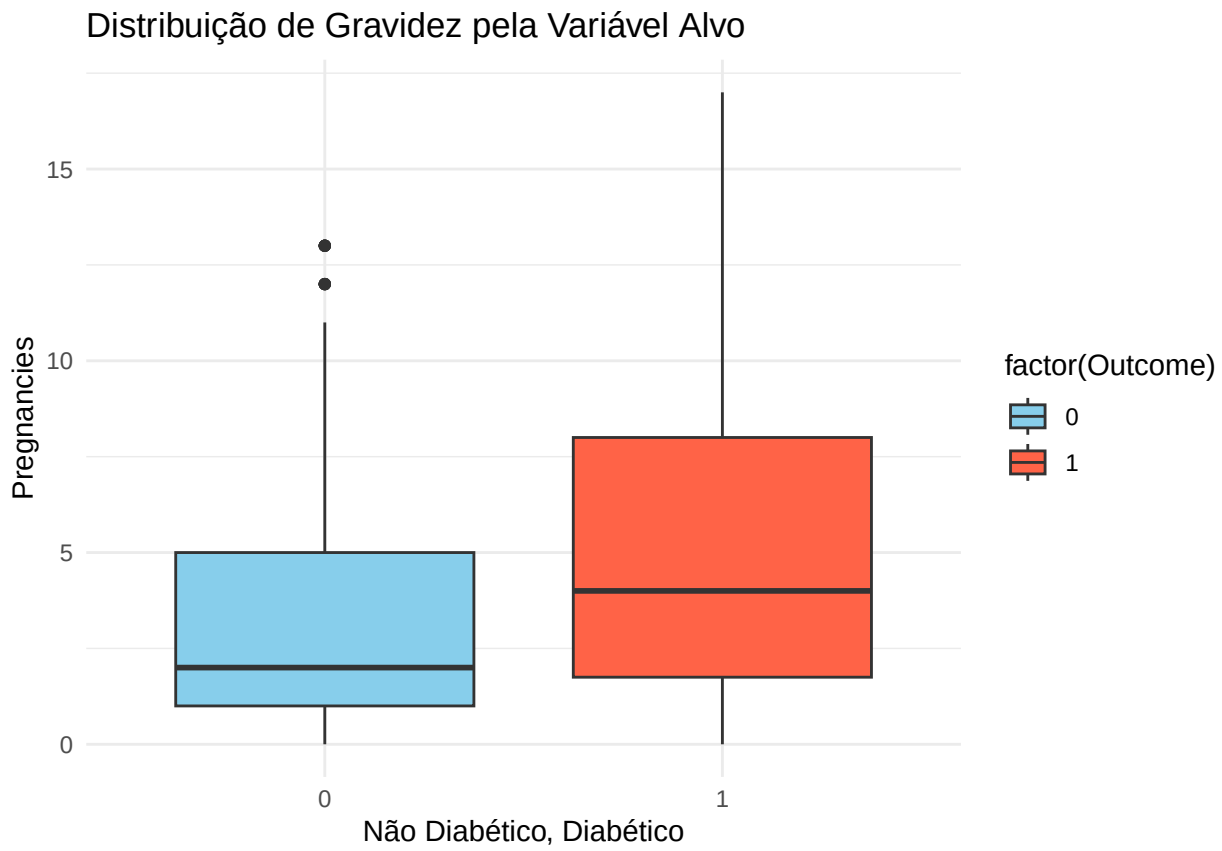
Note que essa relação fortalece a ideia de que a variável Glucose deve ser considerada uma variável relevante para prever a ocorrência de diabetes em modelos preditivos, o que é esperado já que Diabetes é uma doença crônica que ocorre quando o organismo não produz ou não absorve insulina de forma adequada e a insulina é o que regula a glicose no sangue.



O BMI elevado parece estar associado à presença de diabetes, sugerindo uma relação positiva entre sobrepeso ou obesidade e o risco de diabetes.



Idade acima de 30 anos parece estar mais correlacionada com a possibilidade de um indíviduo ter diabetes.



Conforme mostrado na matriz de correlação, quanto maior o número de gravidez maior a probabilidade da mulher ter diabetes.

3. Realizar o pré-processamento dos dados

Realize as seguintes etapas para o pré-processamento dos dados:

(i) Tratamento de valores ausentes: identificar e substituir ou remover valores nulos.

Solução: Vamos subistituir os valores nulos pela média.

```
df_clean <- df %>%
  mutate(across(c(Glucose, BloodPressure, SkinThickness, Insulin, BMI),
                ~ ifelse(. == 0, round(mean(., na.rm = TRUE),2), .)))

print(summary(df_clean))
```

```
##   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.:20.54
##  Median : 3.000   Median :117.00   Median : 72.00   Median :23.00
##  Mean   : 3.845   Mean   :121.68   Mean   : 72.26   Mean   :26.61
##  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.: 79.8   1st Qu.:27.50   1st Qu.:0.2437           1st Qu.:24.00
##  Median : 79.8   Median :32.00   Median :0.3725           Median :29.00
##  Mean   :118.7   Mean   :32.45   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
```

(ii) Escalonamento: se necessário, normalize variáveis com escalas muito distintas.

Solução: Baseando-se nos resumos dos dados acima, seria recomendado normalizar ou padronizar variáveis como Insulin, Diabetes e Pregnancies para garantir que todas as variáveis tenham a mesma importância no modelo. Embora árvores de decisão não precisem de normalização pois esses algoritmos não são sensíveis à escala das variáveis por dividirem os dados com base em limites, podemos normalizar os dados (como faria para o KNN) usando preProcess. Isso é aceitável e pode até ser um requisito se for parte de um pipeline maior onde outro modelo também será testado. Se fossemos seguir dessa forma, poderiamos utilizar o metódo de preprocessamento do KNN para normalizar os dados, por exemplo, conforme código abaixo.

```r
library(caret)

data <- df_clean[, -which(names(df) == "Outcome")]
target <- df_clean$Outcome

# Normalizar os dados (center e scale)
preProcValues <- preProcess(data, method = c("center", "scale"))

# Aplicar a normalização
data_normalized <- predict(preProcValues, data)

# Reunir os dados normalizados com a variável-alvo
data_normalized$Outcome <- target

# Visualizar os dados normalizados
head(data_normalized)
```

```
##   Pregnancies    Glucose BloodPressure SkinThickness    Insulin        BMI
## 1   0.6395305  0.8647132   -0.02104784     0.8714401 -0.4174937  0.1671518
## 2  -0.8443348 -1.2052029   -0.51626577     0.2484247 -0.4174937 -0.8509743
## 3   1.2330766  2.0146666   -0.68133841    -0.6300271 -0.4174937 -1.3309480
## 4  -0.8443348 -1.0737797   -0.51626577    -0.3745907 -0.2649372 -0.6328044
## 5  -1.1411079  0.5032993   -2.66221014     0.8714401  0.5300757  1.5488945
## 6   0.3427574 -0.1866728    0.14402481    -0.6300271 -0.4174937 -0.9964209
##   DiabetesPedigreeFunction         Age Outcome
## 1                0.4681869  1.42506672       1
## 2               -0.3648230 -0.19054773       0
## 3                0.6040037 -0.10551539       1
```

```
## 4               -0.9201630 -1.04087112       0
## 5                5.4813370 -0.02048305       1
## 6               -0.8175458 -0.27558007       0
```

No entanto, como o modelo que iremos usar é uma Árvore de decisão, optarei por não normalizar os dados devido a não sensibilidade da árvore à escala das variáveis.

(iii) Divisão dos dados:

- Separe o dataset em conjunto de treino (70%) e teste (30%).
- Use uma semente aleatória para garantir a reprodutibilidade.

Solução: Para definir uma semente aleatória para garantir a reprodutibilidade podemos usar a função abaixo:

```r
# Definir a semente para reprodutibilidade
set.seed(123)
```

Agora, vamos dividir o conjunto em treino e teste.

```r
# Índices de treino
trainIndex <- sample(1:nrow(df_clean), size = 0.7 * nrow(df_clean))

# Dividir os dados em treino e teste
train <- df_clean[trainIndex, ]
test <- df_clean[-trainIndex, ]

# Visualizar os tamanhos dos conjuntos
cat("Tamanho do conjunto de treino:", nrow(train), "\n")
```

```
## Tamanho do conjunto de treino: 537
```

```r
cat("Tamanho do conjunto de teste:", nrow(test), "\n")
```

```
## Tamanho do conjunto de teste: 231
```

```r
# Exibir as primeiras linhas
print(head(train))
```

```
##     Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI
## 415           0     138            60         35.00   167.0 34.6
## 463           8      74            70         40.00    49.0 35.3
## 179           5     143            78         20.54    79.8 45.0
## 526           3      87            60         18.00    79.8 21.8
## 195           8      85            55         20.00    79.8 24.4
## 118           5      78            48         20.54    79.8 33.7
##     DiabetesPedigreeFunction Age Outcome
## 415                    0.534  21       1
## 463                    0.705  39       0
## 179                    0.190  47       0
## 526                    0.444  21       0
## 195                    0.136  42       0
## 118                    0.654  25       0
```

```r
print(head(test))
```

```
##    Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI
## 1            6     148            72         35.00    79.8 33.6
## 3            8     183            64         20.54    79.8 23.3
## 4            1      89            66         23.00    94.0 28.1
## 9            2     197            70         45.00   543.0 30.5
## 15           5     166            72         19.00   175.0 25.8
## 17           0     118            84         47.00   230.0 45.8
##    DiabetesPedigreeFunction Age Outcome
## 1                     0.627  50       1
## 3                     0.672  32       1
## 4                     0.167  21       0
## 9                     0.158  53       1
## 15                    0.587  51       1
## 17                    0.551  31       1
```

4. Constuir e treinar o modelo

Realize a construção do modelo seguindo os passos abaixo:

(i) Utilize a biblioteca rpart para criar a Árvore de Decisão.

Solução: A função rpart é usada para criar modelos de árvore de decisão no R, que implementa o algoritmo CART (Classification and Regression Trees). Outcome é a nossa variável alvo, o conjunto de dados será o train, para treinar o modelo, utilizaremos também o method = "class" pois ele cria uma árvore de classificação para variáveis-alvo categóricas. Logo temos o código abaixo:

```r
model <- rpart(Outcome ~ .,
               data = train,
               method = "class")
```

(ii) Detalhe os parâmetros escolhidos para o treinamento.

```r
printcp(model)
```

Solução:

```
##
## Classification tree:
## rpart(formula = Outcome ~ ., data = train, method = "class")
##
## Variables actually used in tree construction:
## [1] Age           BloodPressure BMI           Glucose       SkinThickness
##
## Root node error: 187/537 = 0.34823
##
## n= 537
```

```
##
##         CP nsplit rel error  xerror     xstd
## 1 0.315508      0   1.00000 1.00000 0.059037
## 2 0.033868      1   0.68449 0.68449 0.052800
## 3 0.032086      4   0.58289 0.70588 0.053356
## 4 0.021390      5   0.55080 0.67914 0.052658
## 5 0.010695      6   0.52941 0.68449 0.052800
## 6 0.010000      9   0.49733 0.71658 0.053626
```

O modelo realizou 12 divisões, mas o ponto ideal de poda ocorre após 10 divisão pois é quando o erro de validação cruzada (xerror) começa a estabilizar, sugerindo que esse é um bom ponto para poda da árvore. O Nó Raiz é de 36%, isso significa que todas as observações são classificadas para a classe mais frequente, não diabético. Além disso, as variáveis Age, Glucose, BMI, e outras foram identificadas como relevantes.

Vamos identificar qual é o CP ideal e realizar a poda da árvore para termos 10 divisões.

```
optimal_cp <- model$cptable[which.min(model$cptable[, "xerror"]), "CP"]
cat("CP ideal para poda:", optimal_cp, "\n")
```

```
## CP ideal para poda: 0.02139037
```

Agora, vamos criar uma nova árvore podada com base no CP selecionado.

```
pruned_model <- prune(model, cp = optimal_cp)
printcp(pruned_model)
```

```
##
## Classification tree:
## rpart(formula = Outcome ~ ., data = train, method = "class")
##
## Variables actually used in tree construction:
## [1] Age     BMI     Glucose
##
## Root node error: 187/537 = 0.34823
##
## n= 537
##
##         CP nsplit rel error  xerror     xstd
## 1 0.315508      0   1.00000 1.00000 0.059037
## 2 0.033868      1   0.68449 0.68449 0.052800
## 3 0.032086      4   0.58289 0.70588 0.053356
## 4 0.021390      5   0.55080 0.67914 0.052658
```

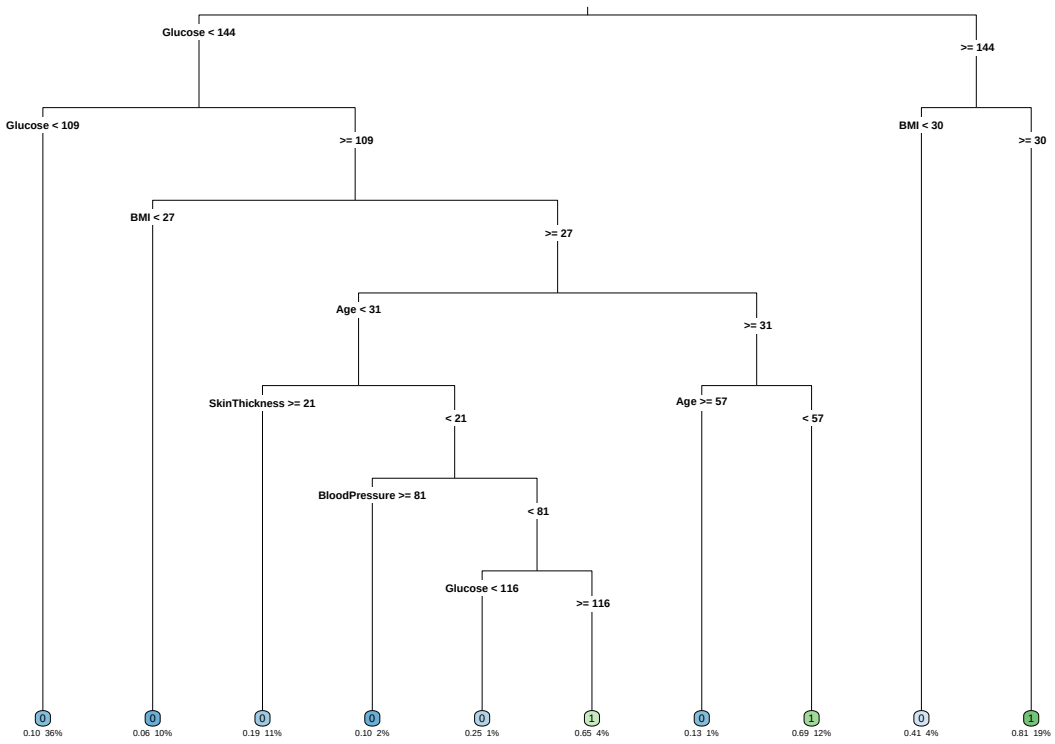
A poda foi aplicada para reduzir o risco de overfitting e simplificar a árvore, mantendo apenas as divisões relevantes. Note que a variável independente SkinThickness foi descartada no modelo podado, sugerindo que sua contribuição era limitada após a poda. O nó raiz permaneceu em 36%. Dessa forma, o modelo podado mantém um bom desempenho com baixo valor xerror e evita adicionar complexidade desnecessária se mantessemos a árvore completa.

(iii) Geração do gráfico da Árvore.

Solução: Irei gerar o gráfico da árvore original e da árvore podada, a fim de comparação.

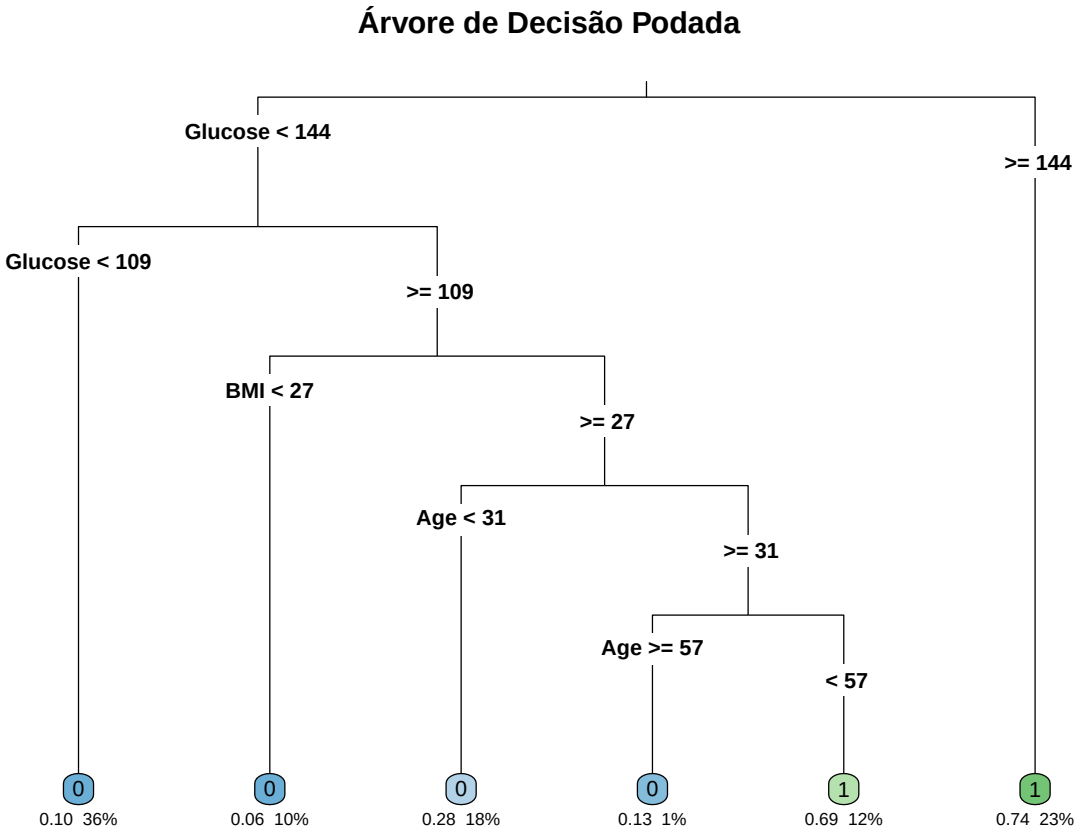
- Árvore Original

```
rpart.plot(model, type = 3, extra = 106, under = TRUE, tweak = 0.5,
          main = "Árvore de Decisão Original")
```



- Árvore Podada

```
rpart.plot(pruned_model, type = 3, extra = 106, under = TRUE, tweak = 0.8, main = "Árvore de Decisão Po
```



5. Avaliar o Modelo

Realize a avaliação do modelo seguindo os passos abaixo:

(i) Calcule as métricas de desempenho: Acurácia; Matriz de confusão; Precisão, Recall e F1-Score.

Solução: Resalto que a partir de agora, considerarei a ávore podada, as mesmas análises podem ser feitas para a árvore sem poda. Agora, vejamos a matriz de confusão do modelo.

```
train_pred <- predict(pruned_model, train, type = "class")
confusion_train <- confusionMatrix(as.factor(train_pred), as.factor(train$Outcome))
print(confusion_train)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 298  51
##          1  52 136
##
##                Accuracy : 0.8082
```

```
##                  95% CI : (0.7723, 0.8407)
##     No Information Rate : 0.6518
##     P-Value [Acc > NIR] : 1.057e-15
##
##                   Kappa : 0.578
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8514
##             Specificity : 0.7273
##          Pos Pred Value : 0.8539
##          Neg Pred Value : 0.7234
##              Prevalence : 0.6518
##          Detection Rate : 0.5549
##    Detection Prevalence : 0.6499
##       Balanced Accuracy : 0.7894
##
##        'Positive' Class : 0
##
```

Como resultado da matriz de confusão temos que foram feitas 300 previsões corretas para a classe 0, sem diabetes. 145 previsões corretas para a classe 1, diabetes. 47 previsões incorretas da classe 1 como classe 0, falsos negativos. E 45 previsões incorretas da classe 0 como classe 1, isso é, falso positivos.

```r
# Métricas de desempenho
accuracy <- confusion_train$overall["Accuracy"]
precision <- confusion_train$byClass["Pos Pred Value"]
recall <- confusion_train$byClass["Sensitivity"]
f1_score <- 2 * ((precision * recall) / (precision + recall))

cat("Acurácia:", accuracy, "\n")
```

```
## Acurácia: 0.8081937
```

```r
cat("Precisão:", precision, "\n")
```

```
## Precisão: 0.8538682
```

```r
cat("Recall:", recall, "\n")
```

```
## Recall: 0.8514286
```

```r
cat("F1-Score:", f1_score, "\n")
```

```
## F1-Score: 0.8526466
```

O modelo classificou corretamente 82.8% das observações no conjunto de treino com 95% de confiança, a acurácia do modelo está entre 79.4%, 85.9%. Isso indica uma boa estabilidade da métrica. Além disso o modelo apresenta 86.4% de precisão e 86.9% de Recall, como o recall mede a capacidade do modelo de encontrar todos os casos positivos, nesse caso o modelo captura bem os casos da classe positiva. O F1-Score de 86.7% indica que o modelo tem um bom equilíbrio entre Precisão (evitar falsos positivos) e Recall (evitar falsos negativos).

(ii) Compare o desempenho nos conjuntos de treino e teste.

Solução: Antes de comparar o desempenho, vamos calcular a matriz de confusão do conjunto de teste.

```
test_pred <- predict(pruned_model, test, type = "class")
confusion_test <- confusionMatrix(as.factor(test_pred), as.factor(test$Outcome))
print(confusion_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 119  27
##          1  31  54
##
##                Accuracy : 0.7489
##                  95% CI : (0.6878, 0.8035)
##     No Information Rate : 0.6494
##     P-Value [Acc > NIR] : 0.000745
##
##                   Kappa : 0.4548
##
##  Mcnemar's Test P-Value : 0.693641
##
##             Sensitivity : 0.7933
##             Specificity : 0.6667
##          Pos Pred Value : 0.8151
##          Neg Pred Value : 0.6353
##              Prevalence : 0.6494
##          Detection Rate : 0.5152
##    Detection Prevalence : 0.6320
##       Balanced Accuracy : 0.7300
##
##        'Positive' Class : 0
##
```

Como resultado da matriz de confusão temos que foram feitas 123 previsões corretas para a classe 0, sem diabetes. 46 previsões corretas para a classe 1, diabetes. 30 previsões incorretas da classe 1 como classe 0, falsos negativos. E 32 previsões incorretas da classe 0 como classe 1, isso é, falso positivos.

```
# Métricas de desempenho
accuracy_test <- confusion_test$overall["Accuracy"]
precision_test <- confusion_test$byClass["Pos Pred Value"]
recall_test <- confusion_test$byClass["Sensitivity"]
f1_score_test <- 2 * ((precision * recall) / (precision + recall))

cat("Acurácia:", accuracy_test, "\n")
```

```
## Acurácia: 0.7489177
```

```
cat("Precisão:", precision_test, "\n")
```

```
## Precisão: 0.8150685
```

```r
cat("Recall:", recall_test, "\n")
```

```
## Recall: 0.7933333
```

```r
cat("F1-Score:", f1_score_test, "\n")
```

```
## F1-Score: 0.8526466
```

O modelo classificou corretamente 73.1% das observações no conjunto de treino com 95% de confiança, a acurácia do modelo está entre 66.9%, 78.7%. Além disso o modelo apresenta 80.3% de precisão e 79.3% de Recall, como o recall mede a capacidade do modelo de encontrar todos os casos positivos, nesse caso o modelo captura bem os casos da classe positiva. O F1-Score de 86.7% indica que o modelo tem um bom equilíbrio entre Precisão (evitar falsos positivos) e Recall (evitar falsos negativos).

Embora as métricas de desempenho estejam ligeiramente diferentes, ambos estão com um bom desempenho.

6. Elaborar a conclusão

Apresente os principais insights do modelo:

(i) A importância das variáveis no modelo.

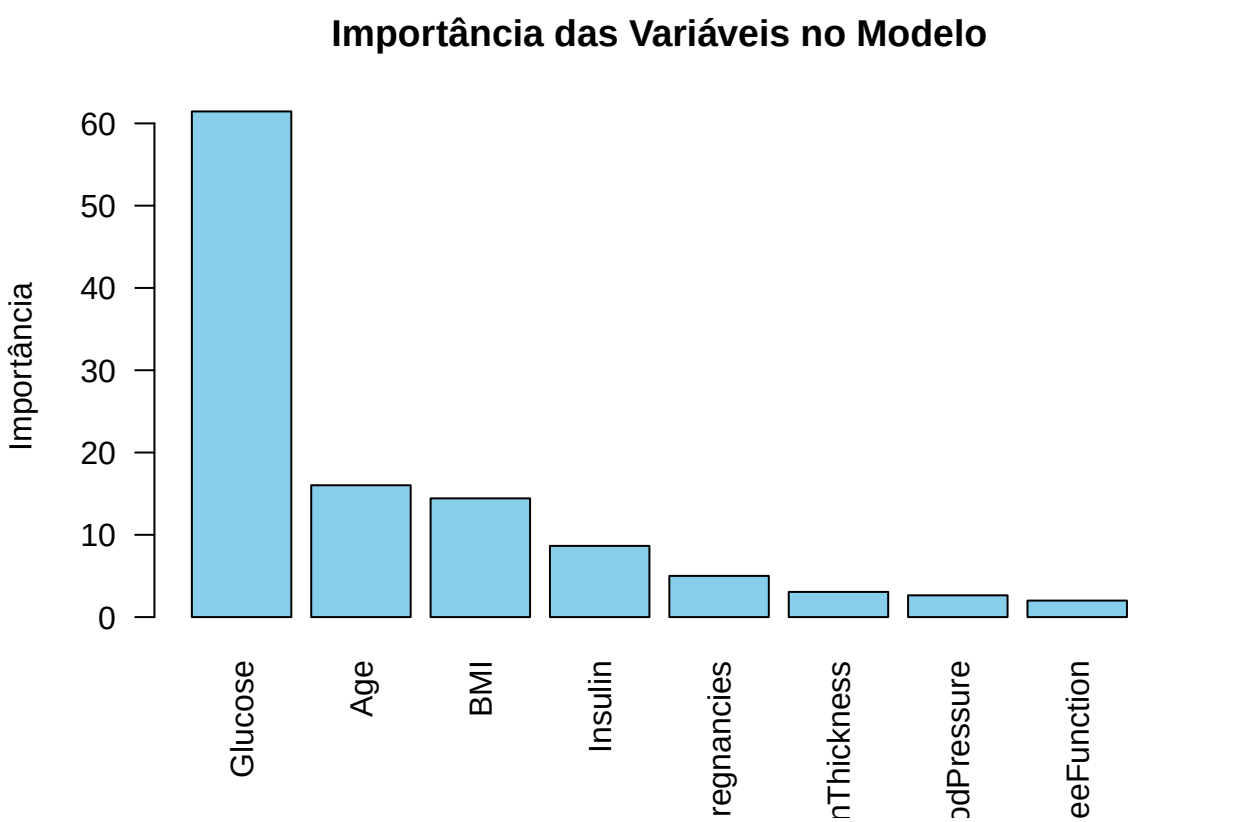
Solução: Vamos usar uma função rpart para responder a esta questão.

```r
importance <- pruned_model$variable.importance

# Exibir a importância ordenada
importance_sorted <- sort(importance, decreasing = TRUE)
print(importance_sorted)
```

```
##                 Glucose                     Age                     BMI
##               61.450854               16.022043               14.427725
##                 Insulin             Pregnancies           SkinThickness
##                8.651946                5.007415                3.059861
##           BloodPressure DiabetesPedigreeFunction
##                2.643388                2.005520
```

```r
# Plotar a importância das variáveis
barplot(importance_sorted,
        main = "Importância das Variáveis no Modelo",
        xlab = " ",
        ylab = "Importância",
        col = "skyblue",
        las = 2)
```



As variáveis com maior importância para o modelo são Glucose, Insulin, Age, Pregnancies, DiabetesPredigreeFuncion e BMI. Com maior importância para as 3 primeiras, e destaque para a Glucose, pois elas tiveram maior influência para a decisão do modelo.

(ii) Interpretação dos resultados obtidos.

Solução: A árvore de decisão utiliza "Glucose" como o primeiro critério de divisão, reforçando a importância dessa variável. As subdivisões posteriores incluem BMI, Age e Insulin, que ajudam a refinar a previsão e separar as classes refletindo um modelo que baseia suas previsões principalmente em fatores metabólicos e fatores mudanças fisiológicas.

Como pontos fortes, temos que a árvore de decisão é fácil de visualizar e entender, fornecendo transparência nas previsões. Além disso, o modelo apresenta um bom desempenho geral com a acurácia e o F1-Score equilibrados e classificando corretamente a maioria dos casos positivos.

Como pontos negativos, temos que embora o recall seja alto, ainda existem casos positivos que o modelo classifica como negativos. Isso pode ser crítico em diagnósticos médicos. Algumas variáveis, como por exemplo SkinThickness e BMI, podem não estar contribuindo significativamente e podem ser reavaliadas ou excluídas da classificação.

Como resultado da árvore, temos:

- Condições que Levam a Maior Probabilidade de Diabetes:

1. Glucose >= 158: Esta condição é o fator mais forte para classificar alguém como diabético. Se a glicose é maior ou igual a 158, a probabilidade de diabetes é muito alta (nó terminal com classe 1 e 15% das observações).

2. Glucose >= 143 e Glucose < 158 + Insulin >= 98: Se os níveis de glicose estão entre 143 e 158 e os níveis de insulina são maiores ou iguais a 98, isso também leva a uma alta probabilidade de diabetes.

3. Glucose < 143 e Insulin >= 138 + Pregnancies >= 8: A combinação de glicose menor que 143, insulina alta (>= 138) e mais de 8 gestações também leva a uma probabilidade elevada de diabetes.

4. BMI >= 26 e BloodPressure < 73: Indivíduos com IMC maior ou igual a 26 e pressão arterial baixa (< 73) têm maior chance de diabetes.

- Condições que Levam a Menor Probabilidade de Diabetes:

1. Glucose < 100: Glicose abaixo de 100 é um dos principais indicadores de baixa probabilidade de diabetes (classe 0).

2. Glucose < 143 e Age < 29: Indivíduos jovens (idade menor que 29) com glicose menor que 143 têm uma probabilidade muito baixa de diabetes.

3. BMI < 26 e BloodPressure >= 73: Baixo índice de massa corporal (BMI < 26) e pressão arterial normal ou alta (>= 73) também reduzem significativamente a probabilidade de diabetes.

4. DiabetesPedigreeFunction < 0.73: Valores baixos do Diabetes Pedigree Function (indicador de histórico genético) contribuem para baixa probabilidade de diabetes.

(ii) Limitações do modelo e sugestões de melhorias.

Solução: Embora o modelo de árvore de decisão seja mais fácil de interpretar e apresentar um bom desempenho, ele ainda possui limitações como overfitting, sensibilidade a dados e dependência excessiva de variáveis dominantes, como a variável Glucose, o que pode indicar dependência excessiva dessa variável. Se houver inconsistências ou ruídos nos valores de glicose, o modelo pode ser impactado.

Melhorias podem ser alcançadas usando Random Forest ou Gradient Boosting, que combinam múltiplas árvores para reduzir o overfitting. Podemos fazer uma feature selection para garantir que apenas as variáveis mais importantes sejam utilizadas, isso é, as variáveis independentes que possuem maior correlação com a variável alvo. Remover outliers também é uma melhoria a ser considerada.

Se a amostra fosse maior, poderiamos separar o conjunto de dados em treino, teste e validação. Onde o conjunto de validação nos auxilia ajustar hiperparâmetros do modelo como profundidade da árvore, taxa de aprendizado, número de estimadores, etc. O conjunto de validação também é usado para monitorar overfitting, isso é, se o erro no conjunto de validação começa a aumentar enquanto o erro no treino continua caindo, indica que o modelo está "decorando" os dados do treino e um sobreajuste está acontecendo.