TP1_TDVI

Tomás Gallo, Nadia Molina y Jazmín Sneider

2024-08-29

Ejercicio 1: Introducción al problema

Para nuestro análisis, seleccionamos un archivo CSV que contiene 36,285 observaciones de reservas y 17 variables relacionadas con estas reservas. Este archivo incluye información detallada sobre las reservas de un hotel, como el precio de la habitación, el número de personas que se iban a hospedar, entre otros datos relevantes. Además, nos proporciona información sobre si la reserva fue cancelada o no. Aproximadamente un tercio de las reservas en el archivo fueron canceladas, mientras que las restantes dos terceras partes no lo fueron.

El objetivo de nuestro análisis es predecir si una reserva será cancelada o no, basándonos en las distintas variables asociadas a cada observación.

Las variables o columnas del CSV son las siguientes:

Booking_ID: Identificador único de cada reserva. Decidimos no utilizar esta variable para la predicción.

number.of.adults: Número de adultos incluidos en la reserva.

number.of.children: Número de niños incluidos en la reserva.

number.of.week.nights: Número de noches entre semana incluidas en la reserva.

number.of.weekend.nights: Número de noches de fin de semana incluidas en la reserva.

type.of.meal: Tipo de plan de comida incluido en la reserva. Esta es una variable categórica con las opciones: “Meal Plan 1”, “Meal Plan 2”, “Meal Plan 3” y “Not Selected”.

car.parking.space: Indica si se solicitó un lugar en el estacionamiento del hotel (1 para sí, 0 para no).

room.type: Tipo de habitación reservada. Es una variable categórica con siete posibles valores: “Room_Type 1”, “Room_Type 2”, “Room_Type 3”, “Room_Type 4”, “Room_Type 5”, “Room_Type 6” y “Room_Type 7”.

lead.time: Número de días entre la fecha de la reserva y la fecha de llegada.

market.segment.type: Tipo de segmento de mercado asociado a la reserva. Es una variable categórica con cinco valores: “Aviation”, “Complementary”, “Corporate”, “Offline” y “Online”.

repeated: Indica si la reserva es repetida, es decir, si ya se había realizado una reserva similar anteriormente (1 para sí, 0 para no).

P.C: Número de reservas canceladas por el cliente antes de esta reserva.

P.not.C: Número de reservas no canceladas por el cliente antes de esta reserva. average.price: Precio promedio de la reserva. special.requests: Número de solicitudes especiales realizadas por el cliente en la reserva.

date.of.reservation: Fecha en la que se realizó la reserva. Originalmente aparecía en formato fecha (mes/dia/año) pero para nuestro análisis decidimos agregar una nueva variable llamada season que contiene la estación del año según su mes. Además, como el csv está hecho con datos de EEUU, decidimos poner las estaciones del año correspondientes al país.

booking.status: Indica si la reserva fue cancelada o no. Originalmente, esta variable presentaba los valores "Canceled" o "Not Canceled", pero para nuestro análisis se codificó como 1 para cancelada y 0 para no cancelada.

Ejercicio 2: Preparación de los datos

Cargamos los datos del csv elegido.

```
datos <- read.csv(file = "booking.csv", header = TRUE)
```

Lo primero que hicimos fue convertir la variable predicha, a una variable binaria, 1 si es "canceled", y 0 en caso contrario. Por otro lado, convertimos las variables char, en tipo factor para tomarlas como categóricas en nuestro modelo. Por último, el csv tiene el detalle de en qué fecha reservó la persona, consideramos que para hacer un análisis de esas fechas debíamos agruparlas según las estaciones del año en el que fueron hechas las reservas.

```
datos$booking.status <- ifelse(datos$booking.status == "Canceled",1,0)
# Convertir variables categóricas a factores
datos$room.type <- as.factor(datos$room.type)
datos$type.of.meal <- as.factor(datos$type.of.meal)
datos$market.segment.type <- as.factor(datos$market.segment.type)

# Asegúrate de que la fecha esté en formato Date
datos$date.of.reservation <- as.Date(datos$date.of.reservation, format = "%m/%d/%Y")

# Agrupar las fechas en estaciones del año
datos$season <- as.factor(
  ifelse(format(datos$date.of.reservation, "%m") %in% c("12", "01", "02"), "Winter",
  ifelse(format(datos$date.of.reservation, "%m") %in% c("03", "04", "05"), "Autumn",
  ifelse(format(datos$date.of.reservation, "%m") %in% c("06", "07", "08"), "Summer", "Spring"))))
```

Para ver cómo interactúan las distintas variables del CSV hicimos una matriz de correlación. Primero lo hicimos con las variables numéricas y luego con las categóricas, esta división se debe a que hay demasiadas variables para un sólo gráfico y si poníamos todas juntas no se lograba apreciar bien la correlación.

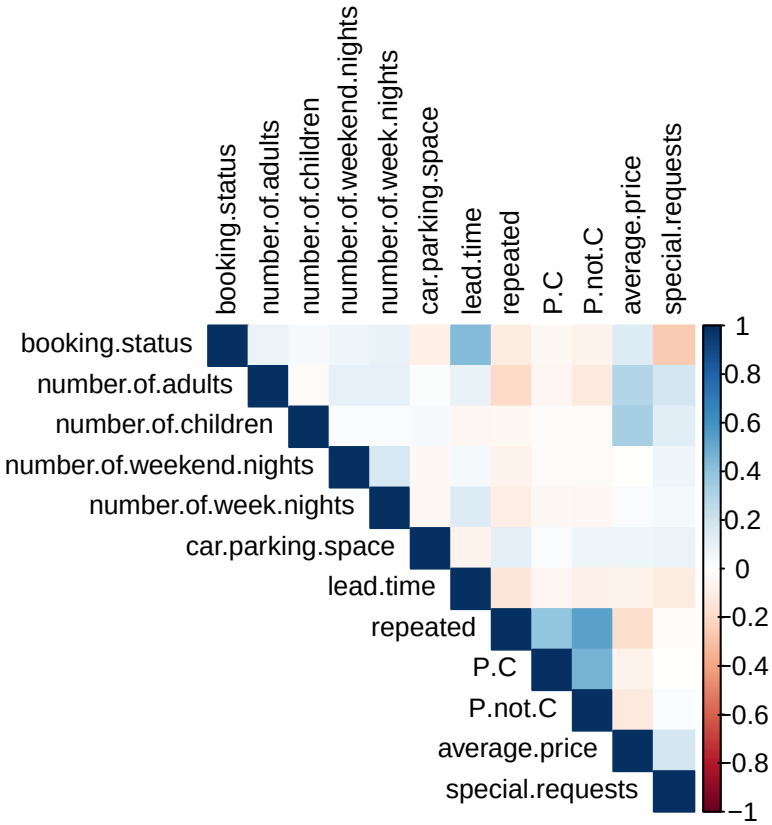
Tenemos en rojo la cercanía a correlación -1 y en azul a 1. Mas cercano a 1 implica relación positiva y cercano a -1 implica relación negativa. Pudimos observar que la variable con mayor relación positiva con booking.status era lead.time. Estar relacionada positivamente con booking.status hace referencia a que contribuye más a la cancelación, mientras que estar relacionado negativamente hace referencia a que contribuye a que la reserva no sea cancelada. Además de lead.time, la variable special.requests también parece tener una correlación moderada negativa con booking.status. Average.price también muestra una correlación positiva con nuestra variable a predecir. No es casualidad que más adelante en el árbol de decisión éstas sean las primeras variables que aparecen y generan divisiones.

```
#importamos la libreria corr
library(corrplot)
```

```
## corrplot 0.94 loaded
```

```
# Seleccionar solo las columnas numéricas
variables_numericas <- datos[, c('booking.status', 'number.of.adults', 'number.of.children', 'number.of

# Calcular la matriz de correlación para variables numéricas
matriz_de_correlaciones_vn <- cor(variables_numericas, use = "complete.obs")

# Visualizar la matriz de correlación numérica usando corrplot
corrplot(matriz_de_correlaciones_vn, method = "color", type = "upper", tl.col = "black",
```

```
        tl.cex = 0.8, title = "Matriz de Correlación - Variables Numéricas", mar = c(0,0,1,0))
```

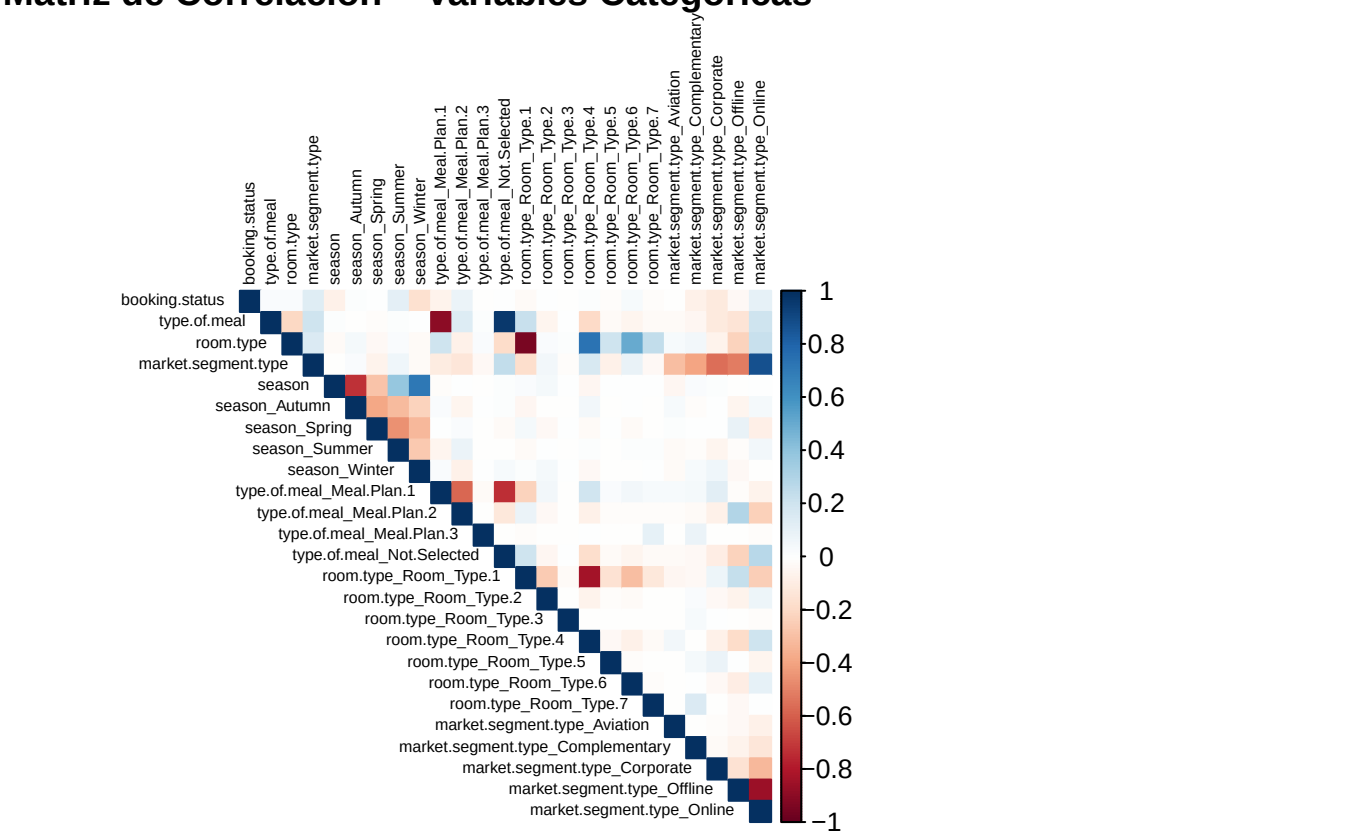


Para entender mejor la relación entre las variables categóricas y el estado de las reservas (**booking.status**), creamos una matriz de correlaciones específica para estas variables. Esta matriz no solo presenta cada variable categórica de manera individual, sino que también desglosa sus categorías, permitiéndonos observar cómo cada una de ellas influye en la predicción de cancelación de reservas. Al analizar la matriz de correlaciones, observamos que las estaciones del año en las que se hicieron las reservas (**season**) muestran una correlación significativa con **booking.status**. De acuerdo con el cuadro de colores de la matriz, las reservas tienden a ser más canceladas en verano (indicado por un color azul claro, que representa valores de correlación entre 0 y 0.2). Por otro lado, en invierno, las reservas muestran una menor tendencia a ser canceladas (reflejado por un color rojo claro, con valores de correlación entre 0 y -0.2).

Además, detectamos que la variable **market.segment.type** también presenta una influencia notable en la cancelación de reservas, algo que confirmaremos más adelante al analizar los resultados del árbol de decisión.

```
library(fastDummies)

# Crear las variables dummy sin eliminar ninguna categoría
variables_categoricas <- dummy_cols(datos, select_columns = c("season", "type.of.meal", "room.type", "ma

# Seleccionar solo las variables dummy generadas y la variable booking.status
variables_categoricas <- variables_categoricas[, c("booking.status", grep("type.of.meal|room.type|marke

# Asegurarse de que todas las columnas sean numéricas
variables_categoricas <- data.frame(lapply(variables_categoricas, as.numeric))

# Calcular la matriz de correlaciones
matriz_de_correlaciones_vc <- cor(variables_categoricas, use = "complete.obs")
```

```r
# Generar el corrplot
corrplot(matriz_de_correlaciones_vc, method = "color", type = "upper", tl.col = "black",
         tl.cex = 0.5, title = "Matriz de Correlación - Variables Categóricas", mar = c(0,0,1,0))
```



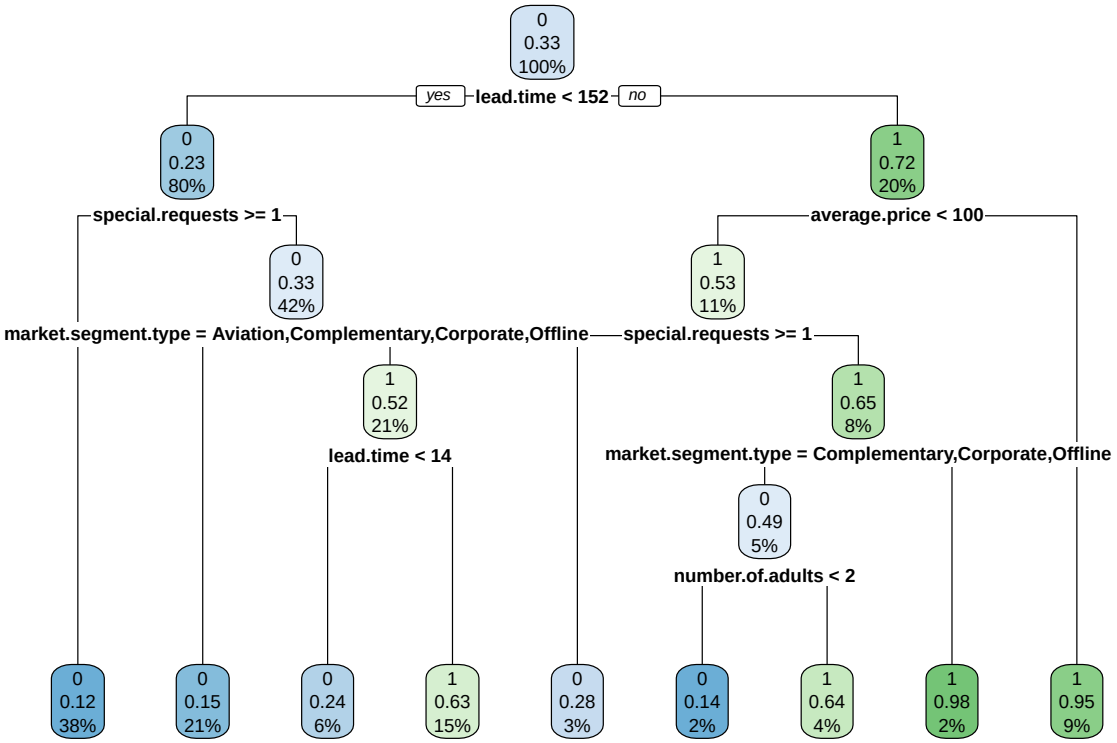
Ejercicio 3: Construcción de un árbol de decisión básico

Primero realizamos la división del conjunto de datos [70%, 15%, 15%]

```r
observaciones_totales <- nrow(datos)
set.seed(500)
conjunto_total<-sample(1:observaciones_totales, size=observaciones_totales)
# Calcular los índices de corte
indice_entrenamiento <- round(0.7 * observaciones_totales)
indice_validacion <- round(0.85 * observaciones_totales)
# Crear los subconjuntos
conjunto_entrenamiento <- conjunto_total[1:indice_entrenamiento]
conjunto_validacion <- conjunto_total[(indice_entrenamiento + 1):indice_validacion]
conjunto_test <- conjunto_total[(indice_validacion + 1):observaciones_totales]

data_entrenamiento <- datos[conjunto_entrenamiento, ]
data_validacion <- datos[conjunto_validacion, ]
data_test <- datos[conjunto_test, ]
```

Luego realizamos el arbol e imprimimos los valores de los hiperparametros con los valores por defecto para analizarlos:

```
library(rpart)
library(rpart.plot)
arbol <- rpart(booking.status ~ number.of.adults + number.of.children + number.of.weekend.nights + numbe
rpart.plot(arbol)
```




```
print(arbol$control)
```

```
## $minsplit
## [1] 20
##
## $minbucket
## [1] 7
##
## $cp
## [1] 0.01
##
## $maxcompete
## [1] 4
##
## $maxsurrogate
## [1] 5
##
## $usesurrogate
## [1] 2
##
## $surrogatestyle
## [1] 0
##
## $maxdepth
## [1] 30
```

```
##
## $xval
## [1] 10
```

Observando el árbol que obtuvimos dejando todos los hiperparámetros por defecto notamos que: - El primer nodo del árbol usa la variable **lead.time** como su principal criterio de división. Esto confirma lo que notamos en la matriz de correlación, donde la variable **lead.time** era la que más correlación tenía con nuestra variable de interés. Usando esta variable el árbol se divide en dos ramas principales: una para <152 y otra para >=152. Esto sugiere que reservas realizadas con más de 152 días de anticipación tienen un comportamiento significativamente diferente en términos de probabilidad de ser o no canceladas. - Para reservas con **lead.time** >= 152, la siguiente división significativa es por **average.price**. Para precios promedio menores a 100, la proba de cancelación es menor. Esto nos podría estar diciendo que los clientes que reservan con mucha anticipación y a un precio más bajo tienden a no cancelar sus reservas. - En la otra rama de **lead.time**, donde es <152, observamos que la variable **special.request** tiene importancia a la hora de decidir. Si no hay solicitudes especiales, el árbol se sigue dividiendo aún más usando la variable **market.segment.type**. Se destaca que los segmentos de mercado como "Complementary", "Corporate", "Offline" y "Aviation" tienen diferentes probabilidades de cancelación. Esta misma variable se utiliza también en otra rama del árbol por lo que su importancia destaca. - Otra variable como **number.of.adults** aparece en los niveles inferiores del árbol, sugiriendo que tienen una menor influencia directa pero aún aportan en ciertas combinaciones específicas de condiciones y **lead.time** vuelve a repetirse en los niveles inferiores del árbol volviendo a mostrar otra vez su importancia.

Ejercicio 4: Evaluación del árbol de decisión básico

Realizamos las predicciones en el conjunto de testeo

```
predicciones_original <- predict(arbol, newdata = data_test, type = "class")
```

Ahora que tenemos las predicciones podemos calcular las metricas de performance:

Matriz de confusión:

Podemos decir que, a simple vista, el modelo parece estar haciendo un buen trabajo en la clasificación, ya que los valores en la diagonal de la matriz (que representan los casos en los que el modelo acertó) son considerablemente mayores que los valores fuera de la diagonal (los casos en los que el modelo falló). Notamos que el modelo cometió errores en 949 casos de 5443, lo que representa un 17.4% de error.

```
confusion_matrix <- table(Predicted = predicciones_original, Actual = data_test$booking.status)
print(confusion_matrix)
```

```
##          Actual
## Predicted    0    1
##         0 3258  570
##         1  379 1236
```

Accuracy:

Observando los resultados, podemos decir que a simple vista el modelo parece estar haciendo un buen trabajo, acertando en 4494 de 5443 casos, lo que corresponde a un 82.6% de accuracy. Esto significa que el modelo predice correctamente la clase de los datos en la mayoría de los casos.

```
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(accuracy)
```

```
## [1] 0.8256476
```

Precisión y Recall:

-La precisión del modelo se calcula como la proporción de verdaderos positivos sobre el total de predicciones positivas realizadas por el modelo. En otras palabras, la precisión mide la capacidad del modelo para evitar falsos positivos, es decir, predecir un positivo cuando en realidad no lo es. -El recall mide la proporción de verdaderos positivos sobre todos los casos que eran realmente positivos. Esta métrica refleja la capacidad del modelo para detectar todos los casos positivos presentes en el conjunto de datos. Los resultados obtenidos muestran que el modelo tiene una precisión de 76.53% y un recall de 68.44%. La precisión relativamente alta sugiere que el modelo es eficaz para evitar falsos positivos. Sin embargo, el recall indica que todavía hay un número significativo de verdaderos positivos que el modelo no está capturando por lo que aún hay bastante margen de mejora.

```
# PRECISIÓN = TP/(TP+FP)
TP <-confusion_matrix[2,2]
FP <-confusion_matrix[2,1]
PRECISION <- TP/(TP+FP)
print(PRECISION)
```

```
## [1] 0.7653251
```

```
# RECALL = TP/(TP+FN)
FN <-confusion_matrix[1,2]
RECALL <- TP/(TP+FN)
print(RECALL)
```

```
## [1] 0.6843854
```

F1-score:

El F1-score nos ayuda a balancear las métricas que obtuvimos hasta el momento. Un modelo con alta precisión pero bajo recall no está encontrando todas las cosas que debería. Y un modelo con alto recall pero baja precisión está encontrando muchas cosas, pero muchas no son lo que estamos verdaderamente buscando. La fórmula del F1-score es una manera de combinar precisión y recall en una sola cifra. Es como tomar un promedio ponderado donde queremos que ambas partes sean importantes. La fórmula se asegura de que si uno de los dos es muy bajo, el F1-score también será bajo, porque queremos un buen equilibrio entre ambos. En general obtuvimos un valor, no perfecto, pero que muestra un buen balance entre ambas métricas.

```
F1SCORE <- (2*PRECISION*RECALL)/(PRECISION+RECALL)
print(F1SCORE)
```

```
## [1] 0.7225957
```

Auc-ROC:

La curva ROC traza cómo varía la tasa de verdaderos positivos en relación con la tasa de falsos positivos a medida que se ajusta el threshold del modelo. Un modelo ideal tendría una curva que se acerca al borde superior izquierdo del gráfico, indicando una alta tasa de verdaderos positivos y una baja tasa de falsos positivos.Y la AUC (área bajo la curva) resume el rendimiento global del modelo de clasificación al calcular el área bajo la curva ROC. Mientras más cercano a 1 es el área mejor distingue entre clases el modelo, por lo que nuestro valor está obteniendo buenos resultados en general.

```
library(MLmetrics)
```

```
##
## Attaching package: 'MLmetrics'
```

```
## The following object is masked from 'package:base':
##
```

```
##     Recall
```

```
AUCROC <- AUC(y_pred = predicciones_original, y_true=data_test$booking.status)
print(AUCROC)
```

```
## [1] 0.7900893
```

```
#usamos esta libreria para graficarla
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

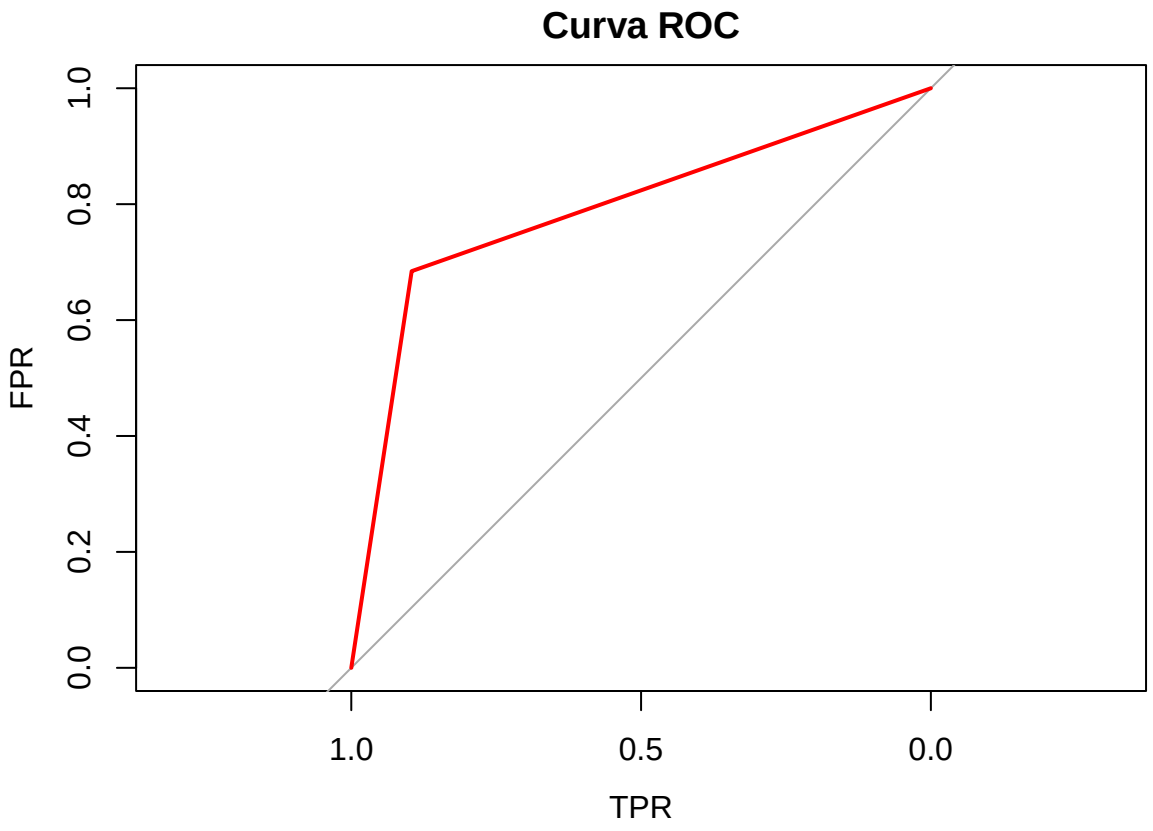
```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
# Calcula la curva ROC
roc_curve <- roc(data_test$booking.status, as.numeric(predicciones_original))
```

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

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

```
# Grafica la curva ROC
plot(roc_curve, main="Curva ROC", col="red",xlab="TPR",ylab="FPR")
```



Ejercicio 5: Optimización del modelo

En el proceso de optimización de nuestro modelo, nos enfocamos en ajustar tres hiperparámetros clave: max_depth, min_split y min_bucket. Cada uno de estos hiperparámetros juega un papel muy importante en la configuración del modelo y su rendimiento final:

max_depth: Este parámetro define la profundidad máxima del árbol de decisión. Un valor mayor permite que el modelo capture más complejidad en los datos, pero también puede llevar a un sobreajuste (overfitting). Optimizamos este parámetro dentro de un rango de 1 a 30 para encontrar el equilibrio adecuado entre precisión y generalización.

min_split: Es el número mínimo de observaciones requeridas para dividir un nodo en el árbol. Este parámetro controla el crecimiento del árbol y ayuda a evitar la creación de nodos con un número insuficiente de observaciones, lo que podría resultar en una alta varianza y sobreajuste. Probamos diferentes valores para encontrar el punto óptimo que balancee el modelo.

min_bucket: Este parámetro determina el tamaño mínimo de los "buckets" o grupos en los que se dividen las observaciones en las hojas del árbol. Un valor adecuado evita la creación de hojas con muy pocas observaciones, que podrían resultar en una alta varianza. Ajustamos este parámetro probando varios valores, desde 1 hasta el número total de observaciones, para asegurar una búsqueda equilibrada.

Dado el amplio rango de valores que pueden tener min_bucket y min_split tuvimos que probar varias opciones para encontrar valores que balancearan una buena búsqueda pero no demoraran demasiado en términos de tiempos de ejecución.

Inicialmente, decidimos investigar cómo cada hiperparámetro individualmente afectaba el rendimiento del modelo mediante la graficación de la curva ROC. Esto nos permitió visualizar cómo variaba la capacidad del modelo para clasificar correctamente las instancias en función de cada parámetro, proporcionando una comprensión más clara del impacto de cada ajuste.

Variación del maxdepth

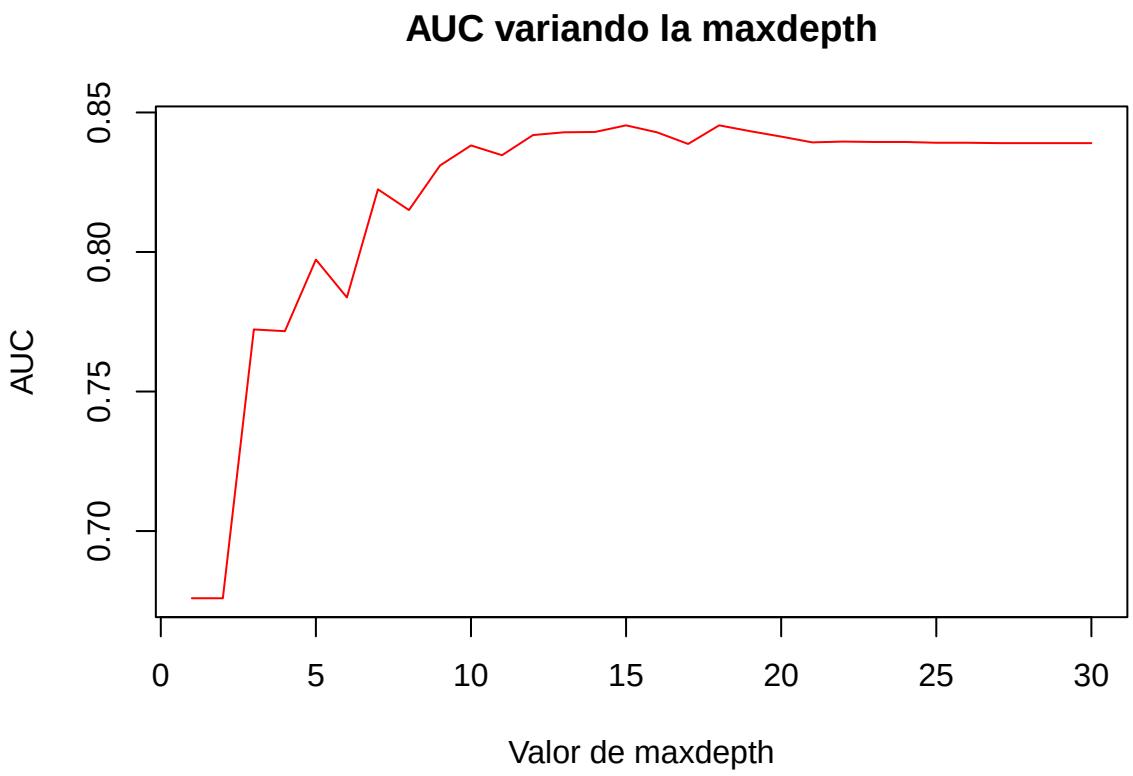
```
# Crear un vector para almacenar los AUC
auc_values_depth <- numeric(30)
# Secuencia de 1 a 29
depths <- 1:30

for (depth in depths) {
  arbol_md <- rpart(booking.status ~ number.of.adults + number.of.children + number.of.weekend.nights +
                cp = 0, xval = 0, data = data_entrenamiento, method = "class", maxdepth = depth)

  predicciones_md <- predict(arbol_md, newdata = data_test, type = "class")
  valorcito <- AUC(y_pred = predicciones_md, y_true = data_test$booking.status)

  # Guardar el valor de AUC en el vector
  auc_values_depth[depth] <- valorcito

  # Imprimir el resultado
  print(paste("El AUC asociado a la depth", depth, "es", valorcito))
}
```

```
## [1] "El AUC asociado a la depth 1 es 0.675924217414776"
## [1] "El AUC asociado a la depth 2 es 0.675924217414776"
## [1] "El AUC asociado a la depth 3 es 0.772243622593067"
## [1] "El AUC asociado a la depth 4 es 0.77161851050374"
## [1] "El AUC asociado a la depth 5 es 0.797278022027208"
## [1] "El AUC asociado a la depth 6 es 0.783706878151252"
```

```
## [1] "El AUC asociado a la depth 7 es 0.822445177852458"
## [1] "El AUC asociado a la depth 8 es 0.815027642864603"
## [1] "El AUC asociado a la depth 9 es 0.830968686238491"
## [1] "El AUC asociado a la depth 10 es 0.838176429589938"
## [1] "El AUC asociado a la depth 11 es 0.834678633620069"
## [1] "El AUC asociado a la depth 12 es 0.841892086105308"
## [1] "El AUC asociado a la depth 13 es 0.842884866410837"
## [1] "El AUC asociado a la depth 14 es 0.843009021040366"
## [1] "El AUC asociado a la depth 15 es 0.845374657718399"
## [1] "El AUC asociado a la depth 16 es 0.842867739009461"
## [1] "El AUC asociado a la depth 17 es 0.838714915089195"
## [1] "El AUC asociado a la depth 18 es 0.845378463807593"
## [1] "El AUC asociado a la depth 19 es 0.843287779012981"
## [1] "El AUC asociado a la depth 20 es 0.841353600606051"
## [1] "El AUC asociado a la depth 21 es 0.839280043212814"
## [1] "El AUC asociado a la depth 22 es 0.839558801185429"
## [1] "El AUC asociado a la depth 23 es 0.839419422199122"
## [1] "El AUC asociado a la depth 24 es 0.839419422199122"
## [1] "El AUC asociado a la depth 25 es 0.839144470315701"
## [1] "El AUC asociado a la depth 26 es 0.839144470315701"
## [1] "El AUC asociado a la depth 27 es 0.839006994373991"
## [1] "El AUC asociado a la depth 28 es 0.839006994373991"
## [1] "El AUC asociado a la depth 29 es 0.839006994373991"
## [1] "El AUC asociado a la depth 30 es 0.839006994373991"
```

Graficamos el maxdepht

```
profundidades <- seq(1, 30, by = 1)

par(mar = c(5, 5, 4, 2) + 0.1)
plot(profundidades, auc_values_depth, type = "l", col = "red",
     xlab = "Valor de maxdepth", ylab = "AUC", main = "AUC variando la maxdepth")
```



Notamos que al aumentar el valor de max_depth del árbol de decisión, la AUC-ROC también tiende a incrementarse. Esto se debe a que un mayor max_depth permite que el árbol desarrolle más nodos, capturando mejor las complejidades de los datos. Al permitir divisiones más profundas, el modelo puede segmentar los datos en grupos más específicos y aprender patrones más detallados, lo que mejora su capacidad para distinguir entre clases. Esta mayor discriminación se refleja en una AUC-ROC más alta, indicando un mejor rendimiento del modelo en términos de clasificación.

Sin embargo, aumentar la profundidad del árbol también puede llevar a que el modelo se ajuste demasiado a los datos de entrenamiento, un problema conocido como sobreajuste (overfitting).Además, un árbol más profundo aumenta la complejidad computacional, requiriendo más tiempo y recursos para su entrenamiento, lo cual puede ser ineficiente en conjuntos de datos grandes.

Variación del minsplit

```
auc_values_split <- numeric(70)
index <- 1

split <- 1
while (split <= 34000) {
  arbol_ms <- rpart(booking.status ~ number.of.adults + number.of.children + number.of.weekend.nights +
                cp = 0, xval = 0, data = data_entrenamiento, method = "class", minsplit = split)

  predicciones_ms <- predict(arbol_ms, newdata = data_test, type = "class")
  valor <- AUC(y_pred = predicciones_ms, y_true = data_test$booking.status)

  auc_values_split[index] <- valor

  print(paste("El AUC asociado a la minsplit", split, "es", valor))

  split <- split + 500
```

```
  index <- index + 1
}
```

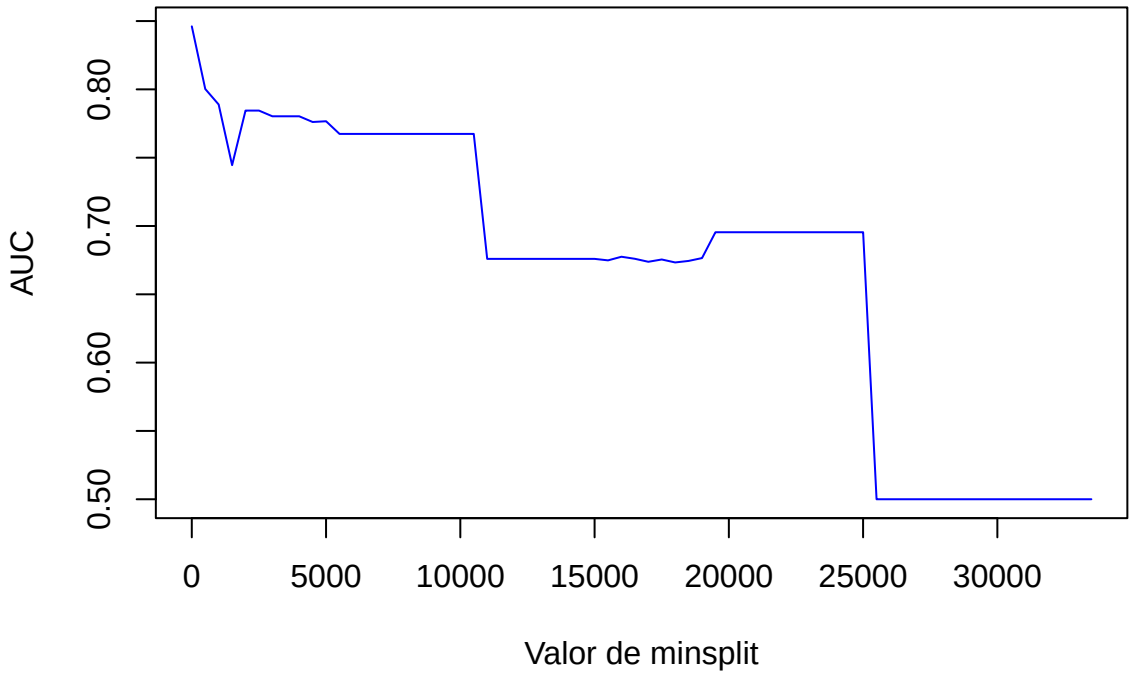
```
## [1] "El AUC asociado a la minsplit 1 es 0.846122934854064"
## [1] "El AUC asociado a la minsplit 501 es 0.800289171432652"
## [1] "El AUC asociado a la minsplit 1001 es 0.788930507205536"
## [1] "El AUC asociado a la minsplit 1501 es 0.744508117779278"
## [1] "El AUC asociado a la minsplit 2001 es 0.784475632046784"
## [1] "El AUC asociado a la minsplit 2501 es 0.784475632046784"
## [1] "El AUC asociado a la minsplit 3001 es 0.78029761181605"
## [1] "El AUC asociado a la minsplit 3501 es 0.78029761181605"
## [1] "El AUC asociado a la minsplit 4001 es 0.78029761181605"
## [1] "El AUC asociado a la minsplit 4501 es 0.776123854405213"
## [1] "El AUC asociado a la minsplit 5001 es 0.776654727726081"
## [1] "El AUC asociado a la minsplit 5501 es 0.767385834832171"
## [1] "El AUC asociado a la minsplit 6001 es 0.767385834832171"
## [1] "El AUC asociado a la minsplit 6501 es 0.767385834832171"
## [1] "El AUC asociado a la minsplit 7001 es 0.767385834832171"
## [1] "El AUC asociado a la minsplit 7501 es 0.767385834832171"
## [1] "El AUC asociado a la minsplit 8001 es 0.767385834832171"
## [1] "El AUC asociado a la minsplit 8501 es 0.767385834832171"
## [1] "El AUC asociado a la minsplit 9001 es 0.767385834832171"
## [1] "El AUC asociado a la minsplit 9501 es 0.767385834832171"
## [1] "El AUC asociado a la minsplit 10001 es 0.767385834832171"
## [1] "El AUC asociado a la minsplit 10501 es 0.767385834832171"
## [1] "El AUC asociado a la minsplit 11001 es 0.675924217414776"
## [1] "El AUC asociado a la minsplit 11501 es 0.675924217414776"
## [1] "El AUC asociado a la minsplit 12001 es 0.675924217414776"
## [1] "El AUC asociado a la minsplit 12501 es 0.675924217414776"
## [1] "El AUC asociado a la minsplit 13001 es 0.675924217414776"
## [1] "El AUC asociado a la minsplit 13501 es 0.675924217414776"
## [1] "El AUC asociado a la minsplit 14001 es 0.675924217414776"
## [1] "El AUC asociado a la minsplit 14501 es 0.675924217414776"
## [1] "El AUC asociado a la minsplit 15001 es 0.675924217414776"
## [1] "El AUC asociado a la minsplit 15501 es 0.674847246416262"
## [1] "El AUC asociado a la minsplit 16001 es 0.677493544111508"
## [1] "El AUC asociado a la minsplit 16501 es 0.67598511484189"
## [1] "El AUC asociado a la minsplit 17001 es 0.673802627175903"
## [1] "El AUC asociado a la minsplit 17501 es 0.675490399368372"
## [1] "El AUC asociado a la minsplit 18001 es 0.673307911702385"
## [1] "El AUC asociado a la minsplit 18501 es 0.674422943592845"
## [1] "El AUC asociado a la minsplit 19001 es 0.67651172534286"
## [1] "El AUC asociado a la minsplit 19501 es 0.69543065290263"
## [1] "El AUC asociado a la minsplit 20001 es 0.69543065290263"
## [1] "El AUC asociado a la minsplit 20501 es 0.69543065290263"
## [1] "El AUC asociado a la minsplit 21001 es 0.69543065290263"
## [1] "El AUC asociado a la minsplit 21501 es 0.69543065290263"
## [1] "El AUC asociado a la minsplit 22001 es 0.69543065290263"
## [1] "El AUC asociado a la minsplit 22501 es 0.69543065290263"
## [1] "El AUC asociado a la minsplit 23001 es 0.69543065290263"
## [1] "El AUC asociado a la minsplit 23501 es 0.69543065290263"
## [1] "El AUC asociado a la minsplit 24001 es 0.69543065290263"
## [1] "El AUC asociado a la minsplit 24501 es 0.69543065290263"
## [1] "El AUC asociado a la minsplit 25001 es 0.69543065290263"
```

```
## [1] "El AUC asociado a la minsplit 25501 es 0.5"
## [1] "El AUC asociado a la minsplit 26001 es 0.5"
## [1] "El AUC asociado a la minsplit 26501 es 0.5"
## [1] "El AUC asociado a la minsplit 27001 es 0.5"
## [1] "El AUC asociado a la minsplit 27501 es 0.5"
## [1] "El AUC asociado a la minsplit 28001 es 0.5"
## [1] "El AUC asociado a la minsplit 28501 es 0.5"
## [1] "El AUC asociado a la minsplit 29001 es 0.5"
## [1] "El AUC asociado a la minsplit 29501 es 0.5"
## [1] "El AUC asociado a la minsplit 30001 es 0.5"
## [1] "El AUC asociado a la minsplit 30501 es 0.5"
## [1] "El AUC asociado a la minsplit 31001 es 0.5"
## [1] "El AUC asociado a la minsplit 31501 es 0.5"
## [1] "El AUC asociado a la minsplit 32001 es 0.5"
## [1] "El AUC asociado a la minsplit 32501 es 0.5"
## [1] "El AUC asociado a la minsplit 33001 es 0.5"
## [1] "El AUC asociado a la minsplit 33501 es 0.5"
```

```
# Ajustar el vector de AUC para eliminar elementos no utilizados
auc_values_split <- auc_values_split[1:(index-1)]
```

Graficamos la variacion en minsplit

```
minsplit_values <- seq(1, 34000, by = 500)[1:(index-1)]

# Configurar los márgenes de la gráfica
par(mar = c(5, 5, 4, 2) + 0.1)

# Graficar AUC vs. Minsplit
plot(minsplit_values, auc_values_split, type = "l", col = "blue",
     xlab = "Valor de minsplit", ylab = "AUC",
     main = "AUC variando minsplit")
```



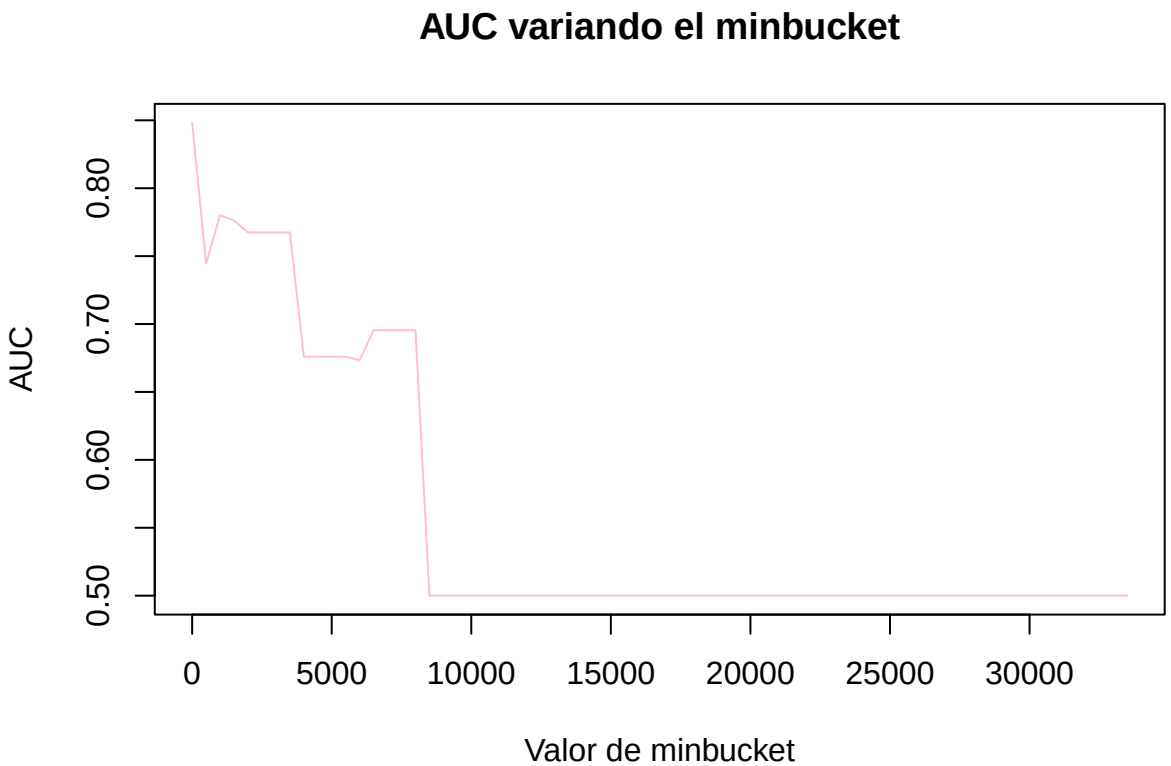

```r
#Ahora variamos el minbucket

auc_values_minbucket <- numeric(70)
indices <- 1

bucketminimo <- 1
while (bucketminimo <= 34000) {
  arbol_mb <- rpart(booking.status ~ number.of.adults + number.of.children + number.of.weekend.nights +
                cp = 0, xval = 0, data = data_entrenamiento, method = "class", minbucket=bucketminimo)

  predicciones_mb <- predict(arbol_mb, newdata = data_test, type = "class")
  valor <- AUC(y_pred = predicciones_mb, y_true = data_test$booking.status)

  auc_values_minbucket[indices] <- valor

  print(paste("El AUC asociado al minbucket", bucketminimo, "es", valor))

  bucketminimo <- bucketminimo + 500
  indices <- indices + 1
}
```

```
## [1] "El AUC asociado al minbucket 1 es 0.848181267890522"
## [1] "El AUC asociado al minbucket 501 es 0.744508117779278"
## [1] "El AUC asociado al minbucket 1001 es 0.78029761181605"
## [1] "El AUC asociado al minbucket 1501 es 0.776123854405213"
## [1] "El AUC asociado al minbucket 2001 es 0.767385834832171"
## [1] "El AUC asociado al minbucket 2501 es 0.767385834832171"
## [1] "El AUC asociado al minbucket 3001 es 0.767385834832171"
## [1] "El AUC asociado al minbucket 3501 es 0.767385834832171"
## [1] "El AUC asociado al minbucket 4001 es 0.675924217414776"
```

```
## [1] "El AUC asociado al minbucket 4501 es 0.675924217414776"
## [1] "El AUC asociado al minbucket 5001 es 0.675924217414776"
## [1] "El AUC asociado al minbucket 5501 es 0.67598511484189"
## [1] "El AUC asociado al minbucket 6001 es 0.673307911702385"
## [1] "El AUC asociado al minbucket 6501 es 0.69543065290263"
## [1] "El AUC asociado al minbucket 7001 es 0.69543065290263"
## [1] "El AUC asociado al minbucket 7501 es 0.69543065290263"
## [1] "El AUC asociado al minbucket 8001 es 0.69543065290263"
## [1] "El AUC asociado al minbucket 8501 es 0.5"
## [1] "El AUC asociado al minbucket 9001 es 0.5"
## [1] "El AUC asociado al minbucket 9501 es 0.5"
## [1] "El AUC asociado al minbucket 10001 es 0.5"
## [1] "El AUC asociado al minbucket 10501 es 0.5"
## [1] "El AUC asociado al minbucket 11001 es 0.5"
## [1] "El AUC asociado al minbucket 11501 es 0.5"
## [1] "El AUC asociado al minbucket 12001 es 0.5"
## [1] "El AUC asociado al minbucket 12501 es 0.5"
## [1] "El AUC asociado al minbucket 13001 es 0.5"
## [1] "El AUC asociado al minbucket 13501 es 0.5"
## [1] "El AUC asociado al minbucket 14001 es 0.5"
## [1] "El AUC asociado al minbucket 14501 es 0.5"
## [1] "El AUC asociado al minbucket 15001 es 0.5"
## [1] "El AUC asociado al minbucket 15501 es 0.5"
## [1] "El AUC asociado al minbucket 16001 es 0.5"
## [1] "El AUC asociado al minbucket 16501 es 0.5"
## [1] "El AUC asociado al minbucket 17001 es 0.5"
## [1] "El AUC asociado al minbucket 17501 es 0.5"
## [1] "El AUC asociado al minbucket 18001 es 0.5"
## [1] "El AUC asociado al minbucket 18501 es 0.5"
## [1] "El AUC asociado al minbucket 19001 es 0.5"
## [1] "El AUC asociado al minbucket 19501 es 0.5"
## [1] "El AUC asociado al minbucket 20001 es 0.5"
## [1] "El AUC asociado al minbucket 20501 es 0.5"
## [1] "El AUC asociado al minbucket 21001 es 0.5"
## [1] "El AUC asociado al minbucket 21501 es 0.5"
## [1] "El AUC asociado al minbucket 22001 es 0.5"
## [1] "El AUC asociado al minbucket 22501 es 0.5"
## [1] "El AUC asociado al minbucket 23001 es 0.5"
## [1] "El AUC asociado al minbucket 23501 es 0.5"
## [1] "El AUC asociado al minbucket 24001 es 0.5"
## [1] "El AUC asociado al minbucket 24501 es 0.5"
## [1] "El AUC asociado al minbucket 25001 es 0.5"
## [1] "El AUC asociado al minbucket 25501 es 0.5"
## [1] "El AUC asociado al minbucket 26001 es 0.5"
## [1] "El AUC asociado al minbucket 26501 es 0.5"
## [1] "El AUC asociado al minbucket 27001 es 0.5"
## [1] "El AUC asociado al minbucket 27501 es 0.5"
## [1] "El AUC asociado al minbucket 28001 es 0.5"
## [1] "El AUC asociado al minbucket 28501 es 0.5"
## [1] "El AUC asociado al minbucket 29001 es 0.5"
## [1] "El AUC asociado al minbucket 29501 es 0.5"
## [1] "El AUC asociado al minbucket 30001 es 0.5"
## [1] "El AUC asociado al minbucket 30501 es 0.5"
## [1] "El AUC asociado al minbucket 31001 es 0.5"
```

```
## [1] "El AUC asociado al minbucket 31501 es 0.5"
## [1] "El AUC asociado al minbucket 32001 es 0.5"
## [1] "El AUC asociado al minbucket 32501 es 0.5"
## [1] "El AUC asociado al minbucket 33001 es 0.5"
## [1] "El AUC asociado al minbucket 33501 es 0.5"
```

```r
# Ajustar el vector de AUC para eliminar elementos no utilizados
auc_values_minbucket<- auc_values_minbucket[1:(indices-1)]

# Crear un vector de splits correspondientes
buckets <- seq(1, bucketminimo - 500, by = 500)

# Graficar
plot(buckets, auc_values_minbucket, type = "l",col = "pink",
     xlab = "Valor de minbucket", ylab = "AUC", main = "AUC variando el minbucket")
```



Por otro lado, al ajustar los hiperparámetros min_bucket y min_split, observamos un comportamiento diferente. A medida que aumentamos estos valores, la AUC-ROC tiende a disminuir. Esto ocurre porque valores más altos de min_bucket y min_split hacen que el árbol sea más restrictivo en su capacidad de dividirse y crecer. Un árbol con menos divisiones tiende a ser más simple, lo que puede resultar en un modelo que no captura completamente la complejidad de los datos, reduciendo su capacidad de discriminación entre clases y, por tanto, disminuyendo la AUC-ROC.

Además. lado restringir el crecimiento del árbol puede ayudar a prevenir el sobreajuste, lo cual mejora la capacidad del modelo para generalizar en datos nuevos. Sin embargo, esta simplificación puede venir acompañada de una pérdida de precisión, ya que el modelo podría no captar suficientes patrones en los datos, llevando a un subajuste (underfitting).

A pesar de los conocimientos obtenidos, reconocimos que la mejor manera de encontrar una combinación óptima de los tres hiperparámetros era explorar todas las posibles combinaciones simultáneamente. Para ello, implementamos un enfoque de búsqueda más exhaustivo utilizando tres ciclos for anidados. Este método permitió evaluar una gran cantidad de combinaciones posibles de los valores para max_depth, min_split y

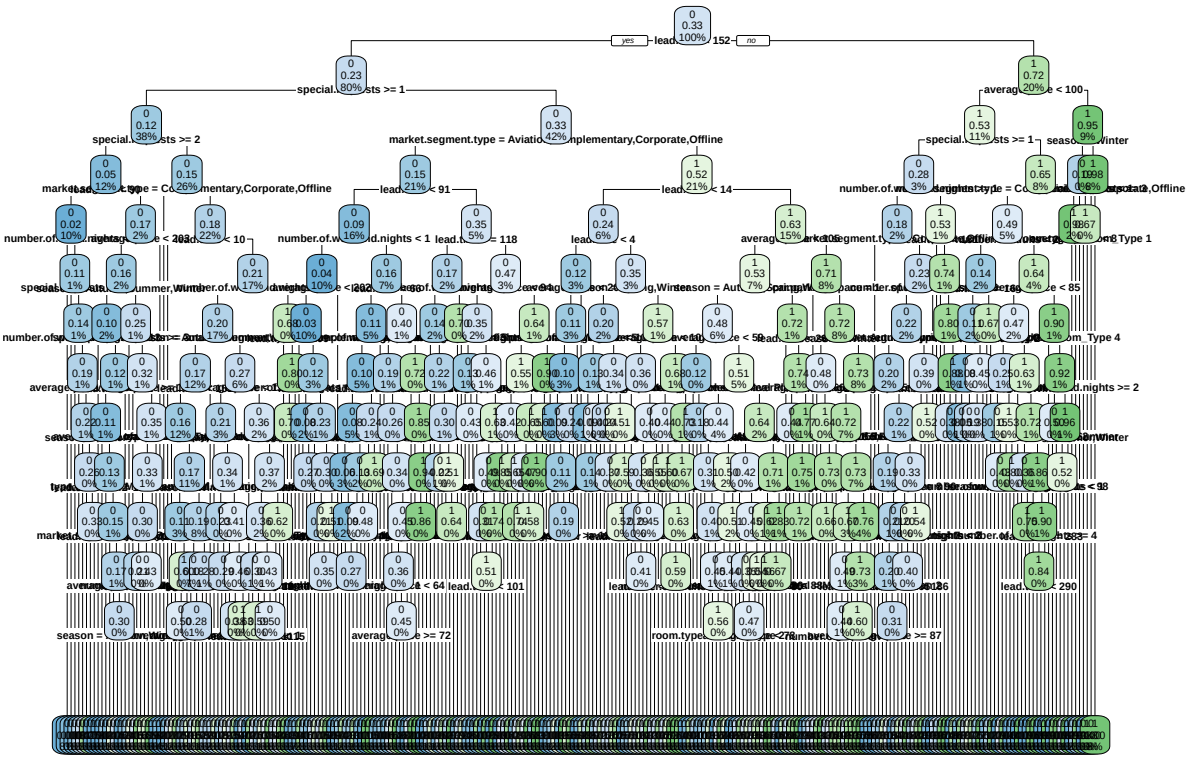
min_bucket, asegurando que encontráramos la configuración que ofreciera el mejor rendimiento general del modelo. En el análisis previo que hicimos de los hiperparámetros por separado pudimos ver que a partir de cierto umbral el valor del AUC-ROC se estanca. Por esto decidimos aplicarle al algoritmo una tolerancia de 0.001, que hace que se corte el ciclo del minsplit si el valor del área de la curva no varió mucho, y sigue analizando con el siguiente. Además, realizamos una función auxiliar que calcula el AUC-ROC de un arbol creado con un maxdepth, minsplit y minbucket pasados por parametro. Notamos con el analisis planteado anteriormente que variando uno de los hiperparametros y dejando los otros por defecto se llegaban a buenos resultados, por lo que quisimos considerarlos. Para eso inicializamos a los hiperparametros en 0, y cuando toman ese valor se los cambia al que deberian tener por defecto. Finalmente, al aplicar esta estrategia, logramos identificar los hiperparámetros óptimos.

Buscamos hiperparametros

```
calcular_auc <- function(data_validation,data_training, max_depth, min_split, min_bucket) {
  #Si alguno de los valores está en 0, entonces le pongo el valor del hiperparametro por defecto
  if(min_split == 0){
    min_split <- 20
  }
  if(min_bucket == 0){
    min_bucket <- 7
  }
  arbol <- rpart(booking.status ~ number.of.adults + number.of.children + number.of.weekend.nights + nu

  predicciones<-predict(arbol, newdata = data_validation, type = "class")
  auc <- AUC(y_pred = predicciones, y_true = data_validation$booking.status)

  return(c(auc,max_depth,min_split,min_bucket))
}


Optimizar_arbol <- function(data_validation,data_training, i, j, k) {

maxdepth1<-1
min_bucket_optimo<-0
max_depth_optimo<-0
min_split_optimo<-0
auc_experimentacion3<-0
#Estos son para que si se estanca podamos saber que tenemos que cortar
auc_anterior <-0
auc_actual <-0

while(maxdepth1<=30){
  minbucket1<-0
  while(minbucket1<=10000){
    minsplit1<-0
    while(minsplit1<=10000){
      auc_actual <- calcular_auc(data_validation,data_training, maxdepth1,minsplit1,minbucket1)
      hola = auc_actual[1]
      if(auc_experimentacion3<auc_actual[1]){
        auc_experimentacion3 <- auc_actual[1]
        max_depth_optimo <- auc_actual[2]
        min_split_optimo <- auc_actual[3]
        min_bucket_optimo <- auc_actual[4]
```

```
      }
      if(abs(auc_actual[1][1] - auc_anterior) <= 0.001){
        break
      }
      auc_anterior<-auc_actual[1]
      minsplit1<-minsplit1+k
      }
    minbucket1<-minbucket1+j
    }
  maxdepth1<-maxdepth1+i
}
  return(c(auc_experimentacion3, max_depth_optimo, min_split_optimo, min_bucket_optimo))
}

Valores_arbol_optimizado <- Optimizar_arbol(data_validacion, data_entrenamiento, 1, 1000, 1000)
auc_optimizado <- Valores_arbol_optimizado[1]
md_optimizado <- Valores_arbol_optimizado[2]
ms_optimizado <- Valores_arbol_optimizado[3]
mb_optimizado <- Valores_arbol_optimizado[4]

arbol_optimizado <- rpart(booking.status ~ number.of.adults + number.of.children + number.of.weekend.nig
rpart.plot(arbol_optimizado,cex = 0.3)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```
print(auc_optimizado)
```

```
## [1] 0.8424775
```

```r
predicciones_optimizadas<-predict(arbol_optimizado, newdata = data_test, type = "class")
auc <- AUC(y_pred = predicciones_optimizadas, y_true = data_test$booking.status)
print(auc)
```

```
## [1] 0.8428849
```

Como podemos observar, el area bajo la curva aumenta por arriba de 0.80 indicando que se logró optimizar el arbol basico planteado en el ejercicio 3.

Para comparar el rendimiento en el conjunto de testeo de de ambos arboles, volvimos a calcular las metricas del ejercicio 4 para el arbol optimizado.

```r
print(paste("maxdepth optimizado = ", md_optimizado))
```

```
## [1] "maxdepth optimizado =  13"
```

```r
print(paste("minsplit optimizado = ", ms_optimizado))
```

```
## [1] "minsplit optimizado =  20"
```

```r
print(paste("minbucket optimizado = ", mb_optimizado))
```

```
## [1] "minbucket optimizado =  7"
```

```r
confusion_matrix_optimizada <- table(Predicted = predicciones_optimizadas, Actual = data_test$booking.s
print(confusion_matrix_optimizada)
```

```
##          Actual
## Predicted    0    1
##         0 3346  423
##         1  291 1383
```

```r
accuracy_optimizada <- sum(diag(confusion_matrix_optimizada)) / sum(confusion_matrix_optimizada)
print(accuracy)
```

```
## [1] 0.8256476
```

```r
# PRECISIÓN = TP/(TP+FP)
TP_opt <-confusion_matrix_optimizada[2,2]
FP_opt <-confusion_matrix_optimizada[2,1]
PRECISION_opt <- TP_opt/(TP_opt+FP_opt)
print(PRECISION_opt)
```

```
## [1] 0.8261649
```

```r
# RECALL = TP/(TP+FN)
FN_opt <-confusion_matrix_optimizada[1,2]
RECALL_opt <- TP_opt/(TP_opt+FN_opt)
print(RECALL_opt)
```

```
## [1] 0.7657807
```

```r
F1SCORE_optimizado <- (2*PRECISION_opt*RECALL_opt)/(PRECISION_opt+RECALL_opt)
print(F1SCORE_optimizado)
```

```
## [1] 0.7948276
```

```r
AUCROC_optimizada <- AUC(y_pred = predicciones_optimizadas, y_true=data_test$booking.status)
print(AUCROC_optimizada)
```

```
## [1] 0.8428849
```

Notamos que en el nuevo árbol optimizado, observamos un aumento en todas las principales métricas de performance. Estas mejoras indican que el modelo es ahora más eficaz para clasificar correctamente las instancias positivas y negativas, reduciendo tanto los falsos positivos como los falsos negativos. Además, el aumento de la AUC-ROC demuestra una mayor capacidad del modelo para distinguir entre las diferentes clases.

Ejercicio 6

En un principio, cabe destacar que el arbol optimizado presenta una bastante mayor profundidad que el original, imposibilitando un análisis exhaustivo del mismo. Sin embargo por lo poco que podemos observar las variables principales del arbol basico se mantienen en el optimizado. Dicha profundidad puede provocar un mayor riesgo de Overfitting. A pesar de ello, todas las metricas de performance indican una mejora en la predicción de datos nuevos, por lo que Se logró la optimización deseada.

A diferencia del arbol basico, este presenta 12 niveles en vez de 5.

Un aspecto interesante que se observa en la comparación entre el árbol de decisión optimizado y el árbol básico es el cambio en el orden de importancia de las variables. En el modelo optimizado, no solo se ha modificado la jerarquía de las variables, sino que también se han incorporado nuevas variables que anteriormente no eran consideradas por el modelo básico. Entre estas nuevas variables se destacan **number.of.week.nights**, **number.of.weekend.nights** y **car.parking.space**, que ahora juegan un papel relevante en el proceso de toma de decisiones del modelo optimizado. Para poder visualizar y comparar la importancia de las variables en cada uno de los árboles, se ejecutaron los siguientes comandos, que permitieron analizar en detalle cómo ha evolucionado el peso de cada variable en la estructura del modelo optimizado respecto al modelo básico.

```
print("importancia de las variables predictoras para el árbol original:")
```

```
## [1] "importancia de las variables predictoras para el árbol original:"
```

```
arbol$variable.importance
```

```
##                lead.time      market.segment.type            average.price
##              2250.784378              1109.262962               751.533118
##         special.requests         number.of.adults             type.of.meal
##               637.194011               366.680024               357.616644
##               room.type                   season       number.of.children
##               286.422770               151.686779                58.451508
##    number.of.week.nights number.of.weekend.nights        car.parking.space
##                12.375528                 5.481202                 2.522631
##                      P.C                   P.not.C
##                 2.028641                 1.099655
```

```
print("importancia de las variables predictoras para el árbol optimizado:")
```

```
## [1] "importancia de las variables predictoras para el árbol optimizado:"
```

```
arbol_optimizado$variable.importance
```

```
##                lead.time            average.price      market.segment.type
##              3138.49956               1653.44980               1244.34642
##         special.requests                   season             type.of.meal
##               802.24164                694.60480                487.81202
##         number.of.adults                room.type number.of.weekend.nights
##               473.87730                409.36721                326.89322
##    number.of.week.nights       number.of.children        car.parking.space
##               276.62952                 75.96372                 64.03296
##                repeated                   P.not.C                      P.C
##                20.32450                 19.69668                 15.29358
```

Ejercicio 7: Impacto de valores faltantes

Primero que nada, para observar que sucedía en las predicciones de un dataset de entrenamiento con valores de variables predictoras faltantes para las observaciones, creamos dataframes con estas características. Para lograr la eliminación de los valores, en este ejercicio, decidimos hacer una función que llenara con NA aleatoriamente el pocentaje de los datos pasados por parámetro.

```
#Creamos una función que elimina el porcentaje pasado por parámetro de cada uno de los datos.
eliminar_porciento <- function(muestras, porcentaje) {
  data <- muestras
  # Cuerpo de la función
  num_filas <- nrow(data)
  num_na <- round(num_filas * porcentaje)

  variables = c("number.of.adults","number.of.children", "number.of.weekend.nights", "number.of.week.nig
  for (variable in variables){
    indices <- sample(1:num_filas, num_na)
    data[indices, variable] <- NA
  }
  return(data)
}

data_entrenamiento_na_20 <- eliminar_porciento(data_entrenamiento,0.2)
data_test_na_20 <- eliminar_porciento(data_test,0.2)
data_validacion_na_20 <- eliminar_porciento(data_validacion,0.2)

data_entrenamiento_na_50 <- eliminar_porciento(data_entrenamiento,0.5)
data_test_na_50 <- eliminar_porciento(data_test,0.5)
data_validacion_na_50 <- eliminar_porciento(data_validacion,0.5)

data_entrenamiento_na_75 <- eliminar_porciento(data_entrenamiento,0.75)
data_test_na_75 <- eliminar_porciento(data_test,0.75)
data_validacion_na_75 <- eliminar_porciento(data_validacion,0.75)
```

Generamos los árboles en base a nuestros nuevos datos de entrenamiento. Primero generamos el árbol para los datasets con 20% de NA en sus columnas.

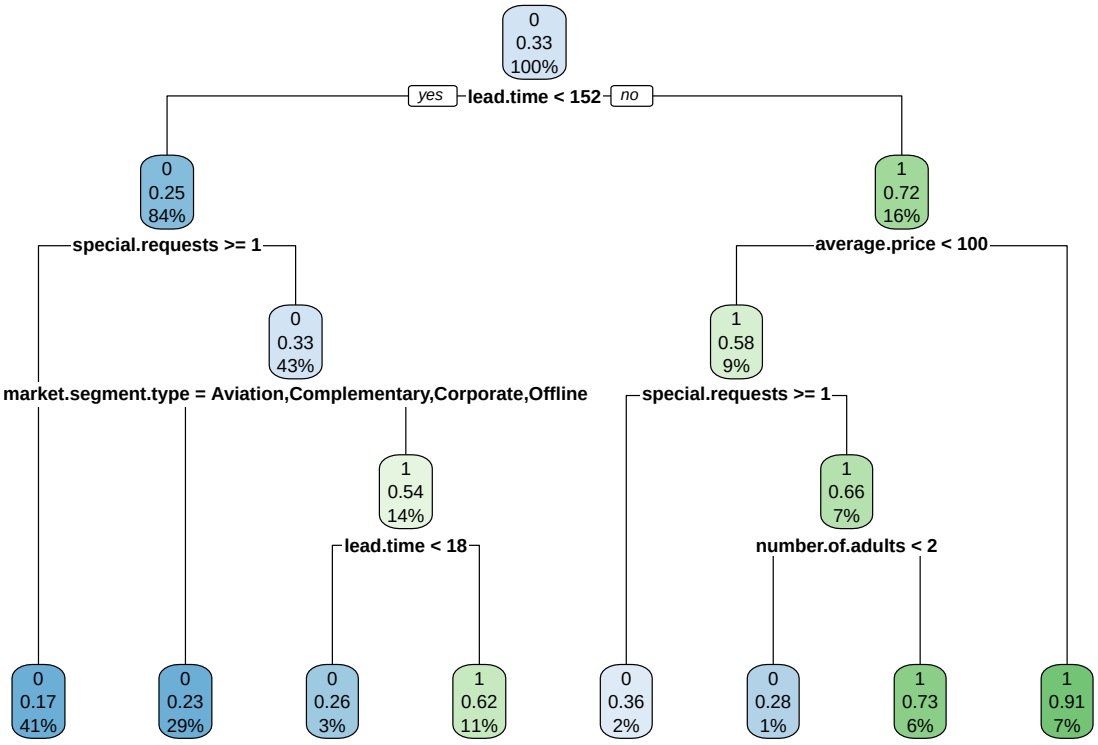
```
#Revisar si no hay que ponerle xval y cp 0
arbol_20 <- rpart(booking.status ~ number.of.adults + number.of.children + number.of.weekend.nights + nu



#Vemos el auc sobre el conjunto de validación con 20% de NA.
predicciones_validacion_na_20 <-predict(arbol_20, newdata = data_validacion_na_20, type = "class")
auc_original_validacion_na_20 <- AUC(y_pred = predicciones_validacion_na_20, y_true = data_validacion_na
print(paste("El AUC para el set de validación con 20% de NA es", auc_original_validacion_na_20))
```

```
## [1] "El AUC para el set de validación con 20% de NA es 0.703294473618856"
```

```
#Vemos el auc sobre el conjunto de test con 20% de NA.
predicciones_test_na_20 <-predict(arbol_20, newdata = data_test_na_20, type = "class")
auc_original_test_na_20 <- AUC(y_pred = predicciones_test_na_20, y_true = data_test_na_20$booking.status
print(paste("El AUC para el set de test con 20% de NA es", auc_original_test_na_20))
```

```
## [1] "El AUC para el set de test con 20% de NA es 0.714154937060987"
```

```
rpart.plot(arbol_20)
```



Luego generamos el árbol para los datasets con 50% de NA en sus columnas.

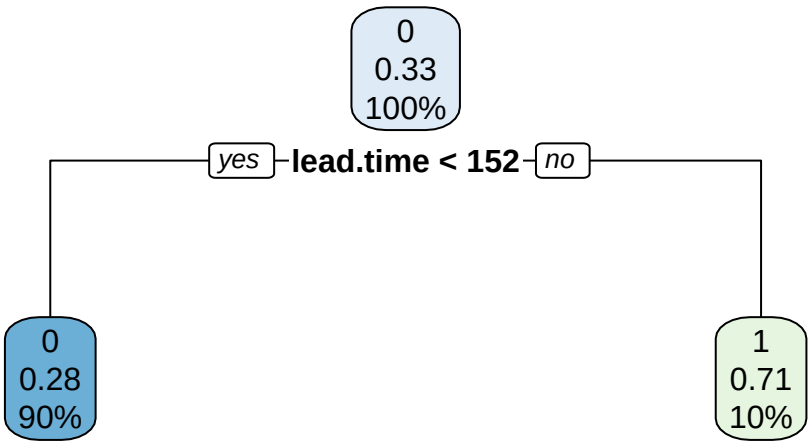
```
arbol_50 <- rpart(booking.status ~ number.of.adults + number.of.children + number.of.weekend.nights + nu

#Vemos el auc sobre el conjunto de validación con 50% de NA.
predicciones_validacion_na_50 <-predict(arbol_50, newdata = data_validacion_na_50, type = "class")
auc_original_validacion_na_50 <- AUC(y_pred = predicciones_validacion_na_50, y_true = data_validacion_na
print(paste("El AUC para el set de validación con 50% de NA es", auc_original_validacion_na_50))
```

```
## [1] "El AUC para el set de validación con 50% de NA es 0.582861441326279"
```

```
#Vemos el auc sobre el conjunto de test con 50% de NA.
predicciones_test_na_50 <-predict(arbol_50, newdata = data_test_na_50, type = "class")
auc_original_test_na_50 <- AUC(y_pred = predicciones_test_na_50, y_true = data_test_na_50$booking.status
print(paste("El AUC para el set de test con 50% de NA es", auc_original_test_na_50))
```

```
## [1] "El AUC para el set de test con 50% de NA es 0.584033577623362"
```

```
rpart.plot(arbol_50)
```



Por último, hacemos el árbol para el conjunto con 75% de NA

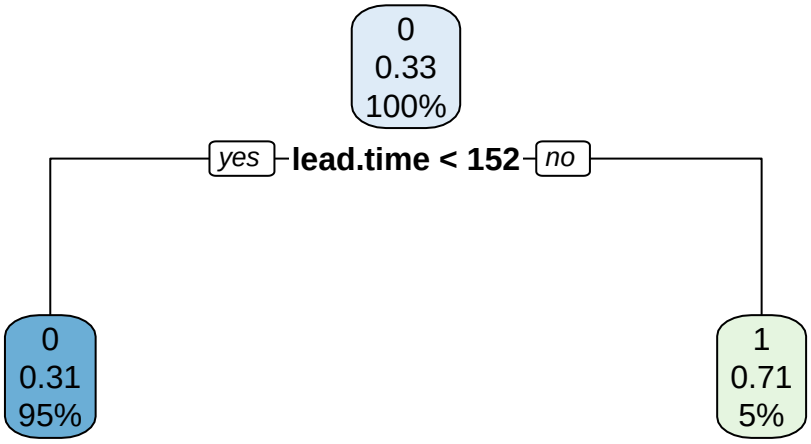
```
arbol_75 <- rpart(booking.status ~ number.of.adults + number.of.children + number.of.weekend.nights + nu

#Vemos el auc sobre el conjunto de validación con 75% de NA.
predicciones_validacion_na_75 <-predict(arbol_75, newdata = data_validacion_na_75, type = "class")
auc_original_validacion_na_75 <- AUC(y_pred = predicciones_validacion_na_75, y_true = data_validacion_na
print(paste("El AUC para el set de validación con 75% de NA es", auc_original_validacion_na_75))
```

```
## [1] "El AUC para el set de validación con 75% de NA es 0.544941506915532"
```

```
#Vemos el auc sobre el conjunto de test con 75% de NA.
predicciones_test_na_75 <-predict(arbol_75, newdata = data_test_na_75, type = "class")
auc_original_test_na_75 <- AUC(y_pred = predicciones_test_na_75, y_true = data_test_na_75$booking.status
print(paste("El AUC para el set de test con 75% de NA es", auc_original_test_na_75))
```

```
## [1] "El AUC para el set de test con 75% de NA es 0.542421604458422"
```

```
rpart.plot(arbol_75)
```



Por un lado, pudimos notar que a medida que fuimos eliminando más porcentaje de valores, el árbol fue haciendose más chico, sospechamos que se debe a que tiene menos valores de cada variable, por lo que se le dificulta encontrar relaciones entre el resultado de la reserva y cada una de las columnas predictoras del archivo. A su vez, quisimos ver qué tanto afectaba a las predicciones del modelo, por lo que imprimimos los valores de AUC-ROC para el conjunto de testeo de cada uno de los modelos, pudimos ver que a medida que eliminamos más datos, el valor disminuye, así afectando el rendimiento de cada uno. Así llegando a valer hasta casi 0.5 en aquellos conjuntos con un 50% y 75% de los valores variables predictoras eliminados, esto

quiere decir que se está prediciendo casi al azar.

```
print(paste("el AUC-ROC para el árbol original (con 100% de los datos) es:",AUCROC))
```

```
## [1] "el AUC-ROC para el árbol original (con 100% de los datos) es: 0.790089309121734"
```

```
print(paste("el AUC-ROC para el árbol con 20% de los datos NA es:",auc_original_test_na_20))
```

```
## [1] "el AUC-ROC para el árbol con 20% de los datos NA es: 0.714154937060987"
```

```
print(paste("el AUC-ROC para el árbol con 50% de los datos NA es:",auc_original_test_na_50))
```

```
## [1] "el AUC-ROC para el árbol con 50% de los datos NA es: 0.584033577623362"
```

```
print(paste("el AUC-ROC para el árbol con 75% de los datos NA es:",auc_original_test_na_75))
```

```
## [1] "el AUC-ROC para el árbol con 75% de los datos NA es: 0.542421604458422"
```

Ahora optimizamos los árboles generados con los na.
Primero optimizamos el árbol del 20%

```
valores_arbol_optimizado_NA20 <- Optimizar_arbol(data_validacion_na_20, data_entrenamiento_na_20,1,1000
auc_optimizado_na20 <- valores_arbol_optimizado_NA20[1]
md_optimizado_na20 <- valores_arbol_optimizado_NA20[2]
ms_optimizado_na20 <- valores_arbol_optimizado_NA20[3]
mb_optimizado_na20 <- valores_arbol_optimizado_NA20[4]


arbol_optimizado_NA20 <- rpart(booking.status ~ number.of.adults + number.of.children + number.of.weeke

print(paste("El AUC para el set de validación con 20% de NA optimizado es", auc_optimizado_na20))
```

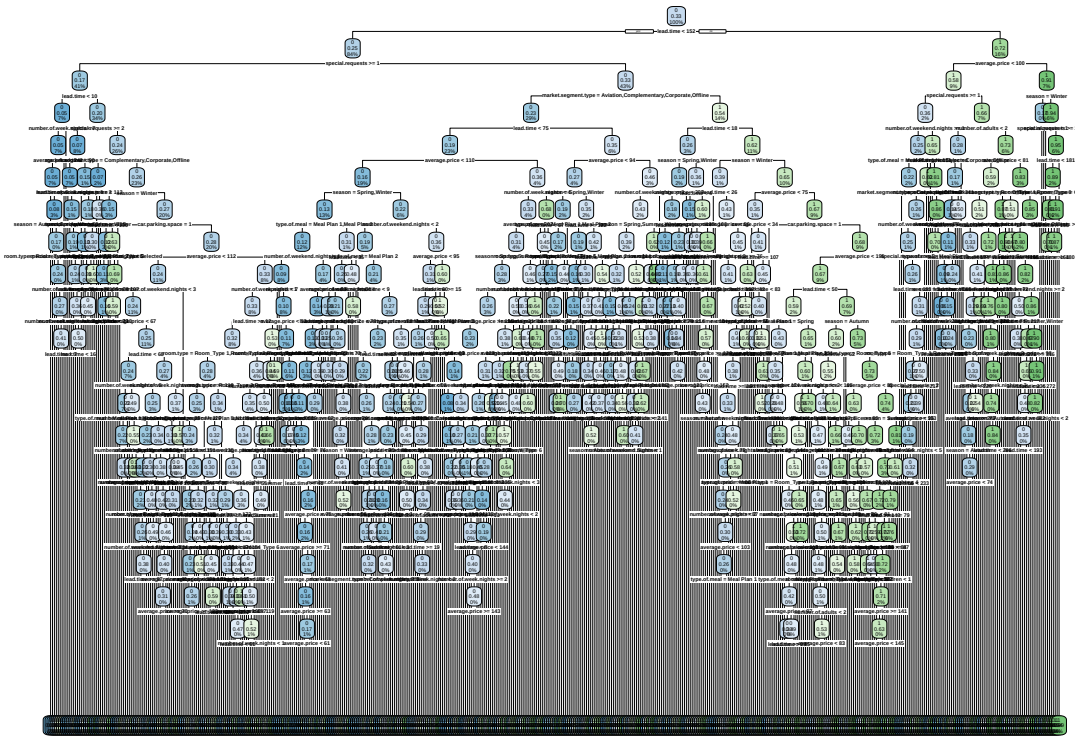
```
## [1] "El AUC para el set de validación con 20% de NA optimizado es 0.746342699867502"
```

```
predicciones_optimizadas_test_na_20 <- predict(arbol_optimizado_NA20, newdata = data_test_na_20, type =
auc_optimizado_test_na_20 <- AUC(y_pred = predicciones_optimizadas_test_na_20, y_true = data_test_na_20
print(paste("El AUC para el set de test con 20% de NA optimizado es", auc_optimizado_test_na_20))
```

```
## [1] "El AUC para el set de test con 20% de NA optimizado es 0.760036276597332"
```

```
rpart.plot(arbol_optimizado_NA20)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Graficamos el auc optimizado y el original

```
# Instalar pROC si no lo tienes ya
# install.packages("pROC")

# Cargar la librería
library(pROC)
# Curva ROC para el modelo optimizado
roc_optimizado <- roc(data_test_na_20$booking.status, as.numeric(predicciones_optimizadas_test_na_20))
```

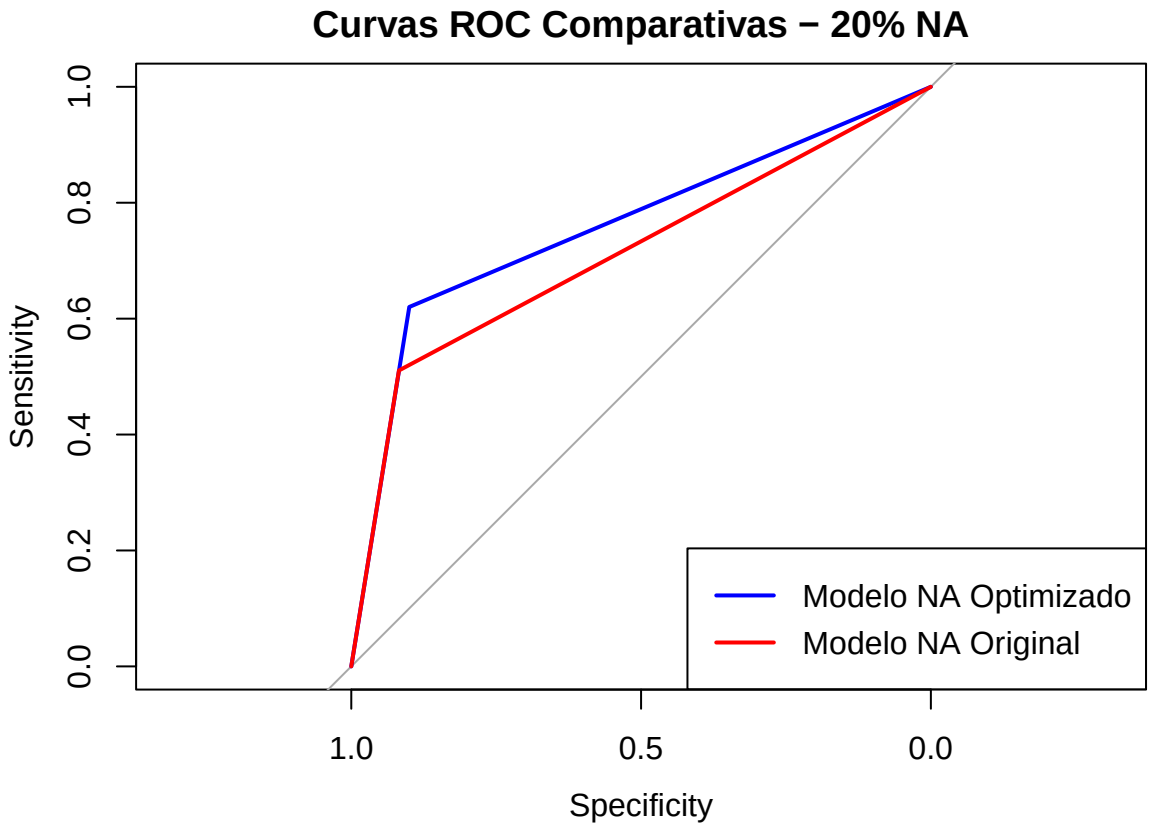
```
## Setting levels: control = 0, case = 1
```

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

```
# Curva ROC para el modelo original
roc_original <- roc(data_test_na_20$booking.status, as.numeric(predicciones_test_na_20))
```

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

```
# Graficar la curva ROC del modelo optimizado
plot(roc_optimizado, col = "blue", main = "Curvas ROC Comparativas - 20% NA", lwd = 2)

# Añadir la curva ROC del modelo original
lines(roc_original, col = "red", lwd = 2)

# Añadir una leyenda para identificar las curvas
legend("bottomright", legend = c("Modelo NA Optimizado", "Modelo NA Original"),
       col = c("blue", "red"), lwd = 2)
```



Hacemos lo mismo con el 50% de NA

```
valores_arbol_optimizado_NA50 <- Optimizar_arbol(data_validacion_na_50, data_entrenamiento_na_50,1,1000
auc_optimizado_na50 <- valores_arbol_optimizado_NA50[1]
md_optimizado_na50 <- valores_arbol_optimizado_NA50[2]
ms_optimizado_na50 <- valores_arbol_optimizado_NA50[3]
mb_optimizado_na50 <- valores_arbol_optimizado_NA50[4]

arbol_optimizado_NA50 <- rpart(booking.status ~ number.of.adults + number.of.children + number.of.weeke

print(paste("El AUC para el set de validación con 50% de NA optimizado es", auc_optimizado_na50))
```

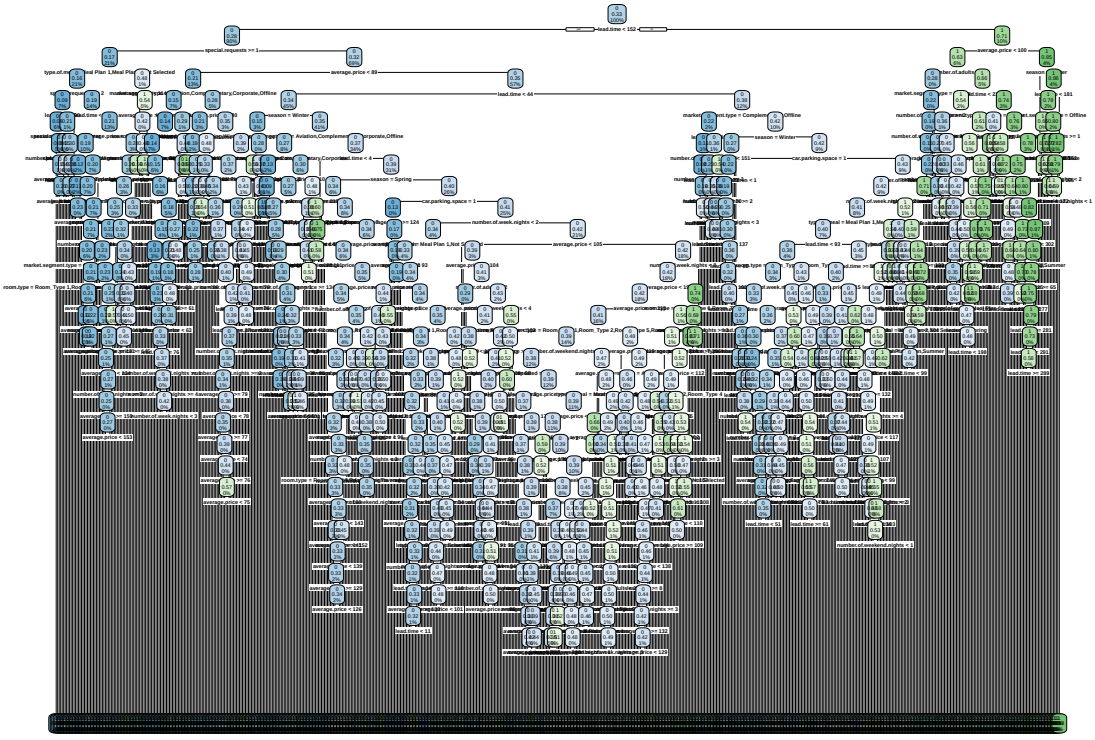
```
## [1] "El AUC para el set de validación con 50% de NA optimizado es 0.63632860240142"
```

```
predicciones_optimizadas_test_na_50 <- predict(arbol_optimizado_NA50, newdata = data_test_na_50, type =
auc_optimizado_test_na_50 <- AUC(y_pred = predicciones_optimizadas_test_na_50, y_true = data_test_na_50
print(paste("El AUC para el set de test con 50% de NA optimizado es", auc_optimizado_test_na_50))
```

```
## [1] "El AUC para el set de test con 50% de NA optimizado es 0.637614559478669"
```

```
rpart.plot(arbol_optimizado_NA50)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Graficamos la curva para el original y el optimizado

```
# Curva ROC para el modelo optimizado
roc_optimizado <- roc(data_test_na_50$booking.status, as.numeric(predicciones_optimizadas_test_na_50))
```

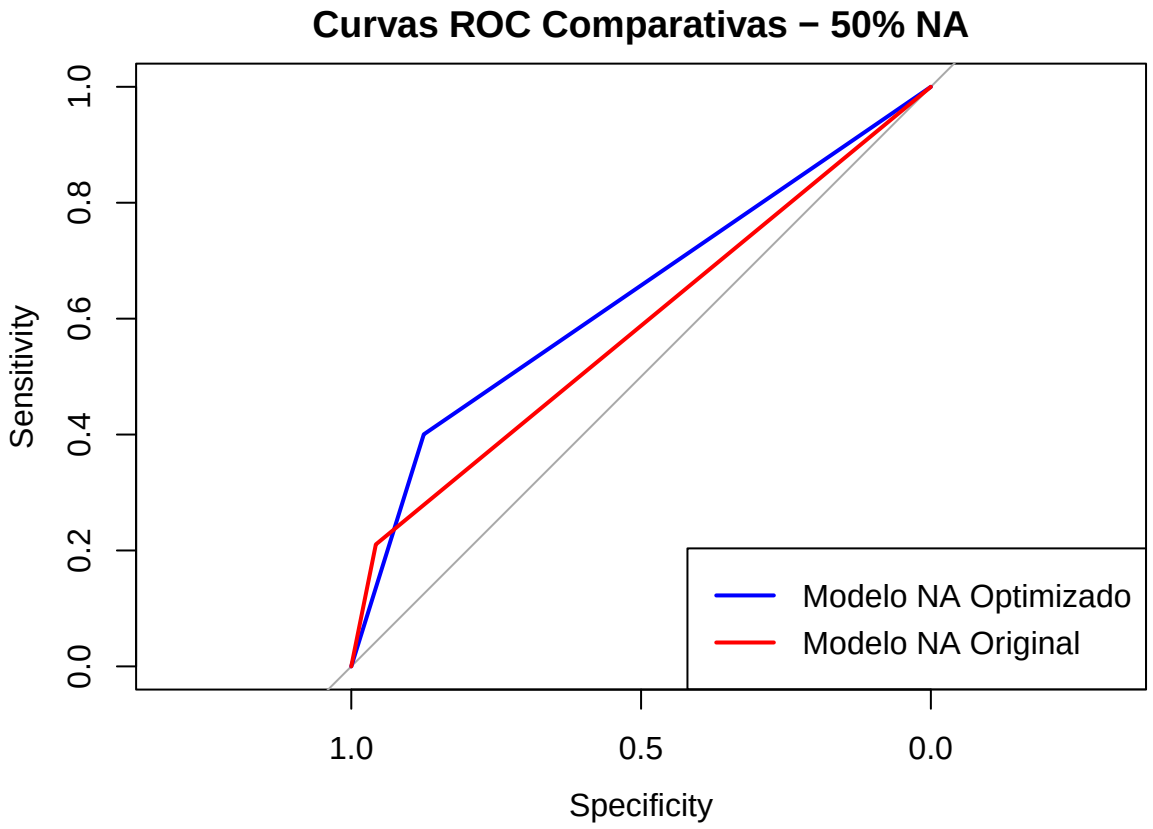
```
## Setting levels: control = 0, case = 1
```

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

```
# Curva ROC para el modelo original
roc_original <- roc(data_test_na_50$booking.status, as.numeric(predicciones_test_na_50))
```

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

```
# Graficar la curva ROC del modelo optimizado
plot(roc_optimizado, col = "blue", main = "Curvas ROC Comparativas - 50% NA", lwd = 2)

# Añadir la curva ROC del modelo original
lines(roc_original, col = "red", lwd = 2)

# Añadir una leyenda para identificar las curvas
legend("bottomright", legend = c("Modelo NA Optimizado", "Modelo NA Original"),
       col = c("blue", "red"), lwd = 2)
```



Por último, optimizamos el que tiene el 75% de los valores eliminados.

```
valores_arbol_optimizado_NA75 <- Optimizar_arbol(data_validacion_na_75, data_entrenamiento_na_75,1,1000
auc_optimizado_na75 <- valores_arbol_optimizado_NA75[1]
md_optimizado_na75 <- valores_arbol_optimizado_NA75[2]
ms_optimizado_na75 <- valores_arbol_optimizado_NA75[3]
mb_optimizado_na75 <- valores_arbol_optimizado_NA75[4]

arbol_optimizado_NA75 <- rpart(booking.status ~ number.of.adults + number.of.children + number.of.weeke

print(paste("El AUC para el set de validación con 75% de NA optimizado es", auc_optimizado_na75))
```

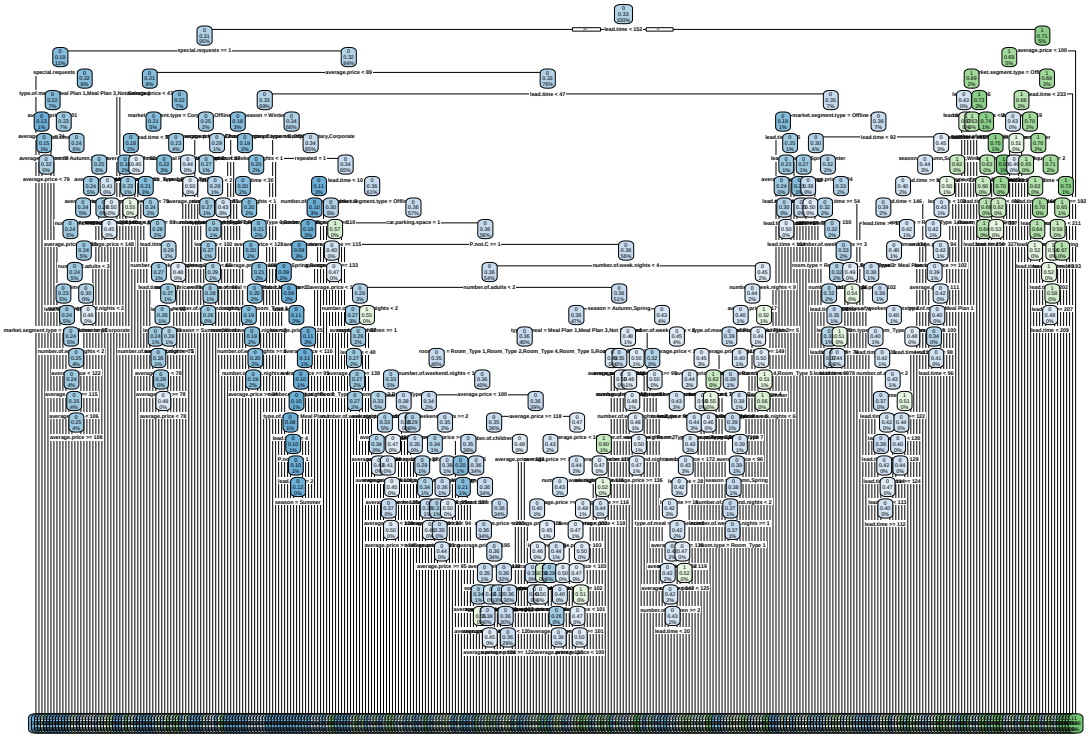
```
## [1] "El AUC para el set de validación con 75% de NA optimizado es 0.565107172416769"
```

```
predicciones_optimizadas_test_na_75 <- predict(arbol_optimizado_NA75, newdata = data_test_na_75, type =
auc_optimizado_test_na_75 <- AUC(y_pred = predicciones_optimizadas_test_na_75, y_true = data_test_na_75
print(paste("El AUC para el set de test con 75% de NA optimizado es", auc_optimizado_test_na_75))
```

```
## [1] "El AUC para el set de test con 75% de NA optimizado es 0.56542865242215"
```

```
rpart.plot(arbol_optimizado_NA75)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Graficamos el auc para el optimizado y el original

```
# Curva ROC para el modelo optimizado
roc_optimizado <- roc(data_test_na_75$booking.status, as.numeric(predicciones_optimizadas_test_na_75))
```

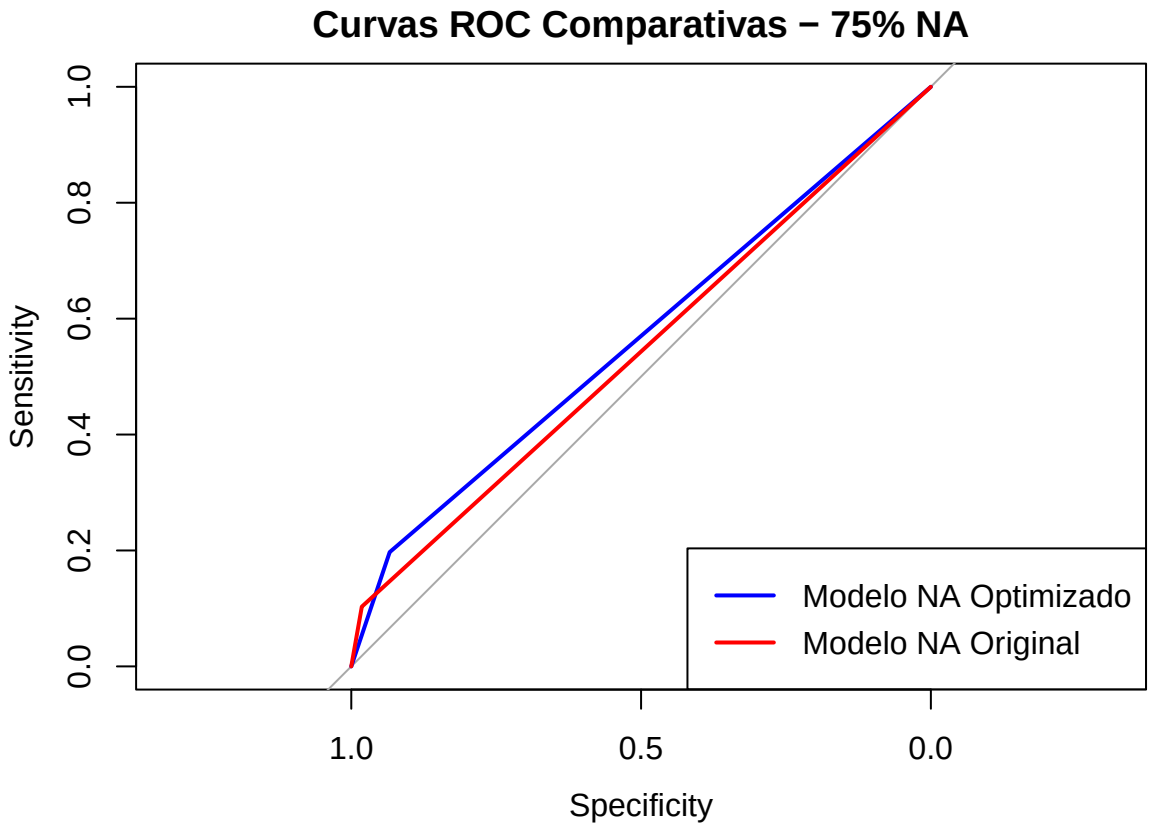
```
## Setting levels: control = 0, case = 1
```

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

```
# Curva ROC para el modelo original
roc_original <- roc(data_test_na_75$booking.status, as.numeric(predicciones_test_na_75))
```

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

```
# Graficar la curva ROC del modelo optimizado
plot(roc_optimizado, col = "blue", main = "Curvas ROC Comparativas - 75% NA", lwd = 2)

# Añadir la curva ROC del modelo original
lines(roc_original, col = "red", lwd = 2)

# Añadir una leyenda para identificar las curvas
legend("bottomright", legend = c("Modelo NA Optimizado", "Modelo NA Original"),
       col = c("blue", "red"), lwd = 2)
```



Comparamos los rendimientos del AUC-ROC de los distintos árboles y sus optimizaciones: Pudimos observar que, en todos los casos la optimización funciona, subiendo el valor del área debajo de la curva de AUC aunque sea un poco. Sin embargo, tampoco puede lograr un valor muy elevado con aquellos modelos cuyo entrenamiento tienen muchos valores predictores faltantes.

```
print(paste("el AUC-ROC para el árbol original:", AUCROC))
```

```
## [1] "el AUC-ROC para el árbol original: 0.790089309121734"
```

```
print(paste("el AUC-ROC para el árbol original optimizado:",auc_optimizado))
```

```
## [1] "el AUC-ROC para el árbol original optimizado: 0.842477533645784"
```

```
print(paste("el AUC-ROC para el árbol con 20% de los datos NA es:",auc_original_test_na_20))
```

```
## [1] "el AUC-ROC para el árbol con 20% de los datos NA es: 0.714154937060987"
```

```
print(paste("el AUC-ROC para el árbol optimizado con 20% de los datos NA es:",auc_optimizado_test_na_20
```

```
## [1] "el AUC-ROC para el árbol optimizado con 20% de los datos NA es: 0.760036276597332"
```

```
print(paste("el AUC-ROC para el árbol con 50% de los datos NA es:",auc_original_test_na_50))
```

```
## [1] "el AUC-ROC para el árbol con 50% de los datos NA es: 0.584033577623362"
```

```
print(paste("el AUC-ROC para el árbol optimizado con 50% de los datos NA es:",auc_optimizado_test_na_50
```

```
## [1] "el AUC-ROC para el árbol optimizado con 50% de los datos NA es: 0.637614559478669"
```

```
print(paste("el AUC-ROC para el árbol con 75% de los datos NA es:",auc_original_test_na_75))
```

```
## [1] "el AUC-ROC para el árbol con 75% de los datos NA es: 0.542421604458422"
```

```r
print(paste("el AUC-ROC para el árbol optimizado con 75% de los datos NA es:",auc_optimizado_test_na_75)
```

```
## [1] "el AUC-ROC para el árbol optimizado con 75% de los datos NA es: 0.56542865242215"
```

Graficamos las 4 curvas para los resultados originales y los optimizados.

```r
# Calcular las curvas ROC
roc_original <- roc(data_test$booking.status, as.numeric(predicciones_optimizadas))
```

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

```r
roc_original_na_20 <- roc(data_test_na_20$booking.status, as.numeric(predicciones_test_na_20))
```

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

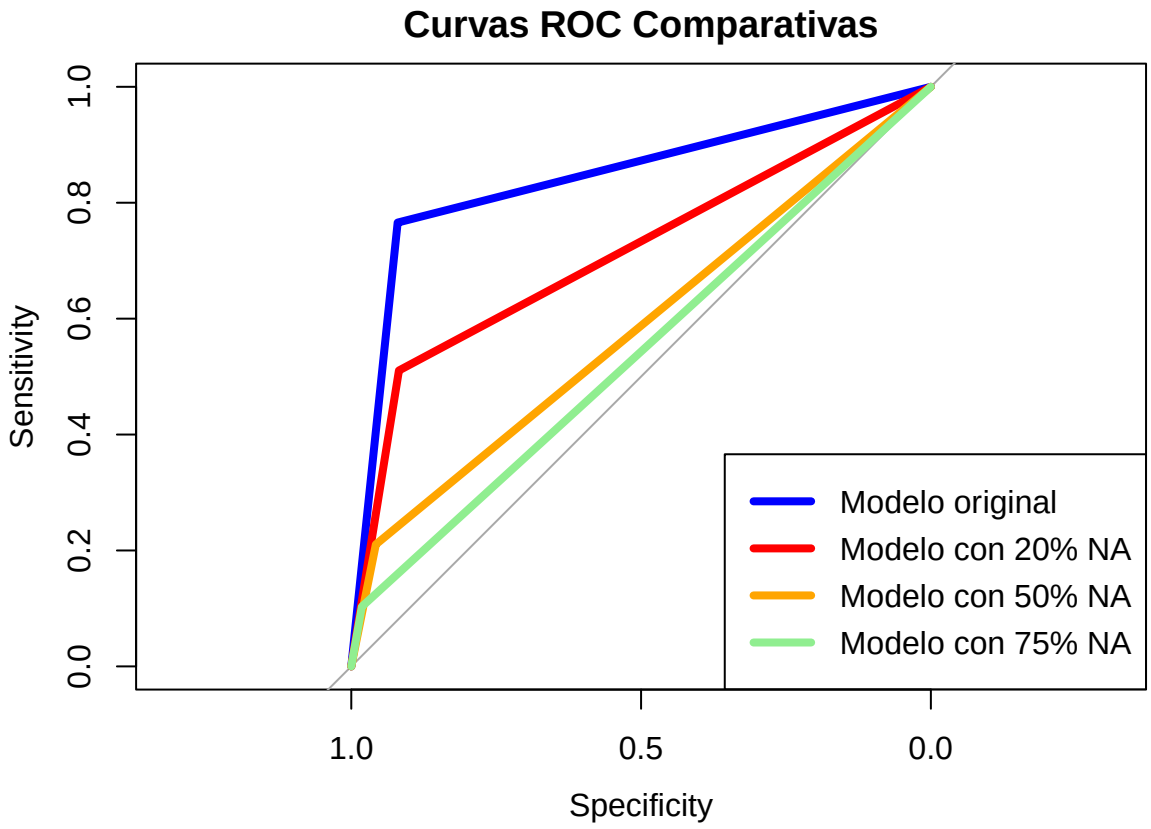
```r
roc_original_na_50 <- roc(data_test_na_50$booking.status, as.numeric(predicciones_test_na_50))
```

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

```r
roc_original_na_75 <- roc(data_test_na_75$booking.status, as.numeric(predicciones_test_na_75))
```

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

```r
# Graficar la curva ROC del modelo original
plot(roc_original, col = "blue", main = "Curvas ROC Comparativas", lwd = 4)

# Añadir la curva ROC del modelo con 20% de NA
lines(roc_original_na_20, col = "red", lwd = 4)

# Añadir la curva ROC del modelo con 50% de NA
lines(roc_original_na_50, col = "orange", lwd = 4)

# Añadir la curva ROC del modelo con 75% de NA
lines(roc_original_na_75, col = "lightgreen", lwd = 4)

# Añadir una leyenda para identificar las curvas
legend("bottomright", legend = c("Modelo original", "Modelo con 20% NA", "Modelo con 50% NA", "Modelo c
       col = c("blue", "red", "orange", "lightgreen"), lwd = 4)
```



Grafi-

camos las 4 curvas optimizadas:

```
# Calcular las curvas ROC
roc_original <- roc(data_test$booking.status, as.numeric(predicciones_optimizadas))
```

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

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

```
roc_original_na_20 <- roc(data_test_na_20$booking.status, as.numeric(predicciones_optimizadas_test_na_20
```

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

```
roc_original_na_50 <- roc(data_test_na_50$booking.status, as.numeric(predicciones_optimizadas_test_na_50
```

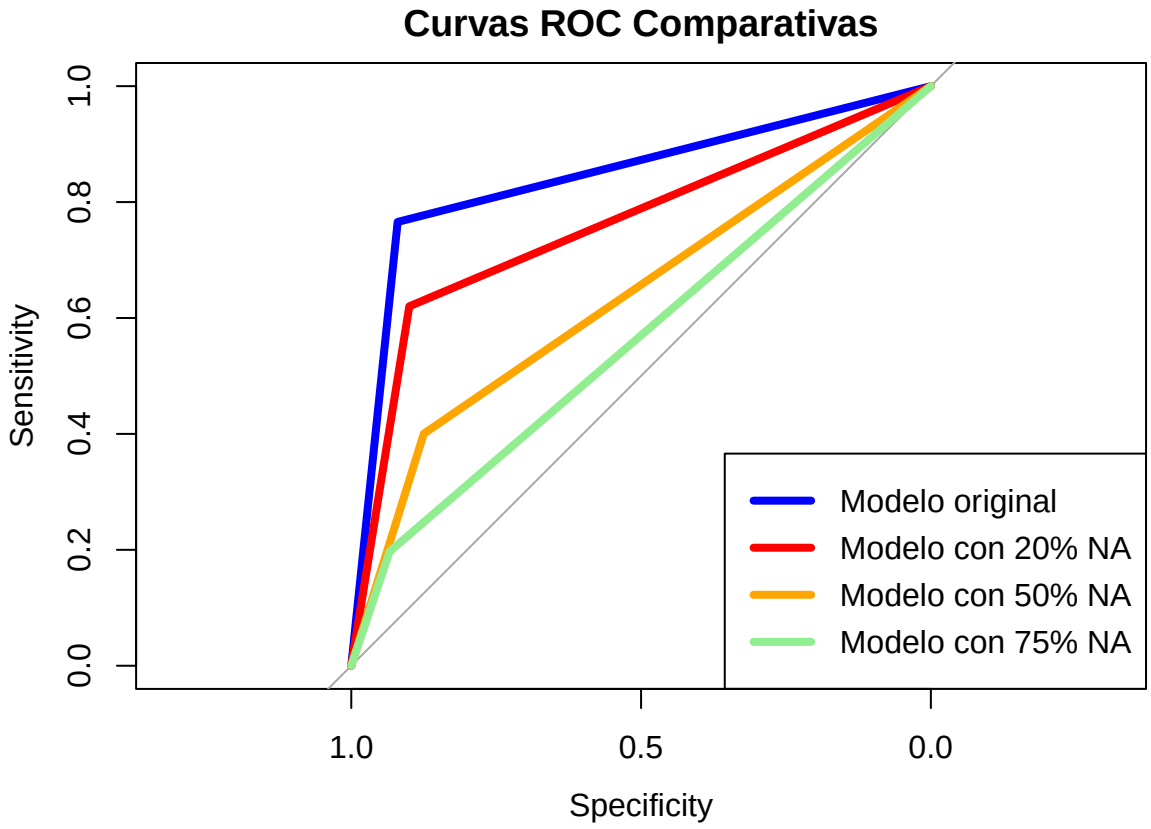
```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
roc_original_na_75 <- roc(data_test_na_75$booking.status, as.numeric(predicciones_optimizadas_test_na_75
```

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

```
# Graficar la curva ROC del modelo original
plot(roc_original, col = "blue", main = "Curvas ROC Comparativas", lwd = 4)

# Añadir la curva ROC del modelo con 20% de NA
lines(roc_original_na_20, col = "red", lwd = 4)

# Añadir la curva ROC del modelo con 50% de NA
lines(roc_original_na_50, col = "orange", lwd = 4)
```

```r
# Añadir la curva ROC del modelo con 75% de NA
lines(roc_original_na_75, col = "lightgreen", lwd = 4)

# Añadir una leyenda para identificar las curvas
legend("bottomright", legend = c("Modelo original", "Modelo con 20% NA", "Modelo con 50% NA", "Modelo c
       col = c("blue", "red", "orange", "lightgreen"), lwd = 4)
```



Ejercicio 8: Conclusión

En este trabajo práctico, uno de los mayores desafíos que enfrentamos fue encontrar una base de datos adecuada que permitiera construir un modelo con buenas métricas de predicción y que generara un árbol de decisión complejo y significativo. En varias ocasiones, probamos con distintos archivos CSV, pero muchos de ellos solo permitían la creación de árboles muy simples, con apenas una o dos hojas. Una posible razón para esto podría ser que esos conjuntos de datos contenían pocas variables predictoras relevantes, carecían de la variabilidad necesaria para segmentar los datos de manera más detallada o estaban desbalanceados.

Fuimos exigentes en la selección del archivo CSV porque queríamos asegurarnos de contar con buenas variables predictoras. El conjunto de datos que finalmente elegimos nos proporcionó resultados prometedores desde el principio: utilizando los hiperparámetros por defecto, obtuvimos un AUC-ROC de 0.79, un punto de partida sólido. Posteriormente, al optimizar los hiperparámetros, logramos mejorar esta métrica a 0.84, lo que refleja una mejora considerable en la capacidad del modelo para distinguir entre las clases.

Aunque el enunciado nos pedía que diferenciáramos los buenos modelos utilizando únicamente la métrica AUC-ROC, también decidimos analizar otras métricas para obtener una evaluación más completa del modelo. Concluimos que la AUC-ROC es una excelente métrica para medir la calidad del modelo, ya que refleja la interacción de todas las variables que afectan a las otras métricas clave: la matriz de confusión, el recall, la precisión, y el F1-score.

Durante el proceso de optimización, descubrimos que es difícil encontrar los hiperparámetros óptimos debido al amplio rango de valores que pueden tomar, especialmente para min_bucket y min_split. Para evitar un costo computacional excesivo, realizamos la búsqueda en incrementos de 1000, 500, o 100. Sin embargo, esta estrategia implica sacrificar la posibilidad de explorar valores intermedios que podrían mejorar el modelo.

Además, aprendimos sobre el impacto significativo que tienen los valores faltantes (NA) en la performance del modelo. A medida que aumenta la cantidad de datos faltantes, el modelo se ve afectado porque tiene menos información disponible para determinar qué variables son importantes para la clasificación. Esto dificulta la construcción de un buen modelo y afecta negativamente la precisión de las predicciones.

Como conclusión final, consideramos que nuestro árbol optimizado presenta numerosas fortalezas y predice con precisión en la mayoría de los casos. Sin embargo, reconocemos que aún hay espacio para seguir mejorando el modelo, ya sea mediante la inclusión de más datos, o la exploración de otros enfoques de optimización de hiperparámetros.