Untitled

2025-04-17

Introduzione

Librerie

```
library(ggplot2)
library(ggcorrplot)
library(scales)
library(randomForest)
library(caret)
library(dplyr)
library(gbm)
library(smotefamily)
```

Import del dataset e analisi preliminare

Attraverso il seguente codice è stato effettuato l'import del dataset scelto. Inoltre utilizzando la lista delle variabili categorieli è stato possibile convertirle in factor attraverso la funzione lapply. Vediamo in oltre un estratto del dataset in basso.

```
ds <- read.csv("StudentPerformanceFactors.csv")
ds = data.frame(ds)

# Lista di variabili categoriali
categorical_vars <- c(
  "Parental_Involvement", "Access_to_Resources", "Extracurricular_Activities",
  "Motivation_Level", "Internet_Access", "Family_Income", "Teacher_Quality",
  "School_Type", "Peer_Influence", "Learning_Disabilities",
  "Parental_Education_Level", "Distance_from_Home", "Gender"
)

ds[categorical_vars] <- lapply(ds[categorical_vars], factor)

head(ds)
```

```
##   Hours_Studied Attendance Parental_Involvement Access_to_Resources
## 1            23         84                  Low               High
## 2            19         64                  Low             Medium
## 3            24         98               Medium             Medium
## 4            29         89                  Low             Medium
## 5            19         92               Medium             Medium
## 6            19         88               Medium             Medium
##   Extracurricular_Activities Sleep_Hours Previous_Scores Motivation_Level
## 1                         No           7              73              Low
## 2                         No           8              59              Low
## 3                        Yes           7              91           Medium
```

```
## 4                       Yes           8              98           Medium
## 5                       Yes           6              65           Medium
## 6                       Yes           8              89           Medium
##   Internet_Access Tutoring_Sessions Family_Income Teacher_Quality School_Type
## 1             Yes                 0           Low          Medium      Public
## 2             Yes                 2        Medium          Medium      Public
## 3             Yes                 2        Medium          Medium      Public
## 4             Yes                 1        Medium          Medium      Public
## 5             Yes                 3        Medium            High      Public
## 6             Yes                 3        Medium          Medium      Public
##   Peer_Influence Physical_Activity Learning_Disabilities
## 1       Positive                 3                    No
## 2       Negative                 4                    No
## 3        Neutral                 4                    No
## 4       Negative                 4                    No
## 5        Neutral                 4                    No
## 6       Positive                 3                    No
##   Parental_Education_Level Distance_from_Home Gender Exam_Score
## 1              High School              Near   Male         67
## 2                  College          Moderate Female         61
## 3             Postgraduate              Near   Male         74
## 4              High School          Moderate   Male         71
## 5                  College              Near Female         70
## 6             Postgraduate              Near   Male         71
```

Descrizione delle variabili Prima di iniziale la trattazione è utili fare un breve riassunto su quello che sono le variabili presenti nel dataset e del loro significato. * **Hours_Studied** Numero di ore spese studiando a settimana. * **Attendance** Percentuale di lezioni frequentate. * **Parental_Involvement** Livello di coinvolgimento genitoriale nella formazione dello studente (Low, Medium, High). * **Access_to_Resources** Disponibilità di risorse educative(Low, Medium, High). * **Extracurricular_Activities** Partecipazione ad attività extracurriculari (Yes, No). * **Sleep_Hours** Numero medio di ore di sonno a notte. * **Previous_Scores** Punteggio degli esami precedenti. * **Motivation_Level** Livello di motivazione dello studente (Low, Medium, High). * **Internet_Access** Disponibilità di accesso ad Internet (Yes, No). * **Tutoring_Sessions** Numero di sessioni di tutoraggio frequentata al mese. * **Family_Income** Livello di reddito familiare (Low, Medium, High). * **Teacher_Quality** Qualità dell'insegnamento (Low, Medium, High). * **School_Type** Tipo di scuola frequentata (Public, Private). * **Peer_Influence** Influenza dei pari sulla performance accademica (Positive, Neutral, Negative). * **Physical_Activity** Numero medio di ore di attività fisica a settimana. * **Learning_Disabilities** Presenza di difficoltà di apprendimento (Yes, No). * **Parental_Education_Level** Livello più alto di educazione dei genitori (High School, College, Postgraduate). * **Distance_from_Home** Distanza da casa a scuola (Near, Moderate, Far). * **Gender** Genere dello studente (Male, Female). * **Exam_Score** Punteggio dell' esame finale.

Il dataset presenta sia variabili numeriche che categoriali, con valori ben distribuiti. Le ore di studio, la frequenza e le ore di sonno mostrano medie intorno a 20, 80 e 7 rispettivamente. La maggior parte degli studenti ha accesso a Internet e partecipa ad attività extracurricolari. Le categorie Parental Involvement, Motivation Level, e Family Income sono abbastanza bilanciate, mentre alcune categorie come Learning Disabilities e Gender mostrano distribuzioni sbilanciate. Il punteggio Exam_Score ha una media di circa 67, con valori compresi tra 55 e 101.

```
summary(ds)
```

```
##  Hours_Studied     Attendance     Parental_Involvement Access_to_Resources
##  Min.   : 1.00   Min.   : 60.00   High  :1908          High  :1975
##  1st Qu.:16.00   1st Qu.: 70.00   Low   :1337          Low   :1313
##  Median :20.00   Median : 80.00   Medium:3362          Medium:3319
```

```
##  Mean   :19.98   Mean   : 79.98
##  3rd Qu.:24.00   3rd Qu.: 90.00
##  Max.   :44.00   Max.   :100.00
##  Extracurricular_Activities  Sleep_Hours     Previous_Scores  Motivation_Level
##  No :2669                   Min.   : 4.000   Min.   : 50.00   High  :1319
##  Yes:3938                   1st Qu.: 6.000   1st Qu.: 63.00   Low   :1937
##                             Median : 7.000   Median : 75.00   Medium:3351
##                             Mean   : 7.029   Mean   : 75.07
##                             3rd Qu.: 8.000   3rd Qu.: 88.00
##                             Max.   :10.000   Max.   :100.00
##  Internet_Access Tutoring_Sessions Family_Income Teacher_Quality  School_Type
##  No : 499        Min.   :0.000     High  :1269         :  78     Private:2009
##  Yes:6108        1st Qu.:1.000     Low   :2672   High  :1947     Public :4598
##                  Median :1.000     Medium:2666   Low   : 657
##                  Mean   :1.494                   Medium:3925
##                  3rd Qu.:2.000
##                  Max.   :8.000
##   Peer_Influence Physical_Activity Learning_Disabilities
##  Negative:1377   Min.   :0.000     No :5912
##  Neutral :2592   1st Qu.:2.000     Yes: 695
##  Positive:2638   Median :3.000
##                  Mean   :2.968
##                  3rd Qu.:4.000
##                  Max.   :6.000
##  Parental_Education_Level Distance_from_Home    Gender       Exam_Score
##              :  90                :  67      Female:2793   Min.   : 55.00
##  College     :1989        Far     : 658      Male  :3814   1st Qu.: 65.00
##  High School :3223        Moderate:1998                    Median : 67.00
##  Postgraduate:1305        Near    :3884                    Mean   : 67.24
##                                                            3rd Qu.: 69.00
##                                                            Max.   :101.00
```

Vediamo il risultato della funzione str sul dataset, utile per comprendere i livelli delle variabili categoriche.

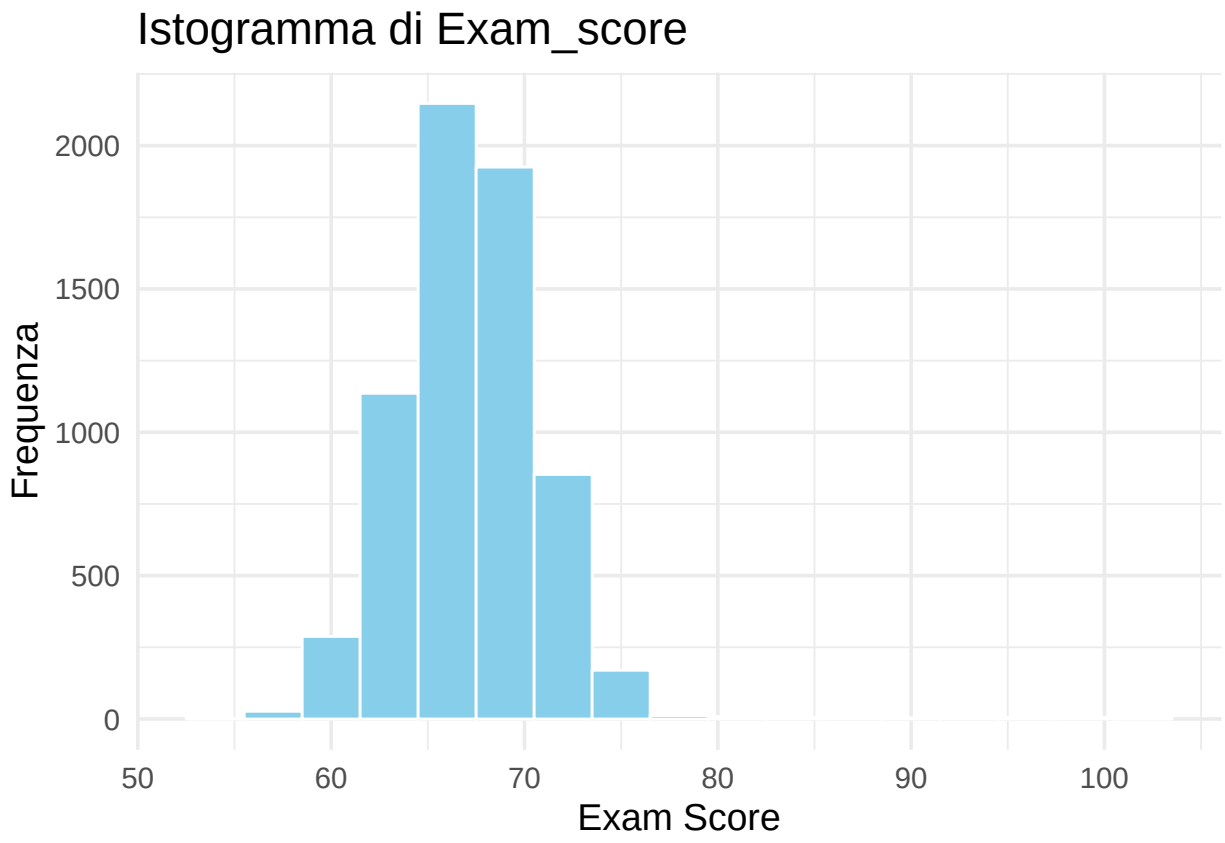
```
str(ds)
```

```
## 'data.frame':    6607 obs. of  20 variables:
##  $ Hours_Studied             : int  23 19 24 29 19 19 29 25 17 23 ...
##  $ Attendance                : int  84 64 98 89 92 88 84 78 94 98 ...
##  $ Parental_Involvement      : Factor w/ 3 levels "High","Low","Medium": 2 2 3 2 3 3 3 2 3 3 ...
##  $ Access_to_Resources       : Factor w/ 3 levels "High","Low","Medium": 1 3 3 3 3 3 2 1 1 3 ...
##  $ Extracurricular_Activities: Factor w/ 2 levels "No","Yes": 1 1 2 2 2 2 2 2 1 2 ...
##  $ Sleep_Hours               : int  7 8 7 8 6 8 7 6 6 8 ...
##  $ Previous_Scores           : int  73 59 91 98 65 89 68 50 80 71 ...
##  $ Motivation_Level          : Factor w/ 3 levels "High","Low","Medium": 2 2 3 3 3 3 3 2 3 1 3 ...
##  $ Internet_Access           : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Tutoring_Sessions         : int  0 2 2 1 3 3 1 1 0 0 ...
##  $ Family_Income             : Factor w/ 3 levels "High","Low","Medium": 2 3 3 3 3 3 2 1 3 1 ...
##  $ Teacher_Quality           : Factor w/ 4 levels "","High","Low",..: 4 4 4 4 2 4 4 2 3 2 ...
##  $ School_Type               : Factor w/ 2 levels "Private","Public": 2 2 2 2 2 2 1 2 1 2 ...
##  $ Peer_Influence            : Factor w/ 3 levels "Negative","Neutral",..: 3 1 2 1 2 3 2 1 2 3 ...
##  $ Physical_Activity         : int  3 4 4 4 4 3 2 2 1 5 ...
##  $ Learning_Disabilities     : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Parental_Education_Level  : Factor w/ 4 levels "","College","High School",..: 3 2 4 3 2 4 3 3 2 3
##  $ Distance_from_Home        : Factor w/ 4 levels "","Far","Moderate",..: 4 3 4 3 4 4 3 2 4 3 ...
```

```
##  $ Gender                    : Factor w/ 2 levels "Female","Male": 2 1 2 2 1 2 2 2 2 2 ...
##  $ Exam_Score                : int  67 61 74 71 70 71 67 66 69 72 ...
```

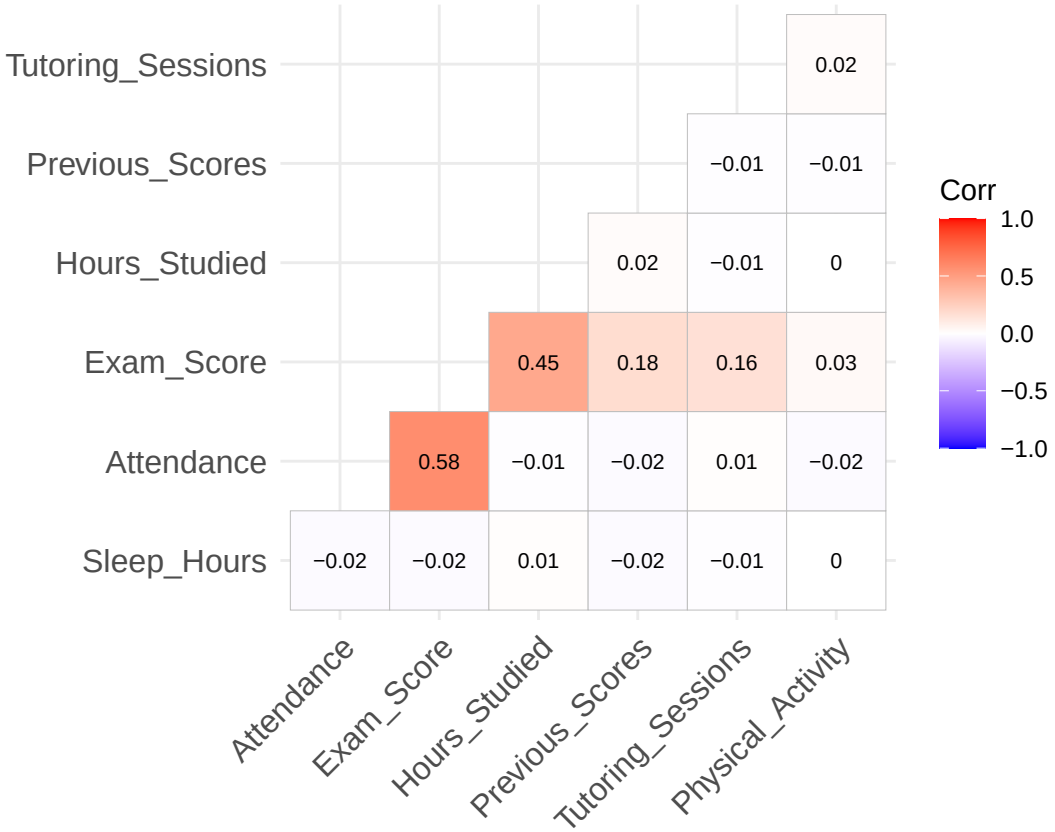
La variabile di interesse in questa analisi è *Exam_Score*. Per comprenderne meglio la sua natura, osserviamo la sua distribuzione. Per fa ciò è stato realizzato un istogramma che ci mostra un andamento quasi normale della variabile in questione, con la maggior parte dei punteggi compresa tra 63 e 75, e una presenza limitata di valori estremi.

```
ggplot(ds, aes(x = Exam_Score)) +
  geom_histogram(
    binwidth = 3,
    fill     = "skyblue",
    color    = "white"
  ) +
  labs(
    x     = "Exam Score",
    y     = "Frequenza",
    title = "Istogramma di Exam_score"
  ) +
  theme_minimal(base_size = 14)
```



Analizzando la matrice di correlazione calcolata sul dataset, si osserva che la variabile Exam_Score mostra una forte correlazione con Attendance (0.58) e Hours_Studied (0.45). La stessa variabile presenta inoltre una correlazione, seppur più debole, anche con Previous_Scores e Tutoring_Sessions. Non emergono invece correlazioni rilevanti tra le altre variabili numeriche del dataset.

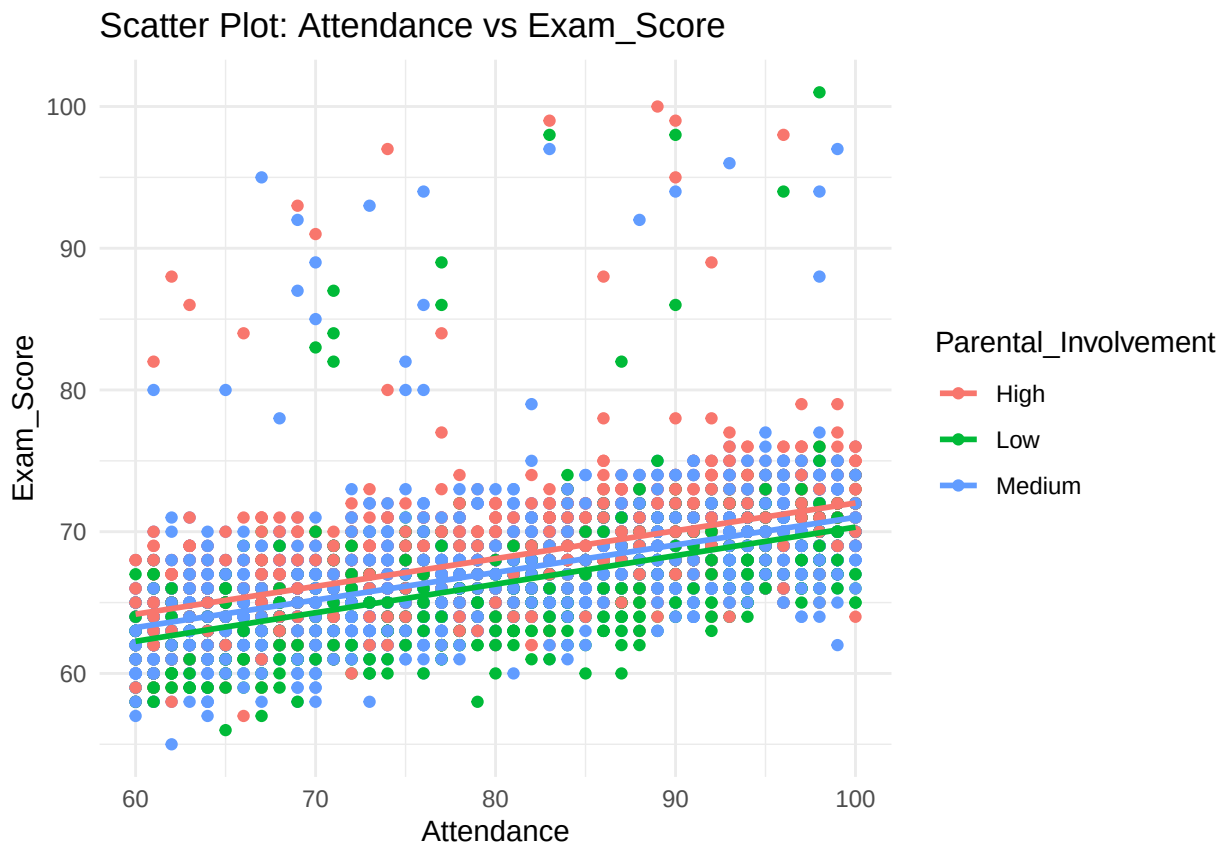
```
matrix_corrplot = round(cor(select_if(ds, is.numeric), method="pearson"),4)
ggcorrplot(matrix_corrplot, hc.order=T, type="lower", lab=T, lab_size = 2.7)
```



Alla luce dei risultati ottenuti dalla matrice di correlazione, è stato deciso di realizzare alcuni scatter plot per analizzare le variabili più significative. In questo grafico, è evidente come la variabile Exam_Score mostri una relazione lineare con la variabile Attendance. Inoltre, si osserva come la relazione tra le due variabili sembri essere influenzata dalla variabile categoriale Parental_Involvement, evidenziando una suddivisione dei bias presenti.

```
ggplot(ds, aes(x = Attendance, y = Exam_Score, color=Parental_Involvement)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(
    x = "Attendance",
    y = "Exam_Score",
    title = "Scatter Plot: Attendance vs Exam_Score"
  ) +
  theme_minimal()
```

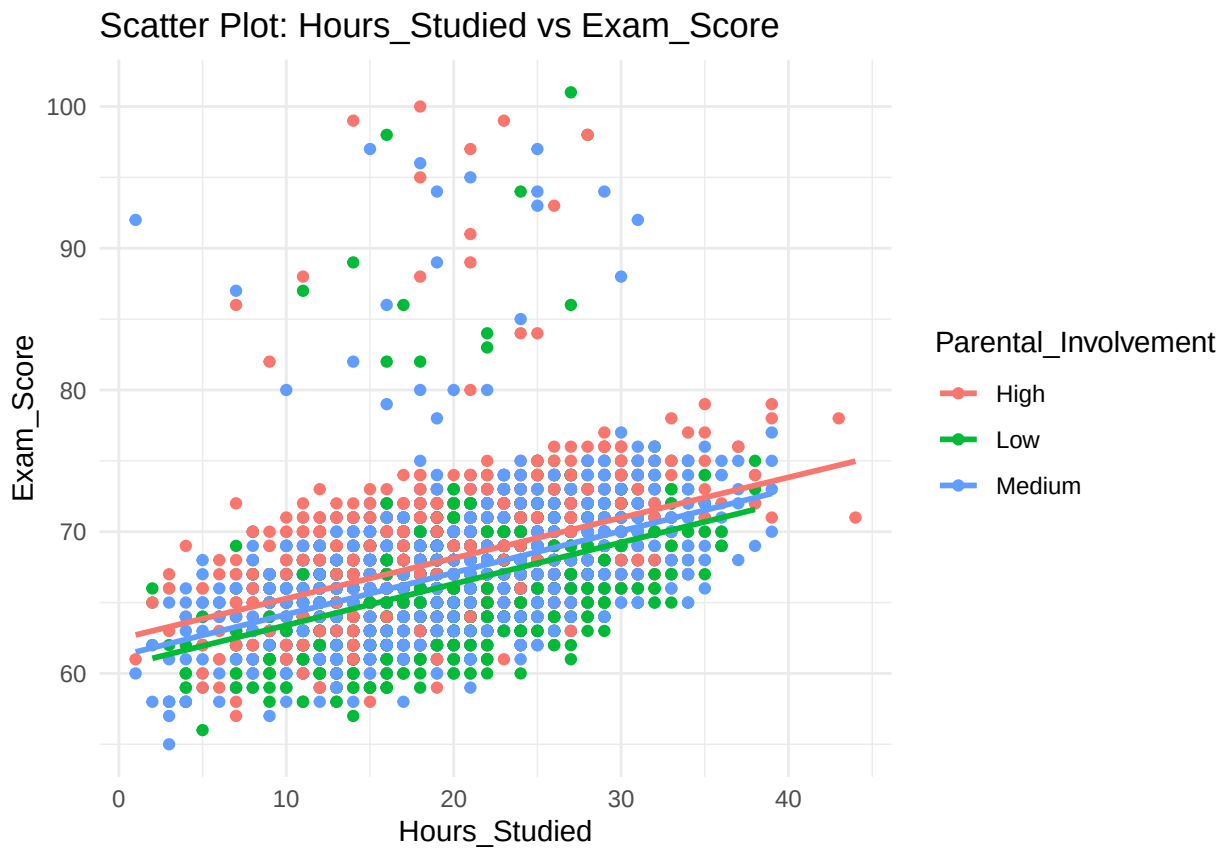
```
## `geom_smooth()` using formula = 'y ~ x'
```



Analogamente a priama, è stata effettuata la analisi precedente ma questa volta sono state considerate le ore di studio settimanali (Hours_Studied). Anche in questo caso si nota una forte dipendenza lineare fra le due variabili con una distinzione in base Parental_Involvement.

```
ggplot(ds, aes(x = Hours_Studied, y = Exam_Score, color=Parental_Involvement)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(
    x = "Hours_Studied",
    y = "Exam_Score",
    title = "Scatter Plot: Hours_Studied vs Exam_Score"
  ) +
  theme_minimal()
```

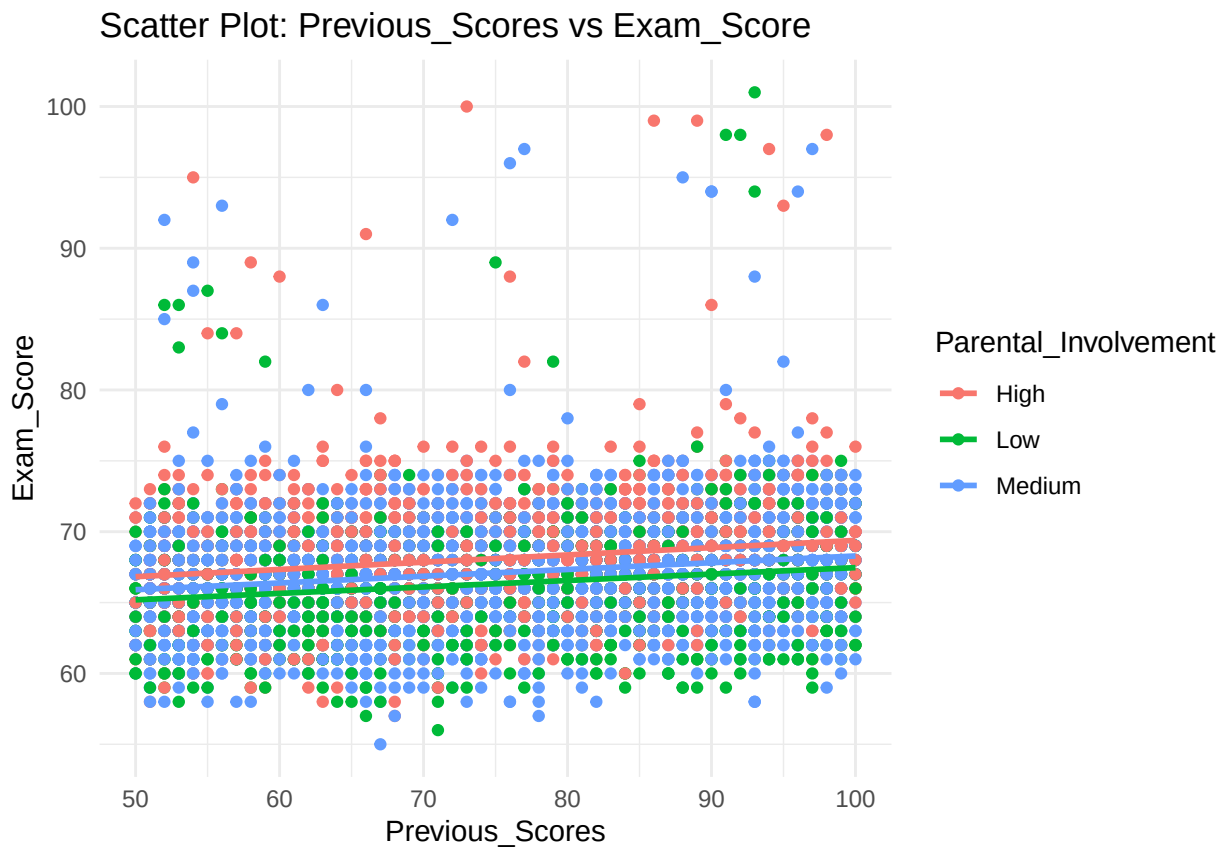
```
## `geom_smooth()` using formula = 'y ~ x'
```



Di particolare interesse è anche la variabile Previous_Scores, che intuitivamente potrebbe sembrare quella più adatta per predire il valore di Exam_Score. Tuttavia, questa analisi mette in luce una notevole variabilità fra i dati, suggerendo che altre variabili potrebbero giocare un ruolo altrettanto rilevante nella previsione dei punteggi dell'esame finale.

```
ggplot(ds, aes(x = Previous_Scores, y = Exam_Score, color=Parental_Involvement)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(
    x = "Previous_Scores",
    y = "Exam_Score",
    title = "Scatter Plot: Previous_Scores vs Exam_Score"
  ) +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

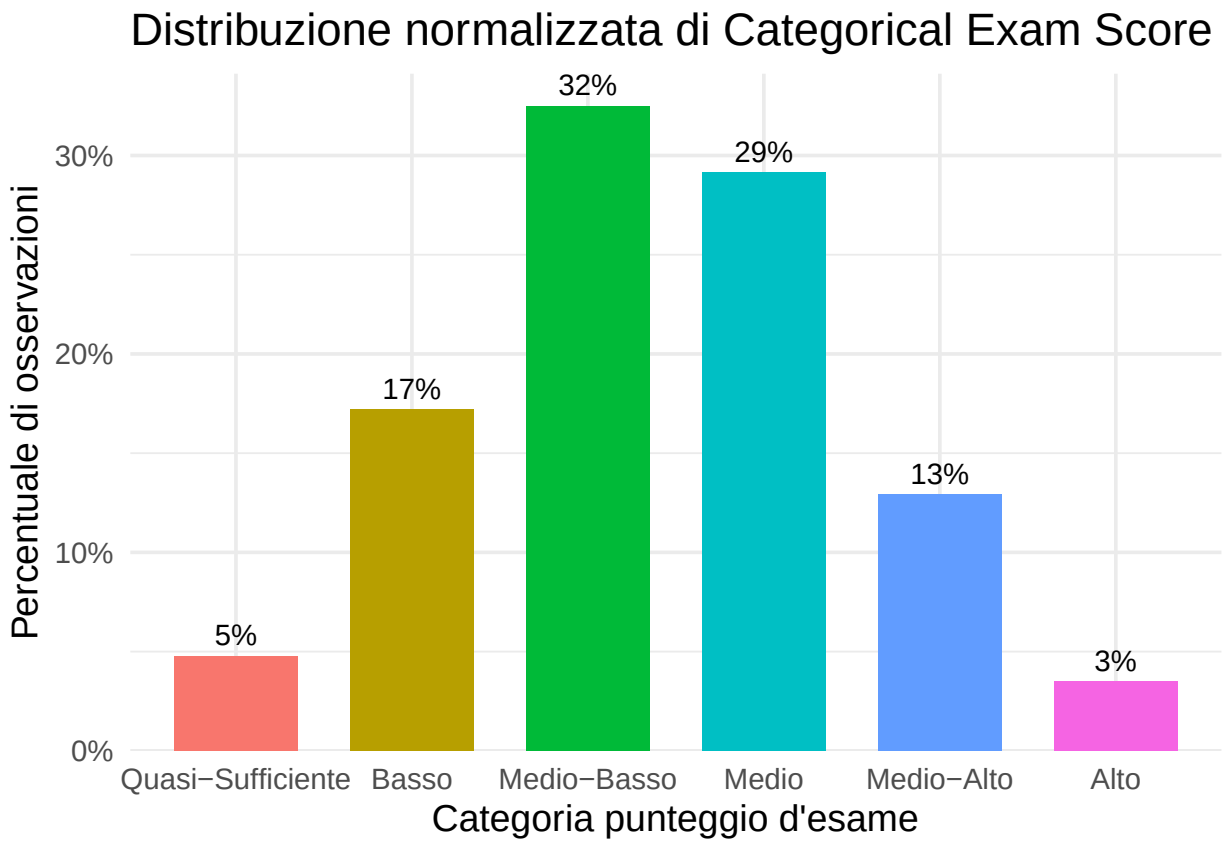


Poiché l'obiettivo di questa analisi è costruire un modello capace sia di individuare le variabili più rilevanti, sia di prevedere il rendimento di uno studente nell'esame finale, si è scelto di trasformare la variabile Exam_Score da numerica a categoriale. Infatti, non è tanto importante stimare il punteggio esatto che uno studente potrebbe ottenere, quanto piuttosto individuare la fascia di voto in cui è più probabile che si collochi. Questo approccio è utile anche per ipotizzare eventuali interventi didattici mirati, con l'intento di migliorare il percorso formativo degli studenti. Per raggiungere questo scopo, il codice seguente effettua tale trasformazione. Tuttavia, come mostrato nell'istogramma precedente, i punteggi sono concentrati in un intervallo molto ristretto. Per questo motivo sono stati creati due dataset con classificazioni differenti: - Il primo suddivide i voti in 6 classi, fornendo una stima più precisa del rendimento ma includendo due classi scarsamente rappresentate. - Il secondo utilizza 4 classi, semplificando il lavoro del modello a scapito però della precisione nella previsione del voto finale.

In entrambi i casi, le fasce sono state definite sfruttando la simmetria della distribuzione attorno al valore medio (67), con l'obiettivo di bilanciare le classi. Inoltre, gli intervalli non sono equidistanti, in quanto le fasce più estreme hanno ampiezze maggiori per compensare la minore densità dei dati in quelle zone.

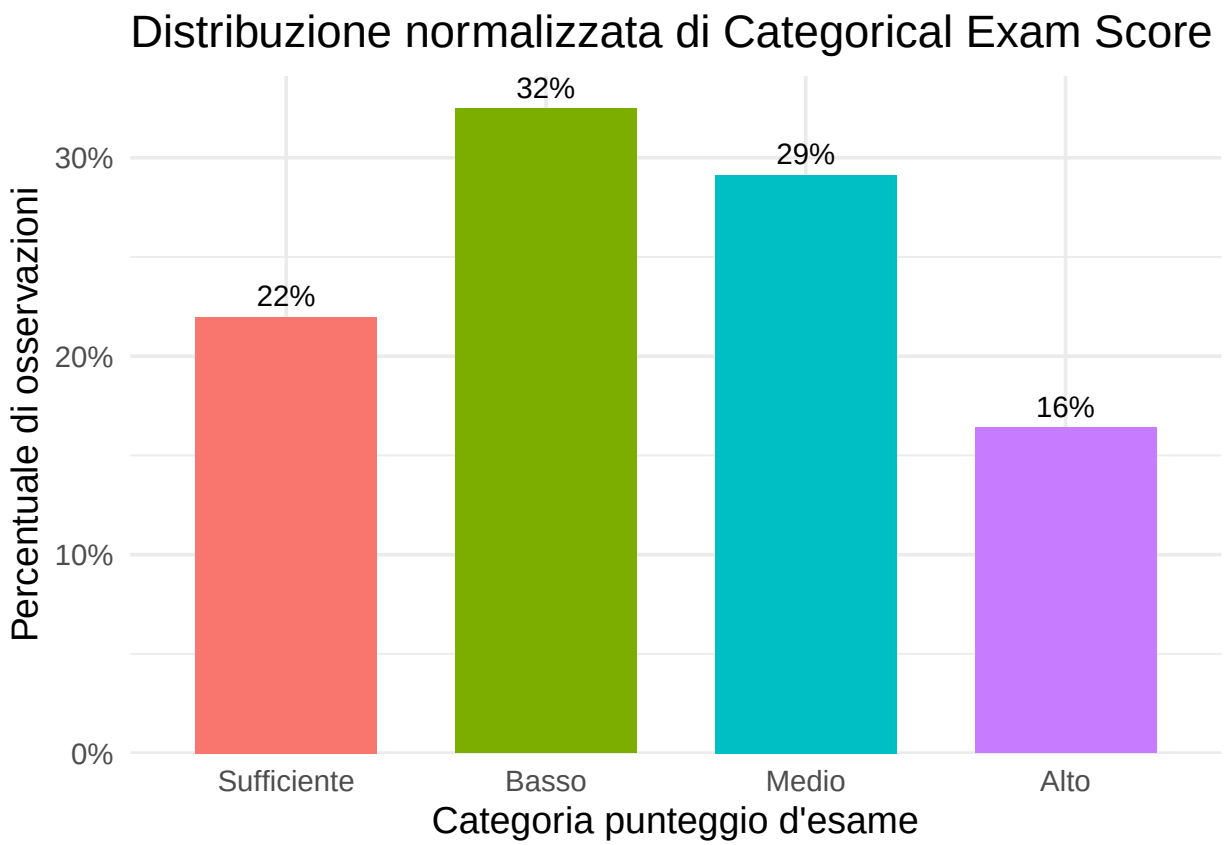
```
ds_2 = ds

ds_2$Categorical_Exam_Score <- cut(
  ds$Exam_Score,
  breaks = c(54, 64, 67, 70, 102),
  labels = c("Sufficiente", "Basso", "Medio", "Alto"),
  include.lowest = FALSE,
  right = TRUE
)
```

```r
ds$Categorical_Exam_Score <- cut(
  ds$Exam_Score,
  breaks = c(54, 61, 64, 67, 70, 73, 102),
  labels = c("Quasi-Sufficiente", "Basso", "Medio-Basso", "Medio", "Medio-Alto", "Alto"),
  include.lowest = FALSE,
  right = TRUE
)

ggplot(ds, aes(x = Categorical_Exam_Score,
               fill = Categorical_Exam_Score)) +
  # barre con proporzione
  geom_bar(
    aes(y = after_stat(count) / sum(after_stat(count))),
    stat = "count",
    width = 0.7,
    show.legend = FALSE
  ) +
  # percentuali sopra le barre
  geom_text(
    aes(
      label = percent(after_stat(count) / sum(after_stat(count)), accuracy = 1),
      y     = after_stat(count) / sum(after_stat(count))
    ),
    stat = "count",
    vjust = -0.5
  ) +
  # scala y in percentuale e un po' di spazio in alto
  scale_y_continuous(
    labels = percent_format(accuracy = 1),
    expand = expansion(mult = c(0, 0.05))
  ) +
  labs(
    x     = "Categoria punteggio d'esame",
    y     = "Percentuale di osservazioni",
    title = "Distribuzione normalizzata di Categorical Exam Score"
  ) +
  theme_minimal(base_size = 14)
```




```r
ggplot(ds_2, aes(x = Categorical_Exam_Score,
                 fill = Categorical_Exam_Score)) +
  # barre con proporzione
  geom_bar(
    aes(y = after_stat(count) / sum(after_stat(count))),
    stat = "count",
    width = 0.7,
    show.legend = FALSE
  ) +
  # percentuali sopra le barre
  geom_text(
    aes(
      label = percent(after_stat(count) / sum(after_stat(count)), accuracy = 1),
      y     = after_stat(count) / sum(after_stat(count))
    ),
    stat = "count",
    vjust = -0.5
  ) +
  # scala y in percentuale e un po' di spazio in alto
  scale_y_continuous(
    labels = percent_format(accuracy = 1),
    expand = expansion(mult = c(0, 0.05))
  ) +
  labs(
    x     = "Categoria punteggio d'esame",
    y     = "Percentuale di osservazioni",
```

```
    title = "Distribuzione normalizzata di Categorical Exam Score"
  ) +
  theme_minimal(base_size = 14)
```



Divisione del dataset in train e test

In questa sezione si è effettuata la suddivisione dei due dataset in train e test. Il seguente codice divide il datsaet con le classi di Categorical_Exam_Score più numerose.

```
set.seed(123)
ds$Exam_Score= NULL

trainIndex <- createDataPartition(ds$Categorical_Exam_Score,
                                  p    = 0.5,
                                  list = FALSE)

train <- ds[ trainIndex, ]
test  <- ds[-trainIndex, ]
```

Mentre il seguente codice divide il datsaet con le classi di Categorical_Exam_Score meno numerose.

```
set.seed(123)
ds_2$Exam_Score= NULL

trainIndex_2 <- createDataPartition(ds_2$Categorical_Exam_Score,
                                  p    = 0.5,
                                  list = FALSE)
```

```
train_2 <- ds_2[ trainIndex, ]
test_2  <- ds_2[-trainIndex, ]
```

Analisi

Data aug

In questo paragrafo ci siamo concentrati sul sottoinsieme di addestramento (train), dove sono presenti le categorie con il maggior numero di osservazioni, con l'obiettivo di incrementare le classi meno rappresentate. A tal fine, abbiamo sfruttato la libreria smotefamily, che mette a disposizione la funzione SMOTE. Lo scopo di questa procedura è verificare se, bilanciando le classi tramite algoritmi di over-sampling, è possibile migliorare le prestazioni predittive dei nostri modelli

Il codice applica la funzione SMOTE per generare nuovi esempi sintetici per le due classi meno rappresentate, "rare", indicate nella lista rare_classes. La funzione trasformando il problema in un'etichettatura binaria, selezionando solo le variabili numeriche e applicando l'algoritmo SMOTE. I campioni sintetici generati vengono successivamente aggiunti al dataset originale, con la pulizia delle colonne temporanee e la verifica della nuova distribuzione delle classi per assicurarsi che l'over-sampling abbia riequilibrato il dataseto quanto meno migliorato il problema .

```
table(train$Categorical_Exam_Score)
```

```
##
## Quasi-Sufficiente             Basso       Medio-Basso             Medio
##               158               568              1074               963
##        Medio-Alto              Alto
##               427               115
```

```
# Quasi-Sufficiente" e "Alto" sono le classi con pochi esempi
rare_classes <- c("Quasi-Sufficiente", "Alto")

apply_smote_to_class <- function(data, class_target, rate = 2) {
  # Crea etichetta binaria
  data$binary_target <- ifelse(data$Categorical_Exam_Score == class_target, 1, 0)

  # SMOTE lavora solo su variabili numeriche → isoliamo features numeriche
  x_vars <- data %>% select(where(is.numeric))
  y_bin  <- data$binary_target

  # Applica SMOTE
  smote_out <- SMOTE(x_vars, y_bin, K = 12, dup_size = rate)

  # Recupera solo i sintetici generati (classe = 1)
  synthetic <- smote_out$syn_data %>%
    mutate(Categorical_Exam_Score = class_target)

  # Rimuove colonna target binaria
  synthetic <- synthetic %>% select(-class)

  return(synthetic)
}

# Applichiamo SMOTE a ciascuna classe rara
synthetics <- lapply(rare_classes, function(cl) {
```

```
  apply_smote_to_class(train, class_target = cl, rate = 2)  # dup_size controlla quanto ne vuoi
})

# Combiniamo i sintetici
synthetic_data <- bind_rows(synthetics)

# Unisci al train originale
train_augmented <- bind_rows(train, synthetic_data)
train_augmented$binary_target = NULL


train_augmented$Categorical_Exam_Score = as.factor(train_augmented$Categorical_Exam_Score)

# Controlla la nuova distribuzione
table(train_augmented$Categorical_Exam_Score)
```

```
##
##              Alto             Basso             Medio        Medio-Alto
##               345               568               963               427
##       Medio-Basso Quasi-Sufficiente
##              1074               474
```

Poiché la funzione SMOTE calcola nuovi valori solo per le variabili numeriche, lasciando vuoti i campi relativi alle variabili categoriali, si è deciso di riempire questi campi con il valore più comune della variabile categoriale per ciascuna classe. Questa scelta riduce la variabilità dei dati, ma, a seguito di valutazioni preliminari, si è constatato che eliminare completamente le variabili categoriali dal dataset comportava un notevole peggioramento delle performance. Pertanto, si è preferito inserire i valori mancanti invece di rimuovere del tutto queste variabili, per mantenere la coerenza e la ricchezza informativa del dataset.

```
impute_categorical_na_by_class_mode <- function(data, class_col, rare_classes) {
  # Identifica colonne categoriali (escluse quelle già numeriche o il target)
  categorical_cols <- data %>% select(where(~is.factor(.) || is.character(.))) %>% select(-all_of(class_

  for (cat_col in categorical_cols) {
    for (rare_class in rare_classes) {
      # Subset dei dati per la classe rara
      subset_class <- data %>%
        filter(!!sym(class_col) == rare_class)

      # Calcola la moda ignorando gli NA
      mode_val <- subset_class %>%
        filter(!is.na(!!sym(cat_col))) %>%
        count(!!sym(cat_col), sort = TRUE) %>%
        slice(1) %>%
        pull(!!sym(cat_col))

      # Sostituisci NA con la moda solo per la classe rara corrente
      data <- data %>%
        mutate(!!sym(cat_col) := ifelse(
          is.na(!!sym(cat_col)) & (!!sym(class_col) == rare_class),
          mode_val,
          !!sym(cat_col)
        ))
    }
  }
```

```
  return(data)
}


train_augmented <- impute_categorical_na_by_class_mode(
  data = train_augmented,
  class_col = "Categorical_Exam_Score",
  rare_classes = rare_classes
)
```

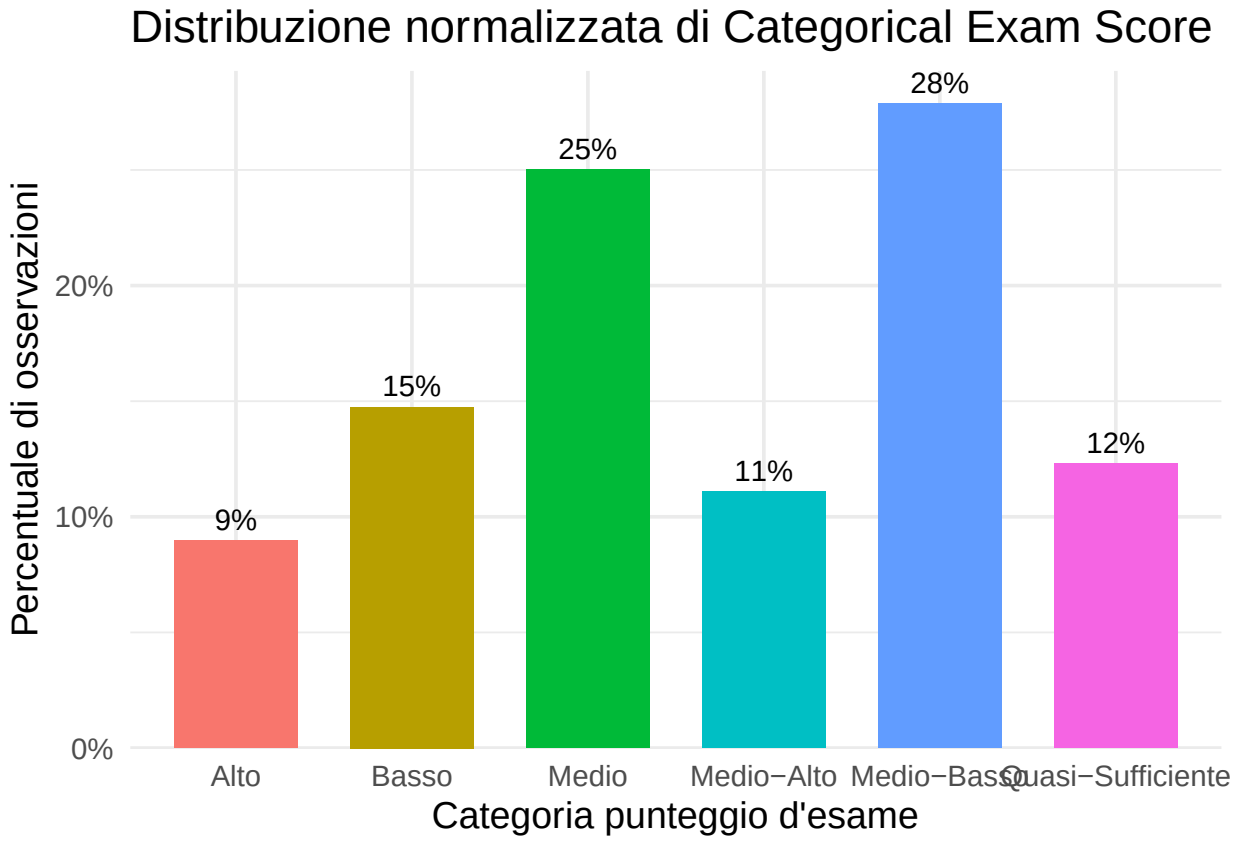
Per motivi di efficienza computazionale, le variabili categoriali presenti nel dataset aumentato nel codice precedente sono state codificate come indici corrispondenti ai livelli delle variabili categoriali stesse. Per rendere nuovamente il dataset facilmente interpretabile, il codice seguente converte questi indici nei rispettivi valori testuali, rendendo il dataset nuovamente leggibile e comprensibile.

```
levels_list <- lapply(train[categorical_vars], function(x) {
  # se sono factor mantieni i livelli, altrimenti estrai i valori unici
  if (is.factor(x)) levels(x) else unique(as.character(x))
})
names(levels_list) <- categorical_vars

# Decodifica in train_augmented gli indici numerici usando levels_list
for (var in categorical_vars) {
  train_augmented[[var]] <- factor(
    train_augmented[[var]],
    levels = seq_along(levels_list[[var]]),
    labels = levels_list[[var]]
  )
}
```

Con il seguente codice è possibile visualizzare la frequenza percentuale delle variabili nel dataset aumentato. Si osserva come l'algoritmo abbia incrementato la rappresentatività delle classi meno numerose, bilanciando meglio la distribuzione delle categorie.

```
ggplot(train_augmented, aes(x = Categorical_Exam_Score,
               fill = Categorical_Exam_Score)) +
  # barre con proporzione
  geom_bar(
    aes(y = after_stat(count) / sum(after_stat(count))),
    stat = "count",
    width = 0.7,
    show.legend = FALSE
  ) +
  # percentuali sopra le barre
  geom_text(
    aes(
      label = percent(after_stat(count) / sum(after_stat(count)), accuracy = 1),
      y     = after_stat(count) / sum(after_stat(count))
    ),
    stat = "count",
    vjust = -0.5
  ) +
  # scala y in percentuale e un po' di spazio in alto
  scale_y_continuous(
    labels = percent_format(accuracy = 1),
```

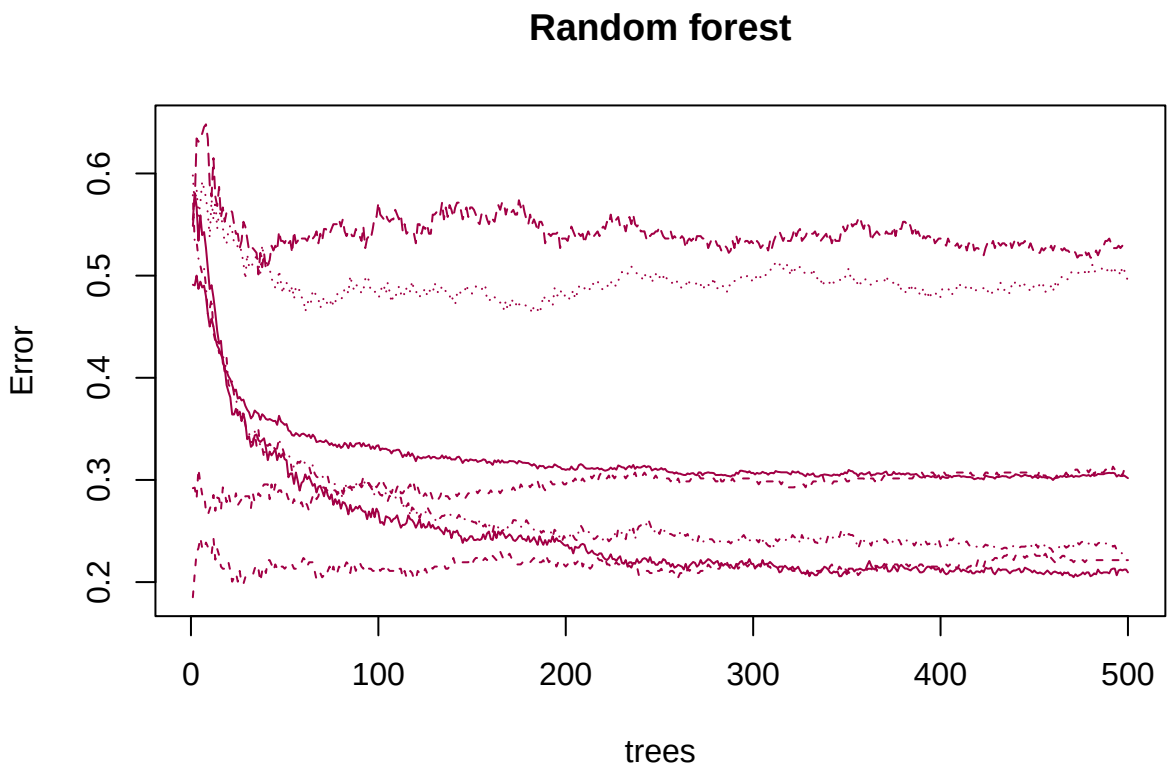
```
    expand = expansion(mult = c(0, 0.05))
  ) +
  labs(
    x     = "Categoria punteggio d'esame",
    y     = "Percentuale di osservazioni",
    title = "Distribuzione normalizzata di Categorical Exam Score"
  ) +
  theme_minimal(base_size = 14)
```



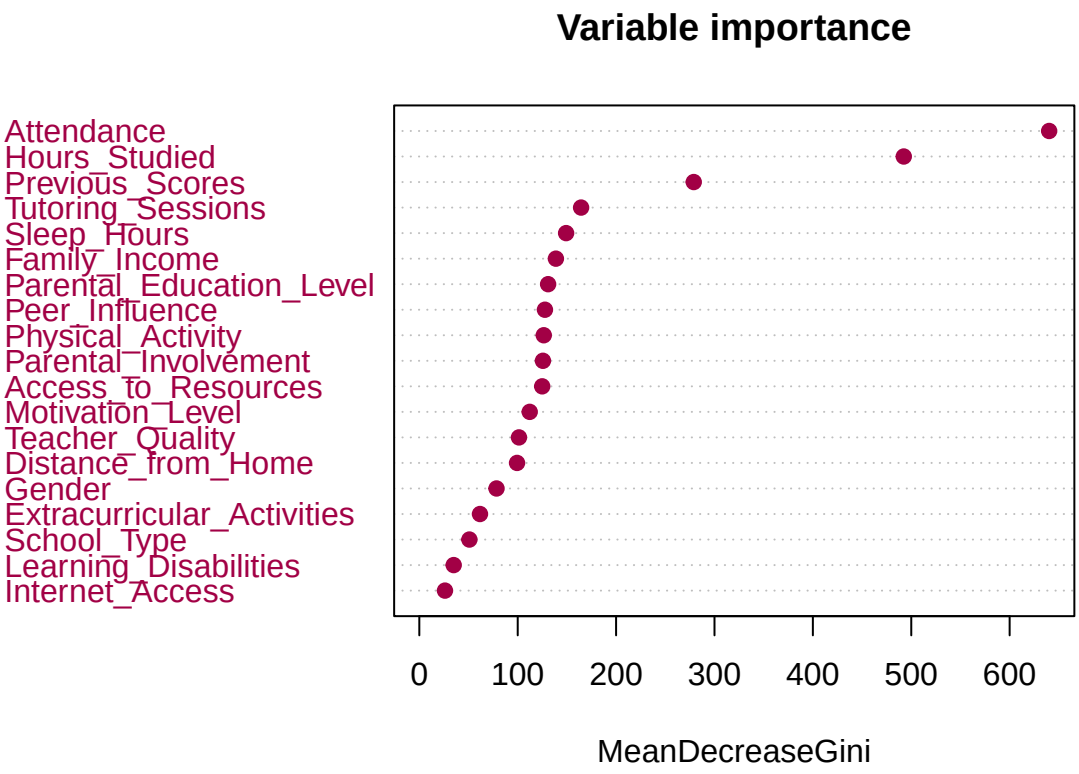
Random Forest

AUg

```
p <- NCOL(train_augmented) - 1


set.seed(11)

rf_1_aug <- randomForest(Categorical_Exam_Score ~ . , data = train_augmented)
plot(rf_1_aug, col="#A20045", main="Random forest")
```




```
varImpPlot(rf_1_aug, main="Variable importance", pch = 19, color="#A20045")
```




```
pred = predict(rf_1_aug)
confusionMatrix(pred, train_augmented$Categorical_Exam_Score)
```

```
## Confusion Matrix and Statistics
##
##                    Reference
## Prediction          Alto Basso Medio Medio-Alto Medio-Basso Quasi-Sufficiente
##   Alto               238     0     1          5           2                 0
##   Basso                3   287     0          0          57               102
##   Medio               14     0   745        216         164                 0
##   Medio-Alto          79     0    15        201           0                 0
##   Medio-Basso         11   274   202          5         849                 3
##   Quasi-Sufficiente    0     7     0          0           2               369
##
## Overall Statistics
##
##                Accuracy : 0.6983
##                  95% CI : (0.6835, 0.7127)
##     No Information Rate : 0.2789
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6157
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
```

```
##
##                      Class: Alto Class: Basso Class: Medio Class: Medio-Alto
## Sensitivity              0.68986      0.50528       0.7736           0.47073
## Specificity              0.99772      0.95065       0.8636           0.97255
## Pos Pred Value           0.96748      0.63920       0.6541           0.68136
## Neg Pred Value           0.97032      0.91740       0.9196           0.93645
## Prevalence               0.08959      0.14749       0.2501           0.11088
## Detection Rate           0.06180      0.07453       0.1935           0.05219
## Detection Prevalence     0.06388      0.11659       0.2958           0.07660
## Balanced Accuracy        0.84379      0.72797       0.8186           0.72164
##                      Class: Medio-Basso Class: Quasi-Sufficiente
## Sensitivity                      0.7905                  0.77848
## Specificity                      0.8218                  0.99733
## Pos Pred Value                   0.6317                  0.97619
## Neg Pred Value                   0.9103                  0.96977
## Prevalence                       0.2789                  0.12308
## Detection Rate                   0.2205                  0.09582
## Detection Prevalence             0.3490                  0.09816
## Balanced Accuracy                0.8061                  0.88791
```

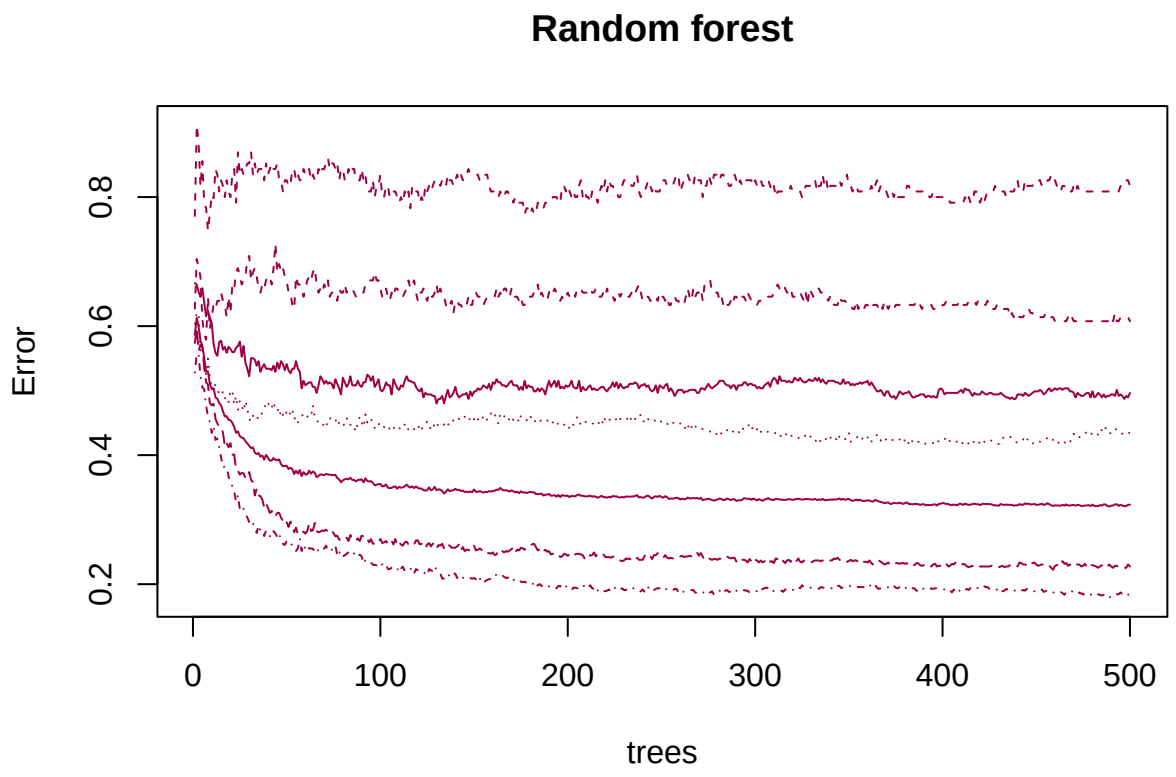
```
pred_test_aug = predict(rf_1_aug, newdata = test)
pred_test_aug <- factor(pred_test_aug, levels = levels(test$Categorical_Exam_Score))
confusionMatrix(pred_test_aug, test$Categorical_Exam_Score)
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction          Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
##   Quasi-Sufficiente                40     4           0     0          0    1
##   Basso                           116   303          61     1          0    9
##   Medio-Basso                       2   260         856   193          1    7
##   Medio                             0     1         156   735        247   11
##   Medio-Alto                        0     0           0    33        177   81
##   Alto                              0     0           0     0          1    6
##
## Overall Statistics
##
##                Accuracy : 0.6411
##                  95% CI : (0.6245, 0.6575)
##     No Information Rate : 0.325
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5091
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Quasi-Sufficiente Class: Basso Class: Medio-Basso
## Sensitivity                           0.25316      0.53345             0.7978
## Specificity                           0.99841      0.93160             0.7923
## Pos Pred Value                        0.88889      0.61837             0.6490
## Neg Pred Value                        0.96377      0.90576             0.8906
## Prevalence                            0.04785      0.17202             0.3250
## Detection Rate                        0.01211      0.09176             0.2592
```

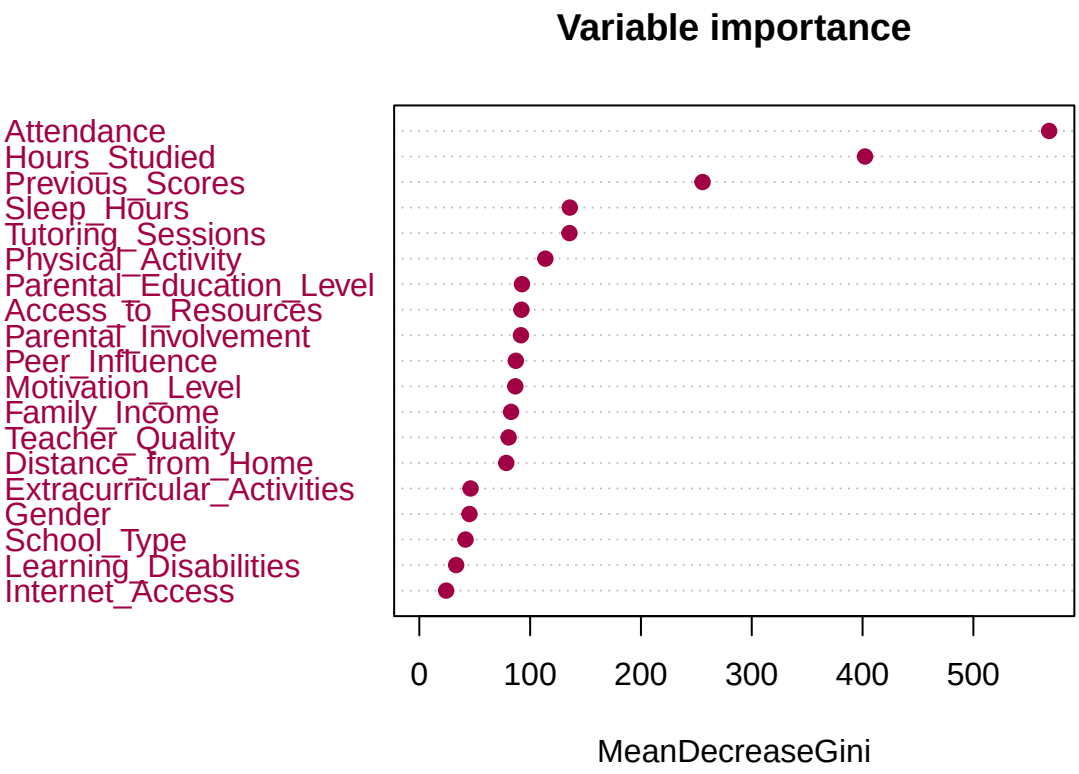
```
## Detection Prevalence                  0.01363      0.14839             0.3995
## Balanced Accuracy                     0.62579      0.73253             0.7950
##                      Class: Medio Class: Medio-Alto Class: Alto
## Sensitivity                0.7640           0.41549    0.052174
## Specificity                0.8226           0.96036    0.999686
## Pos Pred Value             0.6391           0.60825    0.857143
## Neg Pred Value             0.8945           0.91730    0.966920
## Prevalence                 0.2913           0.12901    0.034827
## Detection Rate             0.2226           0.05360    0.001817
## Detection Prevalence       0.3483           0.08813    0.002120
## Balanced Accuracy          0.7933           0.68793    0.525930
```

NON aug

```
set.seed(11)
rf_1 <- randomForest(Categorical_Exam_Score ~ . , data = train)
plot(rf_1, col="#A20045", main="Random forest")
```




```
varImpPlot(rf_1, main="Variable importance", pch = 19, color="#A20045")
```




```
pred = predict(rf_1)
confusionMatrix(pred, train$Categorical_Exam_Score)
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
##   Quasi-Sufficiente                 62     2           0     0          0    0
##   Basso                             92   322          51     0          0    4
##   Medio-Basso                        4   243         873   193          3    9
##   Medio                              0     1         150   744        206   10
##   Medio-Alto                         0     0           0    26        215   71
##   Alto                               0     0           0     0          3   21
##
## Overall Statistics
##
##                Accuracy : 0.6769
##                  95% CI : (0.6606, 0.6928)
##     No Information Rate : 0.325
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5598
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
```

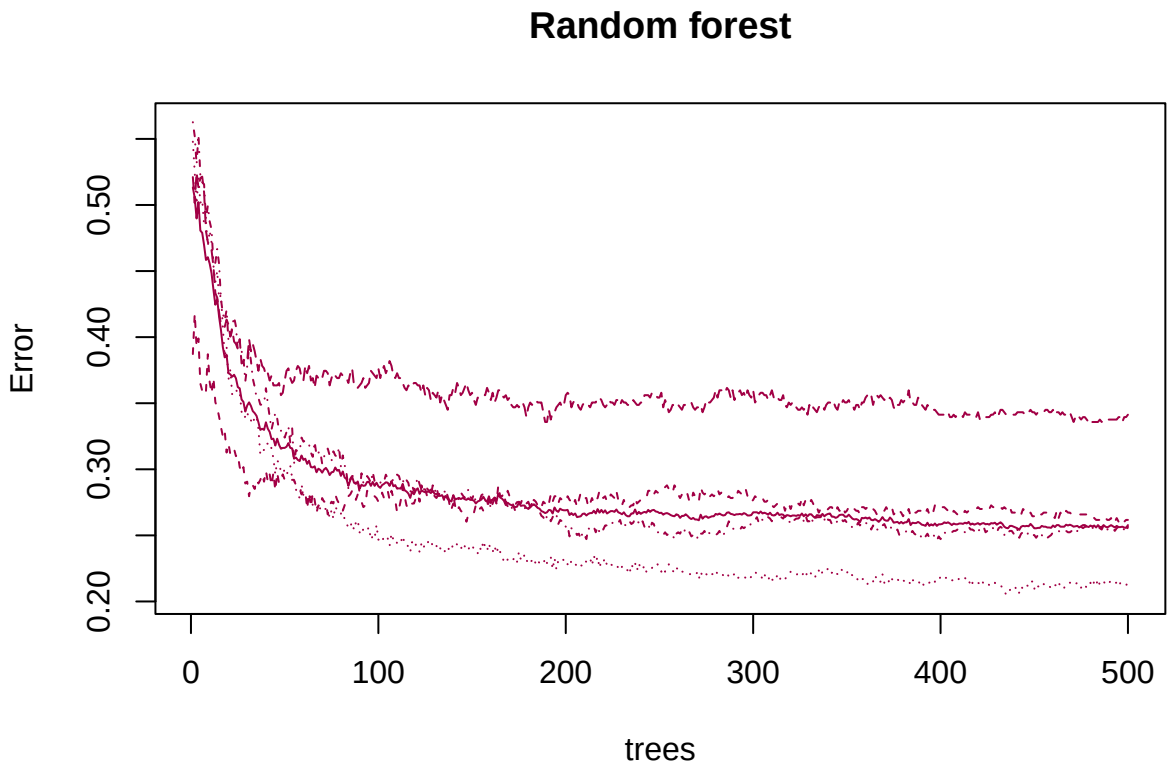
```
##
##                      Class: Quasi-Sufficiente Class: Basso Class: Medio-Basso
## Sensitivity                           0.39241      0.56690             0.8128
## Specificity                           0.99936      0.94629             0.7974
## Pos Pred Value                        0.96875      0.68657             0.6589
## Neg Pred Value                        0.97038      0.91326             0.8985
## Prevalence                            0.04781      0.17186             0.3250
## Detection Rate                        0.01876      0.09743             0.2641
## Detection Prevalence                  0.01936      0.14191             0.4009
## Balanced Accuracy                     0.69588      0.75660             0.8051
##                      Class: Medio Class: Medio-Alto Class: Alto
## Sensitivity                0.7726           0.50351    0.182609
## Specificity                0.8433           0.96630    0.999060
## Pos Pred Value             0.6697           0.68910    0.875000
## Neg Pred Value             0.9002           0.92917    0.971350
## Prevalence                 0.2914           0.12920    0.034796
## Detection Rate             0.2251           0.06505    0.006354
## Detection Prevalence       0.3362           0.09440    0.007262
## Balanced Accuracy          0.8079           0.73490    0.590834
```

```
pred_test = predict(rf_1, newdata = test)
confusionMatrix(pred_test, test$Categorical_Exam_Score)
```

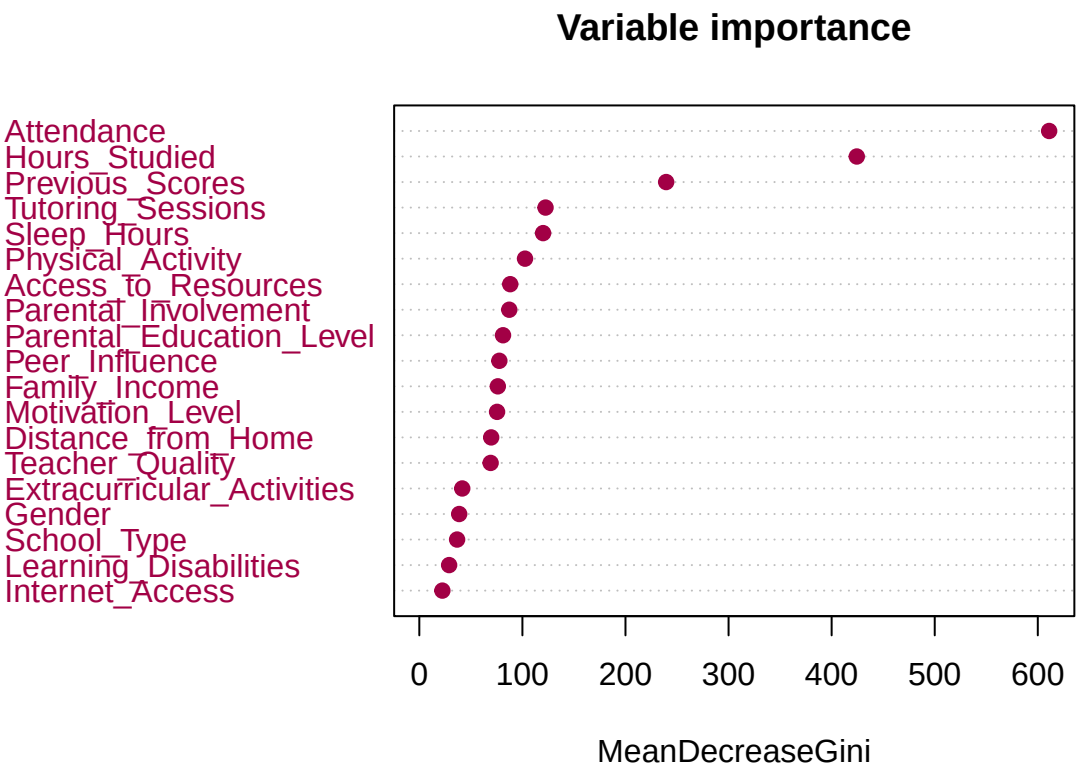
```
## Confusion Matrix and Statistics
##
##                    Reference
## Prediction          Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
##   Quasi-Sufficiente                40     0           0     0          0    1
##   Basso                           117   329          45     0          0    8
##   Medio-Basso                       1   238         885   187          1    9
##   Medio                             0     1         143   745        226    9
##   Medio-Alto                        0     0           0    30        198   80
##   Alto                              0     0           0     0          1    8
##
## Overall Statistics
##
##                Accuracy : 0.6678
##                  95% CI : (0.6514, 0.6838)
##     No Information Rate : 0.325
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5462
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Quasi-Sufficiente Class: Basso Class: Medio-Basso
## Sensitivity                           0.25316      0.57923             0.8248
## Specificity                           0.99968      0.93782             0.8044
## Pos Pred Value                        0.97561      0.65932             0.6699
## Neg Pred Value                        0.96381      0.91473             0.9051
## Prevalence                            0.04785      0.17202             0.3250
## Detection Rate                        0.01211      0.09964             0.2680
## Detection Prevalence                  0.01242      0.15112             0.4001
```

```
## Balanced Accuracy                    0.62642      0.75852            0.8146
##                       Class: Medio Class: Medio-Alto Class: Alto
## Sensitivity                 0.7744           0.46479    0.069565
## Specificity                 0.8380           0.96175    0.999686
## Pos Pred Value              0.6628           0.64286    0.888889
## Neg Pred Value              0.9004           0.92385    0.967507
## Prevalence                  0.2913           0.12901    0.034827
## Detection Rate              0.2256           0.05996    0.002423
## Detection Prevalence        0.3404           0.09328    0.002726
## Balanced Accuracy           0.8062           0.71327    0.534626
```

```r
set.seed(11)

rf_2 <- randomForest(Categorical_Exam_Score ~ . , data = train_2)
plot(rf_2, col="#A20045", main="Random forest")
```




```r
varImpPlot(rf_2, main="Variable importance", pch = 19, color="#A20045")
```




```
set.seed(11)
pred_2 = predict(rf_2)
confusionMatrix(pred_2, train_2$Categorical_Exam_Score)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    Sufficiente Basso Medio Alto
##   Sufficiente         536    77     0    5
##   Basso               190   844   196   11
##   Medio                 0   153   716  169
##   Alto                  0     0    51  357
##
## Overall Statistics
##
##                Accuracy : 0.7422
##                  95% CI : (0.7269, 0.7571)
##     No Information Rate : 0.325
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6445
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

```
##                      Class: Sufficiente Class: Basso Class: Medio Class: Alto
## Sensitivity                      0.7383       0.7858       0.7435      0.6587
## Specificity                      0.9682       0.8221       0.8625      0.9815
## Pos Pred Value                   0.8673       0.6801       0.6898      0.8750
## Neg Pred Value                   0.9293       0.8886       0.8910      0.9361
## Prevalence                       0.2197       0.3250       0.2914      0.1640
## Detection Rate                   0.1622       0.2554       0.2166      0.1080
## Detection Prevalence             0.1870       0.3755       0.3141      0.1234
## Balanced Accuracy                0.8532       0.8040       0.8030      0.8201
```

```r
set.seed(11)
pred_test_2 = predict(rf_2, newdata = test_2)
confusionMatrix(pred_test_2, test_2$Categorical_Exam_Score)
```

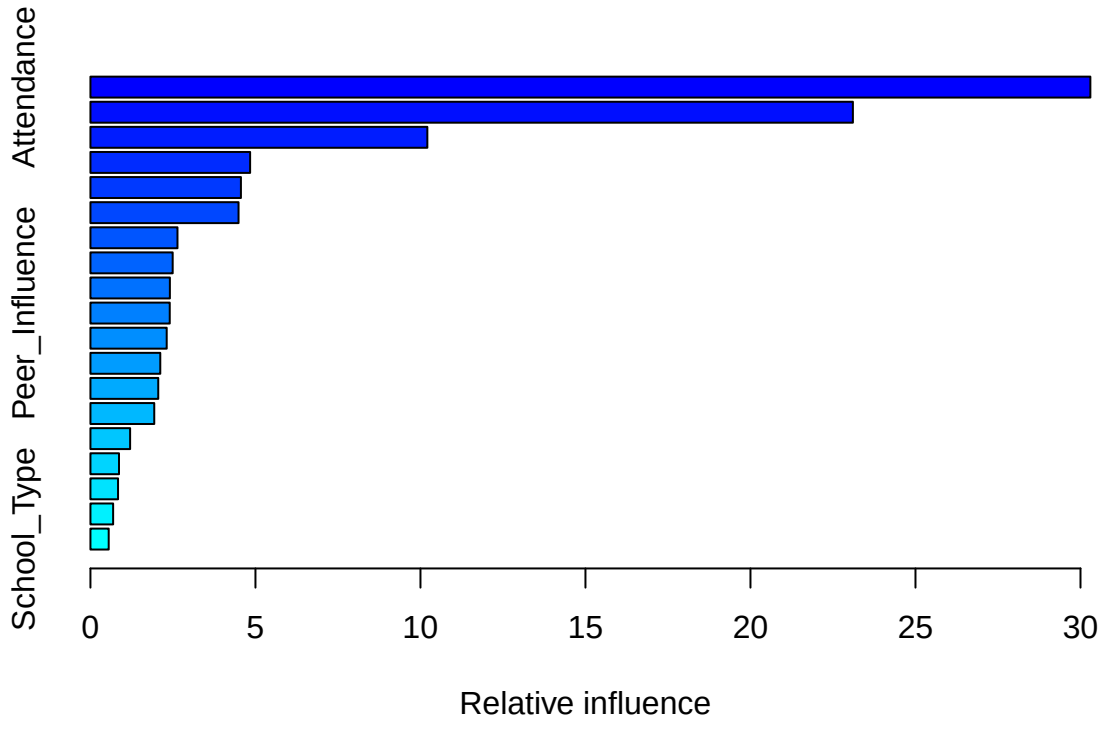
```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    Sufficiente Basso Medio Alto
##   Sufficiente         539    68     0   10
##   Basso               186   860   196    9
##   Medio                 1   145   718  190
##   Alto                  0     0    48  332
##
## Overall Statistics
##
##                Accuracy : 0.7417
##                  95% CI : (0.7264, 0.7565)
##     No Information Rate : 0.325
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6432
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: Sufficiente Class: Basso Class: Medio Class: Alto
## Sensitivity                      0.7424       0.8015       0.7464      0.6137
## Specificity                      0.9697       0.8246       0.8564      0.9826
## Pos Pred Value                   0.8736       0.6875       0.6812      0.8737
## Neg Pred Value                   0.9304       0.8961       0.8915      0.9285
## Prevalence                       0.2199       0.3250       0.2913      0.1638
## Detection Rate                   0.1632       0.2604       0.2174      0.1005
## Detection Prevalence             0.1869       0.3789       0.3192      0.1151
## Balanced Accuracy                0.8561       0.8130       0.8014      0.7981
```

Boosting

Vediamo ora come si comprta l'algoritmo di Boosting sul dateaset 1 contenente 6 classi. Si è deciso di testare solamente questo algorimo sul dataset con più classi quindi con una maggiore difficolta nella capacità di previsione in quanto ci si aspetta che questo sia più performante e che quindi possa darci molte più inforazione nella capacità di previsione con una maggior numero di classi da prevedere. Inizialmente si è proceduti con un modello base con 2000 alberi, con massiamo 4 rami per albero e un learning rate indicato dalla variabile di shrinkage paria a 0.1. Il parametro di shrinkage non è stato cambiato per tutta la trattazione in quanto 0.1 rappresenta il valore massiamo da cui partire e già da questo l'algoritmo inpiega molto tempo per essere

eseguito, percui effettuare un fine tuning risulterebbe troppo difficoltoso. Effettuando il summary del modello possiamo osservare sia la tabella con i valori di importanza relativi a ogni variabile sia un grafico a barre ottenuro da esse. Come visto anche per il modello precedente al primo posso possiamo trovare la variabile Attendance, e a seguire il numero di ore studiate.

```
set.seed(123)

boost.1 <- gbm(Categorical_Exam_Score ~ ., data = train,
               distribution = "multinomial", n.trees = 2000,
               interaction.depth = 4, shrinkage=0.1)
summary(boost.1)
```



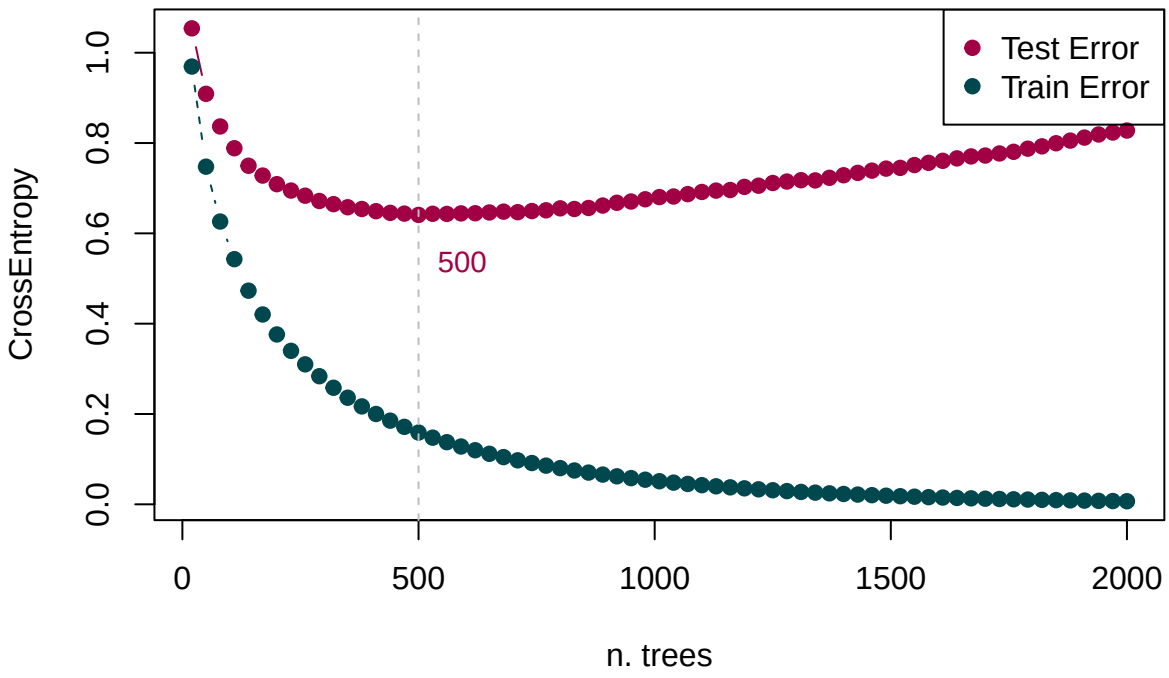

```
##                                                  var    rel.inf
## Attendance                                Attendance 30.2991804
## Hours_Studied                          Hours_Studied 23.1036236
## Previous_Scores                      Previous_Scores 10.2086155
## Tutoring_Sessions                  Tutoring_Sessions  4.8388459
## Access_to_Resources              Access_to_Resources  4.5601296
## Parental_Involvement            Parental_Involvement  4.4876980
## Parental_Education_Level  Parental_Education_Level  2.6370557
## Family_Income                        Family_Income  2.4920223
## Motivation_Level                  Motivation_Level  2.4075285
## Peer_Influence                      Peer_Influence  2.4034831
## Distance_from_Home              Distance_from_Home  2.3112820
## Sleep_Hours                            Sleep_Hours  2.1161358
## Physical_Activity                Physical_Activity  2.0551103
## Teacher_Quality                    Teacher_Quality  1.9339373
```

```
## Extracurricular_Activities Extracurricular_Activities  1.2022736
## Learning_Disabilities           Learning_Disabilities  0.8657044
## Internet_Access                       Internet_Access  0.8364042
## Gender                                         Gender  0.6875161
## School_Type                                School_Type  0.5534537
```

Dal primo modello è stato possibile calcolare l'errore sia sul dataset di training che su quello di test. Per farlo, è stato necessario trasformare i risultati previsti, originariamente espressi come logaritmi delle probabilità, in probabilità vere e proprie comprese tra 0 e 1 per ciascuna classe, assicurandosi che la loro somma fosse pari a 1. A tal fine è stata implementata la funzione softmax. Poiché le etichette erano rappresentate in formato one-hot encoding, per il calcolo dell'errore è stata utilizzata la funzione di perdita cross entropy, confrontando i valori predetti con quelli attesi. Per ciascun numero di alberi (da 20 a 2000, con incrementi di 30), è stata calcolata la media dell'errore su tutte le previsioni, sia per il dataset di training che per quello di test. I risultati sono stati poi visualizzati nel grafico riportato alla fine del blocco di codice. Dal grafico si osserva che l'errore sul test set (curva rossa) raggiunge un minimo attorno ai 500 alberi, per poi aumentare progressivamente. Al contrario, l'errore sul training set (curva nera) mostra un andamento decrescente e monotono, segno che il modello continua ad apprendere anche oltre i 500 alberi. Questo comportamento evidenzia un chiaro caso di overfitting: il modello si adatta sempre meglio ai dati di training, ma a scapito della sua capacità di generalizzazione sui dati di test.

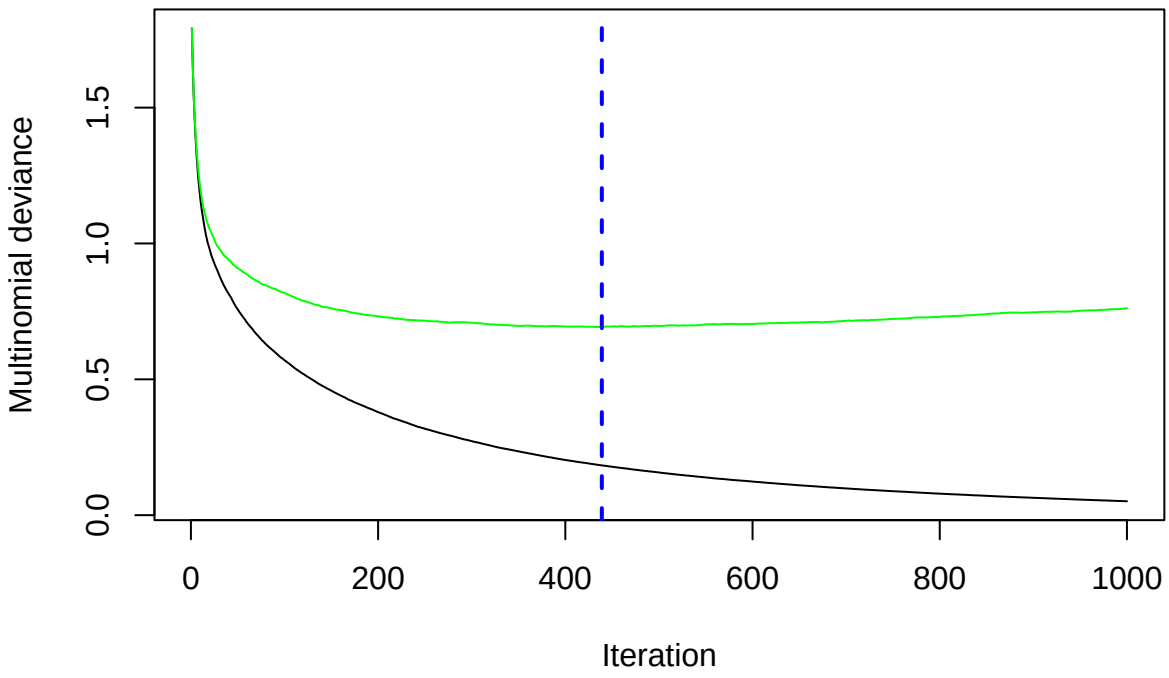
```r
# Calcola l'errore di tipo CrossEntropy
calculate_rowwise_error <- function(real_matrix, predicted_matrix) {
  real_matrix <- as.matrix(real_matrix)
  predicted_matrix <- as.matrix(predicted_matrix)
  errors <- -rowSums(real_matrix * log(predicted_matrix))
  return(errors)
}

# Funzione softmax
softmax <- function(logits) {
  stable_logits <- logits - max(logits) # Stabilità numerica
  exp_logits <- exp(stable_logits)
  return(exp_logits / sum(exp_logits))
}

n.trees.seq <- seq(from = 20, to = 2000, by = 30)

test_matrix_onehot <- model.matrix(~ test$Categorical_Exam_Score - 1)
train_matrix_onehot <- model.matrix(~ train$Categorical_Exam_Score - 1)

Yhat_logits_array_test <- predict(boost.1,
                            newdata = test,
                            n.trees = n.trees.seq
                            )

Yhat_logits_array_train <- predict(boost.1, n.trees = n.trees.seq)



error_matrix_test <- matrix(NA, nrow = 1, ncol = length(n.trees.seq),
                        dimnames = list("MeanCrossEntropy",
                                        paste0("nTrees_", n.trees.seq)))
error_matrix_train <- matrix(NA, nrow = 1, ncol = length(n.trees.seq),
                        dimnames = list("MeanCrossEntropy",
                                        paste0("nTrees_", n.trees.seq)))
```

```r
for (i in 1:length(n.trees.seq)) {
  current_n_trees <- n.trees.seq[i]

  current_logits_test <- Yhat_logits_array_test[,,i]
  current_probs_test <- t(apply(current_logits_test, 1, softmax))

  rowwise_errors_test <- calculate_rowwise_error(test_matrix_onehot, current_probs_test)
  mean_error_test <- mean(rowwise_errors_test)
  error_matrix_test[1, i] <- mean_error_test

  current_logits_train <- Yhat_logits_array_train[,,i]
  current_probs_train <- t(apply(current_logits_train, 1, softmax))

  rowwise_errors_train <- calculate_rowwise_error(train_matrix_onehot, current_probs_train)
  mean_error_train <- mean(rowwise_errors_train)
  error_matrix_train[1, i] <- mean_error_train

}


best_n_trees <- n.trees.seq[which.min(error_matrix_test[1, ])]

matplot(n.trees.seq, cbind(error_matrix_test[1, ], error_matrix_train[1, ]), pch=19 , col=c("#A20045","#
abline(v = best_n_trees, col = "gray", lty = "dashed", lwd = 1)
text(x = best_n_trees, y = mean(par("usr")[3:4]), labels = best_n_trees, pos = 4, col = "#A20045", cex =
legend("topright",legend=c("Test Error","Train Error"),pch=19, col=c("#A20045","#00484D"))
```



Successivamente è stato addestrato un nuovo modello con parametri analoghi a quelli del primo, ad eccezione del numero di alberi, ridotto a 1000. Su questo secondo modello di boosting è stata utilizzata la funzione perf della libreria gbm, che consente di individuare graficamente il numero ottimale di alberi tramite validazione incrociata, minimizzando l'errore. Il risultato è un grafico simile a quello precedente, ma in questo caso viene evidenziato il punto corrispondente al numero ottimale di alberi. Tale valore, salvato nella variabile best.nt, è pari a 439 molto vicino a quanto osservato in precedenza utilizzando il dataset di test. Ricordiamo che il valore di 439 alberi è ottenuto attraverso la cross validation per tanto è meno affidabile di quello precedente basato sul dataset di test in quanto la suddivisione per la valutazione avviene sullo stesso dataset di train.

```
set.seed(123)
boost.2 <- gbm(Categorical_Exam_Score ~ ., data = train, distribution = "multinomial", n.trees = 1000, i

best.nt = gbm.perf(boost.2, method = "cv", plot.it = TRUE)
```




```
best.nt
```

```
## [1] 439
```

Successivamente è stata avviata una procedura di fine-tuning dei parametri, con l'obiettivo di individuare non solo il numero ottimale di alberi, ma anche il valore ideale del parametro depth. Utilizzando il codice mostrato, sono stati testati tutti i valori di depth compresi tra 1 e 7. Per ciascun valore, sono stati calcolati sia l'errore sul dataset di training e su quello di test, sia il numero ottimale di alberi tramite validazione incrociata. Il risultato è una matrice contenente tutte le combinazioni testate, con i relativi valori di errore e numero di alberi ottimale. Da questa matrice è stato possibile individuare la combinazione di parametri che minimizza l'errore sul dataset di test. In basso è riportata la tabella con i risultati ottenuti.

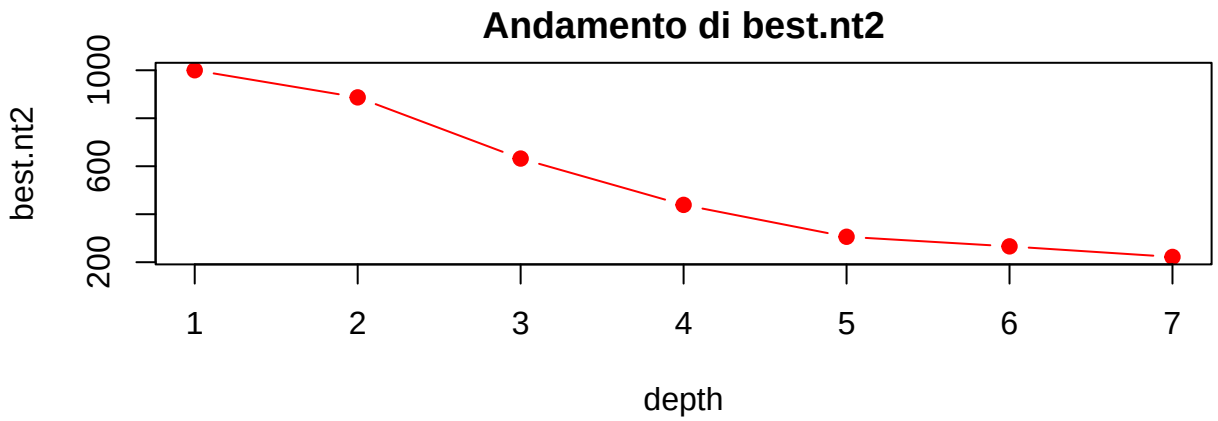
```
myd <- 1:7
test_matrix_onehot <- model.matrix(~ test$Categorical_Exam_Score - 1)
train_matrix_onehot <- model.matrix(~ train$Categorical_Exam_Score - 1)

myEval <- sapply(myd , function(x) {
  set.seed(123)
  boost.3 <- gbm(Categorical_Exam_Score ~ ., data = train, distribution = "multinomial", n.trees = 1000

  # Numero ottimale di alberi
  best.nt2 <- gbm.perf(boost.3, method = "cv", plot.it = FALSE)


  Yhat_test <- predict(boost.3, newdata = test,n.trees = best.nt2)
  Yhat_test = t(apply(Yhat_test, 1, softmax))

  Yhat_train <- predict(boost.3, n.trees = best.nt2)
```

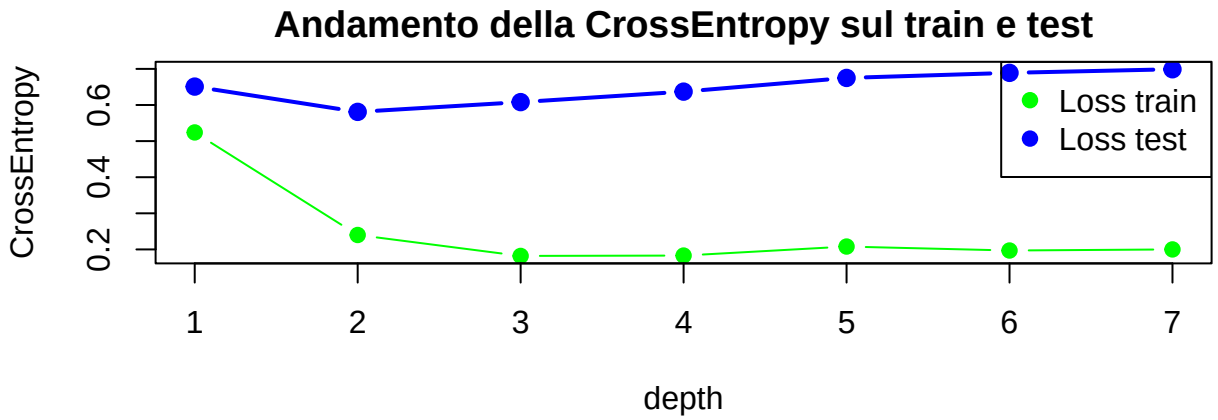
```
  Yhat_train = t(apply(Yhat_train, 1, softmax))

  ### Crossentropy
  loss_train = mean(calculate_rowwise_error(train_matrix_onehot, Yhat_train))
  loss_test = mean(calculate_rowwise_error(test_matrix_onehot, Yhat_test))

  return(c(as.integer(best.nt2), round(loss_train,3), round(loss_test,3)))}
  )

myEval
```

```
##          [,1]    [,2]    [,3]    [,4]    [,5]    [,6]    [,7]
## [1,] 1000.000 887.000 632.000 439.000 306.000 266.000 222.000
## [2,]    0.524   0.240   0.182   0.183   0.208   0.197   0.200
## [3,]    0.651   0.581   0.608   0.637   0.675   0.689   0.699
```

Per una migliore visualizzazione dei risultati, sono stati realizzati due grafici. Il primo mostra come varia il numero ottimale di alberi determinato tramite validazione incrociata in funzione del parametro depth. Mentre il secondo mostra l'errore del modello calcolato sul train e test. In basso sono riportati i valori ottimali di depth e di numero di alberi che minimizzano l'errore.

```
myEval_invertita <- t(myEval)

par(mfrow = c(2, 1), mar = c(4, 4, 2, 1))

# Primo grafico: best.nt2
plot(1:7, myEval_invertita[,1], type = "b", pch = 19, col = "red",
     xlab = "depth", ylab = "best.nt2",
     main = "Andamento di best.nt2")

plot(1:7, myEval_invertita[,2], type = "b", pch = 19, col = "green",
     xlab = "depth", ylab = "CrossEntropy",
     main = "Andamento della CrossEntropy sul train e test",
     ylim = range(c(myEval_invertita[,2], myEval_invertita[,3])))  # Impostare limiti y
lines(1:7, myEval_invertita[,3], type = "b", pch = 19, col = "blue", lwd = 2)

# Legenda
legend("topright",
       legend = c("Loss train", "Loss test"),
       col    = c("green", "blue"),
       pch    = 19)
```






```
indice_minimo <- which.min(myEval_invertita[,3])
best_nt= myEval_invertita[,1][indice_minimo]
best_depth =  indice_minimo
best_depth
```

```
## [1] 2
```

```
best_nt
```

```
## [1] 887
```

Si può osservare come il numero ottimale di alberi tenda a diminuire all'aumentare del parametro depth. Questo comportamento è probabilmente dovuto al fatto che valori elevati di depth aumentano la complessità del modello, rendendolo più soggetto a overfitting. Di conseguenza, un numero inferiore di alberi risulta sufficiente (e talvolta necessario) per contrastare questo effetto e mantenere la generalizzazione. Il secondo grafico mostra l'andamento dell'errore del modello al variare del parametro depth, distinguendo tra il dataset di test (curva blu) e quello di training (curva verde). Per ogni valore di depth, è stato utilizzato il corrispondente numero ottimale di alberi trovato in precedenza. Dal grafico emerge che l'errore sul dataset di test raggiunge il minimo per depth = 2, per poi aumentare, indicando un peggioramento della capacità di generalizzazione. Al contrario, come già osservato nei casi precedenti, la loss sul dataset di training decresce in modo monotono, suggerendo un crescente overfitting con l'aumentare della profondità. Tuttavia, è importante notare che mentre il depth è stato scelto in base alla minima loss sul dataset di test, il numero ottimale di alberi è stato determinato tramite validazione incrociata. Questo significa che i due valori ottimali non sono perfettamente coerenti tra loro, e il numero di alberi potrebbe non essere il migliore possibile in corrispondenza del depth ottimale. Per evitare di appesantire ulteriormente l'analisi e i tempi di calcolo, si è deciso di non ripetere la procedura di ottimizzazione del numero di alberi specificamente per il valore ottimale di depth. Una tale operazione avrebbe probabilmente portato a una stima più accurata, ma a fronte di una maggiore complessità computazionale.

Utilizzando i parametri ottimali identificati in precedenza, è stato addestrato il modello finale di boosting, denominato boost.final.

```
set.seed(123)
boost.final <- gbm(Categorical_Exam_Score ~ ., data = train, distribution = "multinomial", n.trees = bes
```

Da questo modello è stata ricavata la matrice di confusione sia sual dataset di train che di test, e sia altri parametrid i interesse statistico come l'accuracy, Specificity e la Balanced Accuracy. Dai risulatati possiamo osservare come il modella abbia un valore di accuracy molto più elevato sul dataset di train che su quello di test, anche se su quest ultimo raggiunge comunque un accuratezza di quasi l'80%, indicando una discreta capacità predittiva nonostante l'overfitting. Per quanto riguarda Specificity e la Balanced Accuracy queste risultano molto variabili fra le classi. Nel caso della classe quasi sufficiente i valori sembrano discreti mentre per la classe Altro queste assumono valore abbastanza bassi.

Dal modello sono state estratte le matrici di confusione e calcolate le principali metriche: accuracy, specificity e balanced accuracy, sia sul training set sia sul test set. Vediamo alcune considerazioni:

- Training set:
  - Accuracy complessiva: 96,58%
  - Balanced accuracy per classe: da 98,70% (Quasi-Sufficiente) a 88,68% (Alto)
  - Specificity molto elevate per tutte le classi (ad es. 99,94% per Quasi-Sufficiente, 99,97% per Alto)
- Test set:
  - Accuracy complessiva: 79,92%
  - Balanced accuracy per classe: varia da 82,31% (Quasi-Sufficiente) a 67,66% (Alto)
  - Specificity alta anche in test (ad es. 99,43% per Quasi-Sufficiente, 98,81% per Alto)

Nonostante si possa notare dell'overfitting sui dati di train, il modello mantiene quasi l'80% di accuratezza sui dati di test, confermando una discreta capacità predittiva. Tuttavia, le metriche di balanced accuracy e specificity mostrano una forte variabilità tra le classi: La classe Quasi-Sufficiente presenta un'ottima balanced accuracy in training (98,70%), che scende a 82,31% in test, con specificity vicina al 99%.

La classe Alto è invece la più critica: nonostante specificity elevata (98,81%), la sensibilità sui dati di test è solo 36,52%, portando la balanced accuracy al 67,66%.

```
# Effettua le previsioni sul set di training
predictions_train_probs <- predict(boost.final, newdata = train, n.trees = best_nt, type = "response")
predictions_train <- apply(predictions_train_probs, 1, which.max)
predicted_classes_train <- levels(train$Categorical_Exam_Score)[predictions_train]

# Effettua le previsioni sul set di test
predictions_test_probs <- predict(boost.final, newdata = test, n.trees = best_nt, type = "response")
predictions_test <- apply(predictions_test_probs, 1, which.max)
predicted_classes_test <- levels(test$Categorical_Exam_Score)[predictions_test]

# Confusion matrix sul set di training
confusion_matrix_train <- confusionMatrix(factor(predicted_classes_train, levels = levels(train$Categor
print("Confusion Matrix - Training Set:")
```

```
## [1] "Confusion Matrix - Training Set:"
```

```
print(confusion_matrix_train)
```

```
## Confusion Matrix and Statistics
##
##                    Reference
## Prediction          Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
##   Quasi-Sufficiente               154     1           0     0          0    1
##   Basso                             4   545           3     0          0    4
```

```
##   Medio-Basso                      0    22        1057    16          0    6
##   Medio                            0     0          14   945         24    5
##   Medio-Alto                       0     0           0     2        402   10
##   Alto                             0     0           0     0          1   89
##
## Overall Statistics
##
##                Accuracy : 0.9658
##                  95% CI : (0.959, 0.9717)
##     No Information Rate : 0.325
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9548
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Quasi-Sufficiente Class: Basso Class: Medio-Basso
## Sensitivity                           0.97468       0.9595             0.9842
## Specificity                           0.99936       0.9960             0.9803
## Pos Pred Value                        0.98718       0.9802             0.9600
## Neg Pred Value                        0.99873       0.9916             0.9923
## Prevalence                            0.04781       0.1719             0.3250
## Detection Rate                        0.04660       0.1649             0.3198
## Detection Prevalence                  0.04720       0.1682             0.3331
## Balanced Accuracy                     0.98702       0.9777             0.9822
##                      Class: Medio Class: Medio-Alto Class: Alto
## Sensitivity                0.9813            0.9415     0.77391
## Specificity                0.9816            0.9958     0.99969
## Pos Pred Value             0.9565            0.9710     0.98889
## Neg Pred Value             0.9922            0.9914     0.99191
## Prevalence                 0.2914            0.1292     0.03480
## Detection Rate             0.2859            0.1216     0.02693
## Detection Prevalence       0.2989            0.1253     0.02723
## Balanced Accuracy          0.9815            0.9686     0.88680
```

```r
# Calcola l'accuracy sul set di test
confusion_matrix_test <- confusionMatrix(factor(predicted_classes_test, levels = levels(test$Categorical
print("Confusion Matrix - Test Set:")
```

```
## [1] "Confusion Matrix - Test Set:"
```

```r
print(confusion_matrix_test)
```

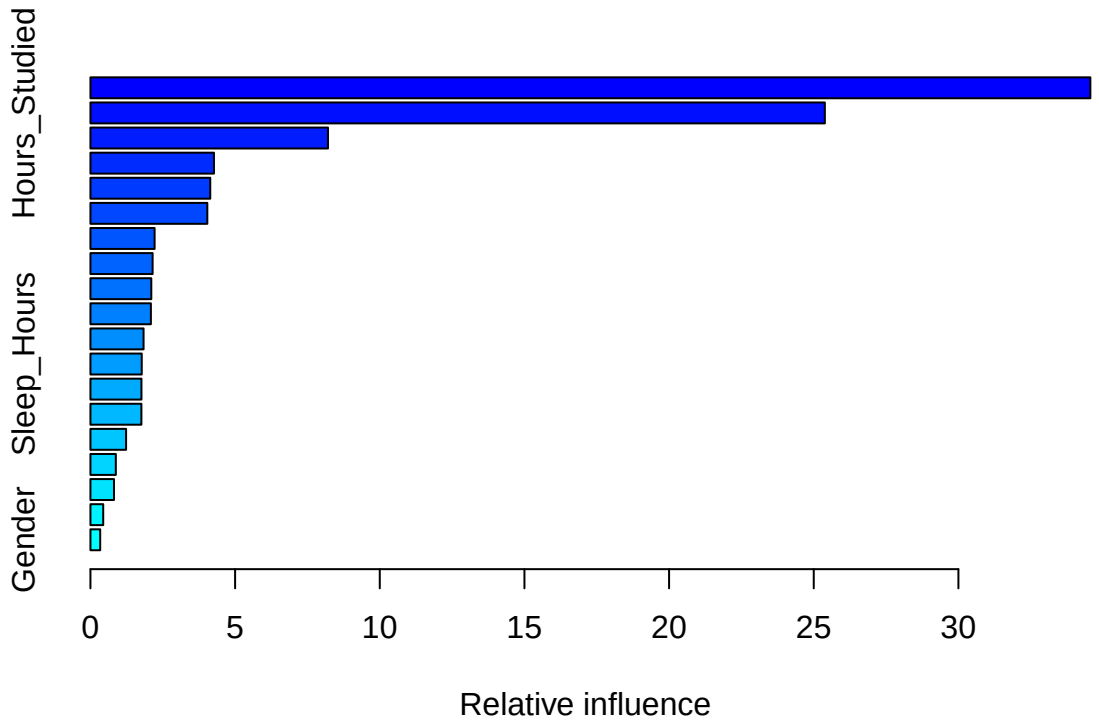
```
## Confusion Matrix and Statistics
##
##                    Reference
## Prediction          Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
##   Quasi-Sufficiente               103    14           1     0          0    3
##   Basso                            55   447          38     0          0    6
##   Medio-Basso                       0   107         948   101          0   11
##   Medio                             0     0          81   820        119    3
##   Medio-Alto                        0     0           0    36        279   50
##   Alto                              0     0           5     5         28   42
```

```
##
## Overall Statistics
##
##                Accuracy : 0.7992
##                  95% CI : (0.7851, 0.8128)
##     No Information Rate : 0.325
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7321
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Quasi-Sufficiente Class: Basso Class: Medio-Basso
## Sensitivity                           0.65190       0.7870             0.8835
## Specificity                           0.99427       0.9638             0.9017
## Pos Pred Value                        0.85124       0.8187             0.8123
## Neg Pred Value                        0.98271       0.9561             0.9415
## Prevalence                            0.04785       0.1720             0.3250
## Detection Rate                        0.03119       0.1354             0.2871
## Detection Prevalence                  0.03664       0.1654             0.3534
## Balanced Accuracy                     0.82309       0.8754             0.8926
##                      Class: Medio Class: Medio-Alto Class: Alto
## Sensitivity                0.8524           0.65493     0.36522
## Specificity                0.9132           0.97010     0.98808
## Pos Pred Value             0.8016           0.76438     0.52500
## Neg Pred Value             0.9377           0.94995     0.97734
## Prevalence                 0.2913           0.12901     0.03483
## Detection Rate             0.2483           0.08449     0.01272
## Detection Prevalence       0.3098           0.11054     0.02423
## Balanced Accuracy          0.8828           0.81251     0.67665
```

Effettuando il summary del modello possiamo osservare l'importanza che il modello ha dato a ciascuna variabile utilizzata. Si può notare che rispetto a prima è crescita l'importanza delle variabili Attendance e Hours_Studied rispetto alla altre. Tenendo conto di queste considerazioni unite a quelle precedenti si può pensare che la difficoltà del modello nel determinare le classi sembra sia da attribuire ai pochi dati nelle classi più lontane dalla media e sia dalla dipendenza lineare che vige tra le variabili più importanti utilizzate dal modello e Exam_Score.

Il summary del modello mette in luce l'importanza attribuita a ciascuna variabile: in particolare, Attendance e Hours_Studied hanno guadagnato peso rispetto alle altre feature e ai valori visti in precedenza. Alla luce di queste evidenze e dei risultati precedenti, si può ipotizzare che le maggiori difficoltà del modello nel classificare correttamente le classi "più estreme" dipendano da due fattori principali: - Scarsa rappresentatività dei dati per le categorie lontane dalla media, che rende il modello meno accurato nel riconoscerle. - Forte correlazione lineare tra le variabili più influenti (Attendance e Hours_Studied) e l'obiettivo (Exam_Score), elemento che può complicare l'apprendimento di pattern distintivi soprattutto nelle classi meno frequenti.

```
summary(boost.final)
```




```
##                                                  var    rel.inf
## Attendance                                Attendance 34.5603677
## Hours_Studied                          Hours_Studied 25.3830110
## Previous_Scores                      Previous_Scores  8.2101842
## Tutoring_Sessions                  Tutoring_Sessions  4.2694557
## Access_to_Resources              Access_to_Resources  4.1411103
## Parental_Involvement            Parental_Involvement  4.0396447
## Family_Income                          Family_Income  2.2180240
## Parental_Education_Level    Parental_Education_Level  2.1493356
## Peer_Influence                        Peer_Influence  2.1032345
## Distance_from_Home                Distance_from_Home  2.0893412
## Motivation_Level                    Motivation_Level  1.8356807
## Sleep_Hours                              Sleep_Hours  1.7736258
## Physical_Activity                  Physical_Activity  1.7635919
## Teacher_Quality                      Teacher_Quality  1.7612849
## Learning_Disabilities          Learning_Disabilities  1.2322316
## Extracurricular_Activities Extracurricular_Activities  0.8777931
## Internet_Access                      Internet_Access  0.8136140
## School_Type                          School_Type  0.4414407
## Gender                                    Gender  0.3370282
```

CART

```
require(rpart)
```

```
## Caricamento del pacchetto richiesto: rpart
```

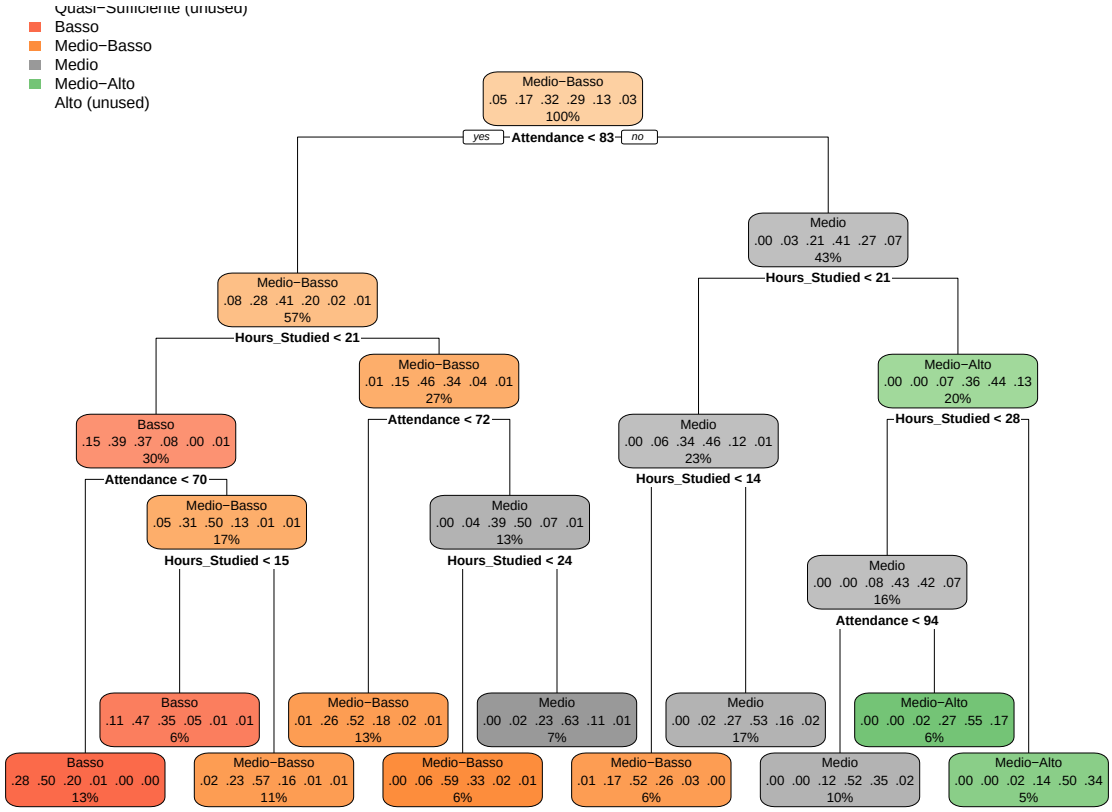
```r
library(rpart.plot)
```

```
## Warning: il pacchetto 'rpart.plot' è stato creato con R versione 4.4.3
```

```r
mod0<- rpart::rpart(Categorical_Exam_Score~., data= train, method="class")
mod0
```

```
## n= 3305
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 3305 2231 Medio-Basso (0.048 0.17 0.32 0.29 0.13 0.035)
##    2) Attendance< 82.5 1876 1104 Medio-Basso (0.083 0.28 0.41 0.2 0.021 0.0075)
##      4) Hours_Studied< 20.5 993  603 Basso (0.15 0.39 0.37 0.077 0.003 0.006)
##        8) Attendance< 69.5 425  211 Basso (0.28 0.5 0.2 0.0094 0 0.0047) *
##        9) Attendance>=69.5 568  286 Medio-Basso (0.055 0.31 0.5 0.13 0.0053 0.007)
##         18) Hours_Studied< 14.5 192  101 Basso (0.11 0.47 0.35 0.052 0.0052 0.0052) *
##         19) Hours_Studied>=14.5 376  161 Medio-Basso (0.024 0.23 0.57 0.16 0.0053 0.008) *
##      5) Hours_Studied>=20.5 883  479 Medio-Basso (0.0068 0.15 0.46 0.34 0.041 0.0091)
##       10) Attendance< 71.5 443  212 Medio-Basso (0.014 0.26 0.52 0.18 0.016 0.009) *
##       11) Attendance>=71.5 440  222 Medio (0 0.036 0.39 0.5 0.066 0.0091)
##         22) Hours_Studied< 23.5 199   82 Medio-Basso (0 0.06 0.59 0.33 0.015 0.005) *
##         23) Hours_Studied>=23.5 241   89 Medio (0 0.017 0.23 0.63 0.11 0.012) *
##    3) Attendance>=82.5 1429  841 Medio (0.0014 0.034 0.21 0.41 0.27 0.071)
##      6) Hours_Studied< 20.5 753  408 Medio (0.0027 0.064 0.34 0.46 0.12 0.013)
##       12) Hours_Studied< 13.5 206   99 Medio-Basso (0.0097 0.17 0.52 0.26 0.029 0.0049) *
##       13) Hours_Studied>=13.5 547  256 Medio (0 0.022 0.27 0.53 0.16 0.016) *
##      7) Hours_Studied>=20.5 676  379 Medio-Alto (0 0 0.067 0.36 0.44 0.13)
##       14) Hours_Studied< 27.5 517  297 Medio (0 0 0.081 0.43 0.42 0.072)
##         28) Attendance< 93.5 327  158 Medio (0 0 0.12 0.52 0.35 0.015) *
##         29) Attendance>=93.5 190   86 Medio-Alto (0 0 0.016 0.27 0.55 0.17) *
##       15) Hours_Studied>=27.5 159   80 Medio-Alto (0 0 0.019 0.14 0.5 0.34) *
```

```r
rpart.plot::rpart.plot(mod0)
```




```
caret::confusionMatrix(table(predicted = predict(mod0, type = "class"), actual = train$Categorical_Exam
```

```
## Confusion Matrix and Statistics
##
##                  actual
## predicted          Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
##   Quasi-Sufficiente                 0     0           0     0          0    0
##   Basso                           141   305         153    14          1    3
##   Medio-Basso                      17   247         670   263         18    9
##   Medio                             0    16         245   612        225   17
##   Medio-Alto                        0     0           6    74        183   86
##   Alto                              0     0           0     0          0    0
##
## Overall Statistics
##
##                Accuracy : 0.5356
##                  95% CI : (0.5184, 0.5527)
##     No Information Rate : 0.325
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3686
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

```
##                      Class: Quasi-Sufficiente Class: Basso Class: Medio-Basso
## Sensitivity                           0.00000      0.53697             0.6238
## Specificity                           1.00000      0.88601             0.7517
## Pos Pred Value                            NaN      0.49433             0.5474
## Neg Pred Value                        0.95219      0.90216             0.8059
## Prevalence                            0.04781      0.17186             0.3250
## Detection Rate                        0.00000      0.09228             0.2027
## Detection Prevalence                  0.00000      0.18669             0.3703
## Balanced Accuracy                     0.50000      0.71149             0.6878
##                      Class: Medio Class: Medio-Alto Class: Alto
## Sensitivity                0.6355           0.42857      0.0000
## Specificity                0.7852           0.94232      1.0000
## Pos Pred Value             0.5489           0.52436         NaN
## Neg Pred Value             0.8397           0.91746      0.9652
## Prevalence                 0.2914           0.12920      0.0348
## Detection Rate             0.1852           0.05537      0.0000
## Detection Prevalence       0.3374           0.10560      0.0000
## Balanced Accuracy          0.7104           0.68545      0.5000
```

```
mytab <- table(predicted = predict(mod0, type = "class"), actual = train$Categorical_Exam_Score)
mctest = (abs(mytab[2,1] - mytab[1,2]) -1)^2/(mytab[2,1] + mytab[1,2])
mctest
```

```
## [1] 139.0071
```

```
pchisq(mctest,1, lower.tail = FALSE)
```

```
## [1] 4.388793e-32
```

```
mcnemar.test(table(predicted = predict(mod0, type = "class"), actual = train$Categorical_Exam_Score), c
```

```
##
##  McNemar's Chi-squared test
##
## data:  table(predicted = predict(mod0, type = "class"), actual = train$Categorical_Exam_Score)
## McNemar's chi-squared = NaN, df = 15, p-value = NA
```

Use of the loss matrix

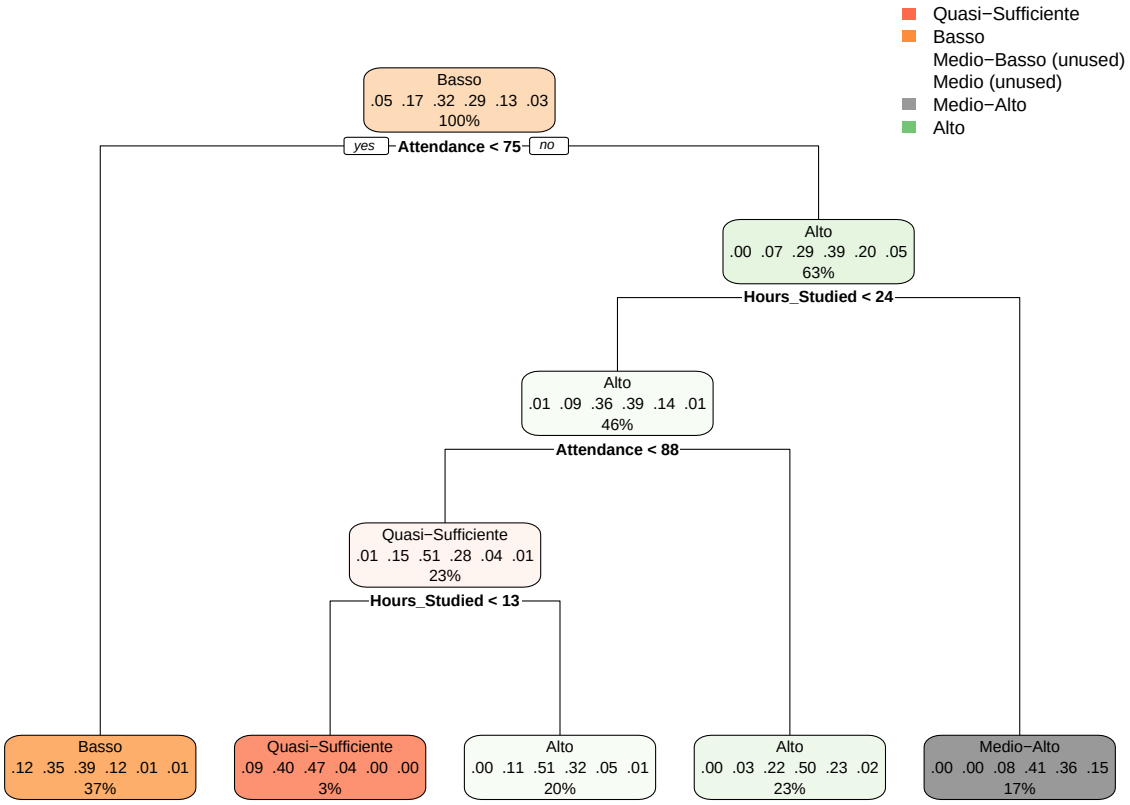
```
m = matrix(c(0,0,0,0,0,1.0,
             0,0,0,0,0.2,0,
             0,0,0,0.1,0,0,
             0,0,0.1,0,0,0,
             0,0.2,0,0,0,0,
             1.0,0,0,0,0,0),
           byrow=TRUE, nrow=6)
m
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0  0.0  0.0  0.0  0.0    1
## [2,]    0  0.0  0.0  0.0  0.2    0
## [3,]    0  0.0  0.0  0.1  0.0    0
## [4,]    0  0.0  0.1  0.0  0.0    0
## [5,]    0  0.2  0.0  0.0  0.0    0
## [6,]    1  0.0  0.0  0.0  0.0    0
```

```
mod0_loss<- rpart::rpart(Categorical_Exam_Score~., data= train, method="class", parms=list(loss=m))
mod0_loss
```

```
## n= 3305
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 3305 85.4 Basso (0.048 0.17 0.32 0.29 0.13 0.035)
##    2) Attendance< 74.5 1217  3.2 Basso (0.12 0.35 0.39 0.12 0.013 0.0066) *
##    3) Attendance>=74.5 2088 10.0 Alto (0.0048 0.067 0.29 0.39 0.2 0.051)
##      6) Hours_Studied< 23.5 1524 10.0 Alto (0.0066 0.091 0.36 0.39 0.14 0.014)
##       12) Attendance< 87.5 773  5.0 Quasi-Sufficiente (0.013 0.15 0.51 0.28 0.039 0.0065)
##         24) Hours_Studied< 12.5 109  0.0 Quasi-Sufficiente (0.092 0.4 0.47 0.037 0 0) *
##         25) Hours_Studied>=12.5 664  0.0 Alto (0 0.11 0.51 0.32 0.045 0.0075) *
##       13) Attendance>=87.5 751  0.0 Alto (0 0.027 0.22 0.5 0.23 0.023) *
##      7) Hours_Studied>=23.5 564  0.0 Medio-Alto (0 0 0.078 0.41 0.36 0.15) *
```

```
rpart.plot::rpart.plot(mod0_loss)
```




```
caret::confusionMatrix(table(predicted = predict(mod0_loss, type = "class"), actual =  train$Categorical
```

```
## Confusion Matrix and Statistics
##
##                   actual
## predicted           Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
##   Quasi-Sufficiente                10    44          51     4          0    0
##   Basso                           148   429         476   140         16    8
##   Medio-Basso                       0     0           0     0          0    0
##   Medio                             0     0           0     0          0    0
```

```
##   Medio-Alto                        0     0          44   230        205   85
##   Alto                              0    95         503   589        206   22
##
## Overall Statistics
##
##                Accuracy : 0.2015
##                  95% CI : (0.188, 0.2156)
##     No Information Rate : 0.325
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.111
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Quasi-Sufficiente Class: Basso Class: Medio-Basso
## Sensitivity                          0.063291       0.7553              0.000
## Specificity                          0.968541       0.7121              1.000
## Pos Pred Value                       0.091743       0.3525                NaN
## Neg Pred Value                       0.953692       0.9334              0.675
## Prevalence                           0.047806       0.1719              0.325
## Detection Rate                       0.003026       0.1298              0.000
## Detection Prevalence                 0.032980       0.3682              0.000
## Balanced Accuracy                    0.515916       0.7337              0.500
##                      Class: Medio Class: Medio-Alto Class: Alto
## Sensitivity                0.0000           0.48009    0.191304
## Specificity                1.0000           0.87526    0.563323
## Pos Pred Value                NaN           0.36348    0.015548
## Neg Pred Value             0.7086           0.91901    0.950794
## Prevalence                 0.2914           0.12920    0.034796
## Detection Rate             0.0000           0.06203    0.006657
## Detection Prevalence       0.0000           0.17065    0.428139
## Balanced Accuracy          0.5000           0.67768    0.377314
```

Conclusioni