El hábito de los jovenes bajo el Machine Learning

Juan Ramón Baeza Borrego

2025-07-18

Índice

Generación de datos

```
data <- source("Generación Dataset.R")
```

```
## 'data.frame':    200 obs. of  11 variables:
##  $ Age                     : int  22 22 29 21 25 30 24 23 25 28 ...
##  $ Social_Media_Hours      : num  0.5 0.4 4.4 3.8 4.1 4.9 0.5 0.5 4 3.9 ...
##  $ Physical_Exercise_Hours : num  0.2 1.4 2.1 0.3 3 0.8 0.1 2.1 2.4 1.1 ...
##  $ Daily_Study_Hours       : num  3.8 3.2 0.8 1.7 3.5 1 0.6 2.6 2.1 5.1 ...
##  $ Stress_Level            : num  6 5 6 4 7 5 5 5 8 2 ...
##  $ Monthly_Expenses        : num  1520 1809 1346 1687 1552 ...
##  $ Income_Level            : num  1808 3075 3003 2696 2057 ...
##  $ Num_Online_Courses_Taken: int  2 1 0 8 2 1 8 9 10 9 ...
##  $ Favorite_Genre          : Factor w/ 3 levels "Action","Comedy",..: 2 1 2 1 2 3 1 2
1 3 ...
##  $ Career_Track            : Factor w/ 4 levels "DataScientist",..: 1 2 1 4 4 2 2 4 4
4 ...
##  $ Binary_Status           : Factor w/ 2 levels "No","Yes": 1 1 2 2 1 2 1 1 2 1 ...
```

Como pobemos observar, tenemos 200 observaciones y 11 variables principales. Quedando el siguiente principio del dataset.

```
head(student_data)
```

```
##   Age Social_Media_Hours Physical_Exercise_Hours Daily_Study_Hours Stress_Level
## 1  22               0.5                     0.2              3.8            6
## 2  22               0.4                     1.4              3.2            5
## 3  29               4.4                     2.1              0.8            6
## 4  21               3.8                     0.3              1.7            4
## 5  25               4.1                     3.0              3.5            7
## 6  30               4.9                     0.8              1.0            5
##   Monthly_Expenses Income_Level Num_Online_Courses_Taken Favorite_Genre
## 1             1520         1808                        2         Comedy
## 2             1809         3075                        1         Action
## 3             1346         3003                        0         Comedy
## 4             1687         2696                        8         Action
## 5             1552         2057                        2         Comedy
## 6             1490         1814                        1          Drama
##    Career_Track Binary_Status
## 1 DataScientist            No
## 2    Freelancer            No
## 3 DataScientist           Yes
## 4        WebDev           Yes
## 5        WebDev            No
## 6    Freelancer           Yes
```

1. Análisis de Componentes Principales (PCA)

Realizaremos un PCA sobre las variables numéricas del conjunto de datos (por ejemplo, horas de uso de redes sociales, horas de ejercicio, niveles de estrés, gastos mensuales, ingresos, etc.).

```
library(factoextra)
```

```
## Loading required package: ggplot2
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
data_numericos <- student_data[, sapply(student_data, is.numeric)]
```

Las variables númericas por ende son:

```
head(data_numericos)
```

```
##   Age Social_Media_Hours Physical_Exercise_Hours Daily_Study_Hours Stress_Level
## 1  22               0.5                     0.2              3.8            6
## 2  22               0.4                     1.4              3.2            5
## 3  29               4.4                     2.1              0.8            6
## 4  21               3.8                     0.3              1.7            4
## 5  25               4.1                     3.0              3.5            7
## 6  30               4.9                     0.8              1.0            5
##   Monthly_Expenses Income_Level Num_Online_Courses_Taken
## 1             1520         1808                        2
## 2             1809         3075                        1
## 3             1346         3003                        0
## 4             1687         2696                        8
## 5             1552         2057                        2
## 6             1490         1814                        1
```

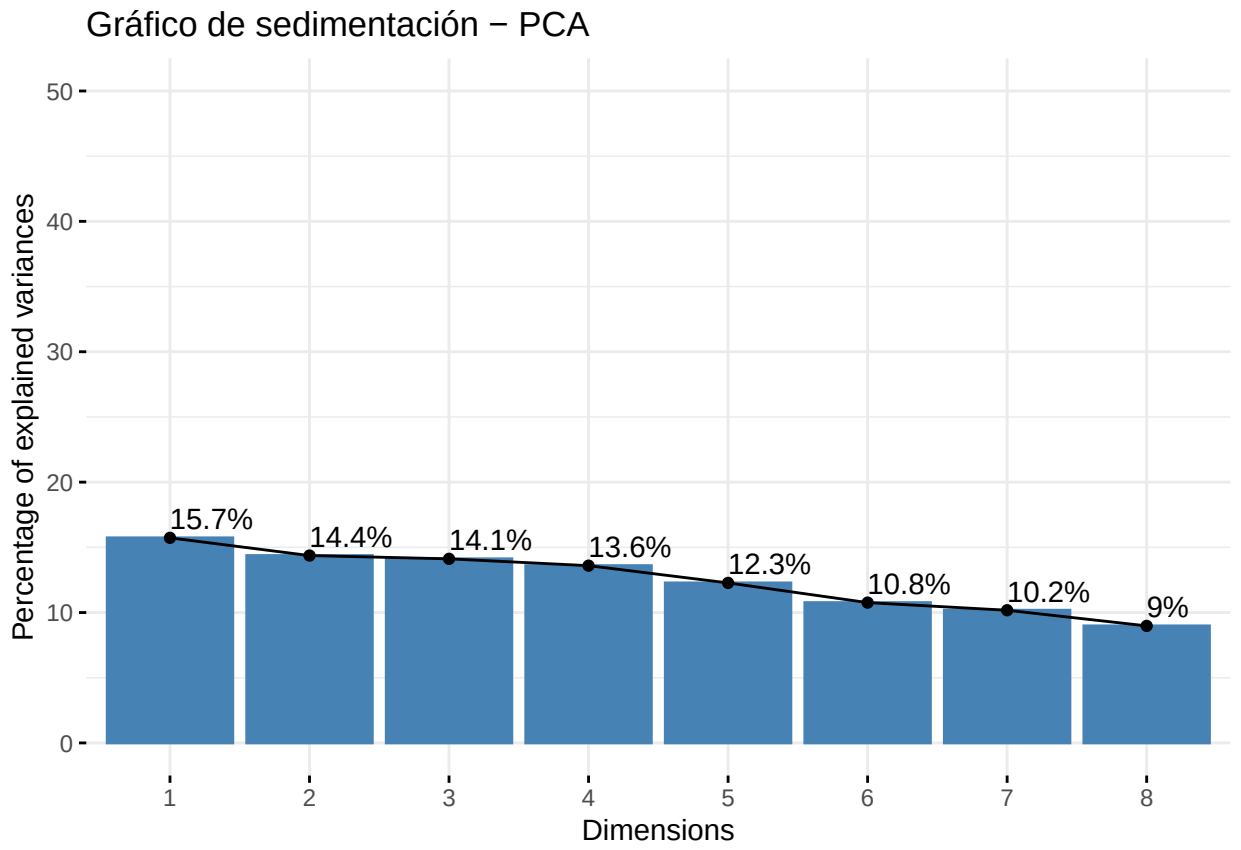
A continuación vamos a estandarizar los datos como requisito para hacer nuestro PCA.

```
datos_esc <- scale(data_numericos)
PCA <- prcomp(datos_esc, center = TRUE, scale. = TRUE)
summary(PCA)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4    PC5    PC6    PC7     PC8
## Standard deviation     1.1217 1.0722 1.0628 1.0429 0.9909 0.9279 0.9023 0.84743
## Proportion of Variance 0.1573 0.1437 0.1412 0.1360 0.1227 0.1076 0.1018 0.08977
## Cumulative Proportion  0.1573 0.3010 0.4422 0.5781 0.7009 0.8085 0.9102 1.00000
```

Basandonos en estos datos en el criterio de la varianza acumulada, nos quedaríamos con los 6 primeros componentes puesto que ya explican un porcentaje alto de varianza con el menor número posible de variables. Sin embargo, haremos uso del **gráfico de scree plot** para hallar algún "codo".

```
fviz_eig(PCA, addlabels = TRUE, ylim = c(0, 50), main = "Gráfico de sedimentación - PCA")
```



De esta forma observamos que hay dos pequeños "codos" sin ser un cambio muy brusco: tras el componente 4 y tras el componente 6.

Otro método es mediante **eigenvalues** donde nos dirá la importancia de cada componente. Es decir, cuanta varianza de los datos totales se explica con estos componentes.

```
eigenvalores <- get_eigenvalue(PCA)
eigenvalores
```

```
##       eigenvalue variance.percent cumulative.variance.percent
## Dim.1  1.2583000        15.728750                    15.72875
## Dim.2  1.1496347        14.370433                    30.09918
## Dim.3  1.1294674        14.118342                    44.21753
## Dim.4  1.0876538        13.595673                    57.81320
## Dim.5  0.9817865        12.272331                    70.08553
## Dim.6  0.8609310        10.761638                    80.84717
## Dim.7  0.8140959        10.176199                    91.02337
## Dim.8  0.7181307         8.976634                   100.00000
```

Partido de los resultados obtenidos, ¿cuántos componentes principales deberiamos retener basándonos en el gráfico de sedimentación (scree plot) o los valores propios (eigenvalues)? Bajo mi punto de vista, optaría por los primeros cuatro componentes, puesto que podemos proceser a un analisis con buena cobertura y eigenvalues $>$ 1,58% de varianza explicada.

A continuación, realizaremos una interpretación de los dos primeros componentes principales. Para ello observaremos las cargas de dichas componentes.

```
round(PCA$rotation[, 1:2], 3)
```

```
##                              PC1    PC2
## Age                       -0.417 -0.088
## Social_Media_Hours         0.186  0.563
## Physical_Exercise_Hours    0.383 -0.090
## Daily_Study_Hours         -0.035 -0.259
## Stress_Level               0.564  0.176
## Monthly_Expenses          -0.343  0.581
## Income_Level               0.413 -0.235
## Num_Online_Courses_Taken   0.193  0.421
```

En el primer componente vemos una alta carga en: *Physical_Exercise_Hours* (0.383), *Stress_Level* (0.564) y *Income_Level (0.413)*, siendo opuesto a *Age* (-0.417), *Daily_Study_Hours* (-0.035) y *Monthly_Expenses (-0.343)*. Esto puede hacernos pensar en una correlación de la actividad personal del individuo

En el segudo componente, observamos una alta carga en: *Monthly_Expenses* (0.581), *Social_Media_Hours (0.563)*, *Num_Online_Courses_Taken* (0.421). Por lo que podriamos pensar en cierta correlación a nivel de consumo online.

2. Análisis de Correspondencias (CA)

Para dicha tarea nos valdremos de las variables categóricas del conjunto de datos (género favorito, área de estudios y realizaremos un análisis de correspondencias. Esta es una técnica estadística multivariante. Su análogo es la técnica vista anterior, el PCA, con diferencia que el anterior, como hemos trabajado, utiliza datos numericos continuos y este categóricos. Además, los datos de entrada del PCA es una matriz de datos, mientra que en el CA es una tabla de contigencia donde se recopilan las frecuencias cruzadas entre distintas variables.

En este caso estudiaremos para las categorías género favorito, *Favorite_Genre*, y área de estudios, *Career_Track*, quedando nuestra tabla de contigencia tal que así.

```
#install.packages("FactoMineR")
#install.packages("factoextra")

library(FactoMineR)
library(factoextra)

tabla_contigencia <- table(student_data$Career_Track, student_data$Favorite_Genre)
tabla_contigencia
```

```
##
##                 Action Comedy Drama
##   DataScientist     20     26    19
##   Freelancer        14     16     9
##   Manager           10     17    14
##   WebDev            16     22    17
```

La distribución porcentual por filas quedaría:

```
prop.table(tabla_contigencia, 1)
```

```
##
##                    Action    Comedy     Drama
##   DataScientist 0.3076923 0.4000000 0.2923077
##   Freelancer    0.3589744 0.4102564 0.2307692
##   Manager       0.2439024 0.4146341 0.3414634
##   WebDev        0.2909091 0.4000000 0.3090909
```

Mientras que por columnas:

```
prop.table(tabla_contigencia, 2)
```

```
##
##                    Action    Comedy     Drama
##   DataScientist 0.3333333 0.3209877 0.3220339
##   Freelancer    0.2333333 0.1975309 0.1525424
##   Manager       0.1666667 0.2098765 0.2372881
##   WebDev        0.2666667 0.2716049 0.2881356
```

A continuación realizaremos el CA a dicha tabla.

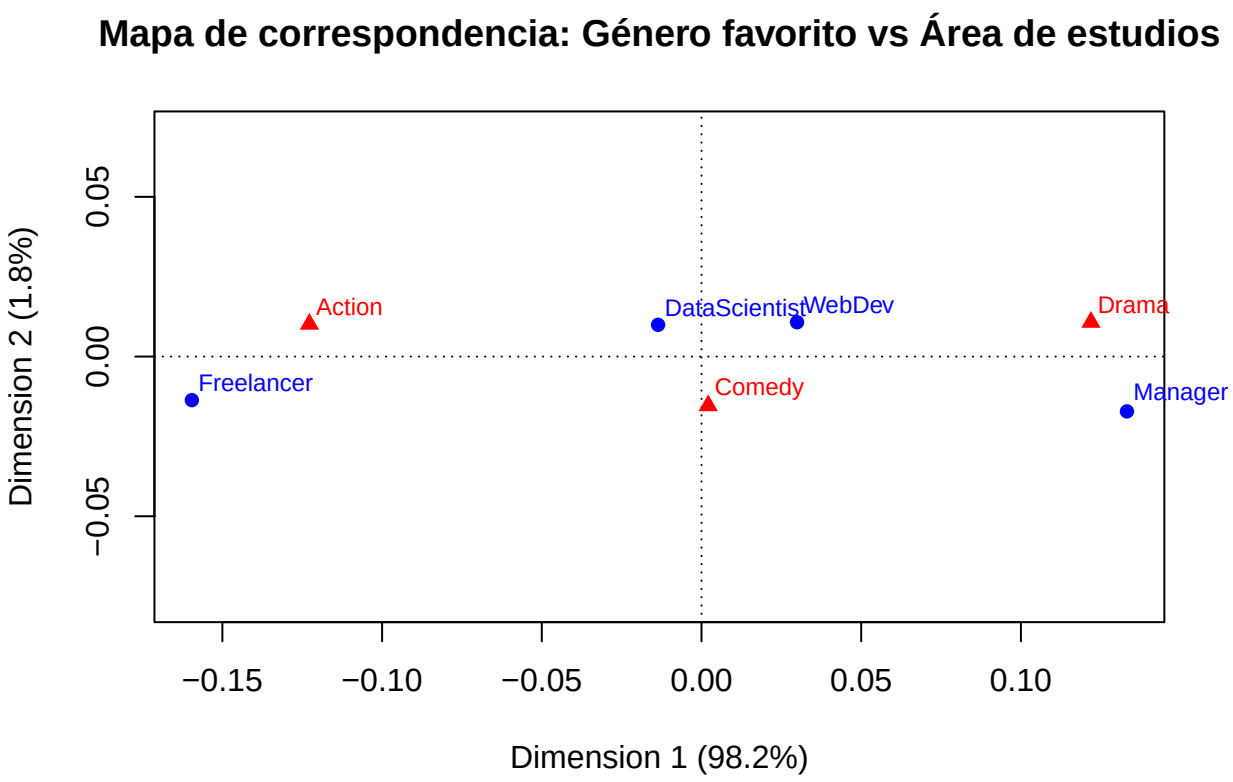
```r
#install.packages("ca")

library(ca)
anacor <- ca(tabla_contigencia)
summary(anacor)
```

```
##
## Principal inertias (eigenvalues):
##
##  dim    value      %   cum%   scree plot
##  1      0.008908  98.2  98.2  ************************
##  2      0.000161   1.8 100.0
##         -------- -----
##  Total: 0.009069 100.0
##
##
## Rows:
##     name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
## 1 | DtSc |  325 1000   10 |  -14 652   7 |   10 348 200 |
## 2 | Frln |  195 1000  551 | -160 993 557 |  -14   7 226 |
## 3 | Mngr |  205 1000  408 |  133 984 408 |  -17  16 377 |
## 4 | WbDv |  275 1000   31 |   30 886  28 |   11 114 197 |
##
## Columns:
##     name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
## 1 | Actn |  300 1000  502 | -123 993 507 |   10   7 193 |
## 2 | Cmdy |  405 1000   11 |    2  18   0 |  -15 982 595 |
## 3 | Dram |  295 1000  487 |  122 992 492 |   11   8 213 |
```

Ahora, visualizaremos las categorías de filas y columnas en 2D.

```r
plot(anacor, main = "Mapa de correspondencia: Género favorito vs Área de estudios")
```



Como cierrre de este apartado haremos una interpretación de las categorías. Aunque con la tabla de contigencia se podría vaticinar, del siguiente gráfico sacamos las siguientes conclusiones. Por un lado debemos declarar que con la dim1, con el 98.2% de la variabilidad, por lo que las asociaciones se interpretan principalmente en el eje horizontal. Partiendo de esta base, observamos que las preferencias entre *Manager* y *Freelancer* son opuestas debido a su separación. De hecho en la correlación k=1 son los que más apartan a las dim1. Por el contrario *DataScientist* y *WebDev* son los más cercanos tanto al centro como entre ellos, por lo que su preferencia es más cercana.

De la misma forma analizando los generos, vemos que los perfiles que elegien tanto *Action* como *Drama* son completamente distitno debido a su distancia, en -123 el priemro y 122 el segundo, completamente opuesto. En cambio, *Comedy* se presenta como un género más bien transversal al situarse en el centro.

3. Algoritmos Genéticos para la selección de variables

Nuestro objetivo será predecir el *Nivel_de_Estrés* de un individuo utilizando las demás variables del conjunto de datos. Comenzaremos cargando nuestra libreria *GA* y seleccionando las variables predictoras.

```
#install.packages("GA")
library(GA)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Package 'GA' version 3.2.4
## Type 'citation("GA")' for citing this R package in publications.

##
## Attaching package: 'GA'

## The following object is masked from 'package:utils':
##
##     de
```

```
vars <- names(student_data)
pred_vars <- vars[vars != "Stress_Level"]

X <- student_data[, vars]
y <- student_data$Stress_Level
```

A continuación, generaremos una función basada en en el **Criterio de Información de Akaike**, de esta forma, penalizan los modelos más complejos. Esto se basa en:

$$\text{AIC} = -2 \cdot \log(\text{verosimilitud}) + 2 \cdot \text{npar}$$

```
fitness_AIC <- function(bitstring) {
  if (sum(bitstring) == 0) return(Inf)
  sel <- pred_vars[as.logical(bitstring)]
  modelo <- lm(Stress_Level ~ ., data = student_data[, sel, drop = FALSE])
  AIC(modelo)
}
```

Una vez generada, ya podemos generar nuestro algoritmo genético.

```
GA_model <- ga(
  type = "binary",
  fitness = function(bits) -fitness_AIC(bits),
  nBits = length(pred_vars),
  popSize = 50,
  maxiter = 50,
  run = 50,
  seed = 123,
  monitor = TRUE
)
```

```
## GA | iter = 1 | Mean = -818.6726 | Best = -811.2232
## GA | iter = 2 | Mean = -818.3969 | Best = -809.8737
## GA | iter = 3 | Mean = -817.8215 | Best = -809.8737
## GA | iter = 4 | Mean = -817.1266 | Best = -809.8737
## GA | iter = 5 | Mean = -816.6934 | Best = -809.8737
## GA | iter = 6 | Mean = -816.4273 | Best = -809.8737
## GA | iter = 7 | Mean = -815.7666 | Best = -809.8737
## GA | iter = 8 | Mean = -815.8870 | Best = -809.8737
## GA | iter = 9 | Mean = -815.8267 | Best = -809.8737
## GA | iter = 10 | Mean = -815.2357 | Best = -809.8737
## GA | iter = 11 | Mean = -815.3560 | Best = -809.8737
## GA | iter = 12 | Mean = -815.0041 | Best = -809.8737
## GA | iter = 13 | Mean = -813.8655 | Best = -809.3451
## GA | iter = 14 | Mean = -813.2320 | Best = -809.3451
## GA | iter = 15 | Mean = -812.7413 | Best = -808.9236
## GA | iter = 16 | Mean = -813.0105 | Best = -808.9236
## GA | iter = 17 | Mean = -812.5303 | Best = -808.9236
## GA | iter = 18 | Mean = -812.8673 | Best = -808.9236
## GA | iter = 19 | Mean = -812.4670 | Best = -808.9236
## GA | iter = 20 | Mean = -811.6998 | Best = -808.9236
## GA | iter = 21 | Mean = -811.3776 | Best = -808.9236
## GA | iter = 22 | Mean = -811.3777 | Best = -808.9236
## GA | iter = 23 | Mean = -811.2504 | Best = -808.9236
## GA | iter = 24 | Mean = -810.8557 | Best = -808.9236
## GA | iter = 25 | Mean = -811.0385 | Best = -808.9236
## GA | iter = 26 | Mean = -811.0948 | Best = -808.9236
## GA | iter = 27 | Mean = -811.2120 | Best = -808.9236
## GA | iter = 28 | Mean = -811.4274 | Best = -808.9236
## GA | iter = 29 | Mean = -811.3752 | Best = -808.9236
## GA | iter = 30 | Mean = -811.4021 | Best = -808.9236
## GA | iter = 31 | Mean = -811.4413 | Best = -808.9236
## GA | iter = 32 | Mean = -811.3918 | Best = -808.9236
## GA | iter = 33 | Mean = -811.3515 | Best = -808.9236
## GA | iter = 34 | Mean = -811.0096 | Best = -808.9236
## GA | iter = 35 | Mean = -810.9794 | Best = -808.9236
## GA | iter = 36 | Mean = -811.1345 | Best = -808.9236
## GA | iter = 37 | Mean = -810.7431 | Best = -808.9236
## GA | iter = 38 | Mean = -810.2681 | Best = -808.9236
## GA | iter = 39 | Mean = -809.9048 | Best = -808.9236
## GA | iter = 40 | Mean = -809.7802 | Best = -808.9236
## GA | iter = 41 | Mean = -809.7907 | Best = -808.9236
## GA | iter = 42 | Mean = -809.7514 | Best = -808.9236
## GA | iter = 43 | Mean = -809.9724 | Best = -808.9236
## GA | iter = 44 | Mean = -809.9934 | Best = -808.9236
## GA | iter = 45 | Mean = -809.8068 | Best = -808.9236
## GA | iter = 46 | Mean = -809.9991 | Best = -808.9236
## GA | iter = 47 | Mean = -809.8451 | Best = -808.9236
## GA | iter = 48 | Mean = -809.9484 | Best = -808.9236
## GA | iter = 49 | Mean = -810.3158 | Best = -808.9236
## GA | iter = 50 | Mean = -810.0579 | Best = -808.9236
```

```r
best_bits  <- GA_model@solution[1, ]
seleccion  <- pred_vars[as.logical(best_bits)]
seleccion
```

```
## [1] "Social_Media_Hours"      "Physical_Exercise_Hours"
## [3] "Favorite_Genre"          "Binary_Status"
```

Finalmente, construimos el modelo con las variables seleccionadas.

```
modelo_final <- lm(Stress_Level ~ .,data = student_data[, seleccion, drop = FALSE])
```

Recapitulando lo realizando anteriomente, hemos comenzando haciendo una selección de variables, obviando la variable que queremos predecir, con el fin de buscar dicho subconjunto óptimo. Seguidamente, hemos generado una función *fitness_AIC* la cual evalua que tan bueno es un oconjunto de variables. Como criterio podriamos haber optado por R^2, sin embargo, pretendemos que se penalice el modelo con muchas variables, optando por el AIC. Dicha función la evaluamos dentro del algoritmo genético el cual busca el mejor AIC, en este caso en 50 iteracciones. Finalmente, nos arroja que el mejor subconjunto óptimo de variables para predecir *Stress_Level*, es: *Social_Media_Hours*, *Physical_Exercise_Hours*, *Favorite_Genre* y *Binary_Status*.

Para comparar modelos, propondremos uno que partad el subconjunto final de variables seleccionado y la precisión del modelo (u otra métrica relevante). Utilizaremos un tamaño de población igual a 30, Un número máximo de interacciones igual a 20, y run = 10, seed = 123.

Comenzaremos generando la función **Fitness** y el algoritmo genético con las condiciones dadas.

```
fitness <- function(string) {
  selected_vars <- pred_vars[as.logical(string)]
  if (length(selected_vars) == 0) return(Inf)

  formula <- as.formula(paste("Stress_Level ~", paste(selected_vars, collapse = " + ")))
  modelo <- lm(formula, data = student_data)

  return(AIC(modelo))
}

GA_model <- ga(
  type = "binary",
  fitness = function(string) -fitness(string),
  nBits = length(pred_vars),
  popSize = 30,
  maxiter = 20,
  run = 10,
  seed = 123,
  monitor = TRUE
)
```

```
## GA | iter = 1 | Mean = -818.4008 | Best = -812.1269
## GA | iter = 2 | Mean = -817.6610 | Best = -812.1269
## GA | iter = 3 | Mean = -816.7403 | Best = -812.1269
## GA | iter = 4 | Mean = -815.8191 | Best = -811.0440
## GA | iter = 5 | Mean = -816.0118 | Best = -811.0440
## GA | iter = 6 | Mean = -814.6651 | Best = -809.8676
## GA | iter = 7 | Mean = -813.9080 | Best = -809.8676
## GA | iter = 8 | Mean = -814.0138 | Best = -809.8676
## GA | iter = 9 | Mean = -812.7122 | Best = -809.8676
## GA | iter = 10 | Mean = -812.0255 | Best = -809.3451
## GA | iter = 11 | Mean = -811.5126 | Best = -809.3451
```

```
## GA | iter = 12 | Mean = -811.3387 | Best = -809.3451
## GA | iter = 13 | Mean = -811.2311 | Best = -809.3451
## GA | iter = 14 | Mean = -811.0043 | Best = -809.3451
## GA | iter = 15 | Mean = -811.1993 | Best = -809.3451
## GA | iter = 16 | Mean = -811.0583 | Best = -809.3451
## GA | iter = 17 | Mean = -810.7981 | Best = -809.3451
## GA | iter = 18 | Mean = -810.2849 | Best = -809.3451
## GA | iter = 19 | Mean = -810.2810 | Best = -809.3451
```

Una vez generados, veremos cual es el subconjunto de variables;

```
best_solution <- GA_model@solution[1, ]
selected_vars <- vars[as.logical(best_solution)]
selected_vars
```

```
## [1] "Social_Media_Hours"       "Physical_Exercise_Hours"
## [3] "Monthly_Expenses"         "Num_Online_Courses_Taken"
## [5] "Career_Track"
```

```
final_formula <- as.formula(paste("Stress_Level ~", paste(selected_vars, collapse = " +
 ↪  ")))
model_DATOS <- lm(final_formula, data = student_data)
```

En este caso, observamos como nos propone 5 varibales predictoras, a diferencia del otro que simplemente nos propone 4. A continuación analizaremos las métricas de ambos modelos.

```
comparacion_modelos <- data.frame(
  Modelo = c("Modelo 1", "Modelo Condiciones b"),
  AIC = c(AIC(modelo_final), AIC(model_DATOS)),
  R_squared = c(summary(modelo_final)$r.squared, summary(model_DATOS)$r.squared)
)
print(comparacion_modelos)
```

```
##                 Modelo      AIC  R_squared
## 1             Modelo 1 808.9236 0.08048085
## 2 Modelo Condiciones b 821.7229 0.03912232
```

Como resultado obtenemos que que el Modelo 1, que es el anteriormente generado tiene un mejor AIC, al ser menos complejo, pero por contra tiene un peor R^2, mientras que en Modelo Condiciones b pasa justamente lo contrario.

4. Métodos de Regresión

Para esta técnica utilizaremos el *Nivel_de_Estrés* como respuesta, construyendo modelos de regresión utilizando el resto de variables con los métodos *Stepwise*. Además, compararemos su desempeñoo en un conjunto de prueba y discuteremos las diferencias en la selección de variables que ha originado cada método.

En el Análisis de Componentes Principales solo se utiliza las variables predictoras, por lo que la variable respuesta no interviene. Con la Regresión PLS el objetivo es encontrar aquella dimensión que explique ambas variables. Comenzaremos haciendo la división del conjunto de datos.

```
set.seed(123)
n <- nrow(student_data)
train_index <- sample(1:n, size = floor(0.75 * n))
train_data <- student_data[train_index, ]
test_data  <- student_data[-train_index, ]
```

A continuación generamos los límites de búsqueda de nuestro algoritmo. Estos serán:

- *Modelo nulo*: donde no usa ninguna variable predictora, solamente *Nivel_de_Estrés*.
- *Modelo completo*: usa todas las variables disponibles.

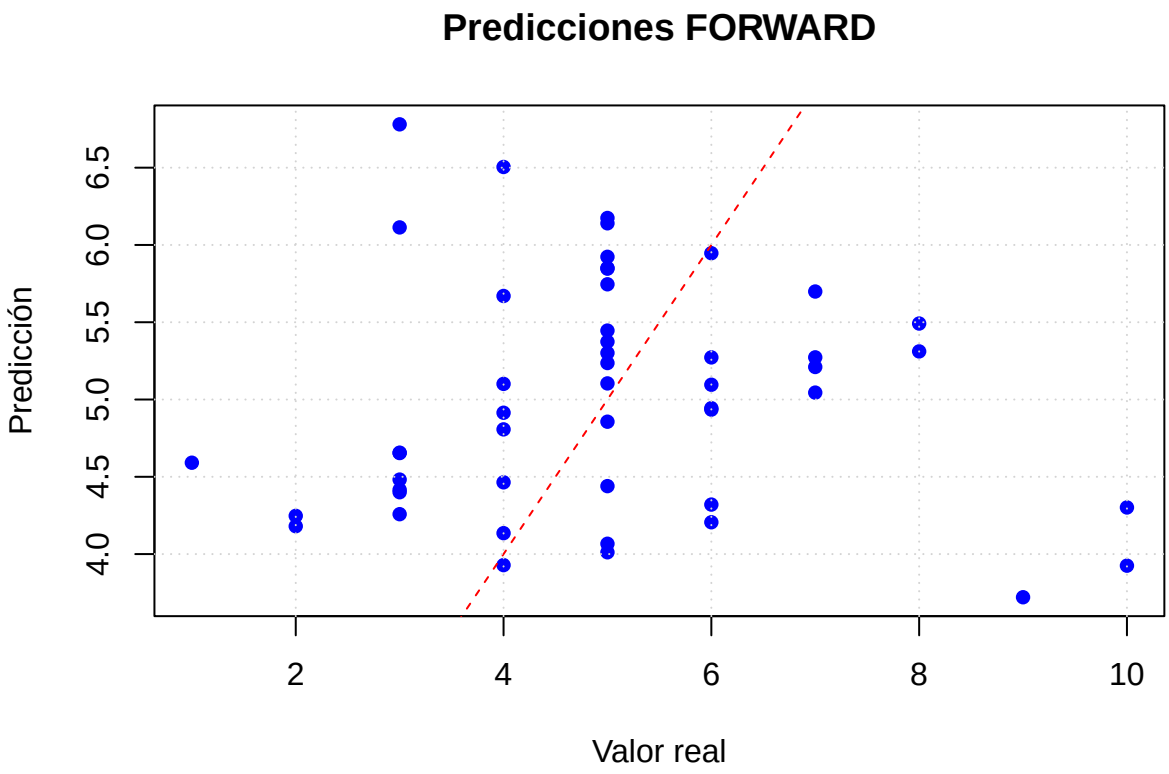
```
modelo_nulo <- lm(Stress_Level ~ 1, data = train_data)
modelo_full <- lm(Stress_Level ~ ., data = train_data)
```

Una vez definidos los límites, comenzaremos desde el *modelo_nulo* y se irán incluyendo variables confome si mejora el modelo según AIC. Para ello declararemos *modelo_forward* y *modelo_backward* y observaremos sus predicciones vs los valores reales.

```
modelo_forward <- step(modelo_nulo, scope = formula(modelo_full), direction = "forward",
 ↪  trace = FALSE)
modelo_backward <- step(modelo_full, direction = "backward", trace = FALSE)

pred_forward <- predict(modelo_forward, newdata = test_data)
pred_backward <- predict(modelo_backward, newdata = test_data)

plot(test_data$Stress_Level, pred_forward, main = "Predicciones FORWARD", xlab = "Valor
 ↪  real", ylab = "Predicción", col = "blue", pch = 16)
abline(a = 0, b = 1, col = "red", lty = 2)
grid()
```



Por último, observaremos y compararemos las métricas.

```
RMSE <- function(y_true, y_pred) sqrt(mean((y_true - y_pred)^2))
R2 <- function(y_true, y_pred) cor(y_true, y_pred)^2

rmse_forward <- RMSE(test_data$Stress_Level, pred_forward)
r2_forward   <- R2(test_data$Stress_Level, pred_forward)

rmse_backward <- RMSE(test_data$Stress_Level, pred_backward)
r2_backward   <- R2(test_data$Stress_Level, pred_backward)

Comparacion_pls <- data.frame(
  Método = c("Forward", "Backward"),
  RMSE = c(rmse_forward, rmse_backward),
  R2   = c(r2_forward, r2_backward)
)

print(Comparacion_pls)
```

```
##     Método     RMSE           R2
## 1  Forward 2.060232 0.0009050973
## 2 Backward 2.060232 0.0009050973
```

Podemos observar que ambos modelos nos arrojan los mismos valores, en este sentido nos deberiamos preguntar si son identicos. Para ello lo verificaremos de la siguiente forma.

```r
formula(modelo_forward)
```

```
## Stress_Level ~ Favorite_Genre + Physical_Exercise_Hours + Social_Media_Hours +
##     Daily_Study_Hours + Age
```

```r
formula(modelo_backward)
```

```
## Stress_Level ~ Age + Social_Media_Hours + Physical_Exercise_Hours +
##     Daily_Study_Hours + Favorite_Genre
```

Como podemos observar son las mismas variables, aunque en distinto orden cuestión que no afecta en absoluto. Las variables que han seleccionado cada modelo son: *Age*,*Social_Media_Hours*,*Physical_Exercise_Hours*,*Daily_Stu* y *Favorite_Genre*. Es por ello que obtenemos unas mismas métricas: $R^2 \approx 0.0009$ y RMSE ≈ 2.06, y aunque las variables óptimas esten claras, esto nos arroja que la variable *Stress_Level* no se explica bien mediante regresión lineal con las variables presentes.

5. Máquinas de Vectores de Soporte (SVM)

Construiremos un modelo de regresión SVM para predecir *Nivel_de_Estrés*s de un individuo. Para ello, eligeremos la función kernel que mejor se ajusta al problema con el resto de parámetros por defecto. Por otro lado, compararemos el desempeñoo del SVM con los métodos de regresión del apartado anterior.

Los kernels son funciones matematicas que nos permiten actuar en modelos no lineales, es por ello que debemos realizar una inmersión de las instancias del conjunto de aprendizaje en un espacio de dimensión superior, para así poder modelarlo de manera lineal. A continuación generaremos los distintos modelos con distintas kernel.

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following objects are masked _by_ '.GlobalEnv':
##
##     R2, RMSE
```

```
library(e1071)

svm_linear <- svm(Stress_Level ~ ., data = train_data, kernel = "linear")
svm_radial <- svm(Stress_Level ~ ., data = train_data, kernel = "radial")
svm_poly <- svm(Stress_Level ~ ., data = train_data, kernel = "polynomial")
svm_sigmoid <- svm(Stress_Level ~ ., data = train_data, kernel = "sigmoid")
```

Una vez determinados dichos modelos con las distintitas kernels, los compararemos entre sí, en función a las métricas de su predicción.

```
eval_model <- function(model, test_data) {
  pred <- predict(model, newdata = test_data)
  rmse <- sqrt(mean((test_data$Stress_Level - pred)^2))
  r2 <- cor(test_data$Stress_Level, pred)^2
  return(c(RMSE = rmse, R2 = r2))
}

res_linear <- eval_model(svm_linear, test_data)
res_radial <- eval_model(svm_radial, test_data)
res_poly   <- eval_model(svm_poly, test_data)
res_sigmod <- eval_model(svm_sigmoid, test_data)

Resultados_SVM <- rbind(
  Linear = res_linear,
  Radial = res_radial,
  Polynomial = res_poly,
  Sigmoid = res_sigmod
)

Resultados_SVM
```

```
##                RMSE           R2
## Linear     2.084422 4.303807e-05
## Radial     2.033903 3.541449e-04
## Polynomial 1.969214 2.190062e-04
## Sigmoid    1.983760 3.727516e-03
```

De los cuatro modelos SVM entrenados con diferentes kernels (lineal, polinomial, radial y sigmoidal), el kernel polinomial obtuvo el menor RMSE, mientras que el sigmoidal alcanzó el R^2 más alto. No obstante, todos los valores de R^2 son cercanos a cero, lo que indica una capacidad predictiva muy limitada. Esto sugiere que las variables actuales apenas explican la variabilidad en el Nivel_de_Estrés. A pesar de aplicar modelos no lineales, el desempeño sigue siendo bajo. Aún así, el "mejor" modelo sería el polinomial, puesto que obtuvo el menor error de predicción (RMSE = 1.969), aunque la capacidad explicativa global sigue siendo limitada en todos los casos. A partir de este determinaremos el número de vectores soportes utilizados.

```
svm_poly$tot.nSV
```

```
## [1] 133
```

Estos vectores son el conjunto de observaciones que utiliza el modelo del conjunto de entrenamiento, en este caso son 133. Estos son los puntos más cercanos al margen de decisión, por lo que son fundamentales para determinar la función de predicción. Como conclusión de este dato, debemos destacar que sea un número alto ya que necesita de muchos vectores para determinar dicha fontera de predicción. Esto casa con lo anteriomente explicado, y es que esta clasificación lineal es compleja para este conjunto de datos, ya que no es linealmente separable.

6. Árboles de Clasificación y Regresión

Son modelos de predictivos que dividen los datos en ramas según sus condiciones. Estudiaremos dos tipos de arboles:

- Árboles de clasificación: tiene por objetivo predecir una categoría por lo que al final de la rama habrá una oja con una categoría. Dicha rama estará determinada por un punto de corte que separa las categorías. Un ejemplo sería basandonos en la variable *Binary_Status*.

```
library(rpart)

arbol_clas <- rpart(Binary_Status ~ ., data = train_data, method = "class")
```

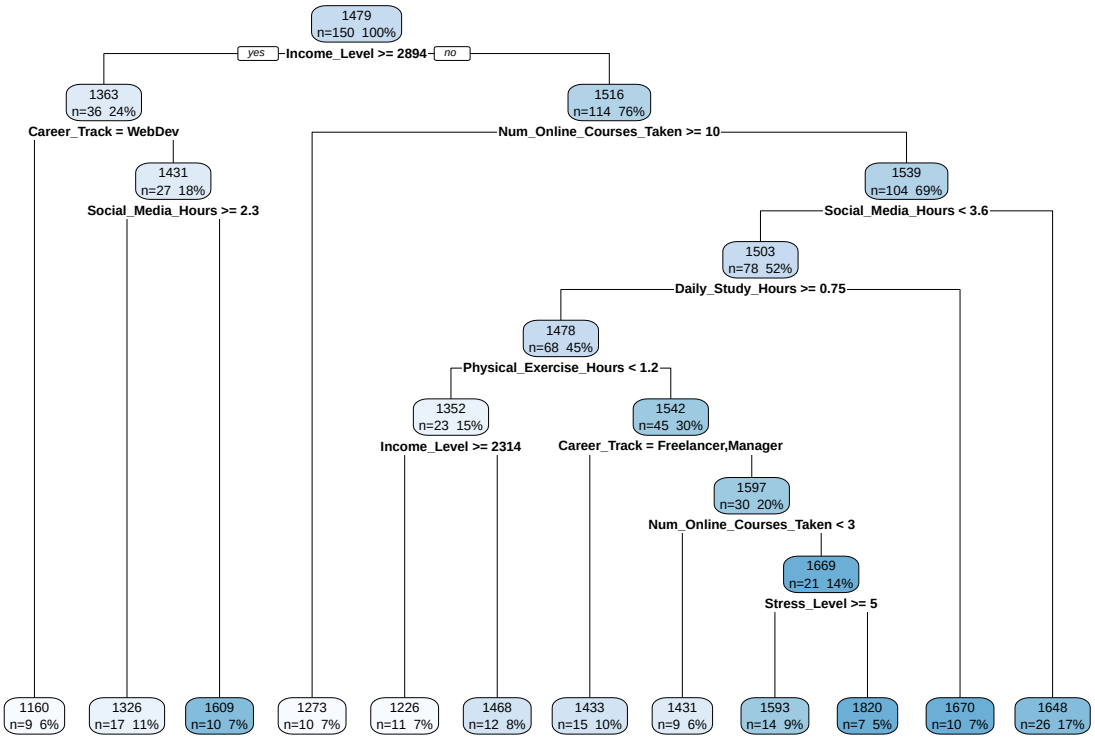
- Árboles de regresión: pretende predecir un valor numérico. En este caso, cada nodo el arbol divide los datos con el fin de minimizar la varianza. La predección por ende es el promedio de los valores de la variable respuesta. Un ejemplo sería basandonos en la variable *Gastos Mensuales*.

```
arbol_reg <- rpart(Monthly_Expenses ~ ., data = train_data, method = "anova")
```

Una vez generados ambos tipos de árboles los visualizaremos e identifica las divisiones más importantes.

```
#3install.packages("rpart.plot")
library(rpart.plot)

rpart.plot(arbol_reg, type = 2, extra = 101, fallen.leaves = TRUE)
```



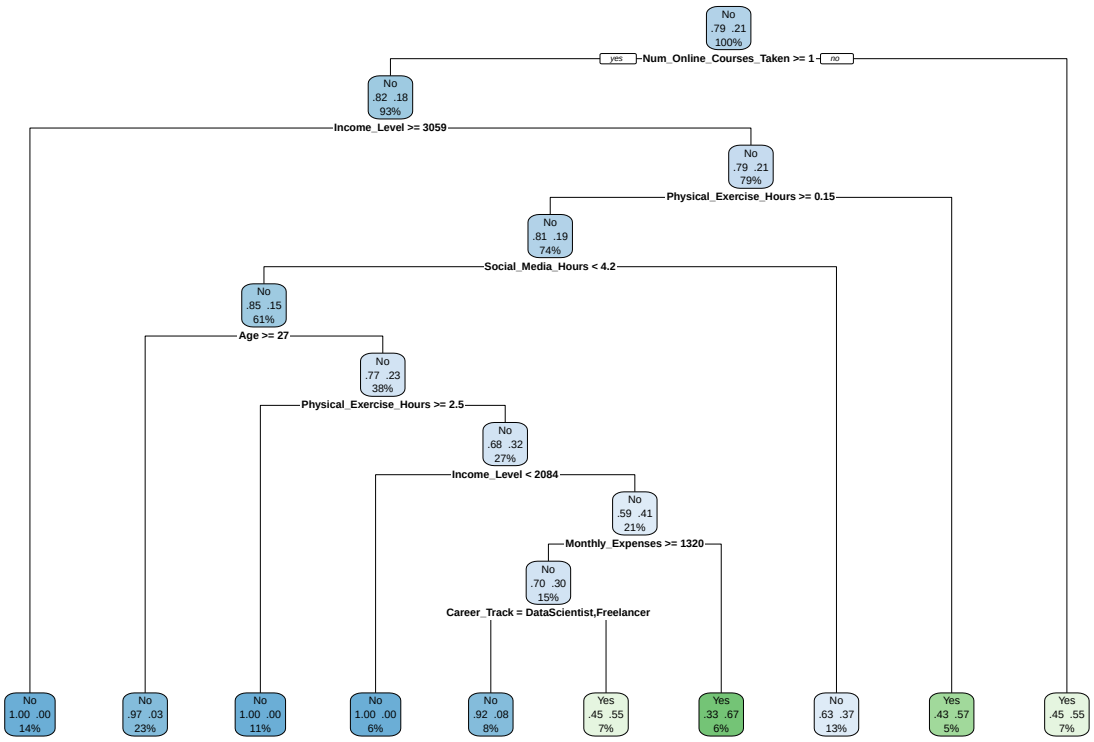
El criterio que se sigue en el primer nodo es *Income_level*, este se subdivide en dos: *Carrer_Track* que justifica en el segundo criterio, y en el caso de que no sea *WebDev* genera otro criterio que es el numero de horas en redes sociales,*Social_Media_Hours*,, siendo este el tercer criterio. En la otra rama depende del número de cursos,*Num_online_courses_taken*. Seguidamente vuelve a salir el mismo critrio de *Social_Media_Hours* aunque con distinto nivel. Este se subidive en *Physical_Exercise_Hours*. Aqui se vuelve a repetir en el esquema pq por un lado tenemos *Income_level*, mientras que del otro *Carrer_Track*. A continuación, veremos que variables resalta R.

```
arbol_reg$variable.importance
```

```
##             Income_Level Num_Online_Courses_Taken       Social_Media_Hours
##                1347048.3                1224040.8                1123176.5
##  Physical_Exercise_Hours            Career_Track       Daily_Study_Hours
##                 837831.2                 764291.8                 554581.3
##                      Age            Stress_Level          Favorite_Genre
##                 288300.4                 240620.0                 135693.0
```

Por último veremos el árbol de clasificación.

```
rpart.plot(arbol_clas, type = 2, extra = 104, fallen.leaves = TRUE)
```



En este caso, nuestro árbol comienza por el criterio *Num_Online_Courses_Taken*. El segundo criterio que sigue es el de *Income_level*. Seguidamente se basa en *Physical_Exercise_Hours*, para seguir con *Social_Media_Hours*, a continuación *Age*. El siguiente criterio vuelve a ser *Physical_Exercise_Hours*, para continuar *Income_Level* y *Monthly_Expenses*. Por último, el criterio es el de *Career_Track*. Por último veremos que variables resalta R.

```
arbol_clas$variable.importance
```

```
##  Physical_Exercise_Hours             Income_Level Num_Online_Courses_Taken
##                5.1295963                4.0636420                3.5039467
##             Career_Track                      Age       Social_Media_Hours
##                2.6399594                2.4294019                2.3386099
##         Monthly_Expenses       Daily_Study_Hours             Stress_Level
##                1.7970037                0.8368974                0.3452196
##           Favorite_Genre
##                0.1972683
```

7. Bosques Aleatorios

Contruiremos un bosque aleatorio desde la variable *Binary Status*. A partir de ahí evaluaremos las métricas de desempeño y discute la importancia de las variables involucradas teniendo presente la siguiente pregunta: ¿cuántas variables explicativas se consideran en cada corte?

A continuación vamos a construir un modelo de clasificación, puesto que la variable a estudiar,*Binary_Status*, es categórica.

```
#install.packages("randomForest")
library(randomForest)
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
rf_model <- randomForest(Binary_Status ~ ., data = train_data, importance = TRUE)
```

Una vez generado el modelo veremos las métricas.

```
rf_pred <- predict(rf_model, newdata = test_data)
conf_mat <- confusionMatrix(rf_pred, test_data$Binary_Status)

conf_mat
```

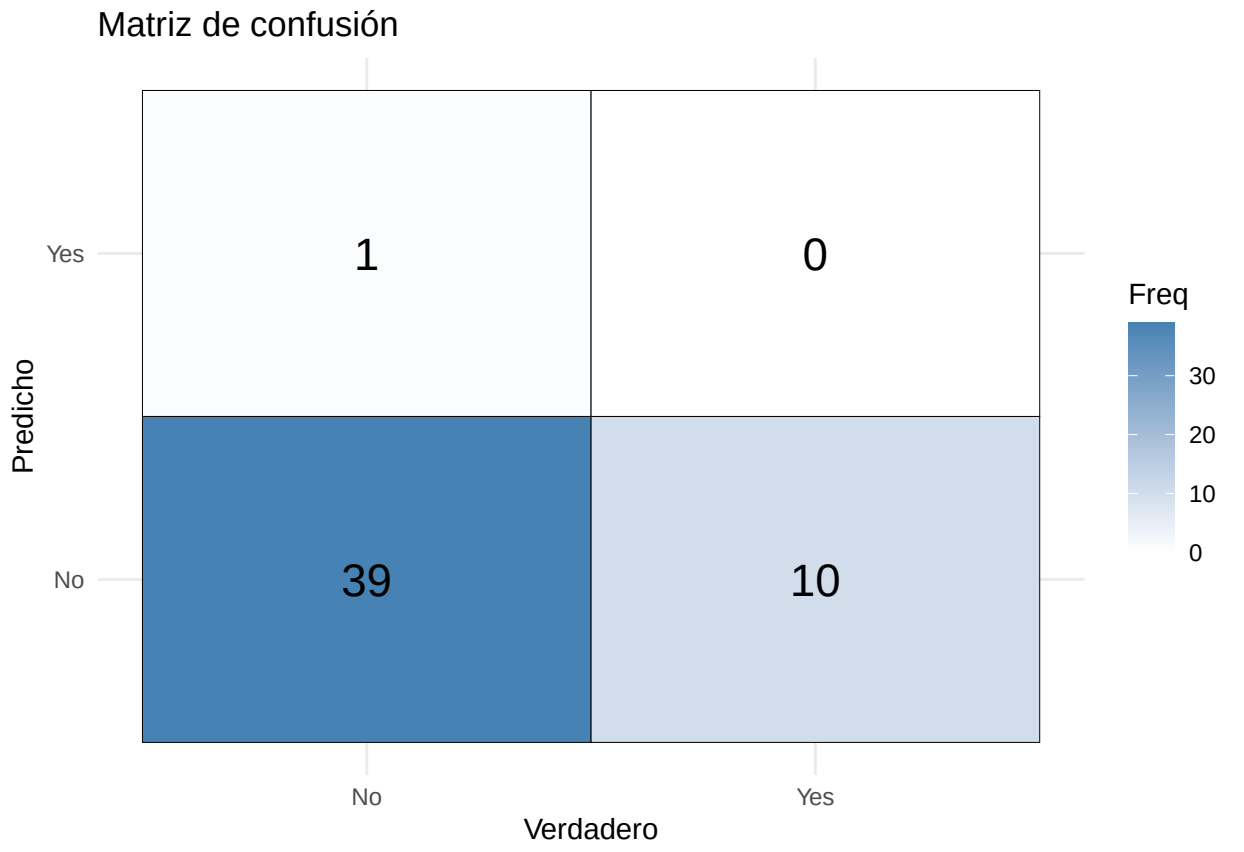
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  39  10
##        Yes  1   0
##
##                Accuracy : 0.78
##                  95% CI : (0.6404, 0.8847)
##     No Information Rate : 0.8
##     P-Value [Acc > NIR] : 0.71067
##
##                   Kappa : -0.0377
##
##  Mcnemar's Test P-Value : 0.01586
##
##             Sensitivity : 0.9750
##             Specificity : 0.0000
##          Pos Pred Value : 0.7959
##          Neg Pred Value : 0.0000
```

```
##              Prevalence : 0.8000
##          Detection Rate : 0.7800
##    Detection Prevalence : 0.9800
##       Balanced Accuracy : 0.4875
##
##        'Positive' Class : No
##
```

Evaluaremos este modelo con los resultados obtenidos de la Matriz de Confunsión. Nos ofrece un 78% de accuracy que a priori sería un buen dato, sin embargo, observamos que casi siempre clasifica como **No**, de hecho si nos fijamos en el No Information Rate este tiene valores de 0.8. De aquí podemos deducir que no es que el modelo tenga una buena forma de entrenamiento sino que se aprovecha del desbalance de los datos para clasificar. Esto coincide con que tengamos un Kappa con valor -0.03 y es que su valor negativo nos viene a decir que categoriza peor que el azar. Finalmente, como conclusión, podemos decir que está altamente sesgado hacia el **No**. Representaremos la matriz de confusión para que nos hagamos una idea más visual de como predice.

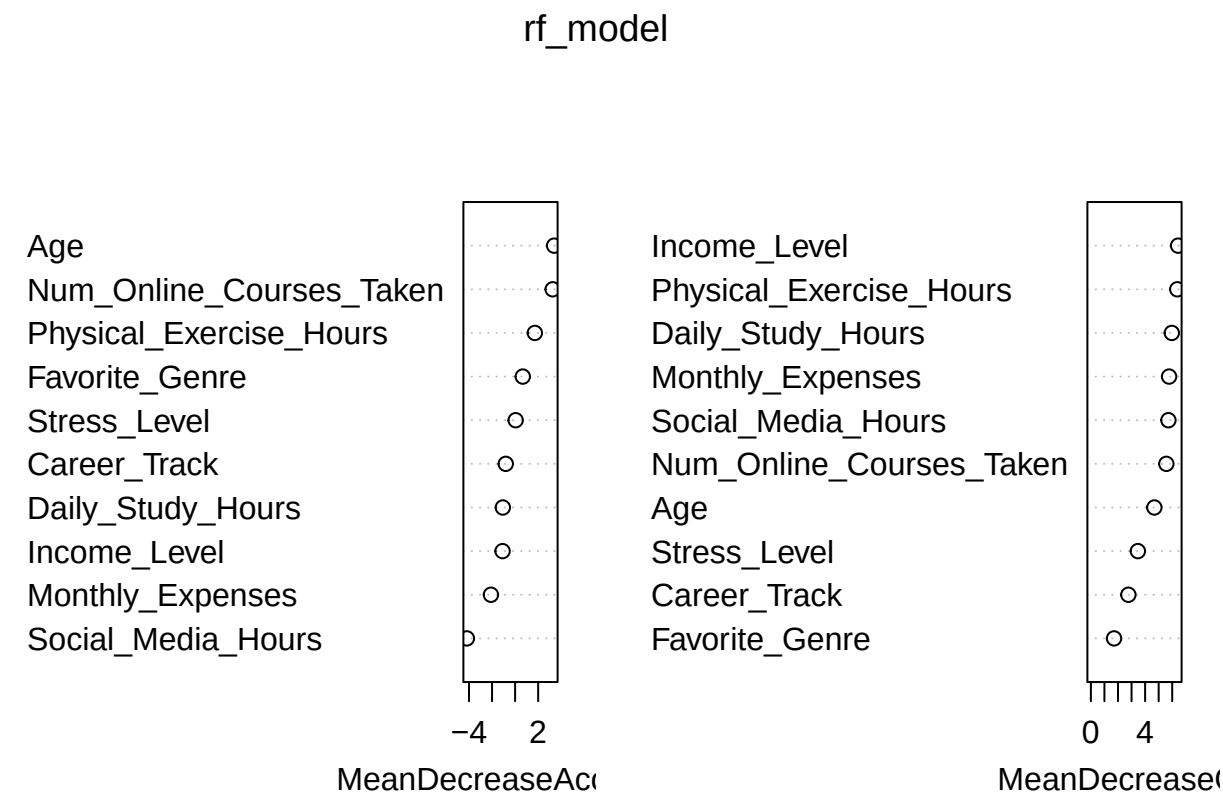
```
cm_df <- as.data.frame(conf_mat$table)

ggplot(cm_df, aes(x = Reference, y = Prediction, fill = Freq)) +
  geom_tile(color = "black") +
  geom_text(aes(label = Freq), size = 6) +
  scale_fill_gradient(low = "white", high = "steelblue") +
  theme_minimal() +
  labs(title = "Matriz de confusión", x = "Verdadero", y = "Predicho")
```



Respecto a las variables explicativas.

```
varImpPlot(rf_model)
```



Las variables que más aporta a la hora predecir en este modelo son: *Physical_ExerciseHours*,*Age* y *Num_Online_Courses_Taken*, mientras que las que menos importan son: *Carrer_Track*, *Daily_Study_Hours* y *Social_Media_Hours*. Por el otro lado, aquellas variables que más han aparecido a la hora de definir los nodos son: *Income_Level*, *Physical_Exercise_Hours* y *Daily_Study_Hours*. Por otro lado las que menos son: *Stress_Level*,*Career_Track* y *Favorite_Genre*. En cuanta a la pregunta de cuántas variables predictoras se tienen en cuenta en cada nodo lo veremos con la siguiente función. Es decir, en cada división de cada árbol dentro del bosque, solo se consideran 3 variables aleatorias de todas las disponibles. El algoritmo escoge la que mejor separa los datos. Esto introduce variabilidad entre los árboles y ayuda a que el bosque sea menos propenso al overfitting.

```
rf_model$mtry
```

```
## [1] 3
```

Como observamos nos arroja el valor 3, es decir, en cada división de cada árbol dentro del bosque, solo se consideran 3 variables aleatorias de todas las disponibles. El algoritmo escoge la que mejor separa los datos. Esto introduce variabilidad entre los árboles y ayuda a que el bosque sea menos propenso al overfitting.

8. Perceptrones Multicapa / Deep Learning con *h2o*

Comenzaremos cargando el paquete *h2o*, que es una plataforma de código abierto donde se nos permite entrenar modelos predictivos.

```
#install.packages("h2o", repos = "https://cloud.r-project.org")

library(h2o)
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit https://docs.h2o.ai
##
## ----------------------------------------------------------------------
```

```
##
## Attaching package: 'h2o'
```

```
## The following objects are masked from 'package:stats':
##
##     cor, sd, var
```

```
## The following objects are masked from 'package:base':
##
##     &&, %*%, %in%, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         21 hours 38 minutes
##     H2O cluster timezone:       Europe/Madrid
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.46.0.7
##     H2O cluster version age:    4 months and 25 days
##     H2O cluster name:           H2O_started_from_R_juanramonbaezaborrego_lgy729
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   1.81 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     H2O cluster healthy:        TRUE
```

```
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.4.1 (2024-06-14)
```

Una vez activado y puesto en funcionamiento el paquete *h2o*, entrena un modelo MLP para predecir *Nivel_de_Estrés* (regresión). Es por ello que definiremos las variables y los conjutno de entrenamiento y test. Previamente, convertiremos los datos a formato *h2o*.

```
student_h2o <- as.h2o(student_data)
```

```
##   |                                                                              |
|   0%  |
|======================================================================| 100%
```

```
y <- "Stress_Level"
x <- setdiff(names(student_data), y)


splits <- h2o.splitFrame(student_h2o, ratios = 0.8, seed = 123)
train <- splits[[1]]
test <- splits[[2]]
```

Los hiperparametros seleccionacionados serán:

- Función de activación de tipo Relu (activation = "Rectifier").
- El número de capas oculatas, 2 con 10 neuronas en ese caso, (hidden = c(10, 10)).
- Número de pasadas de los datos (epochs = 100).

Quedandonos nuestra siguiente modelo MLP.

```
modelo_nn <- h2o.deeplearning(
  x = x,
  y = y,
  training_frame = train,
  validation_frame = test,
  activation = "Rectifier",
  hidden = c(10, 10),
  epochs = 100,
  stopping_rounds = 5,
  stopping_metric = "RMSE",
  seed = 123
)
```

```
##   |                                                                              |
|   0%  |
|======================================================================| 100%
```

A continuación, observaremos las métricas como resultado de dicho modelo. Haremos uso de la matriz de confusión.

```
perf <- h2o.performance(modelo_nn, newdata = test)
cat("El RSME de este modelo es:", h2o.rmse(perf), "\n")
```

```
## El RSME de este modelo es: 1.84124
```

```
cat("El coeficiente de determinación del modelo es: ", h2o.r2(perf))
```

```
## El coeficiente de determinación del modelo es:  -0.07755031
```

A la vista de los resultados, podemos decir que este primer modelo generado no captura bien los modelos, es por ello que pasaremos a hacer un ajuste de hiperparametros que se ajusten mejor.

Respecto al ajuste de hiperparametrnos nos basaremos anteriores:

- Función de activación que ahora la probaremos añadiremos tangente hiperbólica y Maxout.
- El número de capas oculatas, igulamente serian con 2 capas pero con 10, 20 y 50 neuronas cada una.
- Número de pasadas de los datos.

Para ello utilizaremos la función Grid Search de *h2o* que encontrará los mejores hiperparametros.

```
hyper_params <- list(
  activation = c("Rectifier", "Tanh", "Maxout"),
  hidden = list(c(10,10), c(20,20), c(50,50)),
  epochs = c(50, 100, 200)
)

grids <- h2o.ls()
if ("nn_grid" %in% grids$key) {
  h2o.rm("nn_grid")
}

grid <- h2o.grid(
  algorithm = "deeplearning",
  grid_id = "nn_grid",
  x = x,
  y = y,
  training_frame = train,
  validation_frame = test,
  hyper_params = hyper_params,
  stopping_metric = "RMSE",
  stopping_rounds = 5,
  seed = 123
)
```

```
##   |                                                                              |
|   0%  |
|==============================================================================| 100%
```

Finalmente, observaremos cuales son las métricas de cada una de las combincaciones.

```
grid_perf <- h2o.getGrid(
  grid_id = "nn_grid",
  sort_by = "RMSE",
  decreasing = FALSE
)

grid_perf
```

```
## H2O Grid Details
## ================
##
## Grid ID: nn_grid
## Used hyper parameters:
##   -  activation
##   -  epochs
##   -  hidden
## Number of models: 27
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing RMSE
##   activation    epochs   hidden        model_ids    rmse
## 1     Maxout 100.00000 [50, 50] nn_grid_model_24 1.66791
## 2     Maxout  50.00000 [20, 20] nn_grid_model_12 1.69980
## 3  Rectifier  50.00000 [50, 50] nn_grid_model_19 1.76541
## 4     Maxout 100.00000 [20, 20] nn_grid_model_15 1.77593
## 5       Tanh  50.00000 [10, 10]  nn_grid_model_2 1.79975
##
## ---
##    activation    epochs   hidden        model_ids    rmse
## 22  Rectifier 200.00000 [50, 50] nn_grid_model_25 2.05040
## 23       Tanh  50.00000 [50, 50] nn_grid_model_20 2.15826
## 24  Rectifier 100.00000 [20, 20] nn_grid_model_13 2.18257
## 25     Maxout 200.00000 [10, 10]  nn_grid_model_9 2.21156
## 26     Maxout 100.00000 [10, 10]  nn_grid_model_6 2.22190
## 27       Tanh 200.00000 [50, 50] nn_grid_model_26 2.30428
```

Una vez evaluados los modelos se han imprimido 27 modelos distintos. Sin embargo, de todos los visto y de todas las opciones planteadas, la mejor opción es aquella que combina la función de activación tangente hiperbólica, que se pasara 50 veces por los datos y que haya 2 capas ocultas de 10 nueronas cada una. Siendo este

```
modelo_best <- h2o.deeplearning(
  x = x,
  y = y,
  training_frame = train,
  validation_frame = test,
  activation = "Tanh",
  hidden = c(10, 10),
  epochs = 50,
  stopping_rounds = 5,
  stopping_metric = "RMSE",
  seed = 123
)
```

```
##   |                                                                      |
|   0%  |
|======================================================================| 100%
```

A continuación, hallaremos las métricas de dicho modelo.

```
metric_best_model <- h2o.performance(modelo_best, newdata = test)
cat("El RSME del modelo co mejores hiperparametro es:", h2o.rmse(metric_best_model),
 ↪  "\n")
```

```
## El RSME del modelo co mejores hiperparametro es: 1.721211
```

```
cat("El coeficiente de determinación del modelo co mejores hiperparametro es: ",
 ↪  h2o.r2(metric_best_model))
```

```
## El coeficiente de determinación del modelo co mejores hiperparametro es:  0.0583603
```

Aunque estos datos mejoran lo anterior, al explicarse mejor la variabilidad de los datos y fallar menos en promedio el número de equivocaciones por unidades, sigue siendo bajo para poder llegar a hacer predicciones fiables.

9. Manejo de Datos Desequilibrados

La tarea que seleccionaremso corresponde a la número 7, de random forest, la cual utlizaba la variable categórica *Binary_Status*, y se presentaba desbalanceada por la gran cantidad de "No" que exisitían en ella. Pretendemos consiguir balancear la clase minoritaria de nuestro dataset, para así evitar el sobremuestreo. A continuación crearemos ese dataset. Utilizaré el paquete ROSE, debido a que el paquete SMOTE solo funciona con variables númericas y al hacer la conversión me daba diversos fallos relacionados con la función para balancear.

```
#install.packages("ROSE")
library(ROSE)
```

```
## Loaded ROSE 0.0-4
```

```
set.seed(123)


train_data_balanced <- ROSE(Binary_Status ~ ., data = train_data, seed = 1)$data

table(train_data_balanced$Binary_Status)
```

```
##
##  No Yes
##  76  74
```

Ahora ya observamos que el número de "No" y "Yes" ya están, más equiparados. Veremos a continuación si esto repercute en el modelo como esperamos.

```
rf_model_balanced <- randomForest(Binary_Status ~ ., data = train_data_balanced,
 ↪  importance = TRUE)

rf_pred_balanced <- predict(rf_model_balanced, newdata = test_data)

conf_mat_balanced <- confusionMatrix(rf_pred_balanced, test_data$Binary_Status)
conf_mat_balanced
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  27   1
##        Yes 13   9
##
##                Accuracy : 0.72
##                  95% CI : (0.5751, 0.8377)
##     No Information Rate : 0.8
##     P-Value [Acc > NIR] : 0.939278
##
##                   Kappa : 0.3966
##
##  Mcnemar's Test P-Value : 0.003283
##
```

```
##             Sensitivity : 0.6750
##             Specificity : 0.9000
##          Pos Pred Value : 0.9643
##          Neg Pred Value : 0.4091
##              Prevalence : 0.8000
##          Detection Rate : 0.5400
##    Detection Prevalence : 0.5600
##       Balanced Accuracy : 0.7875
##
##        'Positive' Class : No
##
```

Tras aplicar ROSE para balancear las clases, el modelo Random Forest mostró una reducción en la precisión global (de 78% a 70%), pero una mejora significativa en métricas más relevantes para un dataset desbalanceado. El Kappa pasó de -0.03 a 0.33, indicando que el modelo es útil. Balanced Accuracy mejoró de 0.48 a 0.74, reflejando un modelo mucho más equitativo en la predicción de ambas clases. Además, la Specificity mejoró de 0% a 80%, lo que evidencia que el modelo ahora también es capaz de detectar la clase minoritaria ("Yes"), lo que antes no hacía. Por ende, podemos concluir, que hemos mejorado el modelo y su capacidad de predicción.