Detección de cuentas falsas de Instagram

Eulogio Quemada Torres

2024-05-26

Table of contents

1 Introducción

Este libro ha sido escrito por Eulogio Quemada Torres, alumno de Laboratorio de Computación Cíentífica como asignatura optativa en el grado de Ingeniería del Software, impartida por el profesor Ángel Mora Bonilla.

El propósito del mismo es analizar los datos propuestos en el dataset de Kaggle. Este dataset contiene información sobre distintas cuentas de Instagram y si son falsas o reales.

Antes de comenzar vamos a describir las distintas variables que tenemos en nuestro dataset. Será de vital importancia entender el significado cada una para poder seguir el hilo de las técnicas de análisis de datos que se aplicarán durante este libro.

1.1 Variables

1.1.1 profile pic

Indica si la cuenta tiene foto de perfil o no. Los usuarios nuevos por defecto no tienen ninguna foto de perfil, lo que las cuentas falsas puede que luego no usen.

1.1.2 nums/length username

Es la proporción de números sobre la longitud total de carácteres en el nombre de usuario. Ciertas cuentas reales tienen algún número en la cuenta, algunos su día de nacimiento, su año, algún número que les guste. Pero tener muchos números puede ser algo raro.

1.1.3 fullname words

Número de palabras en el nombre completo. Un nombre de una persona europea suele tener 2 o 3 palabras, en función de los apellidos, algunos más, algunos menos, pero cercano a esto.

1.1.4 nums/length fullname

Es la proporción de números sobre la longitud del nombre completo. Un nombre de persona, en la mayoría de países del mundo, no puede tener números, así que la presencia de estos puede ser indicativo de que algo está pasando.

1.1.5 name==username

Indica si el nombre de usuario y el nombre real de la cuenta es el mismo. No es lo normal que sea exactamente el mismo, puede ser indicativo de algo extraño.

1.1.6 description length

Representa el número de carácteres en la descripción de la cuenta de Instagram.

1.1.7 external URL

Indica si la cuenta tiene un enlace externo o no, en su perfil.

1.1.8 private

Indica si la cuenta es privada o no.

1.1.9 posts

Número de publicaciones realizadas por la cuenta de Instagram.

1.1.10 followers

Número de seguidores de la cuenta de Instagram.

1.1.11 follows

Número de personas a las que sigue la cuenta de Instagram.

1.1.12 fake

Es la clasificación. Un 1 para las cuentas clasificadas como falsas, un 0 para las reales.

1.2 Próximos pasos

Durante este libro intentaremos aplicar las técnicas aprendidas en la asignatura para el análisis de datos. Algunas técnicas pueden que no sean aplicables, otras pueden ser mejores o peores, pero intentaremos extraer conocimiento de los datos y sacar conclusiones interesantes.

2 Análisis exploratorio de datos

El análisis exploratorio de datos es una etapa crucial en cualquier proyecto de análisis de datos o modelado predictivo. Consiste en explorar y comprender los datos disponibles antes de aplicar cualquier técnica de modelado o inferencia. El objetivo principal del análisis exploratorio de datos es revelar patrones, tendencias, anomalías y relaciones dentro de los datos, lo que proporciona una base sólida para la toma de decisiones y la construcción de modelos.

```
library(dplyr)
library(ggplot2)
library(tidyr)
library(corrplot)
library(psych)
library(gridExtra)
```

En esta sección nos centraremos en explorar la estructura de los datos disponibles y entender la distribución y características de las variables. En el siguiente apartado, nos adentraremos en la visualización de datos para comprender mejor cómo se distribuyen y relacionan las diferentes variables.

```
dataset <- read.csv("datasets/train.csv")
```

2.1 Exploración inicial del dataset

Vamos a comenzar viendo los aspectos más generales de nuestro dataset, que involucren toda la información, luego nos centraremos en aspectos más concretos.

2.1.1 Estructura del dataset

```
str(dataset)
```

```
'data.frame':   576 obs. of  12 variables:
 $ profile.pic         : int  1 1 1 1 1 1 1 1 1 1 ...
```

```
$ nums.length.username: num  0.27 0 0.1 0 0 0 0 0 0 0 ...
$ fullname.words      : int  0 2 2 1 2 4 2 2 0 2 ...
$ nums.length.fullname: num  0 0 0 0 0 0 0 0 0 0 ...
$ name..username      : int  0 0 0 0 0 0 0 0 0 0 ...
$ description.length  : int  53 44 0 82 0 81 50 0 71 40 ...
$ external.URL        : int  0 0 0 0 0 1 0 0 0 1 ...
$ private             : int  0 0 1 0 1 0 0 0 0 0 ...
$ X.posts             : int  32 286 13 679 6 344 16 33 72 213 ...
$ X.followers         : int  1000 2740 159 414 151 669987 122 1078 1824 12945 ...
$ X.follows           : int  955 533 98 651 126 150 177 76 2713 813 ...
$ fake                : int  0 0 0 0 0 0 0 0 0 0 ...
```

Vemos que contamos con 576 observaciones, es decir, 576 cuentas de Instagram con su clasificación y 12 variables que son precisamente los datos que tenemos de estas cuentas.

Inicialmente podemos ver que todas nuestras variables son númericas, ya sean números enteros o decimales. Para el análisis de datos esto es un plus, pues los números son más manejables para la mayoría de técnicas de análisis de datos.

2.1.2 Valores NA

```
sapply(dataset, function(x) sum(is.na(x)))
```

```
        profile.pic nums.length.username       fullname.words
                  0                     0                    0
nums.length.fullname       name..username   description.length
                  0                     0                    0
        external.URL              private              X.posts
                  0                     0                    0
         X.followers            X.follows                 fake
                  0                     0                    0
```

Sorprendentemente no hay valores nulos, en los datasets suele haber columnas con valores NA con los que debemos tener cuidado, pero en nuestro caso, no tenemos ninguno, lo que de nuevo, nos facilitará el trabajo a lo largo de este libro.

2.1.3 Resumen estadístico

```
summary(dataset)
```

```
 profile.pic     nums.length.username fullname.words  nums.length.fullname
Min.   :0.0000   Min.   :0.0000       Min.   : 0.00   Min.   :0.00000
1st Qu.:0.0000   1st Qu.:0.0000       1st Qu.: 1.00   1st Qu.:0.00000
Median :1.0000   Median :0.0000       Median : 1.00   Median :0.00000
Mean   :0.7014   Mean   :0.1638       Mean   : 1.46   Mean   :0.03609
3rd Qu.:1.0000   3rd Qu.:0.3100       3rd Qu.: 2.00   3rd Qu.:0.00000
Max.   :1.0000   Max.   :0.9200       Max.   :12.00   Max.   :1.00000
name..username    description.length  external.URL       private
Min.   :0.00000   Min.   :  0.00     Min.   :0.0000   Min.   :0.0000
1st Qu.:0.00000   1st Qu.:  0.00     1st Qu.:0.0000   1st Qu.:0.0000
Median :0.00000   Median :  0.00     Median :0.0000   Median :0.0000
Mean   :0.03472   Mean   : 22.62     Mean   :0.1163   Mean   :0.3819
3rd Qu.:0.00000   3rd Qu.: 34.00     3rd Qu.:0.0000   3rd Qu.:1.0000
Max.   :1.00000   Max.   :150.00     Max.   :1.0000   Max.   :1.0000
   X.posts         X.followers         X.follows           fake
Min.   :   0.0   Min.   :       0   Min.   :   0.0   Min.   :0.0
1st Qu.:   0.0   1st Qu.:      39   1st Qu.:  57.5   1st Qu.:0.0
Median :   9.0   Median :     150   Median : 229.5   Median :0.5
Mean   : 107.5   Mean   :   85307   Mean   : 508.4   Mean   :0.5
3rd Qu.:  81.5   3rd Qu.:     716   3rd Qu.: 589.5   3rd Qu.:1.0
Max.   :7389.0   Max.   :15338538   Max.   :7500.0   Max.   :1.0
```

Observamos que hay algunas variables binarias, otras con pocos valores y otras con un rango amplio. Vamos a usar las funciones apply para entrar un poco más en profundidad.

```
minimums <- sapply(dataset, min)
maximums <- sapply(dataset, max)
unique_values <- sapply(dataset, function(x) length(unique(x)))

dataset_values <- data.frame(
  min=minimums,
  max=maximums,
  unique_values=unique_values
)

dataset_values
```

```
                     min         max unique_values
profile.pic            0        1.00             2
nums.length.username   0        0.92            54
fullname.words         0       12.00             9
nums.length.fullname   0        1.00            25
name..username         0        1.00             2
description.length     0      150.00           104
external.URL           0        1.00             2
private                0        1.00             2
X.posts                0     7389.00           193
X.followers            0 15338538.00           372
X.follows              0     7500.00           400
fake                   0        1.00             2
```

Identificamos fácilmente que hay 5 variables binarias, 3 no binarias con menos de 100 valores distintos y el resto con 100 o más valores distintos. Estas variables son:

```
binary_vars <- rownames(dataset_values %>% filter(unique_values == 2))
```

```
Variables binarias son: profile.pic, name..username, external.URL, private, fake
```

```
mid_vars <- rownames(dataset_values %>% filter(unique_values > 2 & unique_values < 100))
```

```
Variables con entre 2 y 100: nums.length.username, fullname.words, nums.length.fullname
```

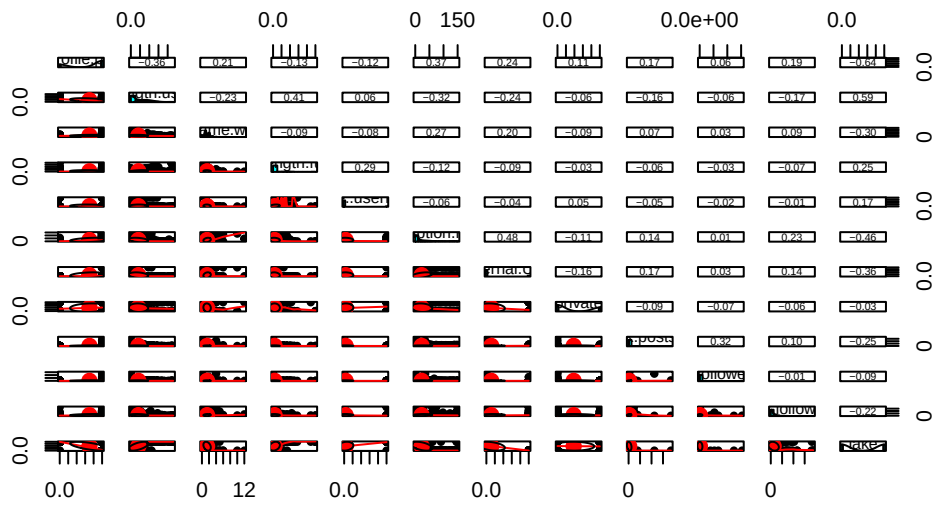
```
big_vars <- rownames(dataset_values %>% filter(unique_values >= 100))
```

```
Variables con 100 o más: description.length, X.posts, X.followers, X.follows
```

2.2 Gráfico de pares

Vamos a visualizar el panel de pares de variables, que nos da información de la relación entre cada par de variables.

```
pairs.panels(dataset, main="Gráfico de pares")
```



Es un poco díficil observar algo ya que tenemos 12 variables y los gráficos se ven muy pequeños. Vamos a observar la matriz de correlación.

2.3 Matriz de correlación

La matriz de correlación nos muestra la correlación entre cada par de variables. Sea M la matriz de correlación y sea (i, j) una posición de la matriz, entonces, la posición M[i, j] es la correlación entre las variables i y j.

```
cor_matrix <- cor(dataset)
cor_matrix
```

```
                    profile.pic nums.length.username fullname.words
profile.pic          1.00000000          -0.36408701     0.21329514
nums.length.username -0.36408701           1.00000000    -0.22547213
fullname.words        0.21329514          -0.22547213     1.00000000
nums.length.fullname -0.13175622           0.40856654    -0.09434799
name..username       -0.12490287           0.05688965    -0.08296878
description.length    0.36789194          -0.32117027     0.27252216
external.URL          0.23672932          -0.23712479     0.19656239
```

```
private               0.11473196          -0.06371257    -0.08907008
X.posts               0.16957023          -0.15744211     0.07335018
X.followers           0.06113663          -0.06278509     0.03322460
X.follows             0.19483278          -0.17241327     0.09485496
fake                 -0.63731535           0.58768653    -0.29879258
                    nums.length.fullname name..username description.length
profile.pic                -0.13175622   -0.124902870        0.367891938
nums.length.username        0.40856654    0.056889649       -0.321170271
fullname.words             -0.09434799   -0.082968780        0.272522165
nums.length.fullname        1.00000000    0.291149086       -0.117521050
name..username              0.29114909    1.000000000       -0.064813853
description.length         -0.11752105   -0.064813853        1.000000000
external.URL               -0.08872418   -0.039232382        0.482313071
private                    -0.03003033    0.046084001       -0.110328832
X.posts                    -0.05771550   -0.049808246        0.144823702
X.followers                -0.02703471   -0.017760756        0.005929455
X.follows                  -0.06797109   -0.009529101        0.226561422
fake                        0.24678210    0.170694729       -0.460824593
                    external.URL     private     X.posts  X.followers
profile.pic           0.23672932  0.11473196  0.16957023  0.061136635
nums.length.username -0.23712479 -0.06371257 -0.15744211 -0.062785090
fullname.words        0.19656239 -0.08907008  0.07335018  0.033224604
nums.length.fullname -0.08872418 -0.03003033 -0.05771550 -0.027034712
name..username       -0.03923238  0.04608400 -0.04980825 -0.017760756
description.length    0.48231307 -0.11032883  0.14482370  0.005929455
external.URL          1.00000000 -0.16261231  0.16500846  0.027188730
private              -0.16261231  1.00000000 -0.08749503 -0.073472710
X.posts               0.16500846 -0.08749503  1.00000000  0.321385480
X.followers           0.02718873 -0.07347271  0.32138548  1.000000000
X.follows             0.14251936 -0.05754247  0.09822504 -0.011065994
fake                 -0.36280938 -0.02858602 -0.24535515 -0.093688783
                      X.follows        fake
profile.pic           0.194832776 -0.63731535
nums.length.username -0.172413275  0.58768653
fullname.words        0.094854964 -0.29879258
nums.length.fullname -0.067971092  0.24678210
name..username       -0.009529101  0.17069473
description.length    0.226561422 -0.46082459
external.URL          0.142519361 -0.36280938
private              -0.057542468 -0.02858602
X.posts               0.098225040 -0.24535515
X.followers          -0.011065994 -0.09368878
X.follows             1.000000000 -0.22483522
```

```
fake                  -0.224835224  1.00000000
```

Aquí podemos buscar cualquier par de variables y ver su correlación. Buscando un poco vemos correlaciones significativas entre profile.pic y fake, la description.length y external.URL

Como hay muchos numeros y no es lo más riguroso buscar a ojo, vamos a hacer unos pequeños cálculos para encontrar todas las variables relacionadas más de un cierto `threshold`.

```r
threshold <- 0.4

cor_table <- data.frame(as.table(cor_matrix)) %>%
  rename(Correlation = Freq)

variables <- colnames(dataset)
n_variables <- length(dataset)
medium_point <- n_variables / 2

# Para que no haya repeticiones simétricas, vamos a poner la restricción de que el
# orden léxicográfico de una variable sea mayor (o menor) que la otra.
# Con un != no valdría porque habría valores filas simétricas
cor_table %>%
  filter(as.character(Var1) > as.character(Var2) & abs(Correlation) > threshold) %>%
  arrange(desc(abs(Correlation)))
```

```
                 Var1                 Var2 Correlation
1          profile.pic                 fake  -0.6373153
2 nums.length.username                 fake   0.5876865
3         external.URL   description.length   0.4823131
4                 fake   description.length  -0.4608246
5 nums.length.username nums.length.fullname   0.4085665
```

Ahora podemos sacar más información que la que habíamos visto de primeras. Hay una importante correlación entre si una cuenta es falsa y si tiene foto de perfil, el ratio de carácteres númericos en el nombre de la cuenta, la longitud de la descripción…

A conitnuación vamos a pasar a un análsis más concreto, para observar en particular cada variable de nuestro dataset.

2.4 Variables binarias

Las variables binarias solo toman dos valores (en nuestro caso todas 0 o 1) y la información que se puede extraer es distinta que el resto. Vamos a comenzar con ellas, como sabemos por

nuestro análisis anterior, estás son `profile.pic`, `name..username`, `external.URL` y `private` y `fake`. Vamos a comenzar con fake ya que es la más relevante, pues es la clasificación de cada fila del dataset.

2.4.1 fake

Toma el valor 1 si la cuenta es falsa, y 0 si no es falsa. Es la variable más importante que tenemos en el dataset, pues es la que tratamos de estimar, de la que queremos a partir de las otras, extraer información para poder predecir esta.

```
table(dataset$fake)
```

```
  0   1
288 288
```

Tenemos un 50 - 50 de clasificaciones, mitad de cuentas son falsas y la otra mitad no.

```
summary(dataset$fake)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0     0.0     0.5     0.5     1.0     1.0
```

La media y la mediana corroboran lo que comentabamos antes.

En resumen:

```
total_count <- length(dataset$fake)
count_fake <- sum(dataset$fake == 1)
count_real <- sum(dataset$fake == 0)
percent_fake <- (count_fake / total_count) * 100
percent_real <- (count_real / total_count) * 100

paste0("Cuentas falsas: ", count_fake, " (", round(percent_fake, 2), "%)")
```

```
[1] "Cuentas falsas: 288 (50%)"
```

```
paste0("Cuentas no falsas: ", count_real, " (", round(percent_real, 2), "%)")
```

```
[1] "Cuentas no falsas: 288 (50%)"
```

2.4.2 profile.pic

Toma el valor 1 para las cuentas con foto de perfil y un 0 para las que no tienen.

```
summary(dataset$profile.pic)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  1.0000  0.7014  1.0000  1.0000
```

La media ya nos indica que en nuestro dataset hay más cuentas con foto de perfil que sin foto.

En resumen:

```
[1] "Cuentas con foto de perfil: 404 (70.14%)"
```

```
[1] "Cuentas sin foto de perfil: 172 (29.86%)"
```

2.4.3 name..username

Toma el valor 1 si el nombre de la persona es igual al username que se ha puesto en Instagram, y 0 en caso contrario.

```
summary(dataset$name..username)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.00000 0.00000 0.03472 0.00000 1.00000
```

La media nos indica que muy pocas cuentas cumplen esta condición.

```
total_count <- length(dataset$name..username)
count_name_equals_username <- sum(dataset$name..username == 1)
count_name_not_equals_username <- sum(dataset$name..username == 0)
percent_name_equals_username <- (count_name_equals_username / total_count) * 100
percent_name_not_equals_username <- (count_name_not_equals_username / total_count) * 100

paste0("Cuentas con mismo nombre y username: ", count_name_equals_username, " (", round(pe
```

```
[1] "Cuentas con mismo nombre y username: 20 (3.47%)"
```

```
paste0("Cuentas con distinto nombre y username: ", count_name_not_equals_username, " (", r
```

```
[1] "Cuentas con distinto nombre y username: 556 (96.53%)"
```

2.4.4 external.URL

Toma el valor 1 si la cuenta tiene un enlace en el perfil, y 0 si no tiene.

```
summary(dataset$external.URL)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  0.0000  0.1163  0.0000  1.0000
```

La mediana es 0 y la media 0.11. Pocas cuentas tienen enlace en el perfil.

```
total_count <- length(dataset$external.URL)
count_with_url <- sum(dataset$external.URL == 1)
count_without_url <- sum(dataset$external.URL == 0)
percent_with_url <- (count_with_url / total_count) * 100
percent_without_url <- (count_without_url / total_count) * 100

paste0("Cuentas con enlace en el perfil: ", count_with_url, " (", round(percent_with_url,
```

```
[1] "Cuentas con enlace en el perfil: 67 (11.63%)"
```

```
paste0("Cuentas sin enlace en el perfil: ", count_without_url, " (", round(percent_without
```

```
[1] "Cuentas sin enlace en el perfil: 509 (88.37%)"
```

2.4.5 private

Toma el valor 1 si la cuenta es privada, y 0 si es pública.

```
summary(dataset$private)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  0.0000  0.3819  1.0000  1.0000
```

```r
total_count <- length(dataset$private)
count_private <- sum(dataset$private == 1)
count_public <- sum(dataset$private == 0)
percent_private <- (count_private / total_count) * 100
percent_public <- (count_public / total_count) * 100

paste0("Cuentas privadas: ", count_private, " (", round(percent_private, 2), "%)")
```

```
[1] "Cuentas privadas: 220 (38.19%)"
```

```r
paste0("Cuentas públicas: ", count_public, " (", round(percent_public, 2), "%)")
```

```
[1] "Cuentas públicas: 356 (61.81%)"
```

2.5 Variables no binarias

Las variables no binarias tienen más información que podemos explorar, las binarias simplemente podíamos ver como se repartían los datos y no mucho más.

Las variables no binarias son:

```r
non_binary <- dataset_values %>%
  filter(unique_values > 2) %>%
  arrange(unique_values)

non_binary
```

```
                     min         max unique_values
fullname.words         0       12.00             9
nums.length.fullname   0        1.00            25
nums.length.username   0        0.92            54
description.length     0      150.00           104
X.posts                0     7389.00           193
X.followers            0 15338538.00           372
X.follows              0     7500.00           400
```

Salvo `fullname.words`, todas las variables tienen un alto número de valores únicos. Vamos a empezar con esta ya que podremos hacer algún que otro gráfico distinto al resto, y seguiremos el orden ascendente del número de valores únicos.

2.5.1 fullname.words

Representa el número de palabras en el nombre completo del usuario.

```
str(dataset$fullname.words)
```

```
int [1:576] 0 2 2 1 2 4 2 2 0 2 ...
```

Vemos que toma valores enteros.

```
summary(dataset$fullname.words)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    1.00    1.00    1.46    2.00   12.00
```

La media es alrededor de la palabra y media, lo que tiene bastante sentido. Sin embargo, la mediana es tener una sola palabra para el nombre.

```
table(dataset$fullname.words)
```

```
  0   1   2   3   4   5   6  10  12
 57 283 187  34   7   4   2   1   1
```

Tenemos bastantes cuentas sin palabras en el nombre completo, lo que es sospechoso.

¿Qué tipo de cuenta tiene más de 1, 2, 3 o como mucho 4 palabras en su nombre completo? Es tan sospechoso como tener 0.

```
total_count <- nrow(dataset)

count_one_word <- sum(dataset$fullname.words == 1)
percent_one_word <- (count_one_word / total_count) * 100

count_multi_word <- sum(dataset$fullname.words > 1)
percent_multi_word <- (count_multi_word / total_count) * 100

count_no_word <- sum(dataset$fullname.words == 0)
percent_no_word <- (count_no_word / total_count) * 100
```

```r
# Mostrar resultados
paste0("Cuentas sin nombre completo: ", count_no_word, " (", round(percent_no_word, 2), "%
```

```
[1] "Cuentas sin nombre completo: 57 (9.9%)"
```

```r
paste0("Cuentas con nombre completo de una palabra: ", count_one_word, " (", round(percent
```

```
[1] "Cuentas con nombre completo de una palabra: 283 (49.13%)"
```

```r
paste0("Cuentas con nombre de más de una palabra: ", count_multi_word, " (", round(percent
```

```
[1] "Cuentas con nombre de más de una palabra: 236 (40.97%)"
```

2.5.2 nums.length.fullname

Ratio del número de carácteres númericos en el nombre completo respecto a la longitud del nombre completo.

```r
str(dataset$nums.length.fullname)
```

```
 num [1:576] 0 0 0 0 0 0 0 0 0 0 ...
```

```r
table(dataset$nums.length.fullname)
```

```
   0 0.08  0.1 0.11 0.12 0.14 0.18  0.2 0.22 0.24 0.25 0.27 0.29 0.31 0.33 0.36
 518    1    1    1    2    1    2    1    3    3    4    1    1    3   11    2
0.38  0.4 0.43 0.44 0.46  0.5 0.57 0.89    1
   1    7    2    2    1    3    1    1    3
```

Tenemos una gran concentración en el número 0 y vemos que la variable toma valores entre 0 y 1, de forma continua.

```r
summary(dataset$nums.length.fullname)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.00000 0.00000 0.03609 0.00000 1.00000
```

Lo normal es no tener carácteres en el nombre completo (recordamos que es el nombre de la persona, no el username).

```
total_count <- nrow(dataset)

count_with_numbers <- sum(dataset$nums.length.fullname > 0)
percent_with_numbers <- (count_with_numbers / total_count) * 100

count_no_numbers <- sum(dataset$nums.length.fullname == 0)
percent_no_numbers <- (count_no_numbers / total_count) * 100

paste0("Cuentas con números en el nombre completo: ",
       count_with_numbers, " (", round(percent_with_numbers, 2), "%)")
```

```
[1] "Cuentas con números en el nombre completo: 58 (10.07%)"
```

```
paste0("Cuentas sin números en el nombre completo: ",
       count_no_numbers, " (", round(percent_no_numbers, 2), "%)")
```

```
[1] "Cuentas sin números en el nombre completo: 518 (89.93%)"
```

2.5.3 nums.length.username

Ratio del número de carácteres númericos en el nombre de usuario respecto a la longitud del nombre de usuario.

Muy parecido a la variable anterior, pero esta vez es respecto al nombre de usuario de Instagram. Probablemente haya ciertas diferencias como que será más común tener algún número en el username.

```
str(dataset$nums.length.username)
```

```
 num [1:576] 0.27 0 0.1 0 0 0 0 0 0 0 ...
```

Toma valores entre 0 y 1, como la variable anterior.

```r
summary(dataset$nums.length.username)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.0000  0.0000  0.0000  0.1638  0.3100  0.9200 
```

Se observa justamente lo que hemos comentado, es más común tener algún número en el username, respecto a los números en el nombre completo de la persona.

```r
total_count <- nrow(dataset)

count_with_numbers <- sum(dataset$nums.length.username > 0)
percent_with_numbers <- (count_with_numbers / total_count) * 100

count_no_numbers <- sum(dataset$nums.length.username == 0)
percent_no_numbers <- (count_no_numbers / total_count) * 100

paste0("Cuentas con números en el nombre completo: ",
       count_with_numbers, " (", round(percent_with_numbers, 2), "%)")
```

```
[1] "Cuentas con números en el nombre completo: 277 (48.09%)"
```

```r
paste0("Cuentas sin números en el nombre completo: ",
       count_no_numbers, " (", round(percent_no_numbers, 2), "%)")
```

```
[1] "Cuentas sin números en el nombre completo: 299 (51.91%)"
```

Prácticamente la mitad de usuarios tiene algún número en el nombre de usuario, mientras que en nombre completo, pocos tenían.

2.5.4 description.length

La longitud de la descripción de la cuenta de Instagram.

```r
str(dataset$description.length)
```

```
 int [1:576] 53 44 0 82 0 81 50 0 71 40 ...
```

Toma valores enteros, como es lógico por el significado de la variable.

```
summary(dataset$description.length)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   0.00    0.00    0.00   22.62   34.00  150.00 
```

La media es de unos 23 carácteres en la descripción, pero probablemente esté muy afectado por las cuentas sin descripción, ya que la mediana es 0.

```
count_with_description <- sum(dataset$description.length > 0)
percent_with_description <- (count_with_description / total_count) * 100

count_no_description <- sum(dataset$description.length == 0)
percent_no_description <- (count_no_description / total_count) * 100

paste0("Cuentas con descripción: ",
       count_with_description, " (", round(percent_with_description, 2), "%)")
```

```
[1] "Cuentas con descripción: 250 (43.4%)"
```

```
paste0("Cuentas sin descripción: ",
       count_no_description, " (", round(percent_no_description, 2), "%)")
```

```
[1] "Cuentas sin descripción: 326 (56.6%)"
```

Hay más cuentas sin descripción que con descripción. Vamos a calcular la media sin tener en cuenta todas esas cuentas que no tienen descripción:

```
cuentas_con_descripcion = dataset %>%
  filter(description.length > 0)

paste0("La media de carácteres en la descripción es: ", mean(cuentas_con_descripcion$descr
```

```
[1] "La media de carácteres en la descripción es: 52.124"
```

2.5.5 X.posts

Representa el número de publicaciones que tiene una cuenta de Instagram.

```r
str(dataset$X.posts)
```

```
 int [1:576] 32 286 13 679 6 344 16 33 72 213 ...
```

De nuevo, valores enteros.

```r
summary(dataset$X.posts)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    0.0     0.0     9.0   107.5    81.5  7389.0 
```

En este caso no pasa como con la descripción que la mediana era directamente 0, pero de nuevo vemos que la media es muy alta y la mediana muy baja.

```r
count_with_posts <- sum(dataset$X.posts > 0)
count_no_posts <- sum(dataset$X.posts == 0)

percent_with_posts <- (count_with_posts / total_count) * 100
percent_no_posts <- (count_no_posts / total_count) * 100

paste0("Cuentas con publicaciones: ", count_with_posts, " (", round(percent_with_posts, 2)
```

```
[1] "Cuentas con publicaciones: 419 (72.74%)"
```

```r
paste0("Cuentas sin publicaciones: ", count_no_posts, " (", round(percent_no_posts, 2), "%
```

```
[1] "Cuentas sin publicaciones: 157 (27.26%)"
```

Tenemos muchas cuentas sin publicaciones teniendo en cuenta que Instagram se usa principalmente para publicar…

Vamos a ver la media de publicaciones quitando las cuentas sin publicaciones y los outliers que tienen más de 1000 publicaciones:

```r
cuentas_menos_1000_posts <- dataset %>%
  filter(X.posts > 0 & X.posts <= 1000)
```

```r
paste0("Media: ", round(mean(cuentas_menos_1000_posts$X.posts), 2))
```

```
[1] "Media: 106.52"
```

2.5.6 X.followers

Representa el número de seguidores que tiene una cuenta de Instagram.

```r
str(dataset$X.followers)
```

```
 int [1:576] 1000 2740 159 414 151 669987 122 1078 1824 12945 ...
```

```r
summary(dataset$X.followers)
```

```
   Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
      0       39      150    85307      716 15338538 
```

Volvemos a tener una media mucho más alta que la mediana. Esta variable tiene un rango de valores muy disperso.

```r
cuentas_con_seguidores <- sum(dataset$X.followers > 0)
cuentas_sin_seguidores <- sum(dataset$X.followers == 0)

percent_con_seguidores <- (cuentas_con_seguidores / total_count) * 100
percent_sin_seguidores <- (cuentas_sin_seguidores / total_count) * 100

paste0("Cuentas con seguidores: ", cuentas_con_seguidores, " (", round(percent_con_seguido
```

```
[1] "Cuentas con seguidores: 558 (96.88%)"
```

```r
paste0("Cuentas sin seguidores: ", cuentas_sin_seguidores, " (", round(percent_sin_seguido
```

```
[1] "Cuentas sin seguidores: 18 (3.12%)"
```

Observamos que hay muy pocas cuentas sin seguidores.

Vamos a ver la media de seguidores quitando las cuentas sin seguidores y las cuentas que tienen más de 10000 seguidores.

```r
cuentas_filtradas_seguidores <- dataset %>%
  filter(X.followers > 0 & X.followers <= 10000)

paste0("Media: ", round(mean(cuentas_filtradas_seguidores$X.followers), 2))
```

```
[1] "Media: 545.35"
```

2.5.7 X.follows

Representa el número de personas que sigue una cuenta de Instagram.

```r
str(dataset$X.follows)
```

```
 int [1:576] 955 533 98 651 126 150 177 76 2713 813 ...
```

```r
summary(dataset$X.follows)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    0.0    57.5   229.5   508.4   589.5  7500.0 
```

En este caso la media y la mediana no están tan alejadas como en los otros casos.

```r
cuentas_con_seguidos <- sum(dataset$X.followers > 0)
cuentas_sin_seguidos <- sum(dataset$X.followers == 0)

percent_con_seguidos <- (cuentas_con_seguidores / total_count) * 100
percent_sin_seguidos <- (cuentas_sin_seguidores / total_count) * 100

paste0("Cuentas con seguidos: ", cuentas_con_seguidos, " (", round(percent_con_seguidos, 2
```

```
[1] "Cuentas con seguidos: 558 (96.88%)"
```

```r
paste0("Cuentas sin seguidos: ", cuentas_sin_seguidos, " (", round(percent_sin_seguidos, 2
```

```
[1] "Cuentas sin seguidos: 18 (3.12%)"
```

Observamos que hay muy pocas cuentas sin seguidos. Vamos a ver si las cuentas que no siguen a nadie son las mismas que tampoco tienen seguidores:

```r
cuentas_sin_seguidos_ni_seguidores <- dataset %>%
  filter(X.followers == 0 & X.follows == 0)

nrow(cuentas_sin_seguidos_ni_seguidores)
```

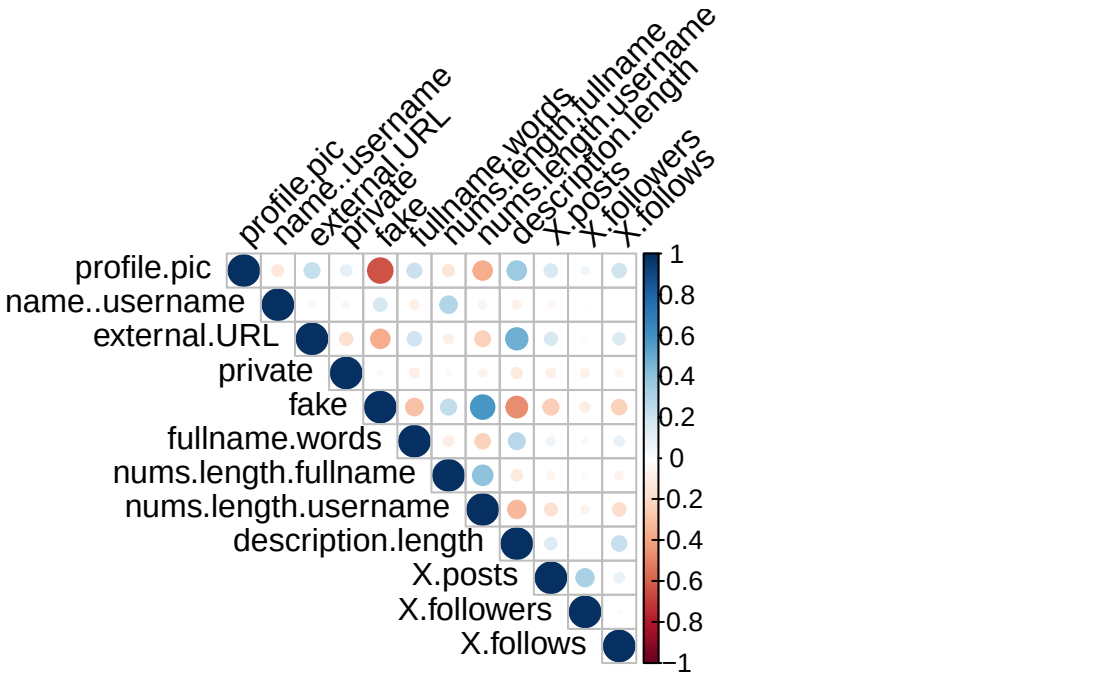
```
[1] 3
```

Pues resulta ser que no, solo hay 3 cuentas que no tienen ni seguidores ni seguidos.

2.6 Relaciones entre variables

Matriz de correlación entre variables binarias y no binarias

```r
cor_bin_nonbin <- cor(dataset[, c(binary_vars, rownames(non_binary))])
corrplot(cor_bin_nonbin, method="circle", type="upper", tl.col="black", tl.srt=45)
```



Estadísticas descriptivas comparativas entre cuentas falsas y reales

```r
fake_stats <- sapply(dataset[dataset$fake == 1, rownames(non_binary)],
                     function(x) c(round(mean(x), 2), median(x), round(sd(x), 2)))
real_stats <- sapply(dataset[dataset$fake == 0, rownames(non_binary)],
                     function(x) c(round(mean(x), 2), median(x), round(sd(x), 2)))

comparison_stats <- data.frame(Fake = fake_stats, Real = real_stats)
rownames(comparison_stats) <- c("Media", "Mediana", "Desviación típica")
t(comparison_stats)
```

```
                          Media Mediana Desviación típica
Fake.fullname.words        1.15     1.0              0.68
Fake.nums.length.fullname  0.07     0.0              0.17
Fake.nums.length.username  0.29     0.3              0.23
Fake.description.length    5.26     0.0             20.29
Fake.X.posts               8.93     0.0             28.01
Fake.X.followers         110.59    40.0            318.41
Fake.X.follows           302.17    70.0            705.06
Real.fullname.words        1.77     2.0              1.25
Real.nums.length.fullname  0.01     0.0              0.03
Real.nums.length.username  0.04     0.0              0.09
Real.description.length   39.98    27.0             42.79
Real.X.posts             206.05    74.0            550.95
Real.X.followers      170503.89   661.5        1282598.58
Real.X.follows           714.60   431.0           1051.60
```

La tabla proporciona una comparación entre las estadísticas descriptivas de las cuentas falsas y reales en Instagram.

Observamos que, en promedio, las cuentas falsas tienden a tener menos palabras en el nombre completo y una longitud de descripción más corta en comparación con las cuentas reales.

Además, las cuentas falsas muestran un número significativamente menor de seguidores y personas seguidas en comparación con las cuentas reales, como indican las medias y medianas mucho más bajas en estas variables.

Por otro lado, las diferencias en las desviaciones estándar sugieren una mayor variabilidad en el número de seguidores y personas seguidas para las cuentas reales en comparación con las falsas.

2.7 Cuestiones generales

Vamos a explorar una serie de cuestiones generales acerca de nuestros datos, para entender mejor como se distribuyen nuestros datos y explorar relaciones interesantes:

- **¿Cuál es la proporción de perfiles privados con foto de perfil?**

```
prop.table(table(
  ifelse(dataset$private == 1, "Privada", "Publica"),
  ifelse(dataset$profile.pic == 1, "Con foto", "Sin foto")))
```

```
          Con foto   Sin foto
Privada 0.29340278 0.08854167
Publica 0.40798611 0.21006944
```

Lo más común es tener una cuenta pública y con foto

- **¿Cuál es la media de seguidores de los perfiles falsos frente a los públicos?**

```
medias <- dataset %>%
  group_by(fake) %>%
  summarise(mean(X.followers))

data.frame(
  Perfil=c("Real", "Falso"),
  "Media de seguidores"=medias$`mean(X.followers)`)
```

```
  Perfil Media.de.seguidores
1   Real          170503.8854
2  Falso             110.5868
```

- **¿Existe una correlación entre la longitud de la descripción y la cantidad de seguidores?**

```
cor(as.numeric(dataset$description.length), as.numeric(dataset$X.followers))
```

```
[1] 0.005929455
```

Parece ser que no.

- **¿Cuál es la media del número de publicaciones según si el perfil es privado o no?**

```
medias <- dataset %>%
  group_by(private) %>%
  summarise(mean(X.posts))

data.frame(
  Cuenta=c("Pública", "Privada"),
  "Media de publicaciones"=medias$`mean(X.posts)`)
```

```
   Cuenta Media.de.publicaciones
1 Pública              135.11798
2 Privada               62.78182
```

- **¿Cuál es la proporción de perfiles con más de 1000 seguidores falsos frente a los reales?**

```
table(dataset$fake,
      dataset$X.followers > 1000) / rowSums(table(dataset$fake,
                                                  dataset$X.followers > 1000))
```

```
       FALSE       TRUE
  0 0.64236111 0.35763889
  1 0.98263889 0.01736111
```

El 98.26% de las cuentas falsas tienen menos de 1000 seguidores. El 35.7% de las cuentas reales tienen más de 1000 seguidores.

- **¿Qué porentaje de cuentas tienen números en el nombre real, en función de perfiles falsos y reales?**

```
medias <- dataset %>%
  mutate(has.nums.fullname=as.numeric(nums.length.fullname > 0)) %>%
  group_by(fake) %>%
  summarise(mean(has.nums.fullname))

data.frame(
  Cuenta=c("Real", "Falso"),
  "Media del ratio de números en el nombre real"=medias$`mean(has.nums.fullname)`)
```

```
  Cuenta Media.del.ratio.de.números.en.el.nombre.real
1   Real                                   0.02777778
2  Falso                                   0.17361111
```

El 2% de las cuentas reales tienen números en su nombre real mientras que el 17% de las cuentas falsas tienen números en el nombre real. Claramente tener números en el nombre real es algo sospechoso.

- **¿Cuáles son las cuentas falsas con más seguidores?**

```r
seguidores_fake <- dataset %>%
  filter(fake == 1) %>%
  select(X.followers, fake) %>%
  arrange(desc(X.followers))

head(seguidores_fake)
```

```
  X.followers fake
1        3033    1
2        3003    1
3        2346    1
4        1489    1
5        1031    1
6         864    1
```

No hay ninguna cuenta falsa con más de 3033 seguidores.

- **¿Cuántas cuentas reales tienen más seguidores que la cuenta falsa con más seguidores?**

```r
seguidores_real <- dataset %>%
  filter(fake == 0 & X.followers > seguidores_fake$X.followers[1]) %>%
  select(X.followers, fake) %>%
  arrange(desc(X.followers))

head(seguidores_real)
```

```
  X.followers fake
1    15338538    0
2    12397719    0
3     6741307    0
```

```
4     5315651    0
5     3896490    0
6      890969    0
```

```
paste0("Hay ", nrow(seguidores_real), " cuentas reales con más de ", seguidores_fake$X.fol
```

```
[1] "Hay 51 cuentas reales con más de 3033 seguidores"
```

Interesante, el 9% del dataset se podría directamente clasificar como cuenta real, al tener más seguidores que la cuenta falsa con más seguidores.

2.8 Conclusiones

Con este análisis exploratorio hemos podido entender mejor la naturaleza de nuestros datos, entendiendo mejor la distribución de los mismos y pudiendo ver las primeras relaciones entre distintas variables.

El carácter númerico de nuestros datos nos permitirá que técnicas como la regresión lineal nos puedan dar buenos resultados, además de que, como hemos visto, hay ciertas relaciones clave entre distintas variables que podremos explotar.

Un buen análisis exploratorio es muy importante y puede marcar el desarrollo del estudio de nuestros datos. Hemos obtenido un conocimiento inicial que nos vendrá muy bien cuando comencemos a aplicar las primeras técnicas de análisis de datos.

En el siguiente apratado, visualización de datos, vamos a visualizar algunos de los puntos que hemos tratado en este análisis exploratorio, para poder hacernos un modelo mental aún más cercano a los datos.

3 Visualización de datos

La visualización de datos es una técnica importante que complementa y apoya el análisis exploratorio. Nos vamos a centrar en crear gráficos que ilustren como se distribuyen nuestros datos, y las relaciones entre los mismos.

Para ello, vamos a usar la librería ggplot2, que permite realizar gráficos de una forma muy intuitiva y sencilla, siendo totalmente configurables a nuestro estilo.

```
library(dplyr)
library(ggplot2)
library(tidyr)
```

Comenzaremos con un análisis univariado de nuestras variables binarias, luego haremos lo mismo para las variables no binarias, y por último realizaremos un análisis multivariado en el que visualicemos relaciones entre distintas variables de nuestro dataset.

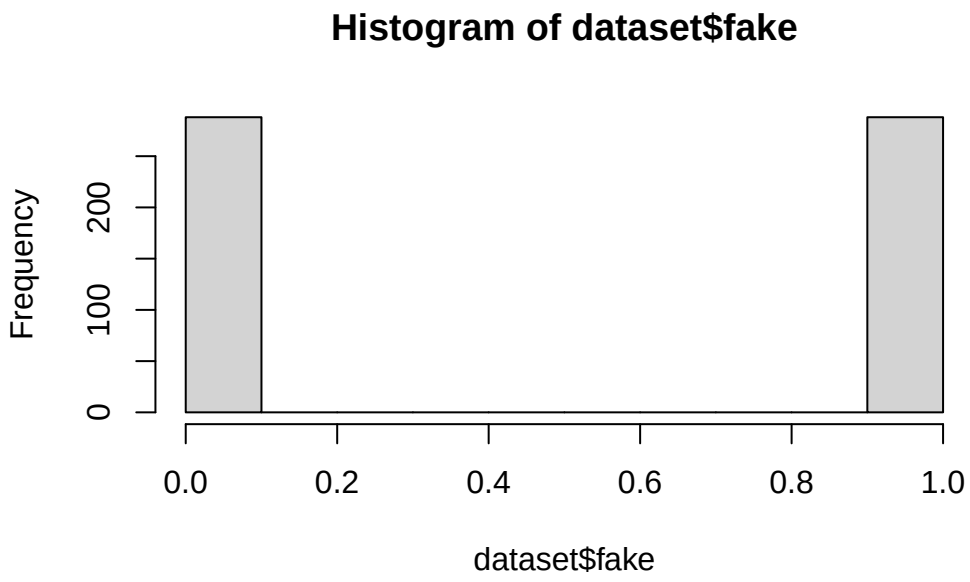
```
dataset <- read.csv("datasets/train.csv")
```

3.1 Variables binarias

Vamos a comenzar visualizando las variables binarias, que como vimos en el análisis exploratorio son fake, profile.pic, name..username, external.URL y private.

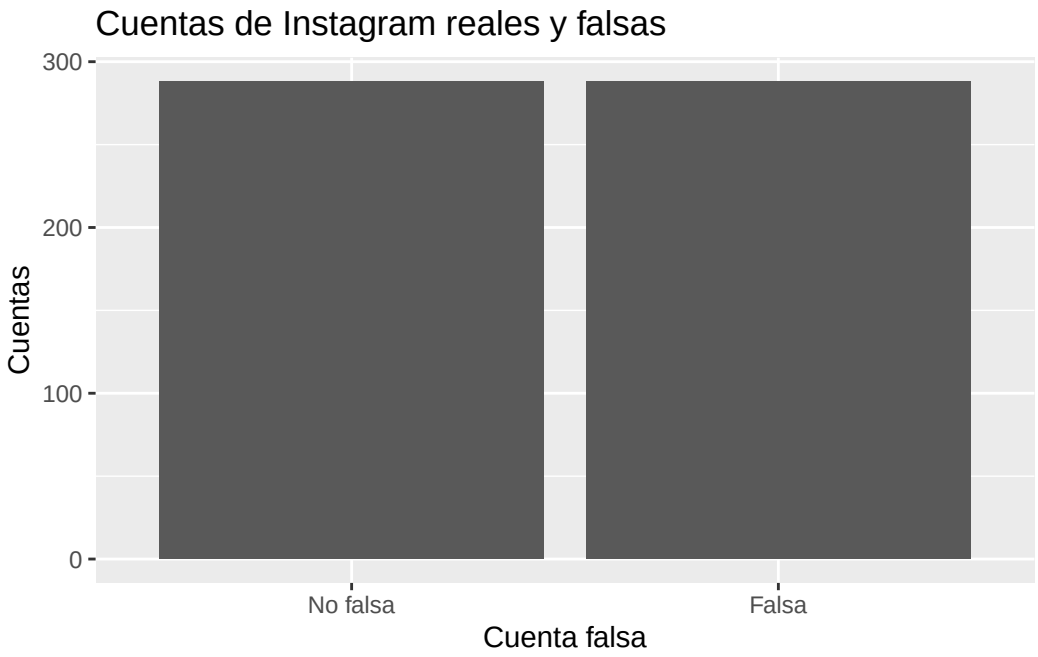
3.1.1 fake

```
hist(dataset$fake)
```



El histograma no es lo más adecuado para visualizar variables binarias (aunque se visualiza claramente), vamos a hacer a partir de ahora los gráficos de las variables binarias con ggplot. Podemos visualizarlo con un gráfico de barras:

```
ggplot(dataset, aes(x = factor(fake))) +
  geom_bar() +
  labs(x = "Cuenta falsa", y = "Cuentas",
       title = "Cuentas de Instagram reales y falsas") +
  scale_x_discrete(labels = c("0" = "No falsa", "1" = "Falsa"))
```



O con un gráfico circular:

```
fake_df <- data.frame(
  category = c("No falsa", "Falsa"),
  count = c(sum(dataset$fake == 0), sum(dataset$fake == 1))
)
ggplot(fake_df, aes(x = "", y = count, fill = category)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  theme_void() +
  labs(title = "Proporción de cuentas reales y falsas",
       fill = "Cuenta falsa") +
  scale_fill_manual(values =
                      c("No falsa" = "#FF9999", "Falsa" = "#99DDFF"))
```

Este gráfico es muy adecuado para la visualización que queremos hacer y lo usaremos en el resto de variables binarias o con pocos valores únicos.

```
[1] "Cuentas falsas: 288 (50%)"
```

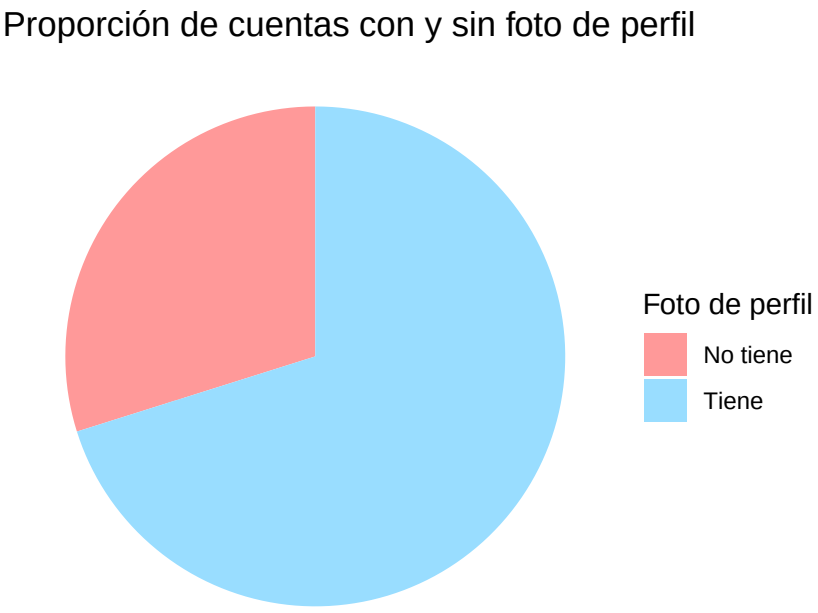
```
[1] "Cuentas no falsas: 288 (50%)"
```

3.1.2 profile.pic

Toma el valor 1 para las cuentas con foto de perfil y un 0 para las que no tienen.

```
profile_pic_df <- data.frame(
  category = c("No tiene", "Tiene"),
  count = c(sum(dataset$profile.pic == 0), sum(dataset$profile.pic == 1))
)
ggplot(profile_pic_df, aes(x = "", y = count, fill = category)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  theme_void() +
  labs(title = "Proporción de cuentas con y sin foto de perfil",
       fill = "Foto de perfil") +
```

```
  scale_fill_manual(values =
                      c("No tiene" = "#FF9999", "Tiene" = "#99DDFF"))
```




```
[1] "Cuentas con foto de perfil: 404 (70.14%)"
```

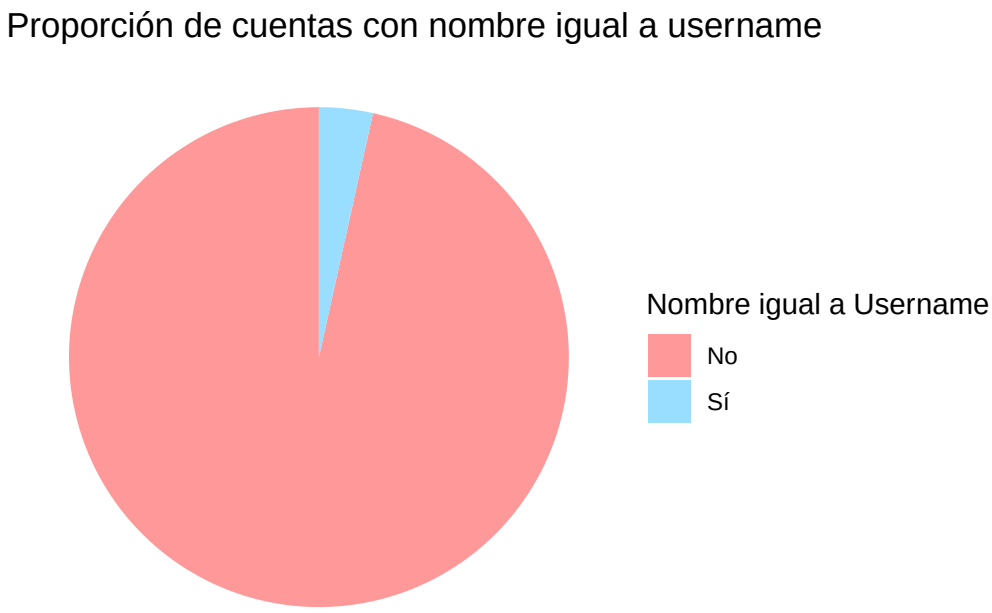
```
[1] "Cuentas sin foto de perfil: 172 (29.86%)"
```

3.1.3 name..username

Toma el valor 1 si el nombre de la persona es igual al username que se ha puesto en Instagram, y 0 en caso contrario.

```
name_username_df <- data.frame(
  category = c("No", "Sí"),
  count = c(sum(dataset$name..username == 0), sum(dataset$name..username == 1))
)
ggplot(name_username_df, aes(x = "", y = count, fill = category)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  theme_void() +
  labs(title = "Proporción de cuentas con nombre igual a username",
```

```
       fill = "Nombre igual a Username") +
  scale_fill_manual(values = c("No" = "#FF9999", "Sí" = "#99DDFF"))
```




```
[1] "Cuentas con mismo nombre y username: 20 (3.47%)"
```

```
[1] "Cuentas con distinto nombre y username: 556 (96.53%)"
```

3.1.4 external.URL

Toma el valor 1 si la cuenta tiene un enlace en el perfil, y 0 si no tiene.

```
external_url_df <- data.frame(
  category = c("No tiene", "Tiene"),
  count = c(sum(dataset$external.URL == 0), sum(dataset$external.URL == 1))
)
ggplot(external_url_df, aes(x = "", y = count, fill = category)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  theme_void() +
  labs(title = "Proporción de cuentas con y sin enlace en el perfil",
       fill = "Enlace en el perfil") +
```

```
  scale_fill_manual(values = c("No tiene" = "#FF9999", "Tiene" = "#99DDFF"))
```

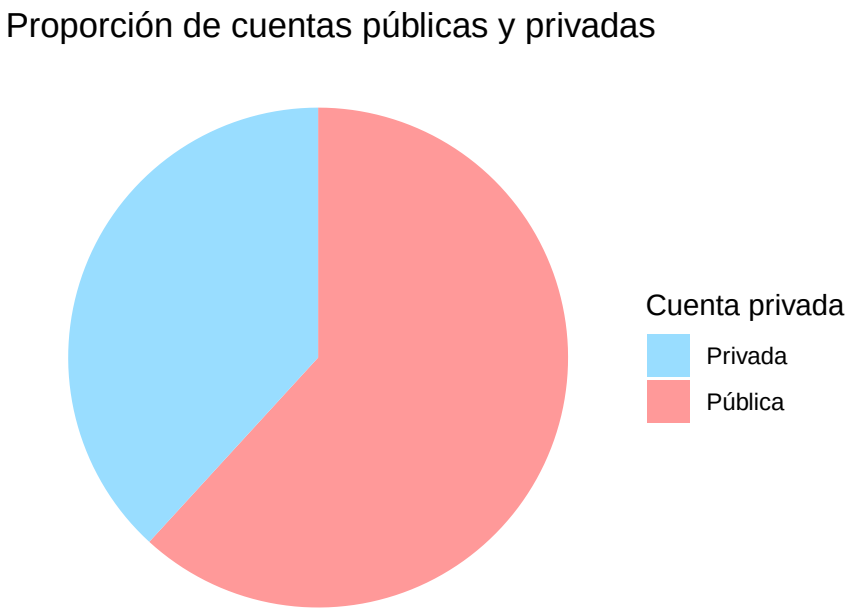

```
[1] "Cuentas con enlace en el perfil: 67 (11.63%)"
```

```
[1] "Cuentas sin enlace en el perfil: 509 (88.37%)"
```

3.1.5 private

Toma el valor 1 si la cuenta es privada, y 0 si es pública.

```
private_df <- data.frame(
  category = c("Pública", "Privada"),
  count = c(sum(dataset$private == 0), sum(dataset$private == 1))
)
ggplot(private_df, aes(x = "", y = count, fill = category)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  theme_void() +
  labs(title = "Proporción de cuentas públicas y privadas", fill = "Cuenta privada") +
  scale_fill_manual(values = c("Pública" = "#FF9999", "Privada" = "#99DDFF"))
```




```
[1] "Cuentas privadas: 220 (38.19%)"
```

```
[1] "Cuentas públicas: 356 (61.81%)"
```

3.2 Variables no binarias

Como vimos en el análisis exploratorio, las variables no binarias son fullname.words, nums.length.fullname, nums.length.username, description.length, X.posts, X.followers y X.follows.

3.2.1 fullname.words

Representa el número de palabras en el nombre completo del usuario.

```
ggplot(dataset, aes(x = fullname.words)) +
  geom_histogram(binwidth = 1, fill = "#99DDFF", color = "black", alpha = 0.7) +
  labs(x = "Longitud de Descripción", y = "Palabras",
       title = "Número de palabras en el nombre completo")
```

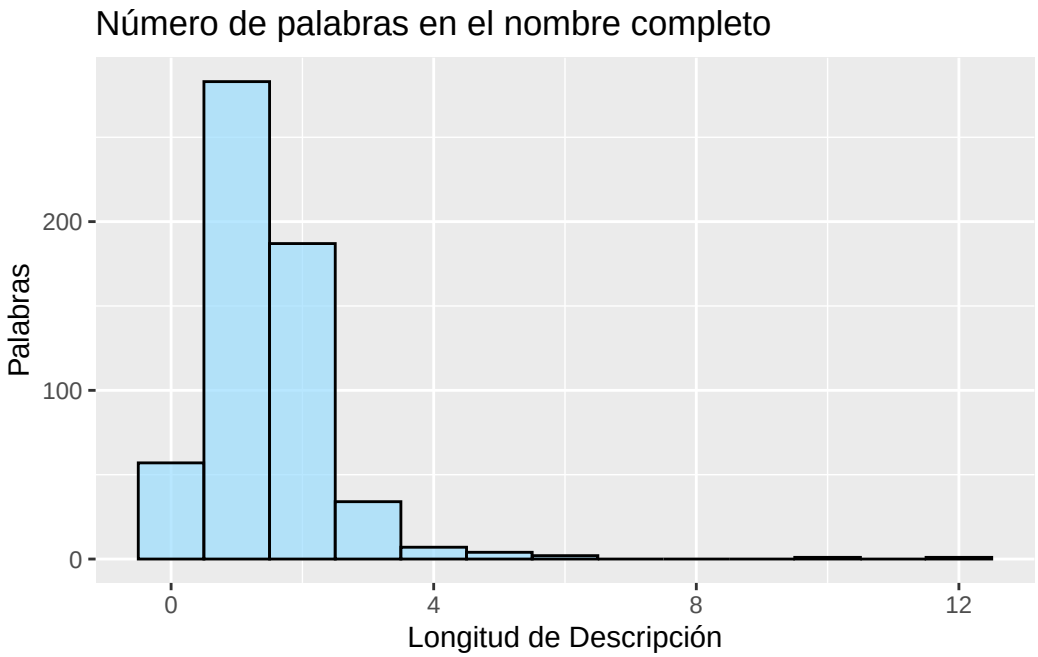


Gráfico circular:

```
fullname_words_df <- as.data.frame(table(dataset$fullname.words))
ggplot(fullname_words_df, aes(x = "", y = Freq, fill = Var1)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  theme_void() +
  labs(title = "Número de palabras en el nombre completo", fill = "Palabras")
```


```
[1] "Cuentas sin nombre completo: 57 (9.9%)"
```

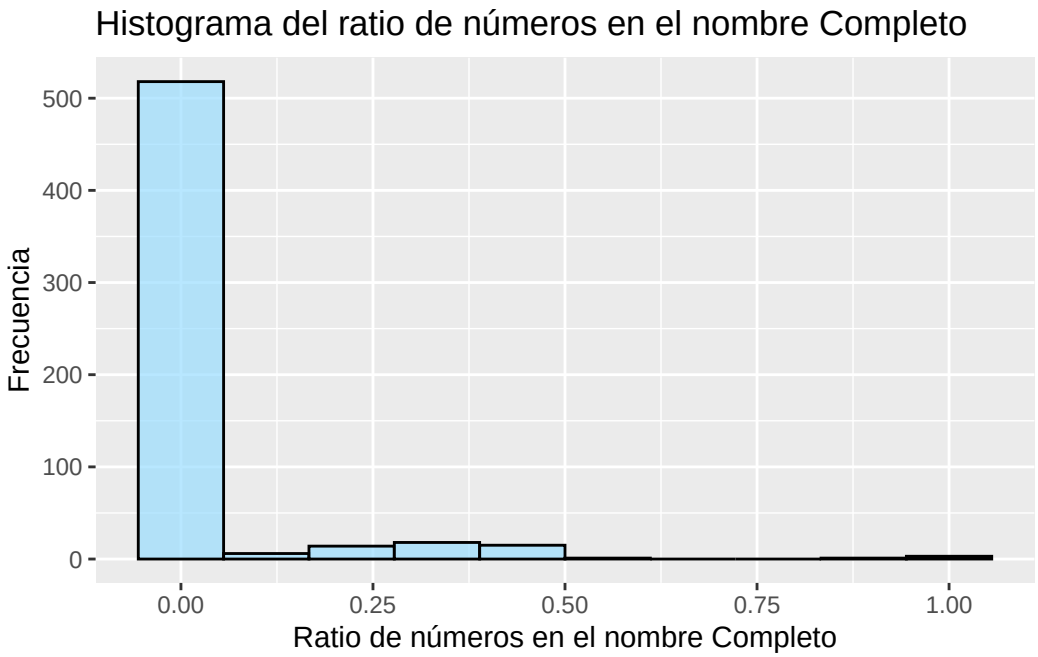
```
[1] "Cuentas con nombre completo de una palabra: 283 (49.13%)"
```

```
[1] "Cuentas con nombre de más de una palabra: 236 (40.97%)"
```

3.2.2 nums.length.fullname

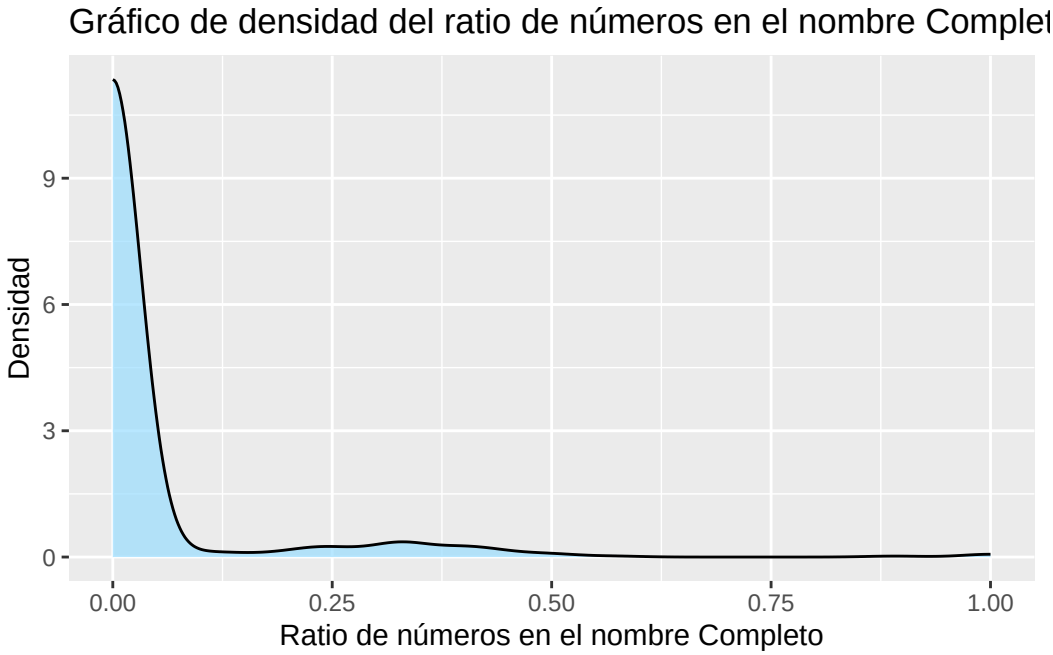
Ratio del número de carácteres númericos en el nombre completo respecto a la longitud del nombre completo.

```
ggplot(dataset, aes(x = nums.length.fullname)) +
  geom_histogram(bins = 10, fill = "#99DDFF", color = "black", alpha = 0.7) +
  labs(x = "Ratio de números en el nombre Completo", y = "Frecuencia",
       title = "Histograma del ratio de números en el nombre Completo")
```



Veamos el gráfico de densidad:

```
ggplot(dataset, aes(x = nums.length.fullname)) +
  geom_density(fill = "#99DDFF", alpha = 0.7) +
  labs(x = "Ratio de números en el nombre Completo",
       y = "Densidad", title =
         "Gráfico de densidad del ratio de números en el nombre Completo")
```



Veamos la cantidad de cuentas con y sin números en el nombre completo.

```
count_summary <- data.frame(
  Categoria = c("Tiene", "No tiene"),
  Cantidad = c(count_with_numbers, count_no_numbers)
)

# Gráfico de barras
ggplot(count_summary, aes(x = Categoria, y = Cantidad, fill = Categoria)) +
  geom_bar(stat = "identity") +
  labs(
    x = "Numeros en el nombre completo",
    y = "Cantidad de cuentas",
    title = "Cuentas con y sin números en el nombre completo"
  ) +
  scale_fill_manual(values = c("Tiene" = "#FF9999", "No tiene" = "#99DDFF"))
```


```
[1] "Cuentas con números en el nombre completo: 58 (10.07%)"
```

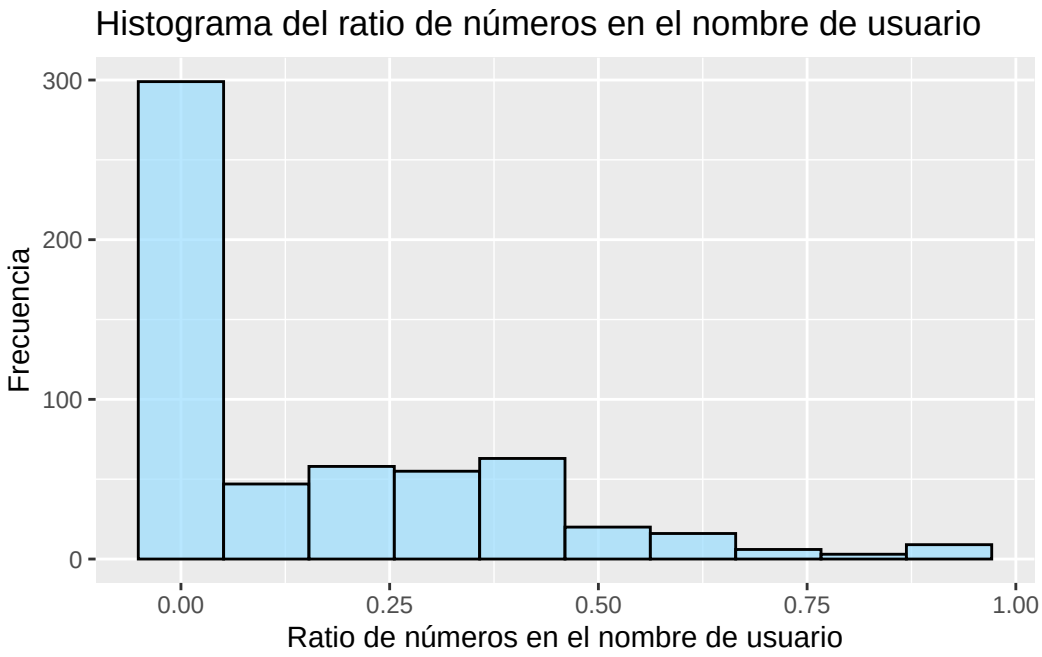
```
[1] "Cuentas sin números en el nombre completo: 518 (89.93%)"
```

3.2.3 nums.length.username

Ratio del número de carácteres númericos en el nombre de usuario respecto a la longitud del nombre de usuario.

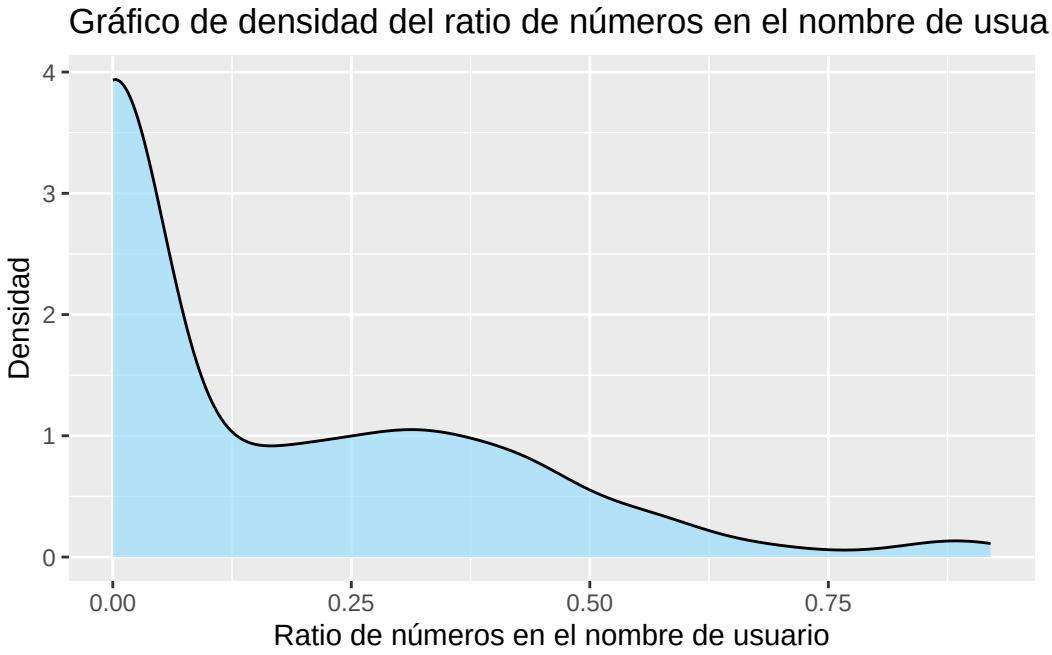
Visualizamos:

```
ggplot(dataset, aes(x = nums.length.username)) +
  geom_histogram(bins = 10, fill = "#99DDFF", color = "black", alpha = 0.7) +
  labs(x = "Ratio de números en el nombre de usuario", y = "Frecuencia",
       title = "Histograma del ratio de números en el nombre de usuario")
```



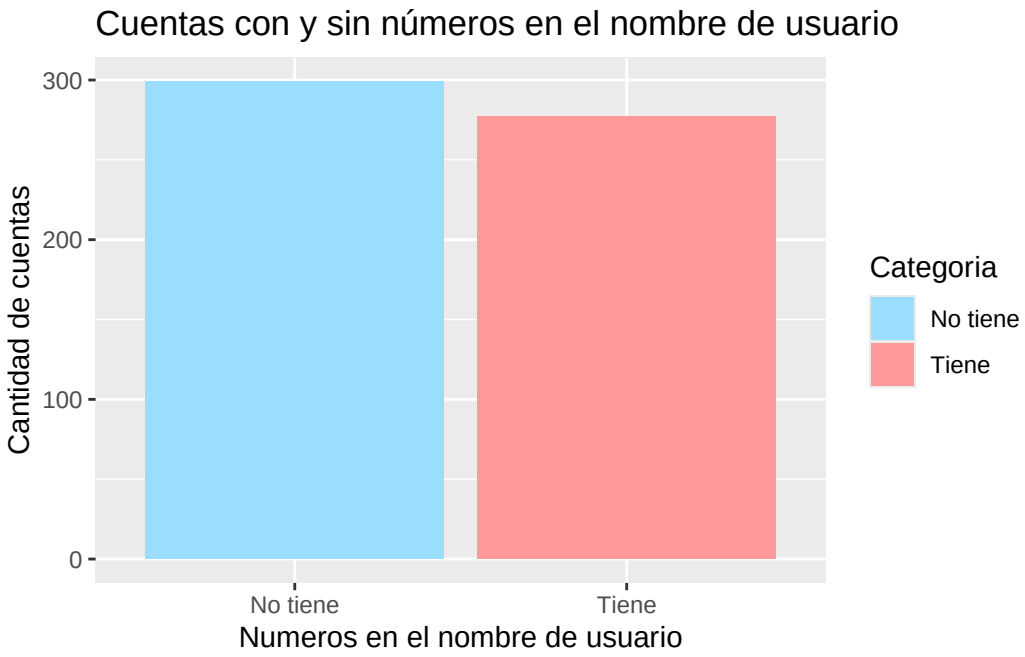
Veamos el gráfico de densidad:

```
ggplot(dataset, aes(x = nums.length.username)) +
  geom_density(fill = "#99DDFF", alpha = 0.7) +
  labs(x = "Ratio de números en el nombre de usuario",
       y = "Densidad", title =
         "Gráfico de densidad del ratio de números en el nombre de usuario")
```



Veamos la cantidad de cuentas con y sin números en el nombre completo.

```r
count_summary <- data.frame(
  Categoria = c("Tiene", "No tiene"),
  Cantidad = c(count_with_numbers, count_no_numbers)
)

# Gráfico de barras
ggplot(count_summary, aes(x = Categoria, y = Cantidad, fill = Categoria)) +
  geom_bar(stat = "identity") +
  labs(
    x = "Numeros en el nombre de usuario",
    y = "Cantidad de cuentas",
    title = "Cuentas con y sin números en el nombre de usuario"
  ) +
  scale_fill_manual(values = c("Tiene" = "#FF9999", "No tiene" = "#99DDFF"))
```



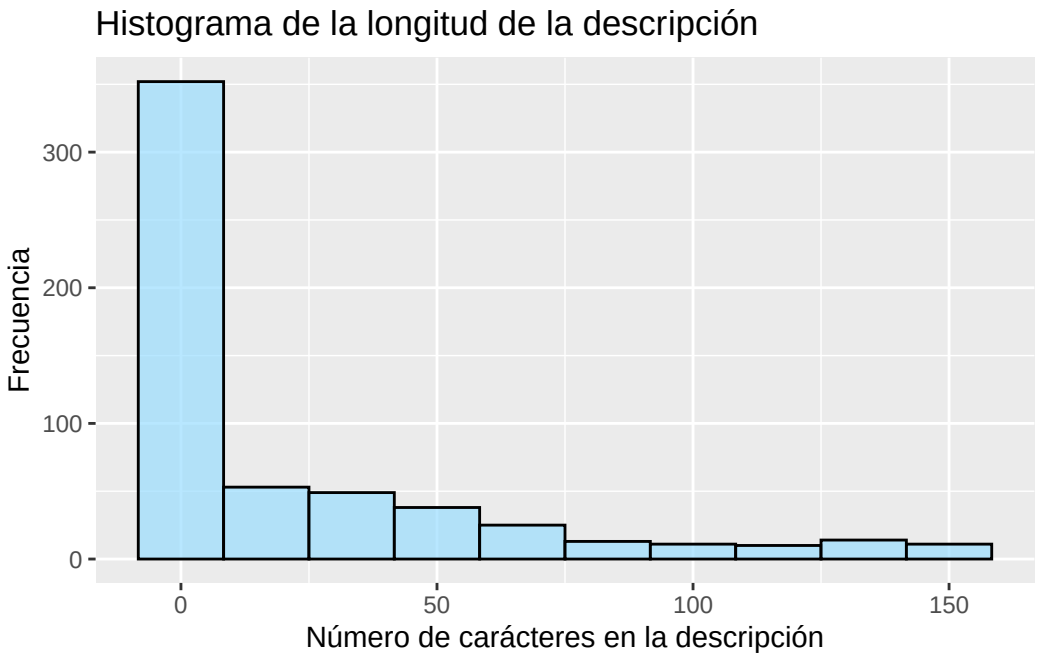

```
[1] "Cuentas con números en el nombre completo: 277 (48.09%)"
```

```
[1] "Cuentas sin números en el nombre completo: 299 (51.91%)"
```

3.2.4 description.length

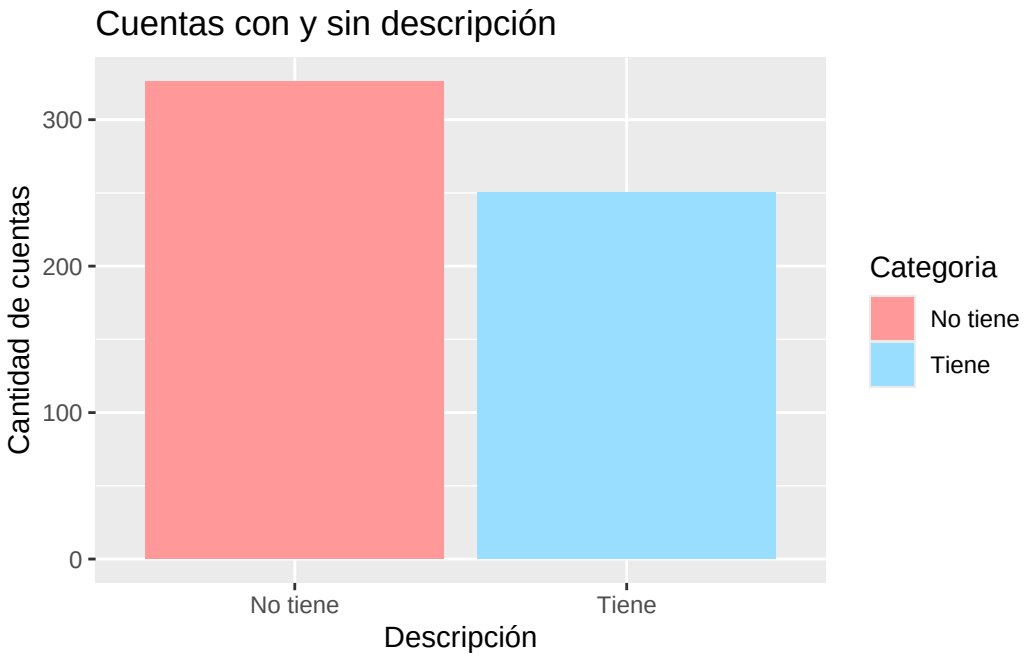
La longitud de la descripción de la cuenta de Instagram.

```
ggplot(dataset, aes(x = description.length)) +
  geom_histogram(bins = 10, fill = "#99DDFF", color = "black", alpha = 0.7) +
  labs(x = "Número de carácteres en la descripción", y = "Frecuencia",
       title = "Histograma de la longitud de la descripción")
```



Se observa claramente que hay muchas cuentas que prácticamente no tienen carácteres en la descripción.

```
count_summary <- data.frame(
  Categoria = c("Tiene", "No tiene"),
  Cantidad = c(count_with_description, count_no_description)
)

ggplot(count_summary, aes(x = Categoria, y = Cantidad, fill = Categoria)) +
  geom_bar(stat = "identity") +
  labs(
    x = "Descripción",
    y = "Cantidad de cuentas",
    title = "Cuentas con y sin descripción"
  ) +
  scale_fill_manual(values = c("No tiene" = "#FF9999", "Tiene" = "#99DDFF"))
```



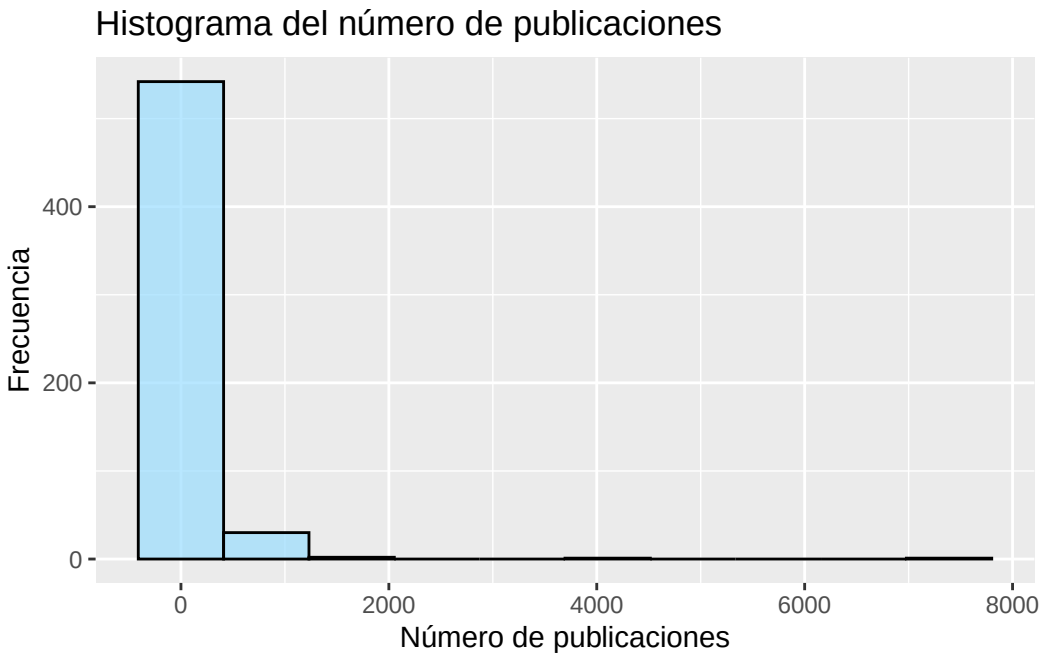

```
[1] "Cuentas con descripción: 250 (43.4%)"
```

```
[1] "Cuentas sin descripción: 326 (56.6%)"
```

3.2.5 X.posts

Representa el número de publicaciones que tiene una cuenta de Instagram.

```r
ggplot(dataset, aes(x = X.posts)) +
  geom_histogram(bins = 10, fill = "#99DDFF", color = "black", alpha = 0.7) +
  labs(x = "Número de publicaciones", y = "Frecuencia",
       title = "Histograma del número de publicaciones")
```

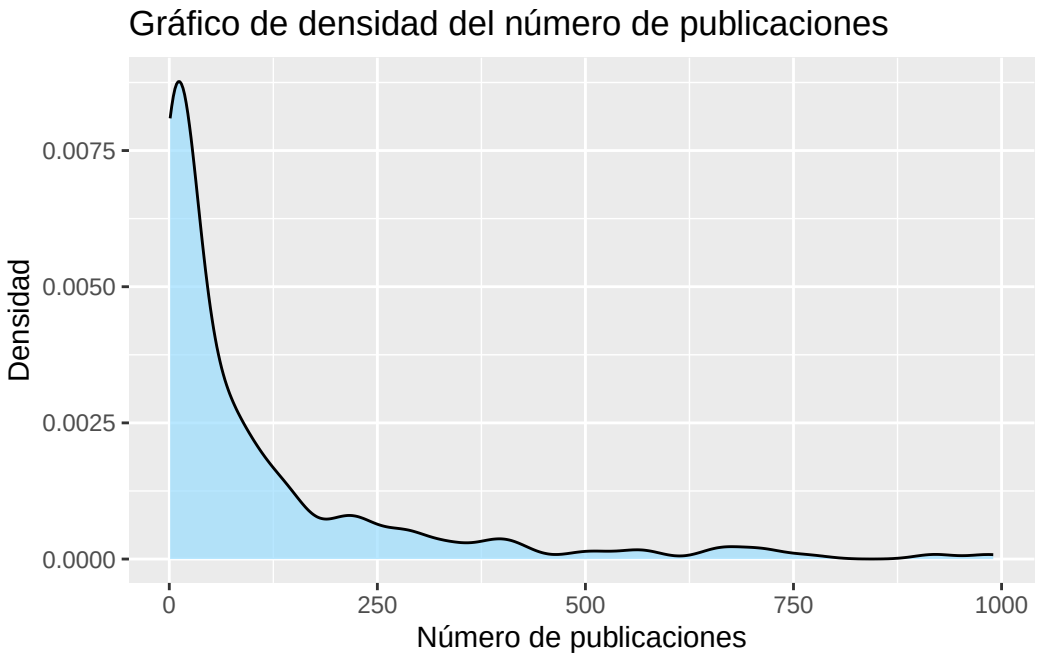


El rango de valores es muy muy amplio y hay mucha concentración en los valores bajos, tenemos que visualizarlo de otra manera, vamos a ver el gráfico de densidad:

```
ggplot(dataset, aes(x = X.posts)) +
  geom_density(fill = "#99DDFF", alpha = 0.7) +
  labs(x = "Número de publicaciones", y = "Densidad",
       title = "Gráfico de densidad del número de publicaciones")
```

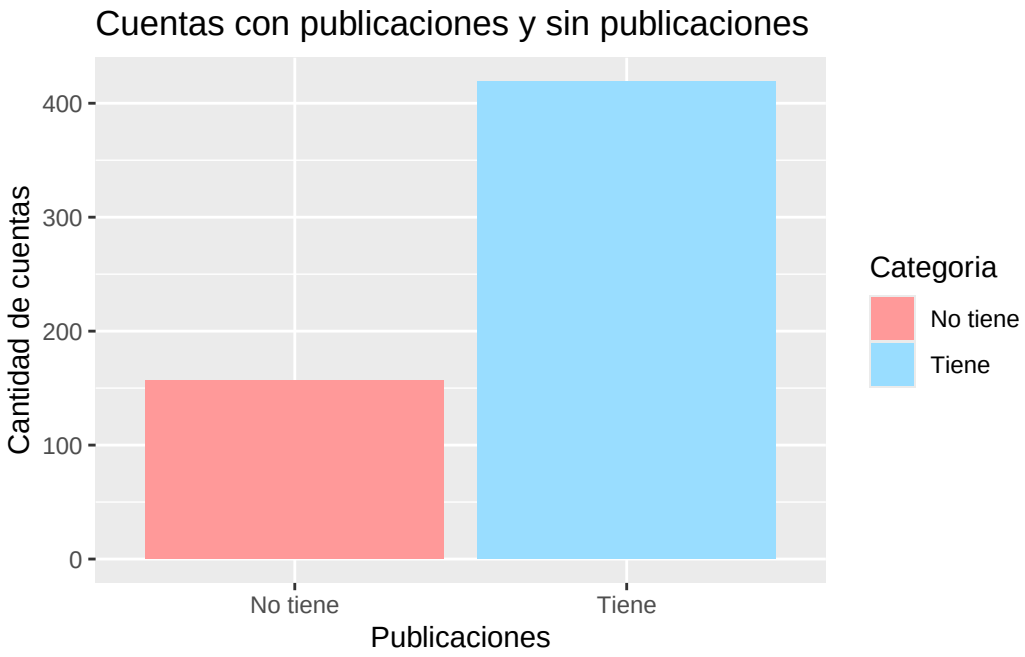
Los valores únicos que tenemos con miles de publicaciones hacen que el gráfico no se pueda visualizar bien. Vamos a realizar el gráfico de nuevo eliminando las cuentas con más de 1000 publicaciones y la cuentas con 0 publicaciones:

```
cuentas_menos_1000_posts <- dataset %>%
  filter(X.posts > 0 & X.posts <= 1000)

ggplot(cuentas_menos_1000_posts, aes(x = X.posts)) +
  geom_density(fill = "#99DDFF", alpha = 0.7) +
  labs(x = "Número de publicaciones", y = "Densidad",
       title = "Gráfico de densidad del número de publicaciones")
```



Ahora se observa el gráfico mejor, y vemos que la mayoría de cuentas tienen menos de 125 publicaciones.

```
count_summary <- data.frame(
  Categoria = c("Tiene", "No tiene"),
  Cantidad = c(count_with_posts, count_no_posts)
)

# Gráfico de barras
ggplot(count_summary, aes(x = Categoria, y = Cantidad, fill = Categoria)) +
  geom_bar(stat = "identity") +
  labs(
    x = "Publicaciones",
    y = "Cantidad de cuentas",
    title = "Cuentas con publicaciones y sin publicaciones"
  ) +
  scale_fill_manual(values = c("No tiene" = "#FF9999", "Tiene" = "#99DDFF"))
```



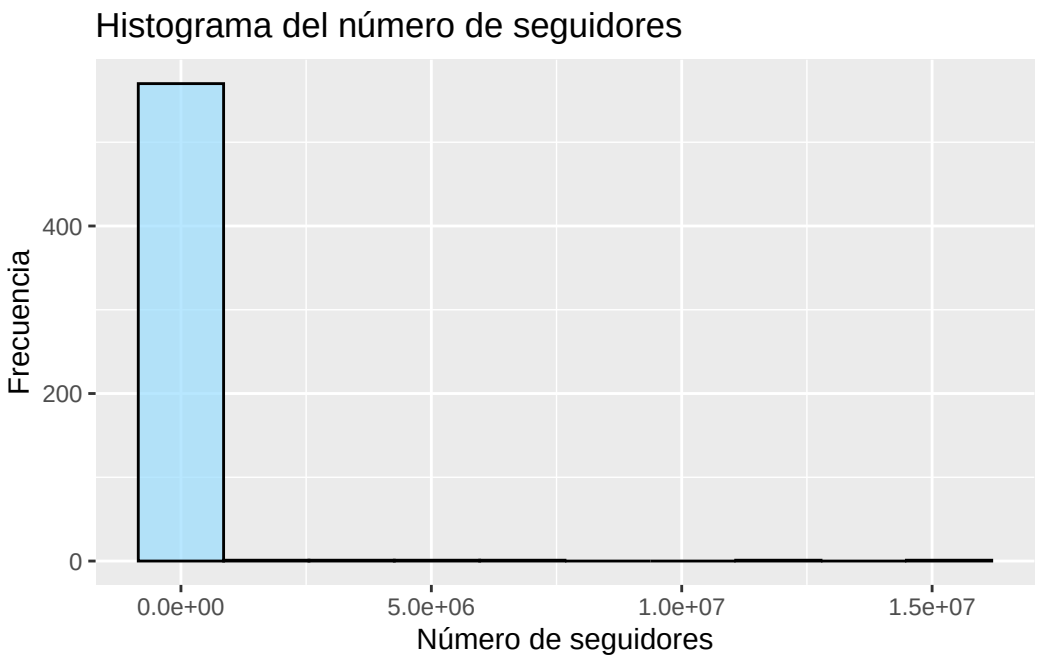

```
[1] "Cuentas con publicaciones: 419 (72.74%)"
```

```
[1] "Cuentas sin publicaciones: 157 (27.26%)"
```

3.2.6 X.followers

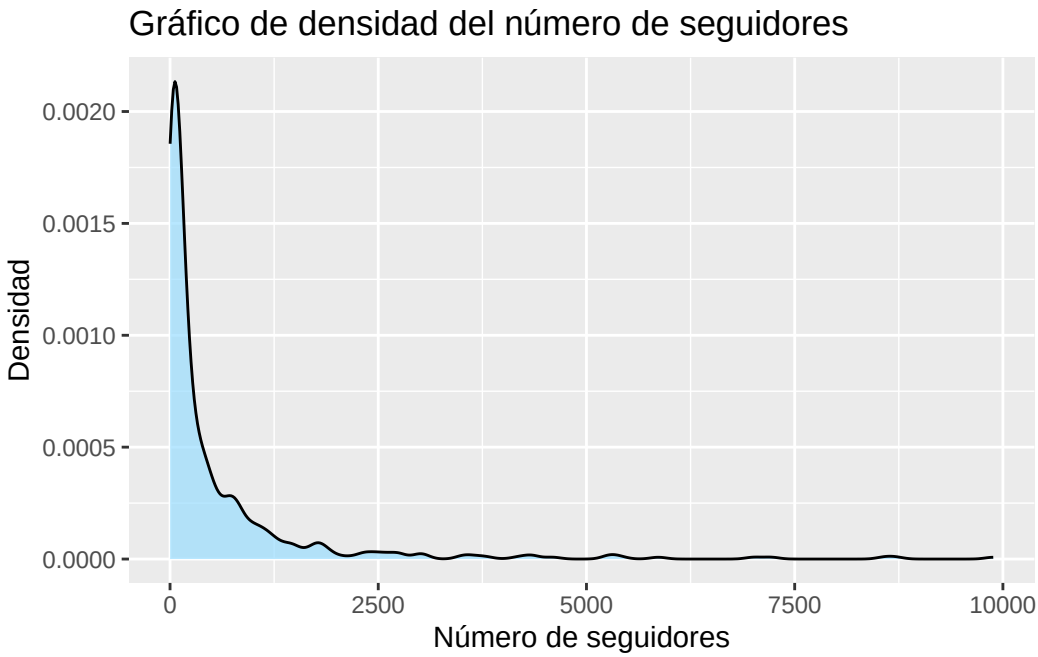
Representa el número de seguidores que tiene una cuenta de Instagram.

```r
ggplot(dataset, aes(x = X.followers)) +
  geom_histogram(bins = 10, fill = "#99DDFF", color = "black", alpha = 0.7) +
  labs(x = "Número de seguidores", y = "Frecuencia",
       title = "Histograma del número de seguidores")
```



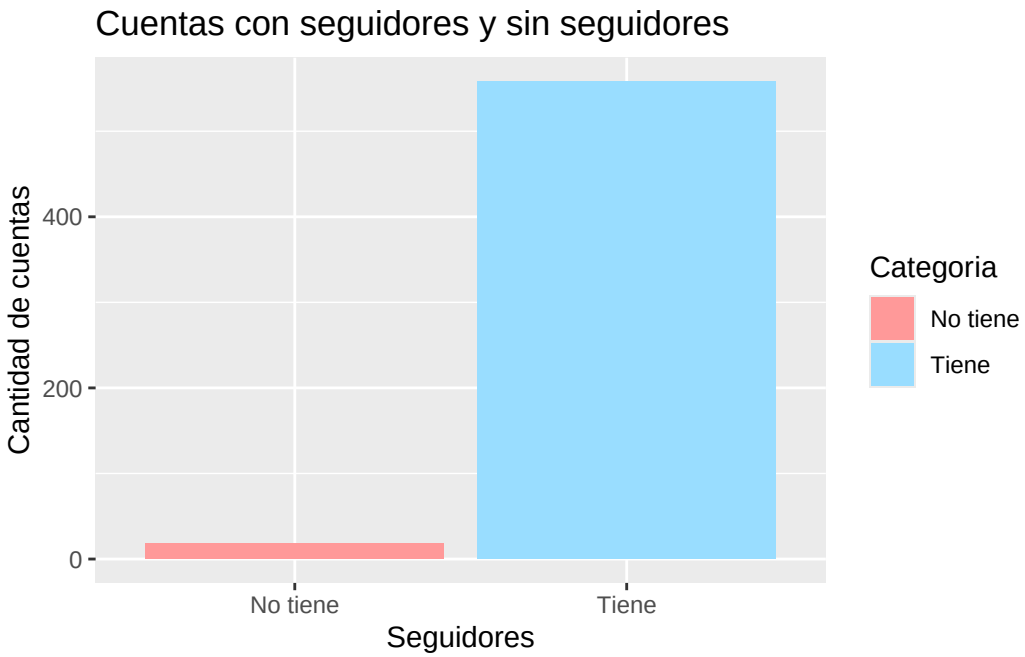
Prácticamente no se ve nada. Vamos a eliminar las cuentas con más de 10000 seguidores y con 0 seguidores.

```
cuentas_filtradas_seguidores <- dataset %>%
  filter(X.followers > 0 & X.followers < 10000)

ggplot(cuentas_filtradas_seguidores, aes(x = X.followers)) +
  geom_density(fill = "#99DDFF", alpha = 0.7) +
  labs(x = "Número de seguidores", y = "Densidad",
       title = "Gráfico de densidad del número de seguidores")
```



La mayoría de cuentas tienen menos de 1000 seguidores.

```
count_summary <- data.frame(
  Categoria = c("Tiene", "No tiene"),
  Cantidad = c(cuentas_con_seguidores, cuentas_sin_seguidores)
)

ggplot(count_summary, aes(x = Categoria, y = Cantidad, fill = Categoria)) +
  geom_bar(stat = "identity") +
  labs(
    x = "Seguidores",
    y = "Cantidad de cuentas",
    title = "Cuentas con seguidores y sin seguidores"
  ) +
  scale_fill_manual(values = c("No tiene" = "#FF9999", "Tiene" = "#99DDFF"))
```



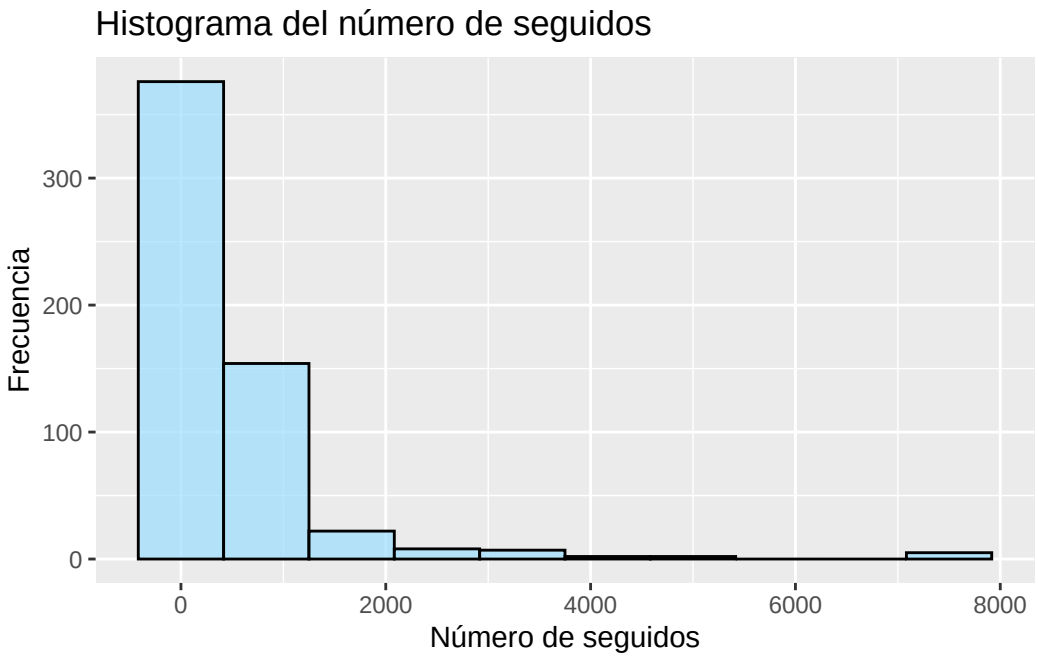

```
[1] "Cuentas con seguidores: 558 (96.88%)"
```

```
[1] "Cuentas sin seguidores: 18 (3.12%)"
```

3.2.7 x.follows

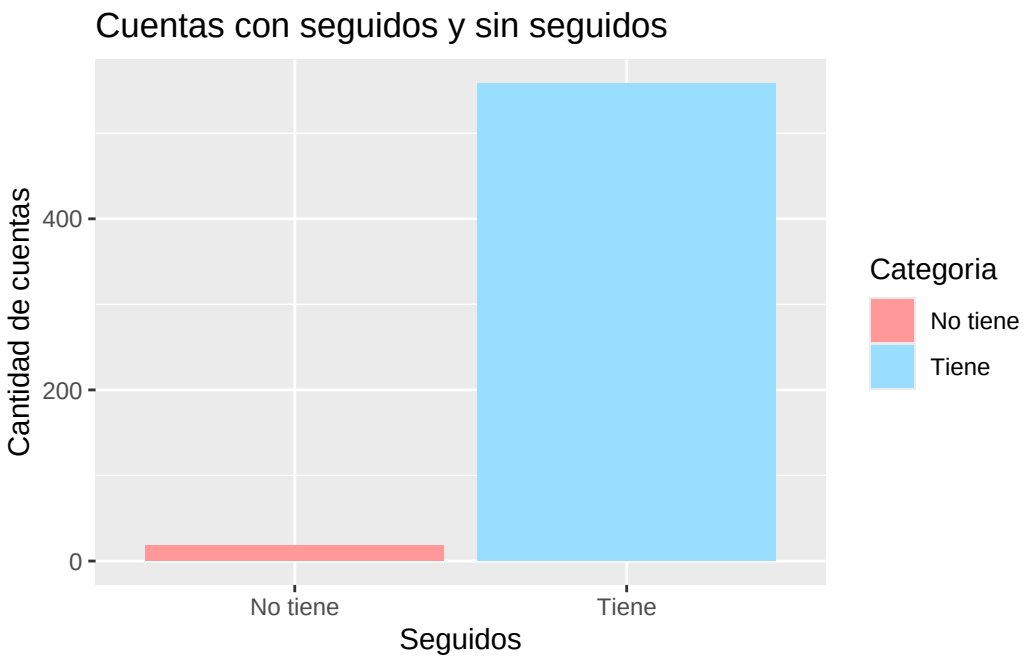
Representa el número de personas que sigue una cuenta de Instagram.

```
ggplot(dataset, aes(x = X.follows)) +
  geom_histogram(bins = 10, fill = "#99DDFF", color = "black", alpha = 0.7) +
  labs(x = "Número de seguidos", y = "Frecuencia",
       title = "Histograma del número de seguidos")
```



La mayoría de cuentas siguen a menos de 1500 personas.

```r
count_summary <- data.frame(
  Categoria = c("Tiene", "No tiene"),
  Cantidad = c(cuentas_con_seguidos, cuentas_sin_seguidos)
)

ggplot(count_summary, aes(x = Categoria, y = Cantidad, fill = Categoria)) +
  geom_bar(stat = "identity") +
  labs(
    x = "Seguidos",
    y = "Cantidad de cuentas",
    title = "Cuentas con seguidos y sin seguidos"
  ) +
  scale_fill_manual(values = c("No tiene" = "#FF9999", "Tiene" = "#99DDFF"))
```




```
[1] "Cuentas con seguidos: 558 (96.88%)"
```

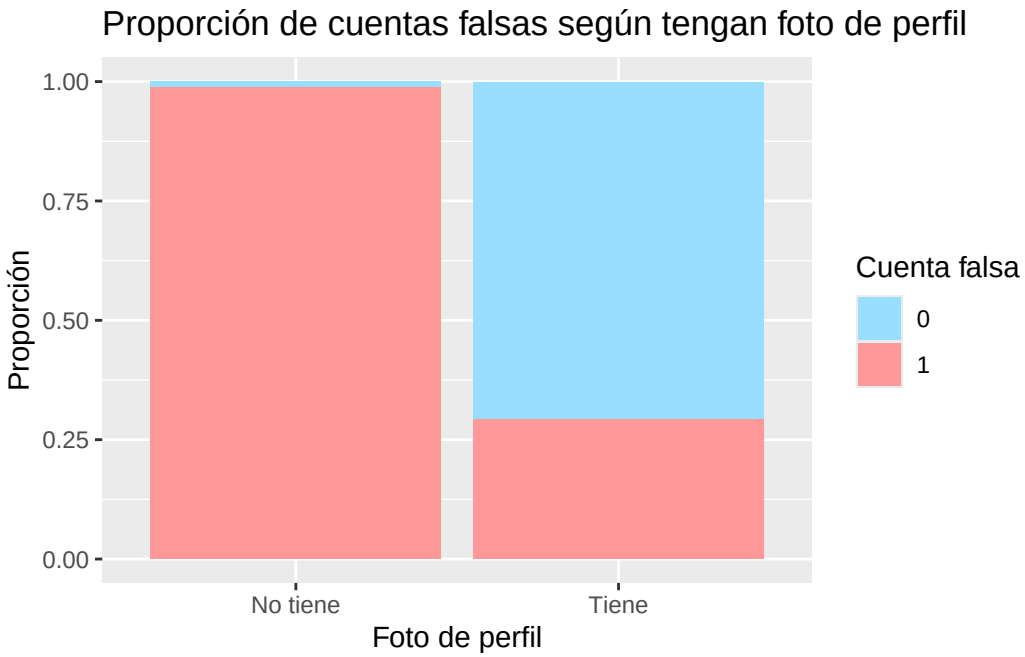
```
[1] "Cuentas sin seguidos: 18 (3.12%)"
```

3.3 Análisis multivariado

Vamos a crear algunos gráficos interesantes que relacionen distintas variables entre sí:

- **Relación entre si la cuenta es falsa y si tiene foto de perfil**

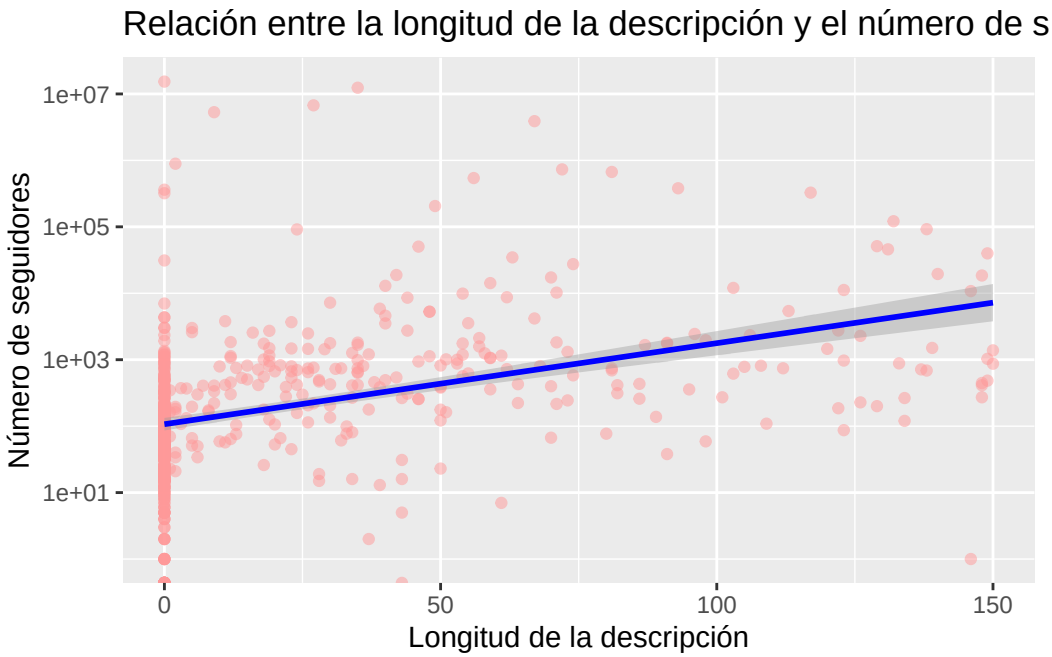
```
ggplot(dataset, aes(x = factor(profile.pic), fill = factor(fake))) +
  geom_bar(position = "fill") +
  labs(x = "Foto de perfil", y = "Proporción", fill = "Cuenta falsa",
       title = "Proporción de cuentas falsas según tengan foto de perfil") +
  scale_x_discrete(labels = c("0" = "No tiene", "1" = "Tiene")) +
  scale_fill_manual(values = c("0" = "#99DDFF", "1" = "#FF9999"))
```



Se observa que la gran mayoria de cuentas que no tienen foto de perfil, son falsas, mientras que para las cuentas con foto de perfil, la mayoría son reales, aunque con una proporción mucho más cercana al punto medio.

- **Relación entre la longitud de la descripción y el número de seguidores**

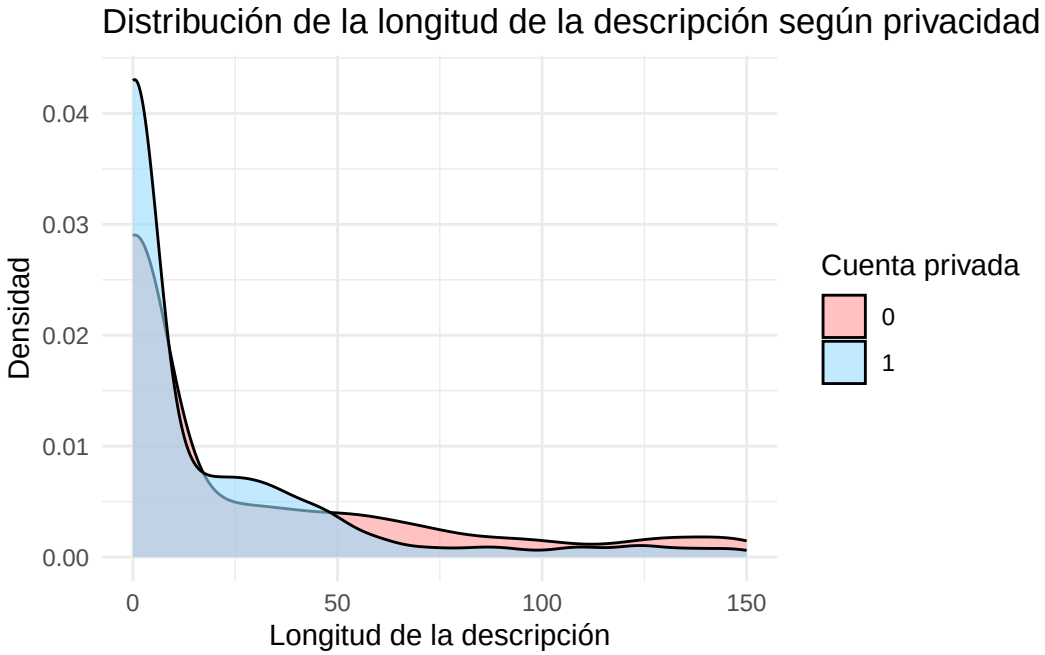
```
ggplot(dataset, aes(x = description.length, y = X.followers)) +
  geom_point(alpha = 0.5, color = "#FF9999") +
  geom_smooth(method = "lm", color = "blue") +
  labs(x = "Longitud de la descripción", y = "Número de seguidores",
       title = "Relación entre la longitud de la descripción y el número de seguidores") +
  scale_y_log10()
```



Observamos una interesante relación entre el número de seguidores y la longitud de la descripción. Sin la escala logarítmica no se apreciaría nada. Parece ser que a más larga la descripción, más famosa es la cuenta. No parece algo extremadamente relevante, ya que tenemos un límite de longitud de descripción en las cuentas de Instagram, pero si tiene cierto sentido que una cuenta famosa albergue más información en la descripción que una cuenta normal.

- **Distribución de la longitud de la descripción según si la cuenta es privada o pública**

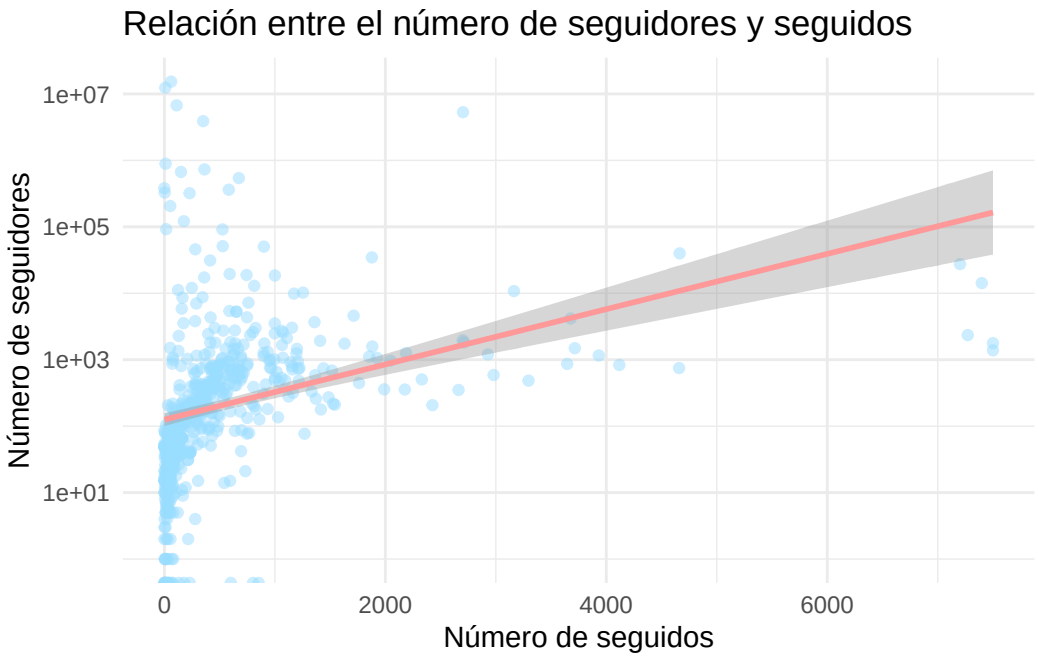
```
ggplot(dataset, aes(x = description.length, fill = factor(private))) +
  geom_density(alpha = 0.6) +
  labs(x = "Longitud de la descripción", y = "Densidad", fill = "Cuenta privada") +
  scale_fill_manual(values = c("0" = "#FF9999", "1" = "#99DDFF")) +
  theme_minimal() +
  labs(title = "Distribución de la longitud de la descripción según privacidad de la cuent
```



Las cuentas públicas suelen tener más descripción.

- **Relación entre el número de seguidores y el número de seguidos**

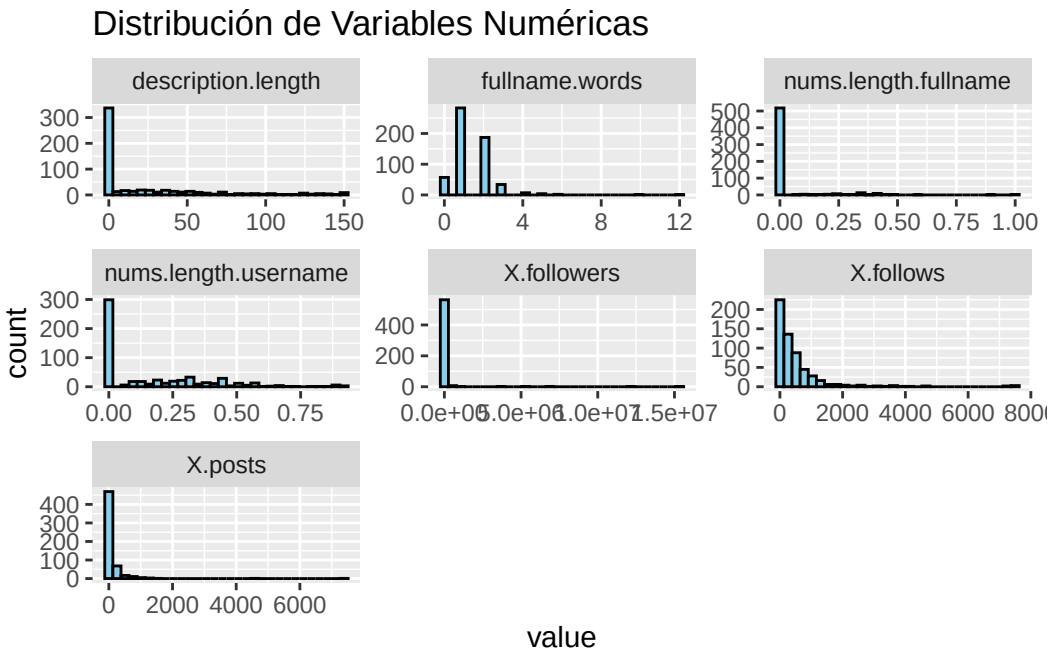
```
ggplot(dataset, aes(x = X.follows, y = X.followers)) +
  geom_point(alpha = 0.5, color = "#99DDFF") +
  geom_smooth(method = "lm", color = "#FF9999") +
  labs(x = "Número de seguidos", y = "Número de seguidores") +
  theme_minimal() +
  labs(title = "Relación entre el número de seguidores y seguidos") +
  scale_y_log10()
```



No parece que haya relación predecible entre los seguidos y los seguidores.

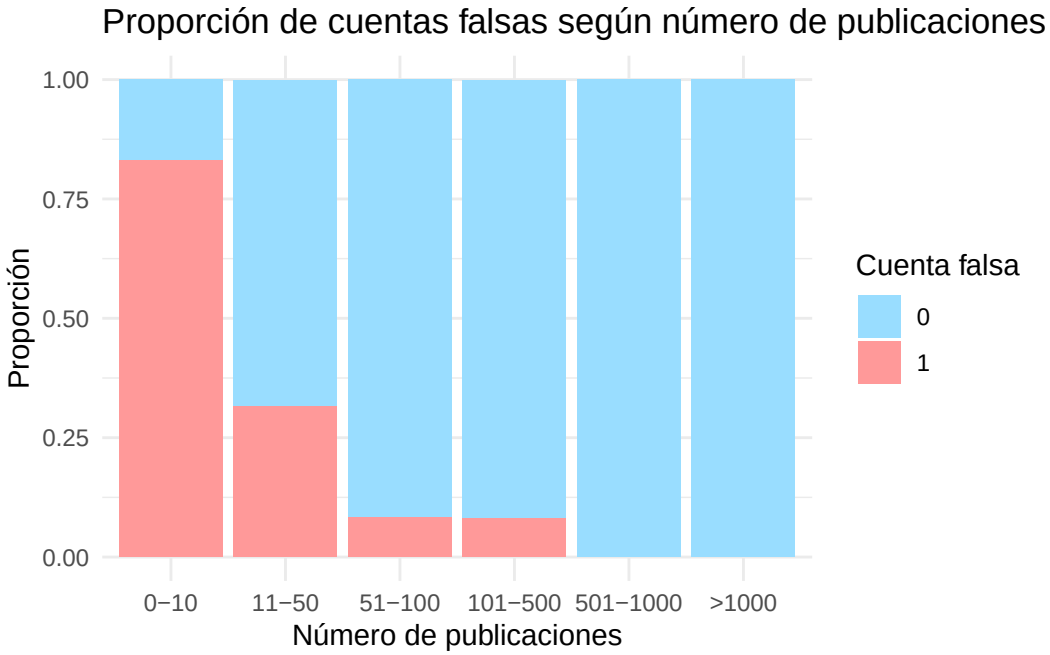
- **Histogramas de variables no binarias**

```
numeric_vars <- dataset %>%
  select(nums.length.username, fullname.words, nums.length.fullname,
         description.length, X.posts, X.followers, X.follows)

numeric_vars %>%
  gather(key = "variable", value = "value") %>%
  ggplot(aes(x = value)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  facet_wrap(~ variable, scales = "free") +
  labs(title = "Distribución de Variables Numéricas")
```



Todas siguen una distribución parecida, independientemente de que signifiquen cosas totalmente distintas, suelen concentrarse en su mayoría, cerca del 0. Esto probablemente se deba a que los más fácil de conseguir para estas variables en Instagram, sea 0, independientemente de lo que signifiquen.

```
dataset <- dataset %>%
  mutate(posts_category = cut(X.posts, breaks = c(-1, 10, 50, 100, 500, 1000, Inf),
                              labels = c("0-10", "11-50",
                                         "51-100", "101-500",
                                         "501-1000", ">1000")))

ggplot(dataset, aes(x = posts_category, fill = factor(fake))) +
  geom_bar(position = "fill") +
  scale_fill_manual(values = c("0" = "#99DDFF", "1" = "#FF9999")) +
  labs(x = "Número de publicaciones", y = "Proporción",
       title = "Proporción de cuentas falsas según número de publicaciones",
       fill = "Cuenta falsa") +
  theme_minimal()
```



No hay cuentas falsas con más de 500 publicaciones realizadas.

3.4 Conclusiones

La visualización de datos nos ha permitido entender mejor la distribución y las relaciones entre las variables del dataset. Utilizando la librería ggplot2, hemos creado gráficos claros y efectivos para variables binarias y no binarias. Los histogramas y gráficos circulares han facilitado la identificación de patrones y proporciones. En el análisis multivariado, hemos observado correlaciones relevantes, como la relación entre cuentas falsas y la presencia de foto de perfil. Estas visualizaciones no solo enriquecen el análisis exploratorio, sino que también sirven como base sólida para futuros modelos predictivos y toma de decisiones informadas.

4 Reglas de asociación

El análisis de reglas de asociación es una técnica fundamental en el campo de la minería de datos y el aprendizaje automático. Se utiliza para descubrir patrones interesantes y relaciones ocultas entre variables en conjuntos de datos. En particular, las reglas de asociación son útiles para identificar asociaciones frecuentes entre diferentes elementos en una base de datos transaccional.

En este trabajo, exploraremos el uso de reglas de asociación para el análisis de cuentas de Instagram. Utilizaremos el paquete `arules` en R para realizar este análisis. Nuestro conjunto de datos contiene una variedad de características de las cuentas de Instagram, como el número de seguidores, el número de publicaciones, si la cuenta es privada o no, entre otros.

A lo largo de este análisis, exploraremos cómo las diferentes características de las cuentas de Instagram están asociadas entre sí, identificando reglas de asociación significativas que nos ayuden a comprender mejor el ecosistema de esta red social.

```
library(arules)
library(dplyr)
library(magrittr)
```

```
dataset <- read.csv("datasets/train.csv")
```

Antes de ponernos manos a la obra, tenemos que saber que arules no se lleva bien con las variables con un gran número de valores. Es más, solo se lleva bien con las binarias.

En nuestro dataset, hemos visto en el análisis exploratorio que hay 5 variables binarias (bien para arules) y las demás no lo son. Por suerte, gracias a los conocimientos que hemso adquirido durante la asignatura, sabemos que esto tiene solución.

Comprendiendo bien el significado de las variables y aplicando técnicas de discretización y agrupación, podemos convertir nuestras variables númericas a directamente binarias, o con pocos valores, y las que tengan pocos valores, podemos transformarlas a variables binarias.

Vamos a realizar un buen trabajo de pre-processing para este apartado, así nuestras reglas serán potentes y tendrán un conocimiento importante:

4.1 Pre-processing

Recordamos del apartado de análisis exploratorio que teníamos:

- **Variables binarias**:
  - profile.pic
  - name..username
  - external.URL
  - private
  - fake
- **Variables no binarias**:
  - nums.length.username
  - fullname.words
  - nums.length.fullname
  - description.length
  - X.posts
  - X.followers
  - X.follows

4.1.1 Variables binarias

Las añadimos directamente al dataset modificado que vamos a ir construyendo

```
datarules <- transmute(dataset,
                       profile.pic=profile.pic,
                       name..username=name..username,
                       external.URL=external.URL,
                       private=private,
                       fake=fake)
```

4.1.2 Variables no binarias

Para las variables no binarias, podemos tener distintos puntos de vista, así que vamos a ir viendo cada variable:

4.1.2.1 nums.length.username

El ratio de números en el nombre de usuario de la cuenta es díficilmente binarizable, ya que no es trivial donde poner el punto medio. En el nombre de persona si resulta extraño tener números, pero en el nombre de usuario puede ser normal, no tiene porque estar relacionado directamente con que la cuenta sea falsa.

Podríamos binarizar simplemente en "tiene números o no tiene números", ya que como vimos en el análisis exploratorio, teníamos prácticamente un 50-50, pero eso no exprimiría todo el conocimiento de los datos, pues lo que viene a representar esta variable es el ratio de carácteres númericos sobre la longitud del nombre.

Vamos a discretizar en 3 categorías:

- "0"
- "(0, 0.5]"
- "(0.5, 1]"

```
lu_discretize <- transmute(dataset,
          nums.length.username=
            ifelse(nums.length.username == 0,
                   "0", ifelse(nums.length.username <= 0.5,
                               "(0, 0.5]", "(0.5, 1]")))

datarules <- mutate(datarules,
       nums.length.username=lu_discretize$nums.length.username)
```

4.1.2.2 nums.length.fullname

Para esta variable, si vamos a binarizar, ya que no es lo normal tener un número en el nombre completo. Pondremos un 0 a las cuenas que no tengan números en el nombre completo y un 1 a las que si tengan,

```
lfn_binary <- transmute(dataset,
                        nums.length.fullname=
                          as.numeric(nums.length.fullname!=0))

datarules <- mutate(datarules,
       has.nums.fullname=lfn_binary$nums.length.fullname)
```

4.1.2.3 fullname.words

Para esta variable vamos a tomar una decisión que quizá no guste a todos los lectores, pero creemos que dará los mejores resultados (recordamos que esta variable representa el número de palabras en el nombre completo)

```
table(dataset$fullname.words)
```

```
  0   1   2   3   4   5   6  10  12
 57 283 187  34   7   4   2   1   1
```

Tener 0 palabras en el nombre no es lo normal, tener entre 1 y 2, lo es, tener 3 o más tampoco es normal. Esto obviamente subjetivo y está basado en la opinión subjetiva del escritor de este libro en base a su experiencia en el uso de Instagram.

Vamos a categorizar esta varaible en 3 rangos:

- "0"
- "[1, 2]"
- "> 2"

```
fnw_discretize <- transmute(dataset,
         fullname.words =
            ifelse(fullname.words == 0,
                   "0", ifelse(fullname.words <= 2,
                               "[1, 2]", "> 2")))

datarules <- mutate(datarules,
       fullname.words=fnw_discretize$fullname.words)
```

4.1.2.4 description.length

Vamos a binarizar esta variable, en el análisis exploratorio vimos que había mas cuentas sin descripción que con descripción y esto puede ser interesante.

```
dl_binary <- transmute(dataset,
                       description.length=
                         as.numeric(description.length!=0))

datarules <- mutate(datarules,
```

```
        has.description=dl_binary$description.length)
```

4.1.2.5 X.posts

Para esta variable pasaremos a 3 categorías, ya que creemos que dividen bien el conjunto de valores:

```
[1] "Cuentas sin publicaciones: 157"
```

```
[1] "Cuentas con entre 0 y 100 publicaciones: 292"
```

```
[1] "Cuentas con más de 100 publicaciones: 576"
```

Por tanto estas categorías son:

- "0"
- "(0, 100]"
- "> 100"

```
p_discretize <- transmute(dataset,
         X.posts =
            ifelse(X.posts == 0,
                   "0", ifelse(X.posts <= 100,
                               "(0, 100]", paste0("> 100"))))

datarules <- mutate(datarules,
       X.posts=p_discretize$X.posts)
```

4.1.2.6 X.followers

Vamos a dividir de nuevo en 3 categorías que hemos considerado:

```
[1] "Cuentas con menos de 20 seguidores: 95"
```

```
[1] "Cuentas con entre 20 y 200 seguidores: 223"
```

```
[1] "Cuentas con más de 200 seguidores: 258"
```

Por tanto estas categorías son:

- "< 20"
- "[20, 200]"
- "> 200"

```
fwrs_discretize <- transmute(dataset,
         X.followers =
            ifelse(X.followers < 20,
                   "< 20", ifelse(X.followers <= 200,
                                  "[20, 200]", paste0("> 200"))))

datarules <- mutate(datarules,
       X.followers=fwrs_discretize$X.followers)
```

4.1.2.7 X.follows

Para los seguidos vamos a dividir en las mismas categorías que para los seguidores:

```
[1] "Cuentas con menos de 20 seguidos: 64"
```

```
[1] "Cuentas con entre 20 y 200 seguidos: 211"
```

```
[1] "Cuentas con más de 200 seguidos: 301"
```

Por tanto estas categorías son:

- "< 20"
- "[20, 200]"
- "> 200"

```
fws_discretize <- transmute(dataset,
         X.follows =
            ifelse(X.follows < 20,
                   "< 20", ifelse(X.follows <= 200,
                                  "[20, 200]", paste0("> 200"))))

datarules <- mutate(datarules,
       X.follows=fws_discretize$X.follows)
```

4.1.3 Resultados del pre-processing

Por tanto, después de nuestro preprocessing, nos quedan como variables de nuestro dataset:

- **profile.pic**
  - Variable binaria que toma el valor 1 si la cuenta tiene foto de perfil, 0 si no tiene.
- **name..username**
  - Variable binaria que toma el valor 1 si el nombre completo de la persona y el de usuario es el mismo, 0 si no.
- **external.URL**
  - Variable binaria que toma el valor 1 si la cuenta tiene un enlace puesto en su perfil, 0 si no.
- **private**
  - Variable binaria que toma el valor 1 si la cuenta es privada, 0 si es pública.
- **nums.length.username**
  - Variable trinaria que toma los valores:
    - "0" si no tiene números en el nombre de usuario.
    - "(0, 0.5]" si el ratio de números es mayor a 0 y menor o igual a 0.5.
    - "(0.5, 1]" si el ratio de números es mayor a 0.5 y menor o igual a 1.
- **has.nums.fullname**
  - Variable binaria que toma el valor 1 si el nombre completo contiene números, 0 si no contiene.
- **fullname.words**
  - Variable categórica que toma los valores:
    - "0" si el nombre completo no tiene palabras.
    - "[1, 2]" si el nombre completo tiene entre 1 y 2 palabras.
    - "> 2" si el nombre completo tiene más de 2 palabras.
- **has.description**
  - Variable binaria que toma el valor 1 si la cuenta tiene una descripción, 0 si no tiene.
- **X.posts**
  - Variable categórica que toma los valores:
    - "0" si la cuenta no tiene publicaciones.
    - "(0, 100]" si la cuenta tiene entre 1 y 100 publicaciones.

        - “> 100” si la cuenta tiene más de 100 publicaciones.

- **X.followers**
    - Variable categórica que toma los valores:
        - “< 20” si la cuenta tiene menos de 20 seguidores.
        - “[20, 200]” si la cuenta tiene entre 20 y 200 seguidores.
        - “> 200” si la cuenta tiene más de 200 seguidores.

- **X.follows**
    - Variable categórica que toma los valores:
        - “< 20” si la cuenta sigue a menos de 20 cuentas.
        - “[20, 200]” si la cuenta sigue a entre 20 y 200 cuentas.
        - “> 200” si la cuenta sigue a más de 200 cuentas.

- **fake**
    - Variable binaria que toma el valor 1 si la cuenta es falsa, 0 si no lo es.

Con este nuevo dataset binarizado y discretizado, estamos listos para proceder al análisis de reglas de asociación utilizando el paquete `arules` en R. Este preprocesamiento nos asegura que las variables están en un formato adecuado para el análisis, lo que facilitará la identificación de patrones y asociaciones significativas.

4.2 Generando reglas

Nuestro dataset ha quedado así:

```
'data.frame':   576 obs. of  12 variables:
 $ profile.pic         : int  1 1 1 1 1 1 1 1 1 1 ...
 $ name..username      : int  0 0 0 0 0 0 0 0 0 0 ...
 $ external.URL        : int  0 0 0 0 0 1 0 0 0 1 ...
 $ private             : int  0 0 1 0 1 0 0 0 0 0 ...
 $ nums.length.username: chr  "(0, 0.5]" "0" "(0, 0.5]" "0" ...
 $ has.nums.fullname   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ fullname.words      : chr  "0" "[1, 2]" "[1, 2]" "[1, 2]" ...
 $ has.description     : num  1 1 0 1 0 1 1 0 1 1 ...
 $ X.posts             : chr  "(0, 100]" "> 100" "(0, 100]" "> 100" ...
 $ X.followers         : chr  "> 200" "> 200" "[20, 200]" "> 200" ...
 $ X.follows           : chr  "> 200" "> 200" "[20, 200]" "> 200" ...
 $ fake                : int  0 0 0 0 0 0 0 0 0 0 ...
```

Ahora arules puede generar las reglas fácilmente a partir de este dataset usando el algoritmo apriori. Directamente se podría aplicar apriori, pero lo suyo es convertir el dataset a un objeto transaccional primero.

4.2.1 Objeto transaccional

Antes de ello, para que la conversion se haga bien, tenemos que poner nuestras variables binarias como variables de tipo factor, si no la conversión a objeto transaccional no se hará bien:

```
datarules$profile.pic <- as.factor(datarules$profile.pic)
datarules$external.URL <- as.factor(datarules$external.URL)
datarules$name..username <- as.factor(datarules$name..username)
datarules$private <- as.factor(datarules$private)
datarules$has.nums.fullname <- as.factor(datarules$has.nums.fullname)
datarules$has.description <- as.factor(datarules$has.description)
datarules$fake <- as.factor(datarules$fake)
```

Ahora si, converitmos:

```
Tdatarules <- as(datarules, "transactions")
Tdatarules@itemInfo$labels
```

```
 [1] "profile.pic=0"                "profile.pic=1"
 [3] "name..username=0"             "name..username=1"
 [5] "external.URL=0"               "external.URL=1"
 [7] "private=0"                    "private=1"
 [9] "nums.length.username=(0, 0.5]" "nums.length.username=(0.5, 1]"
[11] "nums.length.username=0"       "has.nums.fullname=0"
[13] "has.nums.fullname=1"          "fullname.words=[1, 2]"
[15] "fullname.words=> 2"           "fullname.words=0"
[17] "has.description=0"            "has.description=1"
[19] "X.posts=(0, 100]"             "X.posts=> 100"
[21] "X.posts=0"                    "X.followers=[20, 200]"
[23] "X.followers=< 20"             "X.followers=> 200"
[25] "X.follows=[20, 200]"          "X.follows=< 20"
[27] "X.follows=> 200"              "fake=0"
[29] "fake=1"
```

Como vemos, la conversión ha hecho justamente lo que queríamos. Por ejemplo, la variable X.posts, que le dimos tres valores distintos, se han convertido en 3 variables, que cada una

de ellas es binaria. Es una técnica que podríamos haber hecho a mano, pero que ya que el comando as(..., "transactions") lo hace por nosotros, no está mal aprovecharlo. Ahora tenemos 29 variables.

4.2.2 Apriori

```
rules <- apriori(Tdatarules, parameter = list(conf=0.1, supp=0.2, target="rules"))
```

```
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
        0.1    0.1    1 none FALSE            TRUE       5     0.2      1
 maxlen target  ext
     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 115

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[29 item(s), 576 transaction(s)] done [0.00s].
sorting and recoding items ... [21 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 done [0.00s].
writing ... [6751 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

```
length(rules)
```

```
[1] 6751
```

El hecho de haber usado un soporte de 0.05 y 0.1 ha sido nuestra elección para filtrar entre las miles de reglas que hay si no ponemos un límite. El uso de "target = rules" no es necesario, pero es para recordar que estamos buscando reglas. También se pueden buscar otras cosas como los conjuntos de items más frequentes.

Vamos a continuar filtrando estas reglas, para quedarnos con las mejores. Primero vamos a eliminar las reglas redundantes.

```
indices_no_redundantes <- which(!is.redundant(rules))
rules <- rules[indices_no_redundantes]
inspect(head(rules))
```

```
    lhs    rhs                      support   confidence coverage lift count
[1] {}  => {X.posts=> 100}           0.2204861 0.2204861  1        1    127
[2] {}  => {X.posts=0}               0.2725694 0.2725694  1        1    157
[3] {}  => {profile.pic=0}           0.2986111 0.2986111  1        1    172
[4] {}  => {X.follows=[20, 200]}     0.3663194 0.3663194  1        1    211
[5] {}  => {private=1}               0.3819444 0.3819444  1        1    220
[6] {}  => {X.followers=[20, 200]} 0.3871528 0.3871528  1        1    223
```

Y vamos a quedarnos solo con las significativas con el método de fisher:

```
indices_significativos <- is.significant(rules, method="fisher")
rules <- rules[indices_significativos]
```

De esta manera nos hemos quedado con muchas menos reglas, pero con mucha mas relevancia. No sirven de nada miles de reglas si luego ni si quiera podemos trabajar con ellas.

4.2.3 Reglas que inducen cuenta falsa

Lo que nos interesa son las reglas que nos lleven a que la cuenta es fake, vamos a observar estas:

```
reglas_fake <- subset(rules, rhs %in% "fake=1")
```

```
[1] "Tenemos 38 reglas que nos llevan a que la cuenta es falsa."
```

Vamos a observar las 5 mejores reglas ordenadas por lift:

```
reglas_fake_lift <- sort(reglas_fake, by="lift")
inspect(reglas_fake_lift[1:5])
```

```
    lhs                                          rhs      support   confidence
[1] {profile.pic=0, X.posts=0}                => {fake=1} 0.2152778 1.0000000
[2] {profile.pic=0, has.description=0}        => {fake=1} 0.2829861 1.0000000
[3] {profile.pic=0, private=0}                => {fake=1} 0.2100694 1.0000000
[4] {profile.pic=0, fullname.words=[1, 2]} => {fake=1} 0.2725694 0.9936709
[5] {profile.pic=0}                           => {fake=1} 0.2951389 0.9883721
    coverage  lift     count
[1] 0.2152778 2.000000 124
[2] 0.2829861 2.000000 163
[3] 0.2100694 2.000000 121
[4] 0.2743056 1.987342 157
[5] 0.2986111 1.976744 170
```

Por primera vez en lo que llevamos de libro hemos obtenido un conocimiento interesante para la detección de cuentas falsas de Instagram.

La primera regla, por ejemplo, nos está diciendo que con una confianza del 100% en nuestro dataset, las cuentas que no tienen foto de perfil ni posts, son falsas.

Lo mismo con las siguientes reglas que tienen 100% de confianza: - Las cuentas sin foto de perfil ni descripción, son falsas - Las cuentas sin foto de perfil y que son públicas son falsas

Obviamente esto no significan que cualquier cuenta de Instagram que encontremos en Internet es falsa si cumple dichas condiciones, si no que en nuestro dataset siempre es así, y nuestro dataset no es demasiado grande. Necesitaríamos un dataset mucho más grande para poder afirmar esto para un caso general, pero ya son conclusiones interesantes.

También vemos que la quinta regla nos dice que casi siempre que una cuenta no tiene foto de perfil, es falsa. Por ello no podemos hacer caso a lo que dicen las reglas sin más, tenemos que ver más opciones y usar más técnicas de análisis de datos, pero como primeros conocimientos, es un buen resultado.

Otra cosa a tener en cuenta es el soporte de las reglas. Vamos a ver que dicen las reglas con más soporte:

Mejores reglas ordenadas por soporte:

```
reglas_fake_support <- sort(reglas_fake, by="support")
inspect(reglas_fake_support[1:5])
```

```
    lhs                                          rhs      support   confidence
[1] {external.URL=0}                          => {fake=1} 0.5000000 0.5658153
[2] {fullname.words=[1, 2]}                   => {fake=1} 0.4322917 0.5297872
[3] {external.URL=0, fullname.words=[1, 2]} => {fake=1} 0.4322917 0.5942721
```

```
[4] {has.description=0}                      => {fake=1} 0.4305556 0.7607362
[5] {external.URL=0, has.description=0}      => {fake=1} 0.4305556 0.7774295
    coverage  lift     count
[1] 0.8836806 1.131631 288
[2] 0.8159722 1.059574 249
[3] 0.7274306 1.188544 249
[4] 0.5659722 1.521472 248
[5] 0.5538194 1.554859 248
```

Más soporte significa que esos patrones se dan mas veces en el dataset, pero sin tener en cuenta la confianza de estas reglas, no sirve de nada. Por ejemplo, la primera regla nos dice que si una cuenta no tiene enlace externo, entonces es falsa, y se da el 50% de las veces en nuestro dataset. Pero la confianza es solo del 56%. No es bueno fiarse simplemente de esta regla.

En vez de buscar reglas tan simples y generales como "dada una sola cosa entonces la cuenta es fake" podemos buscar reglas que tengan a la izquierda varios items, lo que hará que el soporte sea más bajo, pero puede que nos de reglas mas fiables.

Que una cuenta sea fake no va a depender únicamente de que no tenga foto de perfil o de que no tenga descripción, si no que probablemente cada condición tenga cierto peso y no sea tan sencillo.

Por todo esto no podemos fiarnos solamente del soporte de la regla, y tenemos que mirar también la confianza. O mejor aún, el lift.

4.2.4 Reglas que inducen cuenta real

Antes de buscar reglas con más items en la parte izquierda, vamos a ver las reglas que nos llevan a que la cuenta no es falsa.

```
reglas_reales <- subset(rules, rhs %in% "fake=0")
```

```
[1] "Tenemos 138 reglas que nos llevan a que la cuenta es real."
```

Vamos a ver las reglas con mayor lift:

```
reglas_reales_lift <- sort(reglas_reales, by="lift")
inspect(reglas_reales_lift[1:5])
```

```
    lhs                          rhs        support confidence  coverage     lift count
[1] {nums.length.username=0,
     fullname.words=[1, 2],
     has.description=1,
     X.followers=> 200}       => {fake=0} 0.2013889  0.9830508 0.2048611 1.966102   116
[2] {profile.pic=1,
     name..username=0,
     nums.length.username=0,
     fullname.words=[1, 2],
     has.description=1}       => {fake=0} 0.2378472  0.9785714 0.2430556 1.957143   137
[3] {profile.pic=1,
     nums.length.username=0,
     fullname.words=[1, 2],
     X.followers=> 200,
     X.follows=> 200}         => {fake=0} 0.2222222  0.9770992 0.2274306 1.954198   128
[4] {X.posts=> 100,
     X.followers=> 200}       => {fake=0} 0.2013889  0.9747899 0.2065972 1.949580   116
[5] {profile.pic=1,
     nums.length.username=0,
     has.description=1,
     X.followers=> 200}       => {fake=0} 0.2552083  0.9735099 0.2621528 1.947020   147
```

Son bastantes interesantes y precisamente tienen el componente que comentabamos antes. No son reglas tan generales como "si tiene foto de perfil la cuenta es real", si no que tiene en cuenta mucho mos detalles, como or ejemplo la segunda regla. Con un 97.85% de confianza, si la cuenta tiene foto de perfil, el nombre real y el de usuario son distintos, tiene entre 1 y 2 palabras en el nombre real y tiene descripción, entonces, es real.

Estas reglas son muy interesantes y nos dan pistas y conocimiento sobre que elementos son claves a la hora de ver si una cuenta es falsa o no.

4.2.5 Reglas con más de 6 items

Vamos ahora a buscar reglas que tengan al menos 6 items en total, (con la idea de que tengan 5 en la parte izquierda y 1 en la derecha), para ver que tipo de reglas podemos conseguir.

```
big_rules <- apriori(Tdatarules, parameter =
                       list(conf=0.1, supp=0.2, minlen=6, target="rules"))
```

```
Apriori
```

```
Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
        0.1    0.1    1 none FALSE            TRUE       5     0.2      6
 maxlen target  ext
     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 115

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[29 item(s), 576 transaction(s)] done [0.00s].
sorting and recoding items ... [21 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 done [0.00s].
writing ... [1351 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

Eliminamos las reglas redundantes

```
indices_no_redundantes <- which(!is.redundant(big_rules))
big_rules <- big_rules[indices_no_redundantes]
```

Y nos quedamos con las significativas

```
indices_significativos <- is.significant(big_rules, method="fisher")
big_rules <- big_rules[indices_significativos]
```

Veamos las reglas que nos llevan a que la cuenta es falsa:

```
big_reglas_fake <- subset(big_rules, rhs %in% "fake=1")
```

```
[1] "Tenemos 17 reglas con 6 o más items que nos llevan a que la cuenta es falsa."
```

Las 5 más importantes ordenadas por lift:

```
big_reglas_fake_lift <- sort(big_reglas_fake, by="lift")
inspect(big_reglas_fake_lift[1:3])
```

```
    lhs                          rhs        support confidence  coverage lift count
[1] {profile.pic=0,
     external.URL=0,
     has.nums.fullname=0,
     fullname.words=[1, 2],
     has.description=0}      => {fake=1} 0.2187500          1 0.2187500    2   126
[2] {profile.pic=0,
     name..username=0,
     external.URL=0,
     fullname.words=[1, 2],
     has.description=0}      => {fake=1} 0.2430556          1 0.2430556    2   140
[3] {profile.pic=0,
     name..username=0,
     has.nums.fullname=0,
     fullname.words=[1, 2],
     has.description=0}      => {fake=1} 0.2152778          1 0.2152778    2   124
```

Estamos viendo patrones que creíamos que no se daban en las cuentas falsas, como por ejemplo, que cuentas con 1 o 2 palabras en el nombre completo también pueden llevar a una cuenta falsa, y sin números en el nombre completo, con el nombre completo distinto del nombre de usuario...

Esto nos indica que detectar una cuenta falsa no es tan fácil como buscar lo que pensamos que no es normal, si no que es más complejo.

4.2.6 Reglas entre otros atributos

Ahora vamos a buscar reglas entre los distintos atributos del dataset, que no tengan nada que ver con la clasificación de si la cuenta es falsa o no, con la idea de encontrar reglas que nos permitan derivar unos atributos a partir de otros. Usaremos las reglas generadas y almacenadas en la variable **rules**.

Vamos a filtrar de forma que la regla no contenga nada referente a la variable fake, ni a la izquierda ni a la derecha. Vamos a filtrar también por reglas que tengan al menos 0.8 de confianza:

```
rules_sin_fake <- subset(rules,
                         !(rhs %pin% "fake=") &
                         !(lhs %pin% "fake=") &
                         confidence > 0.8)
paste("Tenemos", length(rules_sin_fake), "reglas con confidence > 0.8
      y sin la variable fake")
```

```
[1] "Tenemos 255 reglas con confidence > 0.8 \n      y sin la variable fake"
```

Ordenamos por lift:

```
rules_sin_fake_confidence <- sort(rules_sin_fake, by="lift")
inspect(rules_sin_fake_confidence[1:4])
```

```
    lhs                           rhs                    support confidence  coverage     lift
[1] {has.description=0,
     X.posts=0}                => {profile.pic=0}     0.2135417  0.8255034 0.2586806 2.764476
[2] {profile.pic=1,
     X.posts=> 100}            => {X.followers=> 200} 0.2048611  0.9440000 0.2170139 2.107535
[3] {X.posts=> 100}            => {X.followers=> 200} 0.2065972  0.9370079 0.2204861 2.091925
[4] {profile.pic=1,
     nums.length.username=0,
     fullname.words=[1, 2],
     X.follows=> 200}          => {X.followers=> 200} 0.2274306  0.9357143 0.2430556 2.089037
```

Vemos relaciones interesantes:

- Cuentas sin descripción ni publicaciones, no tienen foto de perfil
- Cuentas con más de 100 publicaciones, tienen más de 200 seguidores
- Cuentas con foto de perfil, sin numeros en el nombre de usuario, con un nombre real con 1 o 2 palabras y más de 200 personas seguidas, tienen más de 200 seguidores

Estamos viendo que nuestros datos tienen relaciones entre sí, algunas más o menos lógicas, y que hay ciertas cosas que nos dan pistas sobre si una cuenta es falsa o no.

4.2.7 Conjuntos de items frecuentes

Por último, vamos a buscar los conjuntos de items más frecuentes en nuestro dataset. Apriori, además de conseguirnos reglas, también nos permite conseguir los **frequent itemsets**

```
items <- apriori(Tdatarules, parameter = list(
  supp = 0.2,
  conf = 0.3,
  target="frequent itemsets"))
```

```
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
         NA    0.1    1 none FALSE            TRUE       5     0.2      1
 maxlen            target  ext
     10 frequent itemsets TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 115

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[29 item(s), 576 transaction(s)] done [0.00s].
sorting and recoding items ... [21 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 done [0.00s].
sorting transactions ... done [0.00s].
writing ... [1648 set(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

Vamos a ver los más frecuentes (ordenados por soporte)

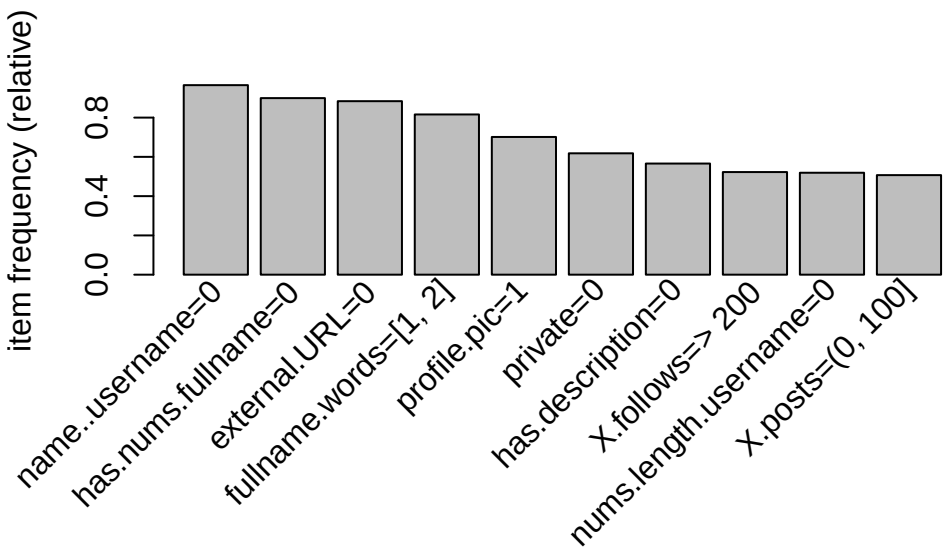
```
frequent_items <- sort(items, by="support")
inspect(head(frequent_items))
```

```
    items                                      support   count
[1] {name..username=0}                         0.9652778 556
[2] {has.nums.fullname=0}                      0.8993056 518
[3] {name..username=0, has.nums.fullname=0}    0.8888889 512
[4] {external.URL=0}                           0.8836806 509
[5] {name..username=0, external.URL=0}         0.8506944 490
[6] {fullname.words=[1, 2]}                    0.8159722 470
```

Observamos que lo más común en nuestro dataset es no tener el nombre de usuario igual al nombre real, con un 96.52% de veces, no tener números en el nombre real, no tener enlace externo...

Vamos a visualizarlo mediante el objeto transaccional. Veamos los 10 items más frecuentes:

```
itemFrequencyPlot(Tdatarules, topN = 10)
```



4.3 Visualización con arulesViz

arulesViz es una herramienta para visualizar y comprender las reglas de asociación generadas con arules en R. Permite representar gráficamente las relaciones entre elementos en nuestros datos, facilitando la identificación de patrones complejos.

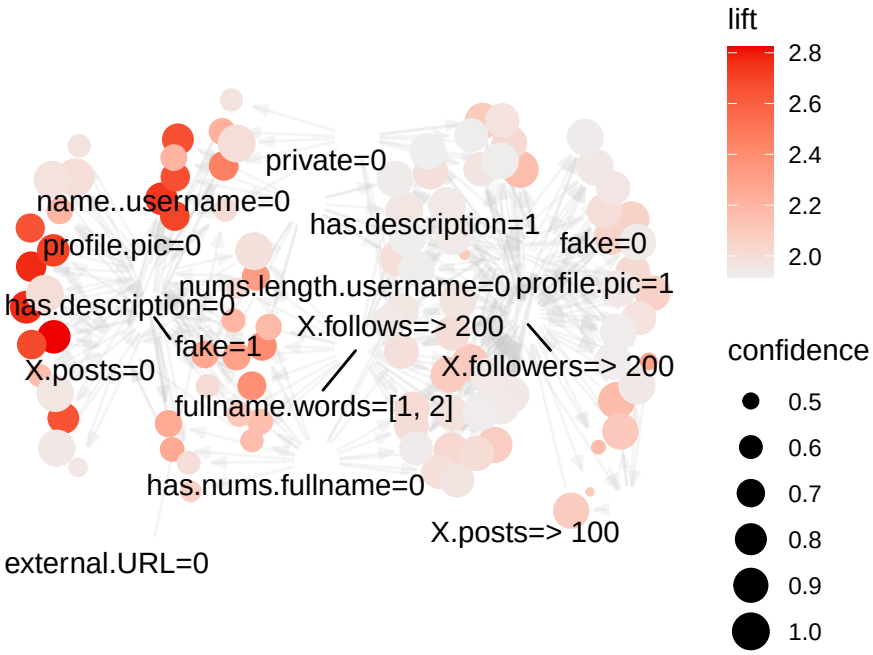
Con arulesViz, podemos visualizar las reglas como grafos o diagramas de coordenadas paralelas, lo que nos ayuda a entender la importancia relativa de cada elemento.

Requieren un mínimo de entendimiento ya que no son gráficos tan intuitivos como uno de barras, pero al entenderlos correctamente, proporcionan mucha información muy útil de manera visual.

```
library(arulesViz)
```

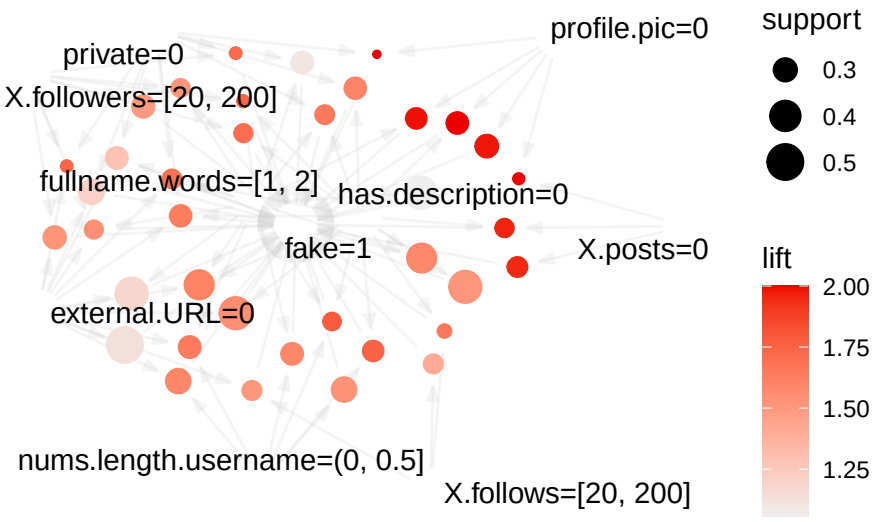
```
plot(rules, method = "graph", control = list(type = "items"),
     shading = "lift", measure = "confidence")
```

```
Available control parameters (with default values):
layout   =  stress
circular     =  FALSE
ggraphdots   =  NULL
edges    =  <environment>
nodes    =  <environment>
nodetext     =  <environment>
colors   =  c("#EE0000FF", "#EEEEEEFF")
engine   =  ggplot2
max  =  100
verbose  =  FALSE
```



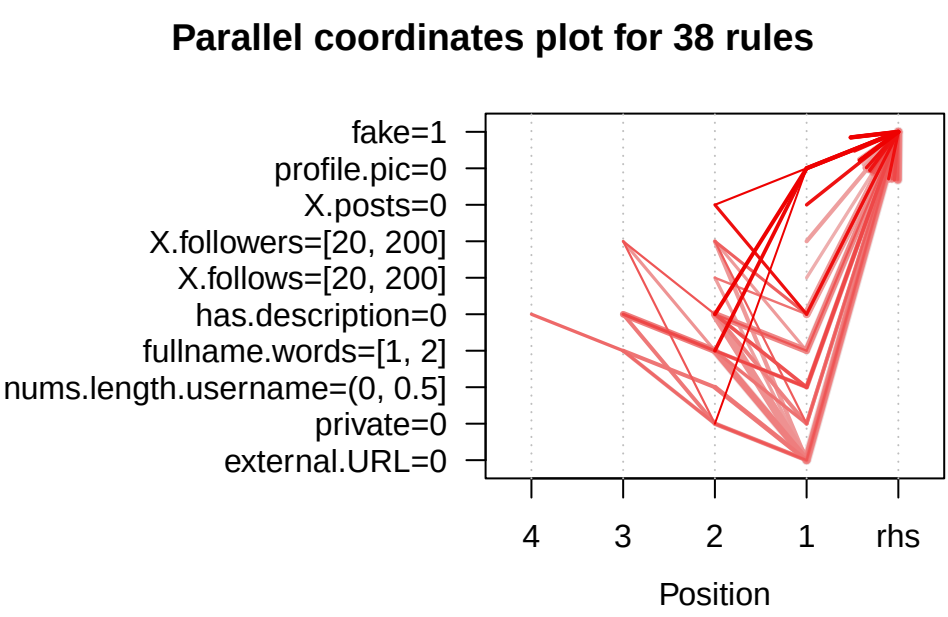
Este gráfico nos muestra cómo ciertos atributos están relacionados entre sí con base en el lift y la confianza. Por ejemplo, se puede observar que "fake=1" tiene una relación significativa con otros atributos como "X.posts=0" y "profile.pic=0", lo que sugiere que estas combinaciones son más comunes de lo esperado y tienen alta confianza en la regla.

```r
plot(reglas_fake, method = "graph")
```



"profile.pic=0" tiene un alto lift y soporte, lo que indica que es común encontrar cuentas sin foto de perfil en el conjunto de datos y que esta característica tiene una alta co-ocurrencia significativa con otros atributos. Las reglas asociadas a "fake=1" también tienen un alto soporte, sugiriendo que estas combinaciones son comunes en los datos.

```
plot(reglas_fake, method = "paracoord", control = list(reorder = TRUE))
```



Podemos ver que las reglas que tienen "X.posts=0" y "fake=1" también suelen tener valores específicos para otras variables como "X.followers" y "profile.pic". Este gráfico facilita la identificación de patrones complejos y de cómo múltiples atributos se combinan en las reglas de asociación.

4.4 Conclusiones

Con las de reglas de asociación, hemos descubierto que nuestros datos contienen un conocimiento valioso y hemos identificado las primeras relaciones significativas entre las características de las cuentas de Instagram. Este análisis nos ha permitido derivar varias reglas que aportan información relevante sobre el comportamiento de estas cuentas y su clasificación como falsas o reales:

1. **Cuentas falsas** (dentro de nuestro dataset):
    - Con una confianza del 100% en nuestro dataset, las cuentas que no tienen foto de perfil ni publicaciones son falsas.
    - Las cuentas sin foto de perfil y sin descripción también son falsas.
    - Las cuentas sin foto de perfil y que son públicas son falsas.
2. **Cuentas reales** (dentro de nuestro dataset):

- Con un 97.85% de confianza en nuestro dataset, las cuentas con foto de perfil, nombre de usuario distinto del nombre real, entre 1 y 2 palabras en el nombre real, y que tienen descripción, son reales.
- Las cuentas con foto de perfil, sin números en el nombre completo, nombre completo diferente del nombre de usuario, y con entre 1 y 2 palabras en el nombre real, son reales con una alta confianza en nuestro dataset.

3. **Reglas de asociación generales**:

   - Las cuentas sin descripción ni publicaciones, no tienen foto de perfil.
   - Las cuentas con más de 100 publicaciones, tienen más de 200 seguidores.
   - Las cuentas con foto de perfil, sin números en el nombre de usuario, con un nombre real de 1 o 2 palabras, y que siguen a más de 200 cuentas, tienen más de 200 seguidores.

4. **Itemsets más frecuentes**:

   - Tener el nombre de usuario distinto del nombre real
   - No tener números en el nombre real
   - No tener enlace en el perfil

Estas reglas nos muestran patrones interesantes y nos ayudan a entender mejor cómo ciertos atributos de las cuentas de Instagram se relacionan entre sí y con la autenticidad de la cuenta.

Sin embargo, es crucial tener en cuenta que estas conclusiones se basan en un conjunto de datos limitado. Para poder generalizar estos hallazgos a una población más amplia de cuentas de Instagram, necesitaríamos un dataset más grande y representativo.

Además, aunque las reglas de asociación nos han proporcionado una buena base de conocimiento, debemos seguir aplicando otras técnicas de análisis de datos para extraer aún más información y verificar la robustez de nuestros hallazgos. Métodos adicionales como la regresión y FCA pueden complementar y profundizar en el conocimiento obtenido hasta ahora, mejorando así nuestra capacidad de detectar cuentas falsas y entender el comportamiento de las cuentas de Instagram en general.

5 FCA

El Análisis Formal de Conceptos (FCA) es una técnica matemática utilizada para descubrir y visualizar estructuras de datos. Se basa en la teoría de conjuntos y la lógica algebraica para identificar patrones y relaciones entre datos en un contexto formal. Un contexto formal consta de un conjunto de objetos, un conjunto de atributos y una relación binaria que indica qué objetos poseen qué atributos.

```
library(fcaR)
library(dplyr)
library(magrittr)
```

En el contexto de análisis de datos de cuentas de Instagram, el FCA nos permite identificar grupos de cuentas que comparten características comunes, lo cual es valioso para detectar patrones de comportamiento, distinguir cuentas auténticas de cuentas falsas, y explorar otros aspectos relevantes.

```
dataset <- read.csv("datasets/train.csv")
```

5.1 Pre-processing

Al igual que para las reglas de asociación, necesitamos hacer un pre-procesado de los datos, ya que para aplicar FCA necesitamos que nuestras variabes sean binarias.

Algunas de ellas ya lo son, pero las que no lo son, tendremos que hacerles el proceso de escalado **(scaling)**.

Vamos a empezar convirtiendo las variables description.length y nums.length.fullname como antes a has.description y has.nums.fullname, ya que creemos que es la información importante que contienen estas variables.

```
dataset <- dataset %>%
  mutate(has.description=as.numeric(description.length > 0)) %>%
  mutate(has.nums.fullname=as.numeric(nums.length.fullname > 0))
```

```
dataset$description.length <- NULL
dataset$nums.length.fullname <- NULL
```

Ahora, para todas las varaibles binarias, vamos a crear su simétrica. Es decir, si tenemos la variable fake que representa que una cuenta es falsa, creamos la varaible no.fake, que representa que una cuenta no es falsa.

¿Por qué simplemente no tener una variable y el valor 1 significa la afirmación de la condición y valor 0 la negación?

Cuando hicimos el objeto transaccional para aplicar el algoritmo apriori del paquete arules, se nos crearon a partir de cada varaible binaria, dos variables. Por ejemplo, para fake, se nos creó fake=1 y fake=0. Es el mismo caso que tenemos ahora, solo que en aquel caso se hizo automáticamente, y ahora tenemos que hacerlo nosotros mismos.

```
dataset <- dataset %>%
  mutate(no.fake = as.numeric(!fake)) %>%
  mutate(no.profile.pic = as.numeric(!profile.pic)) %>%
  mutate(no.name..username = as.numeric(!name..username)) %>%
  mutate(no.external.URL = as.numeric(!external.URL)) %>%
  mutate(no.private = as.numeric(!private)) %>%
  mutate(no.has.description = as.numeric(!has.description)) %>%
  mutate(no.has.nums.fullname = as.numeric(!has.nums.fullname))
```

Esto se hace para que en nuestros conceptos, podamos tener ambos casos, la afirmación y la negación. Con el caso de fake, si solo dejásemos la variable, tendríamos conceptos en los que tenemos que se cumple "fake", pero el no cumplimiento de fake no se representaría y no podríamos formar conceptos con cuentas que no son falsas.

Para las 5 variables no binarias que tenemos, las vamos a escalar a variables nominales.

¿Por qué?

Debido a que vamos a pasarlas a intervalos, y dado el valor de una variable, esta solo estará en uno de los intervalos, y justamente esto lo conseguidmos con las variables nominales.

Para ello, antes tenemos que crear los intervalos. Vamos a usar los mismos que usamos para aplicar las reglas de asociación:

- **nums.length.username**
  - "0"
  - "(0, 0.5]"
  - "(0.5, 1]"

```
dataset <- dataset %>%
  mutate(nums.length.username=
           ifelse(nums.length.username == 0,
                  "0", ifelse(nums.length.username <= 0.5,
                              "(0, 0.5]", "(0.5,1]")))
```

- **fullname.words**
  - "0"
  - "[1, 2]"
  - "> 2"

```
dataset <- dataset %>%
  mutate(fullname.words =
           ifelse(fullname.words == 0,
                  "0", ifelse(fullname.words <= 2,
                              "[1, 2]", "> 2")))
```

- **X.posts**
  - "0"
  - "(0, 100]"
  - "> 100"

```
dataset <- dataset %>%
  mutate(X.posts =
           ifelse(X.posts == 0,
                  "0", ifelse(X.posts <= 100,
                              "(0, 100]", "> 100")))
```

- **X.followers**
  - "< 20"
  - "[20, 200]"
  - "> 200"

```
dataset <- dataset %>%
  mutate(X.followers =
           ifelse(X.followers < 20,
                  "< 20", ifelse(X.followers <= 200,
                              "[20, 200]", "> 200")))
```

- **X.follows**

- "< 20"
- "[20, 200]"
- "> 200"

```
dataset <- dataset %>%
  mutate(X.follows =
           ifelse(X.follows < 20,
                  "< 20", ifelse(X.follows <= 200,
                                 "[20, 200]", "> 200")))
```

Vamos ahora a crear nuestro objeto `FormalContext` del paquete **fcaR**:

```
fc <- FormalContext$new(dataset)
```

Hemos tenido que crear el objeto antes de terminar el pre-processing, ya que para las variables no binarias, es el objeto FormalContext el que nos proporciona los mecanismos de escalado.

Vamosa escalar las 5 variales no binarias que hemos convertido a intervalos, mediante la función scale del objeto FormalContext:

```
fc$scale(type = "nominal", "nums.length.username")
fc$scale(type = "nominal", "X.posts")
fc$scale(type = "nominal", "X.followers")
fc$scale(type = "nominal", "X.follows")
fc$scale(type = "nominal", "fullname.words")
```

Hemos pasado de tener 12 variables, a tener:

```
fc$attributes
```

```
 [1] "profile.pic"                    "nums.length.username = (0, 0.5]"
 [3] "nums.length.username = (0.5,1]" "nums.length.username = 0"
 [5] "fullname.words = [1, 2]"        "fullname.words = > 2"
 [7] "fullname.words = 0"             "name..username"
 [9] "external.URL"                   "private"
[11] "X.posts = (0, 100]"             "X.posts = > 100"
[13] "X.posts = 0"                    "X.followers = [20, 200]"
[15] "X.followers = < 20"             "X.followers = > 200"
[17] "X.follows = [20, 200]"          "X.follows = < 20"
[19] "X.follows = > 200"              "fake"
[21] "has.description"                "has.nums.fullname"
[23] "no.fake"                        "no.profile.pic"
[25] "no.name..username"              "no.external.URL"
```

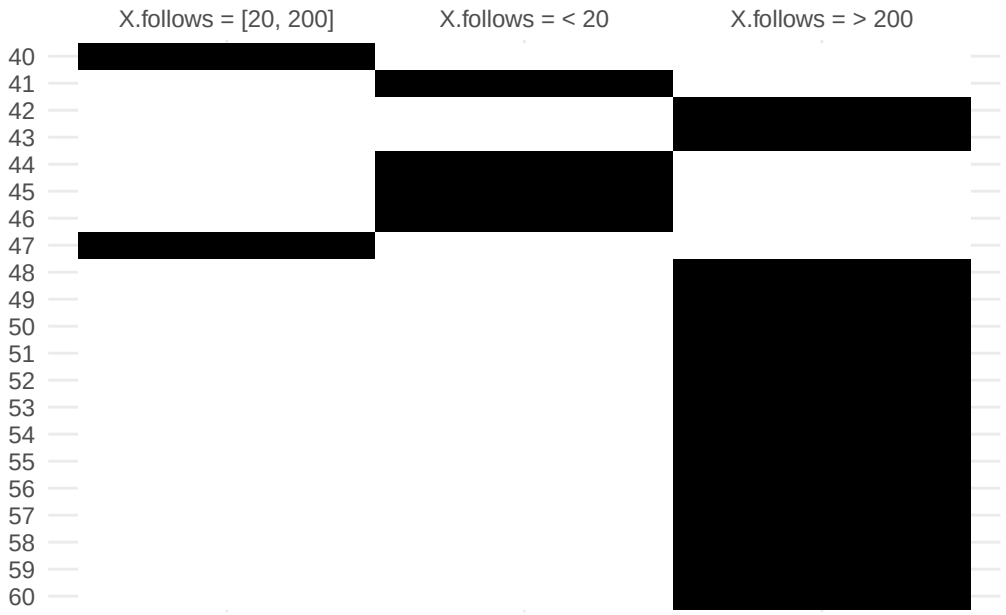
```
[27] "no.private"                          "no.has.description"
[29] "no.has.nums.fullname"
```

Con estos atributos estamos preparados para empezar a crear conceptos y estudiarlos para extraer conocimiento de los datos.

5.2 Primeros pasos

Vamos a observar como hemos transformado las variables que tenían muchos valores, tomando de ejemplo, X.follows:

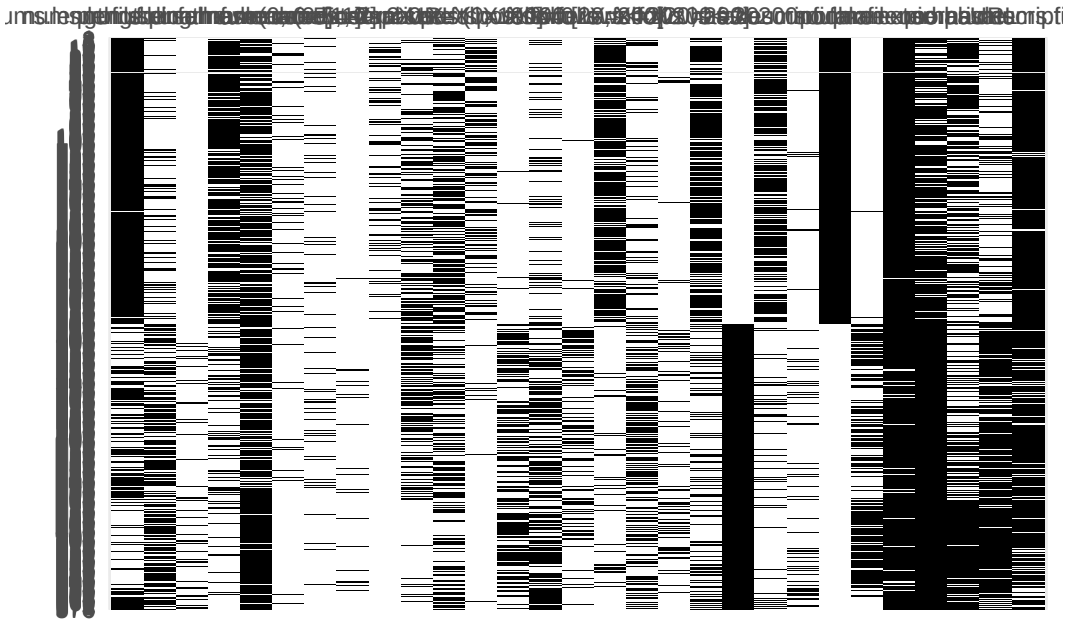
```
plot(fc[40:60][,c(17, 18, 19)])
```



Hemos tomado las observaciones de la 40 a la 60 y las columnas que corresponden con X.follows. Como se puede apreciar, la clasificación es exclusiva, si una persona sigue a a más de 200 personas, entonces no está en "[20, 200]" ni en "< 200", que es justo lo que pensamos que nos dará los mejores conceptos.

Veamos el contexto formal completo:

```
plot(fc)
```



Esto básicamente representa toda la información que tenemos, pero obviamente de esta forma no podemos deducir nada.

Como hemos visto los atributos son todas nuestras variables

```
attr_ig <- fc$attributes
attr_ig
```

```
 [1] "profile.pic"                      "nums.length.username = (0, 0.5]"
 [3] "nums.length.username = (0.5,1]"   "nums.length.username = 0"
 [5] "fullname.words = [1, 2]"          "fullname.words = > 2"
 [7] "fullname.words = 0"               "name..username"
 [9] "external.URL"                     "private"
[11] "X.posts = (0, 100]"               "X.posts = > 100"
[13] "X.posts = 0"                      "X.followers = [20, 200]"
[15] "X.followers = < 20"               "X.followers = > 200"
[17] "X.follows = [20, 200]"            "X.follows = < 20"
[19] "X.follows = > 200"                "fake"
[21] "has.description"                  "has.nums.fullname"
```

```
[23] "no.fake"                          "no.profile.pic"
[25] "no.name..username"                "no.external.URL"
[27] "no.private"                       "no.has.description"
[29] "no.has.nums.fullname"
```

Los objetos son simplemente las observaciones, las filas de nuestro dataset, cada cuenta de Instagram que tenemos.

```
obj_ig <- fc$objects
str(obj_ig)
```

```
 chr [1:576] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12" "13" "14" ...
```

Vamos a calcular manualmente el cierre de algunos atributos, para sacar las primeras conclusiones:

- **Cierre de fake**

```
S <- Set$new(attr_ig)
S$assign(fake = 1)
fc$intent(fc$extent(S))
```

```
{fake, no.external.URL}
```

¿Qué significa esto?

Todas las cuentas que son falsas, no tienen un enlace externo en el perfil. Lo que hemos hecho es calcular el cierre de "fake", que es "fake, no.external.URL".

Vamos a hacer ahora el cierre de external.URL.

- **Cierre de no.external.URL**

```
S <- Set$new(attr_ig)
S$assign(external.URL = 1)
fc$intent(fc$extent(S))
```

```
{profile.pic, external.URL, no.fake}
```

Todas las cuentas que tienen un enlace externo, también tienen foto de perfil, y son reales. Vemos viendo el potencial que tiene el cierre, extra un conocimiento valioso.

5.3 Generando conceptos

Vamos a generar todos los conceptos:

```
fc$find_concepts()
```

Nos han salido:

```
fc$concepts$size()
```

```
[1] 20421
```

Tenemos muchos conceptos. Vamos a visualizar el primero:

```
fc$concepts[1]
```

```
A set of 1 concepts:
1: ({1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 2
```

¿Qué son todos estos números?

Cada número representa una cuenta de Instagram, una observación, una fila de nuestro dataset. Si nos fijamos bien, en esta lista, apareecen todos los números del 1 al 576, y es que justamente tenemos 576 observaciones en nuestro dataset. La otra parte del concepto, es {}. Esto quiere decir que todos los objetos (todas las cuentas) tienen los atributos {} (trivial). No es un concepto útil, vamos a visualizar el que se encuentra en el punto medio:

```
n <- fc$concepts$size()
indice <- floor(n/2)
concepto_medio <- fc$concepts[indice]
concepto_medio
```

```
A set of 1 concepts:
1: ({3, 75, 82, 113, 114, 121, 129, 152, 191, 196, 250, 254, 266, 291, 295, 297, 305, 333, 34
```

Este concepto es más interesante, tenemos ciertas cuentas de Instagram que cumplen una serie de atributos.

```
n_cuentas <- sum(concepto_medio$extents())
attr_medio <- attr_ig[as.logical(concepto_medio$intents())]

paste0("Tenemos que ", n_cuentas, " tienen los atributos:")
```

```
[1] "Tenemos que 39 tienen los atributos:"
```

```
attr_medio
```

```
[1] "nums.length.username = (0, 0.5]" "fullname.words = [1, 2]"
[3] "private"                         "X.posts = (0, 100]"
[5] "no.external.URL"
```

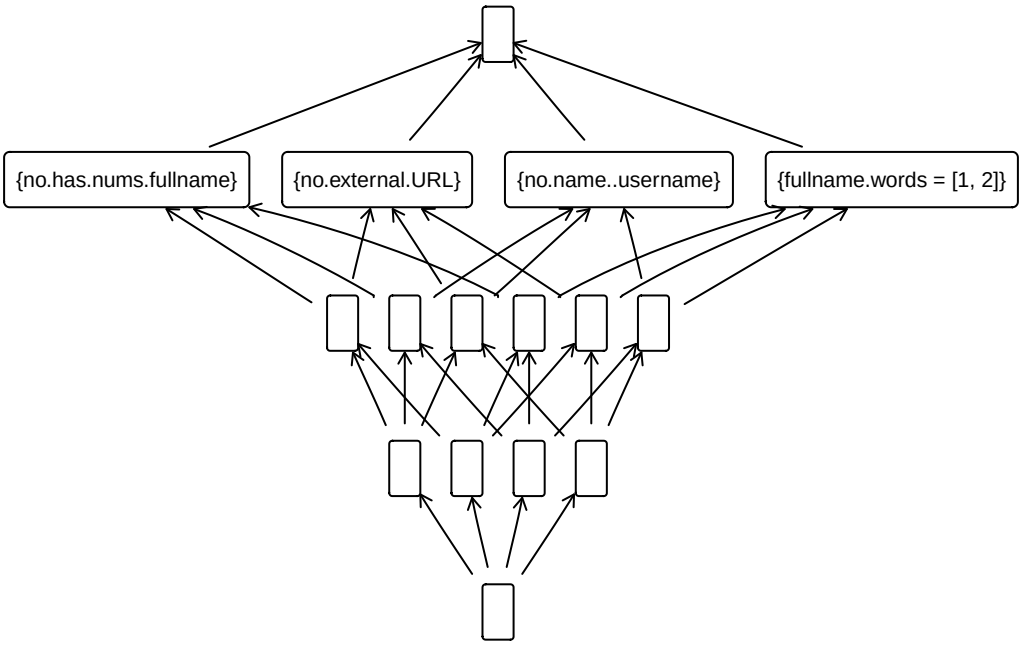
Con fcaR tenemos una forma sencilla y eficiente de obtener conceptos, con los que podemos ver agrupaciones de nuestros datos, donde en nuestro caso encontramos el número de cuentas que comparten una serie de atributos.

Ahora vamos a visualizar nuestros conceptos en forma de retículo, pero para ello, vamos a crear antes un subreticulo, pues tenemos demasiados conceptos.

Vamos a crear el subreticulo de los conceptos con soporte mayor al 0.89, para asi quedarnos con pocos conceptos y poder visualizarlos:

```
idx <- which(fc$concepts$support() > 0.8)
sublaticce <- fc$concepts$sublattice(idx)
```

```
sublaticce$plot()
```



Dado que lo que teníamos antes es un retículo, podemos calcular el ínfimo y el supremo de nuestro subreticulo:

```
fc$concepts$infimum(sublaticce)
```

```
({2, 3, 4, 5, 7, 8, 11, 21, 23, 24, 30, 31, 33, 34, 36, 38, 39, 40, 43, 47, 48,
  49, 52, 57, 58, 59, 60, 61, 63, 67, 68, 71, 72, 73, 74, 75, 76, 77, 78, 80,
  81, 82, 84, 87, 90, 91, 92, 93, 94, 95, 96, 97, 101, 104, 109, 110, 111, 112,
  113, 114, 115, 117, 121, 123, 124, 125, 127, 128, 129, 132, 133, 135, 139,
  142, 144, 145, 146, 149, 151, 152, 153, 154, 155, 156, 158, 159, 161, 163,
  164, 167, 169, 170, 171, 176, 178, 180, 181, 182, 186, 188, 189, 191, 192,
  193, 196, 197, 198, 201, 203, 204, 207, 211, 212, 214, 216, 218, 221, 222,
  223, 224, 225, 226, 227, 229, 230, 231, 232, 236, 237, 239, 240, 241, 242,
  244, 245, 249, 250, 252, 253, 255, 256, 258, 260, 262, 264, 265, 266, 267,
  268, 269, 270, 272, 273, 274, 275, 277, 278, 280, 281, 282, 284, 285, 286,
  287, 288, 289, 290, 291, 294, 296, 297, 298, 299, 300, 301, 303, 304, 305,
  306, 307, 308, 310, 311, 312, 314, 315, 316, 318, 319, 320, 321, 322, 323,
  325, 327, 328, 329, 331, 333, 337, 338, 340, 341, 342, 343, 344, 345, 346,
  350, 352, 353, 356, 359, 361, 362, 364, 366, 370, 371, 372, 373, 375, 376,
  377, 379, 380, 381, 383, 384, 386, 387, 388, 391, 392, 393, 394, 395, 397,
  398, 400, 402, 403, 404, 406, 408, 409, 410, 411, 412, 414, 416, 417, 418,
  419, 420, 421, 423, 424, 426, 427, 428, 430, 431, 432, 433, 435, 437, 438,
```

```
  439, 441, 444, 447, 449, 450, 454, 456, 457, 459, 460, 461, 463, 464, 466,
  467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 478, 479, 480, 482, 483,
  484, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 500,
  501, 502, 503, 504, 505, 506, 507, 508, 509, 511, 512, 514, 519, 521, 522,
  523, 525, 526, 527, 529, 530, 531, 532, 534, 536, 538, 539, 540, 542, 543,
  544, 545, 546, 547, 550, 553, 554, 559, 563, 564, 567, 568, 569, 571, 574,
  575, 576}, {fullname.words = [1, 2], no.name..username, no.external.URL,
  no.has.nums.fullname})
```

```
fc$concepts$supremum(sublaticce)
```

```
({1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
  23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
  42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
  61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
  80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98,
  99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114,
  115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
  130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144,
  145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159,
  160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174,
  175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189,
  190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204,
  205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219,
  220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234,
  235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249,
  250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264,
  265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279,
  280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294,
  295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309,
  310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324,
  325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339,
  340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354,
  355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369,
  370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384,
  385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399,
  400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414,
  415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429,
  430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444,
  445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459,
  460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474,
```

```
475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489,
490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504,
505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519,
520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534,
535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549,
550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564,
565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576}, {})
```

El ínfimo representa

5.4 Generando implicaciones

Con fcaR también podemos generar implicacioens a partir de nuestros conceptos. Son reglas de asociación, pero a diferencia de arules, aquí si se generan reglas con múltiples atributos en el lado derecho.

Podemos generar reglas con el comando find_implications():

```
fc$find_implications()
```

Hemos generado:

```
fc$implications$cardinality()
```

```
[1] 1383
```

Vamos a ver cual es el tamaño medio la parte izquierda y derecha de nuestras reglas:

```
l <- fc$implications$size()
colMeans(l)
```

```
     LHS      RHS
6.704266 1.974693
```

La media de la parte izquierda tiene 6.7 items. En el apartado de reglas de asociación soliamos ver reglas de entre 1, 2 o 3 items como mucho a la izquierda (salvo cuando buscamos con una longitud mínima).

```
sum(colMeans(l))
```

```
[1] 8.678959
```

En promedio el número de items de nuestras reglas son 8.67.

Con fcaR podemos aplicar una serie de simplificaciones, para eliminar reglas redundantes. Estas son; - composition - generalization - simplification - rsimplification

Vamos a aplicarlas a ver si nos eliminamos reglas redundantes

```
fc$implications$apply_rules(rules = c("composition",
                                      "generalization",
                                      "simplification",
                                      "rsimplification"))
```

```
Processing batch

--> Composition: from 1383 to 1383.

--> Generalization: from 1383 to 1383.

--> Simplification: from 1383 to 1383.

--> Right Simplification: from 1383 to 1383.
```

Como podemos leer en la salida del comando, no hemos simplificado nada, lo que significa que no teníamos reglas redundantes ni que se pudiesen simplificar.

Vamos a visualizar las mejores reglas ordenadas por soporte:

```
indices <- order(fc$implications$support(), decreasing = TRUE)
implicaciones_ordenadas <- fc$implications[indices]
head(implicaciones_ordenadas)
```

```
Implication set with 6 implications.
Rule 1: {fake} -> {no.external.URL}
Rule 2: {profile.pic, X.followers = > 200} -> {no.name..username}
Rule 3: {profile.pic, X.follows = > 200, no.has.nums.fullname} ->
  {no.name..username}
Rule 4: {X.followers = > 200, no.fake} -> {no.name..username}
Rule 5: {no.fake, no.external.URL} -> {no.name..username}
Rule 6: {X.follows = > 200, no.fake} -> {no.name..username}
```

Vemos algunas conclusiones como las que sacamos con arules: - Si es falsa no tiene enlace externo - Si tiene foto de perfil y más de 2 seguidores no tiene el nombre completo igual al nombre de usuario...

Vamos a ver las reglas que nos llevan a que la cuenta es falsa:

```
head(fc$implications$filter(rhs = "fake"))
```

```
Implication set with 6 implications.
Rule 1: {no.profile.pic, no.has.description} -> {fake}
Rule 2: {no.profile.pic, no.private} -> {fake}
Rule 3: {has.nums.fullname, no.profile.pic} -> {fullname.words = [1, 2], fake}
Rule 4: {X.follows = < 20, no.profile.pic} -> {fake}
Rule 5: {X.follows = [20, 200], no.profile.pic} -> {fake}
Rule 6: {X.follows = [20, 200], has.nums.fullname} -> {fullname.words = [1, 2],
  fake}
```

Encontramos el mismo conocimiento que con arules.

Con las implicaciones también podemos calcular el cierre de atributos (también podemos con los objetos, pero en este caso no nos interesa pues lo único que podemos sacar es el número de cuentas que habría en el cierre, cada cuenta no tiene nada que la haga especial o distinta de las demás).

- **profile.pic**

```
S <- Set$new(attr_ig)
S$assign(c("no.profile.pic", "no.fake"), values = 1)
fc$implications$closure(S)
```

```
$closure
{nums.length.username = 0, private, X.followers = > 200, X.follows = > 200,
  has.description, no.fake, no.profile.pic, no.name..username, no.external.URL,
  no.has.nums.fullname}
```

El cierre de no tener foto y ser una cuenta real, es: - no tener números en el nombre de usuario, ser privada, tener más de 200 seguidores, seguir a más de 200 personas, tener desripción, tener el nombre real distinto del nombre de usuario, no tener enlace externo y no tener números en el nombre real.

5.5 Conclusiones

FCA es una herramienta poderosa para identificar y visualizar relaciones en conjuntos de datos, permitiendo la detección de patrones y la extracción de conocimiento valioso. FCA nos ha facilitado la diferenciación entre cuentas auténticas y falsas, identificando grupos con características comunes. Hemos necesitado realizar un trabajo de de preprocesamiento como el que hiicmos para las reglas de asociación, con lo que hemos obtenido variables binarias y nominales que hemos usado para crear nuestros conceptos.

El uso de fcaR para generar conceptos y reglas de implicación muestra cómo se pueden descubrir asociaciones no triviales, proporcionando una visión profunda de los datos.

Con toda la información que hemos obtenido hasta ahora, estamos listos para realizar regresión, una técnica que nos permitirá realizar predicciones sobre la veracidad o falsedad de las cuentas de Instagram.

6 Regresión

La regresión es una técnica estadística que nos permite entender y modelar la relación entre una variable dependiente (o de respuesta) y una o más variables independientes (o predictoras). En términos simples, nos ayuda a prever cómo cambia la variable dependiente cuando las variables independientes cambian.

```
library(ggplot2)
library(magrittr)
library(dplyr)
```

En el contexto de nuestro dataset de Instagram, la regresión puede ser extremadamente útil para:

Predecir el número de seguidores: Basándonos en variables como el número de publicaciones, la cantidad de personas a las que sigue una cuenta, la longitud de la descripción, etc. Identificar cuentas falsas: Determinando si ciertos patrones en los datos se asocian con cuentas falsas. Analizar la influencia de la privacidad: Evaluar cómo la privacidad de una cuenta afecta el número de seguidores o seguidos. Análisis de Regresión Vamos a realizar diversos análisis de regresión para explorar nuestro dataset y obtener insights valiosos.

```
dataset <- read.csv("datasets/train.csv")
attach(dataset)
```

6.1 Primeros pasos

Para las técnicas de regresión es importante que todas nuestras variables sean numéricas. No podemos hacer un ajuste con carácteres. Si fuese el caso, podríamos hacer de nuevo un trabajo de pre-processing, pero por suerte:

```
str(dataset)
```

```
'data.frame':   576 obs. of  12 variables:
 $ profile.pic         : int  1 1 1 1 1 1 1 1 1 1 ...
 $ nums.length.username: num  0.27 0 0.1 0 0 0 0 0 0 0 ...
```

```
$ fullname.words      : int  0 2 2 1 2 4 2 2 0 2 ...
$ nums.length.fullname: num  0 0 0 0 0 0 0 0 0 0 ...
$ name..username      : int  0 0 0 0 0 0 0 0 0 0 ...
$ description.length  : int  53 44 0 82 0 81 50 0 71 40 ...
$ external.URL        : int  0 0 0 0 0 1 0 0 0 1 ...
$ private             : int  0 0 1 0 1 0 0 0 0 0 ...
$ X.posts             : int  32 286 13 679 6 344 16 33 72 213 ...
$ X.followers         : int  1000 2740 159 414 151 669987 122 1078 1824 12945 ...
$ X.follows           : int  955 533 98 651 126 150 177 76 2713 813 ...
$ fake                : int  0 0 0 0 0 0 0 0 0 0 ...
```

¡Nuestro dataset tiene todas las variables númericas! Probablemente con regresión tengamos más juego que con FCA.

Aunque no todo es bueno. La variable que queremos predecir, fake, es una variable binaria, y la regresión no es lo más ideal para predecir exactamente entre dos valores.

Vamos a intentar sacar el mayor conocimiento de nuestros datos y encontrar ajustes interesantes.

6.2 Primeras regresiones

Aunque lo que queremos es predecir la variable "fake", es bueno antes de comenzar ver relaciones entre otras variables, y particularmente que variables se relacionan más con fake.

En el análisis exploratorio, construimos una tabla que nos proporcionaba las mayores relaciones entre variables. Vamos a recuperar dicha tabla:

```r
threshold <- 0.4

cor_table <- data.frame(as.table(cor(dataset))) %>%
  rename(Correlation = Freq)

variables <- colnames(dataset)
n_variables <- length(dataset)

# Para que no haya repeticiones simétricas, vamos a poner la restricción de que el
# orden léxicográfico de una variable sea mayor (o menor) que la otra.
# Con un != no valdría porque habría valores filas simétricas
cor_table %>%
  filter(as.character(Var1) > as.character(Var2) & abs(Correlation) > threshold) %>%
  arrange(desc(abs(Correlation)))
```

```
                  Var1                  Var2 Correlation
1          profile.pic                  fake  -0.6373153
2 nums.length.username                  fake   0.5876865
3         external.URL   description.length   0.4823131
4                 fake   description.length  -0.4608246
5 nums.length.username nums.length.fullname   0.4085665
```

Vemos que la mayor correlación (aunque inversa, pero eso no nos importa, lo importante es que hay relación para la regresión). El problema con el que nos encontraremos más adelante, es que estas variables son binarias.

Podemos ir intuyendo que si queremos ajustar una variable a partir de una variable binaria, como dicha variable solo puede tomar dos valores, por muy complejo que sea el ajuste, solo hay dos posibles resultados. Como la variable que también queremos predecir es binaria, solo nos queda asignar un valor de la variable usada en el ajuste para un valor de la variable fake, pero obviamente esto no es un buen modelo.

Vamos a intentar hacer una regresión simple entre estas dos variables.

```r
modelo1 <- lm(fake ~ profile.pic, data = dataset)
summary(modelo1)
```

```
Call:
lm(formula = fake ~ profile.pic, data = dataset)

Residuals:
     Min       1Q   Median       3Q      Max
-0.98837 -0.29208 -0.14023  0.01163  0.70792

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.98837    0.02943   33.58   <2e-16 ***
profile.pic -0.69629    0.03514  -19.81   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.386 on 574 degrees of freedom
Multiple R-squared:  0.4062,    Adjusted R-squared:  0.4051
F-statistic: 392.6 on 1 and 574 DF,  p-value: < 2.2e-16
```

Obtenemos un 0.40 de R^2 ajustado, lo que no está tan mal para la simplicidad del modelo.

Los únicos dos valores que puede dar como predicción nuestro modelo son:

```
unique(predict(modelo1))
```

```
[1] 0.2920792 0.9883721
```

A nosotros nos interesa un valor de 0 o un valor 1, así que podemos decir que el 0.29 es 0, por cercanía, y el 0.98 es 1. Construyamos una tabla:

```
predicciones <- predict(modelo1)
predicciones_binarizado <- ifelse(predicciones < 0.3, 0, 1)

tabla <- data.frame(profile.pic=dataset$profile.pic,
           fake=dataset$fake,
           prediccion=predicciones,
           prediccion_bin=predicciones_binarizado)

head(tabla, 10)
```

```
   profile.pic fake prediccion prediccion_bin
1            1    0  0.2920792              0
2            1    0  0.2920792              0
3            1    0  0.2920792              0
4            1    0  0.2920792              0
5            1    0  0.2920792              0
6            1    0  0.2920792              0
7            1    0  0.2920792              0
8            1    0  0.2920792              0
9            1    0  0.2920792              0
10           1    0  0.2920792              0
```

Si observamos lo que está pasando, es que nuestro modelo de regresión (junto a la binarización), para las cuentas con foto, predice que no es falsa, y para las cuentas sin foto, predice que es falsa.

Vamos a calcular el % de aciertos asumiendo lo anterior:

```
total <- nrow(dataset)
aciertos <- sum(predicciones_binarizado == dataset$fake)

paste0("Hemos obtenido un ", round(100*aciertos/total,2), "% de aciertos")
```

```
[1] "Hemos obtenido un 79.17% de aciertos"
```

Parece un buen resultado, ¿verdad? Pues casi un 80% de aciertos es un buen modelo… ¿no?

Debemos tener en cuenta que nuestro dataset es pequeño, y que queremos un modelo riguroso, no podemos basarnos en una sola característica para predecir si una cuenta de Instagram es falsa.

Si concluyesemos que el modelo anterior es muy bueno, nuestro predictor se basaría simplemente en tener foto de perfil o no. Es demasiado básico. El porcentaje de aciertos viene de que en nuestro dataset esto es un patrón que suele aparecer, pero quizá si el dataset fuese mas grande, no aparecería:

```
casualidades <- sum(dataset$profile.pic == !dataset$fake)
round(100 * casualidades / total, 2)
```

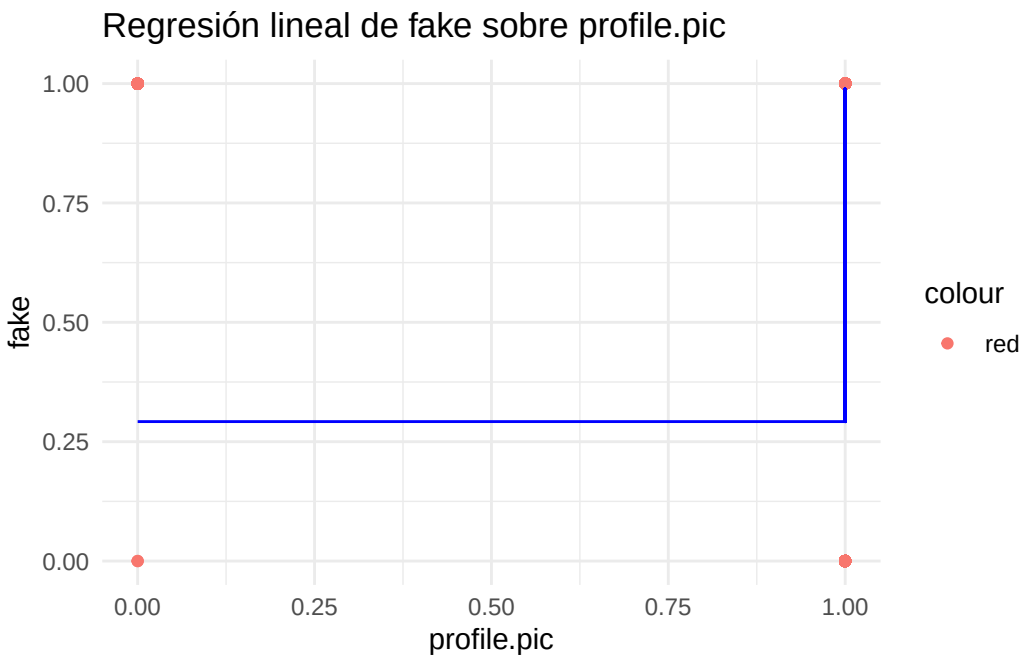
```
[1] 79.17
```

De ahí viene el porcentaje de aciertos.

Y aunque el patrón siguiese dándose en un dataset más grande, es un modelo demasiado simple, no podemos ir a alguien y decirle si su cuenta es falsa o no simplemente por si tiene foto de perfil.

Ya que es un ajuste bidimensional, pero si lo visualizamos, para que veamos que esto por si solo no es algo muy útil:

```
ggplot(dataset, aes(x = profile.pic, y = fake, color="red")) +
  geom_point() +
  geom_line(y = predicciones, color ="blue") +
  labs(title = "Regresión lineal de fake sobre profile.pic",
       x = "profile.pic",
       y = "fake") +
  theme_minimal()
```



La función lineal que nos ha creado el modelo es (donde Y es fake y X es profile.pic):

```
paste0("y = ", modelo1$coefficients[2], "x + ", modelo1$coefficients[1])
```

```
[1] "y = -0.696292885102468x + 0.988372093023256"
```

6.3 Preparando el terreno

Ya hemos visto que predecir usando únicamente variables binarizadas no es una gran opción. También hemos visto, que dado que queremos ajustar una variable binaria (fake), los resultados que queremos son 0 o 1.

Por lo que vamos a crear una serie de funciones con las que podamos binarizar nuestros resultados para probar nuestras predicciones, y también calcular valores estadísticos del modelo teniendo en cuenta esta binarización:

- **Función binarize.predictions** Simplemente dado el conjunto de predicciones, las binariza usando cierto threshold, que por defecto será 0.5:

```
binarize.predictions <- function(predictions, threshold=0.5) {
  ifelse(predictions < 0.5, 0, 1)
```

```
}
```

- **Función aciertos.binarized** Calcula el porcentaje de aciertos para nuestro dataset, binarizando los resultados

```
aciertos.binarized <- function(predictions, real, threshold=0.5) {
  total <- length(real)
  aciertos <- sum(binarize.predictions(predictions) == real)

  100 * aciertos / total
}
```

- **Función rss.binarized** Calcula el RSS con las predicciones binarizadas

```
rss.binarized <- function(model, real, threshold=0.5) {
  sum((real - binarize.predictions(predict(model), threshold))^2)
}
```

- **Función rsquared.binarized** Calcula el parámetro R^2 del modelo, pero usando las predicciones binarizadas (ya que el R^2 que nos proporciona summary, se realiza con los valores sin binarizar, y es un valor diferente)

```
rsquared.binarized <- function(model, real, threshold=0.5) {
  rss <- rss.binarized(model, real, threshold)
  tss <- sum((real - mean(real))^2)

  1 - (rss/tss)
}
```

- **Función rse.binarized** Lo mismo que con rsquared.binarized, pero para calcular RSE (residual standar error):

```
rse.binarized <- function(model, real, threshold=0.5) {
  rss <- rss.binarized(model, real, threshold)
  n <- length(real)

  rse <- sqrt((1/(n-2))*rss)

  return(rse)
}
```

- **Función fstatistic.binarized** Calcula el F-statistic para las predicciones binarizadas:

```r
fstatistic.binarized <- function(model, real) {
  tss <- sum((real - mean(real))^2)
  rss <- rss.binarized(model, real, threshold)

  p <- length(model$coefficients) - 1
  n <- length(real)

  ((tss - rss) / p) / (rss / (n - p - 1))
}
```

6.4 Regresiones previas

Vamos a intentar ahora ajustar ahora algunas de las variables que no son binarias entre sí, a ver si encontramos alguna relación interesante.

Para no probar a lo loco, vamos a coger la tabla del principio pero esta vez la vamos a construir solo con variables que no sean binarias:

```r
unique_values <- sapply(dataset, function(x) length(unique(x)))
binary_vars <- names(which(unique_values == 2))

cor_table %>%
  filter(as.character(Var1) > as.character(Var2)
         & !(Var1 %in% binary_vars)
         & !(Var2 %in% binary_vars)) %>%
  arrange(desc(abs(Correlation)))
```

```
                  Var1                 Var2  Correlation
1  nums.length.username nums.length.fullname  0.408566542
2               X.posts          X.followers  0.321385480
3  nums.length.username   description.length -0.321170271
4        fullname.words   description.length  0.272522165
5             X.follows   description.length  0.226561422
6  nums.length.username       fullname.words -0.225472125
7             X.follows nums.length.username -0.172413275
8               X.posts nums.length.username -0.157442112
9               X.posts   description.length  0.144823702
10 nums.length.fullname   description.length -0.117521050
11              X.posts            X.follows  0.098225040
12            X.follows       fullname.words  0.094854964
```

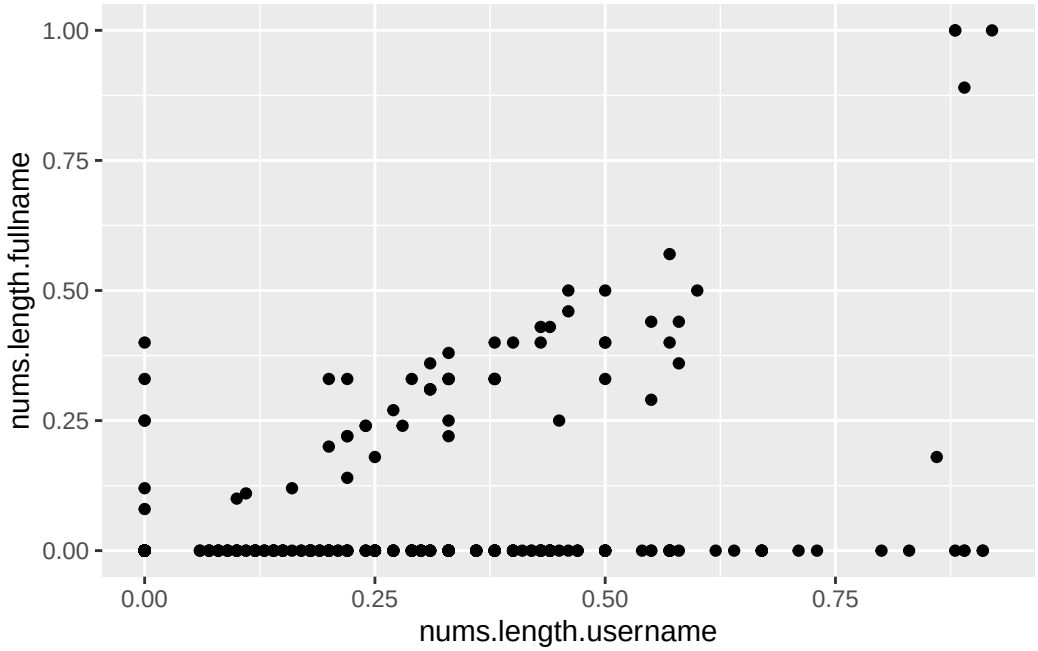
```
13 nums.length.fullname       fullname.words -0.094347995
14              X.posts       fullname.words  0.073350182
15            X.follows nums.length.fullname -0.067971092
16          X.followers nums.length.username -0.062785090
17              X.posts nums.length.fullname -0.057715505
18          X.followers       fullname.words  0.033224604
19          X.followers nums.length.fullname -0.027034712
20            X.follows          X.followers -0.011065994
21          X.followers   description.length  0.005929455
```

Vamos a probar una regresión con el primer par de variables, para explorar como jugar con los datos, y luego pasaremos a intentar predecir la variable fake en función de todo lo demás:

- **Regresión entre nums.length.username y nums.length.fullname**

Primero vamos a visualizar la nube de puntos:

```
ggplot(dataset, aes(x=nums.length.username, y=nums.length.fullname)) +
  geom_point()
```



Se aprecia como hay muchos valores en nums.length.fullname = 0 (y = 0), y algunos en nums.length.username = 0 (x = 0) pero por lo demás, parece haber una clara tendencia lineal alcista.

Creamos un modelo de regresión lineal, pues salvo los valores en los ejes, parece que la nube de puntos dibuja una línea:

```
modelo <- lm(nums.length.fullname ~ nums.length.username, data = dataset)
```

```
summary(modelo)
```

```
Call:
lm(formula = nums.length.fullname ~ nums.length.username, data = dataset)

Residuals:
     Min       1Q   Median       3Q      Max
-0.21426 -0.05667  0.00303  0.00303  0.79291

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)          -0.003026   0.005999  -0.504    0.614
nums.length.username  0.238772   0.022264  10.725   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1143 on 574 degrees of freedom
Multiple R-squared:  0.1669,    Adjusted R-squared:  0.1655
F-statistic:   115 on 1 and 574 DF,  p-value: < 2.2e-16
```

El R^2 es muy malo, un 0.16. Vamos a visualizar la regresión:

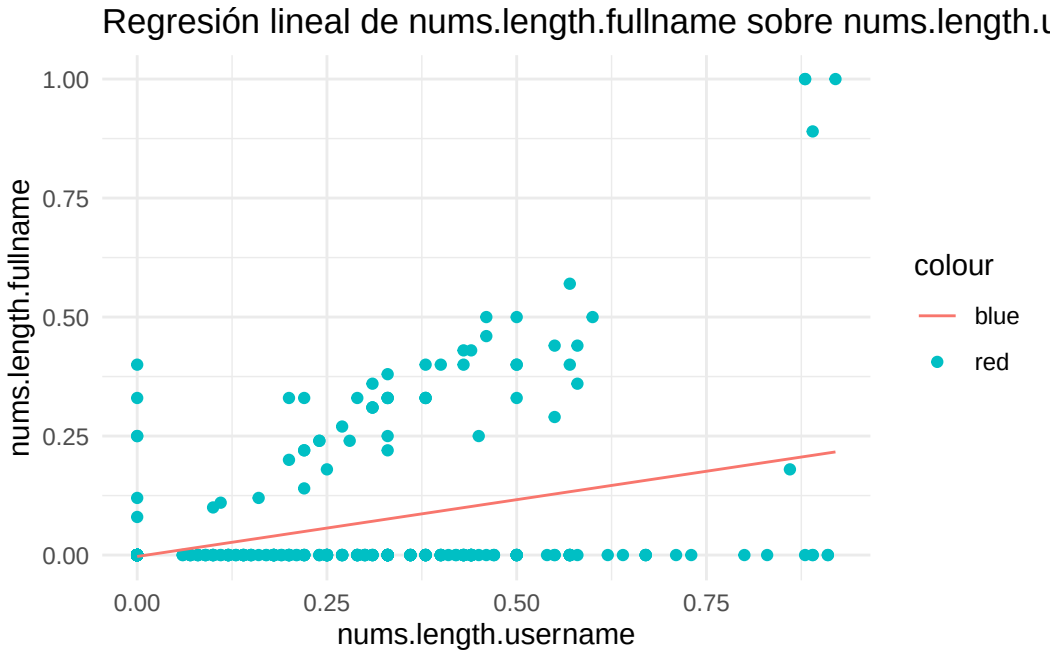
Veamos los valores predichos en una tabla:

```
tabla <- data.frame(
  nums.length.username=nums.length.username,
  nums.length.fullname=nums.length.fullname,
  prediccion=predict(modelo)
)

head(tabla, 10)
```

```
   nums.length.username nums.length.fullname   prediccion
1                  0.27                    0  0.061442570
2                  0.00                    0 -0.003025925
```

```
3                0.10                    0  0.020851295
4                0.00                    0 -0.003025925
5                0.00                    0 -0.003025925
6                0.00                    0 -0.003025925
7                0.00                    0 -0.003025925
8                0.00                    0 -0.003025925
9                0.00                    0 -0.003025925
10               0.00                    0 -0.003025925
```

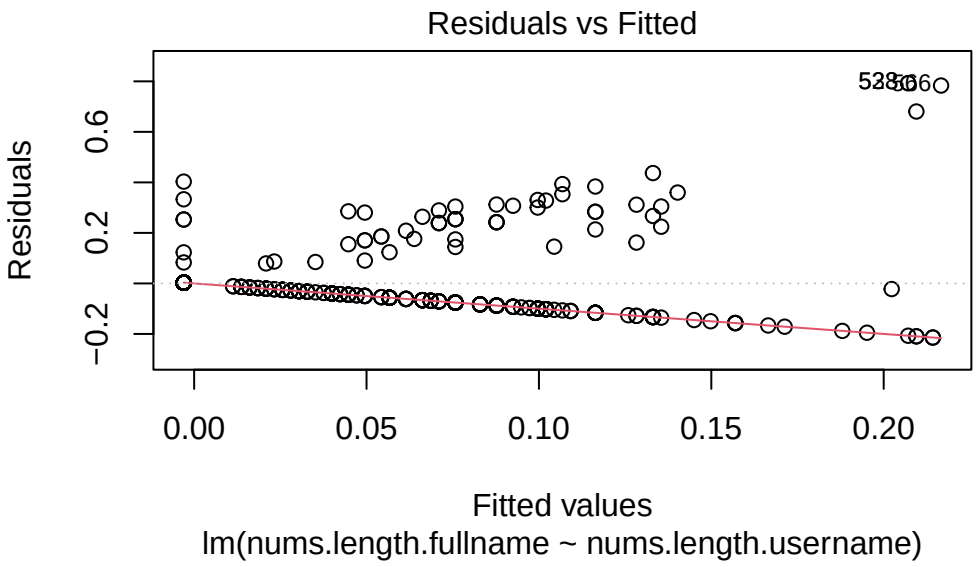
```
ggplot(dataset, aes(x = nums.length.username, y = nums.length.fullname, color="red")) +
  geom_point() +
  geom_line(aes(x = nums.length.username, y = predict(modelo), color ="blue")) +
  labs(title = "Regresión lineal de nums.length.fullname sobre nums.length.username",
       x = "nums.length.username",
       y = "nums.length.fullname") +
  theme_minimal()
```

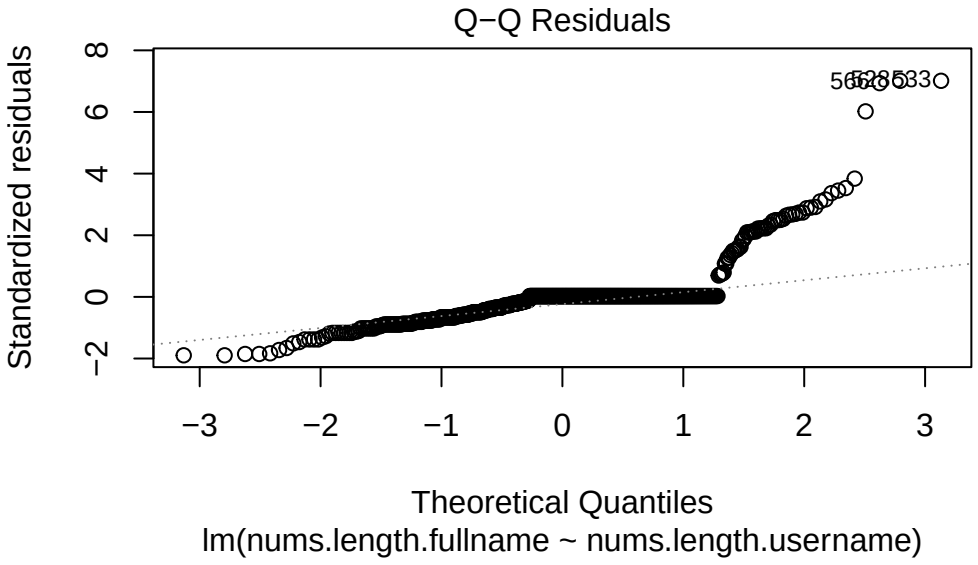


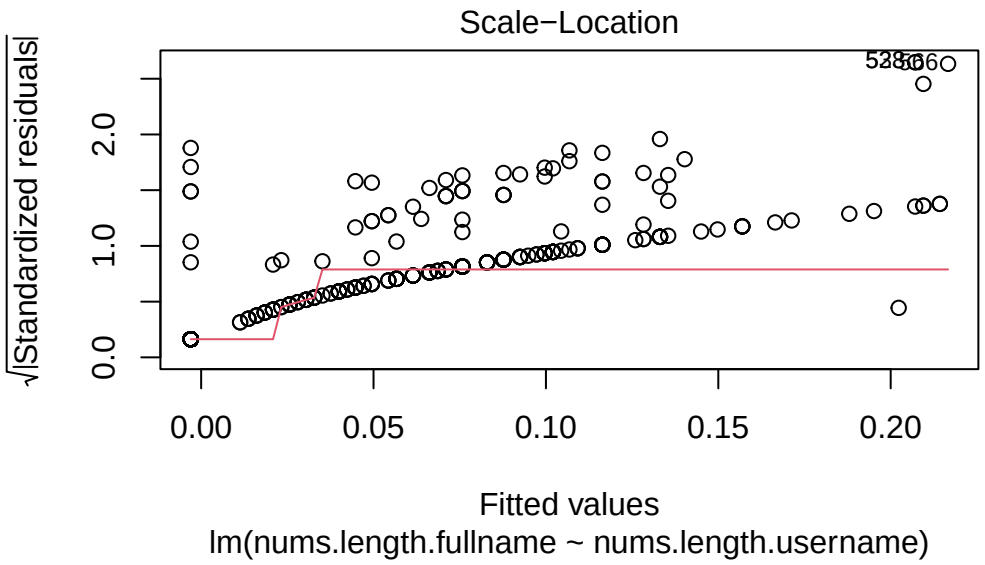
Vemos que la línea se ve influenciada altamente por los valores en los ejes. Se trata de las cuentas sin números en el nombre completo pero con números en el nombre de usuario y viceversa.

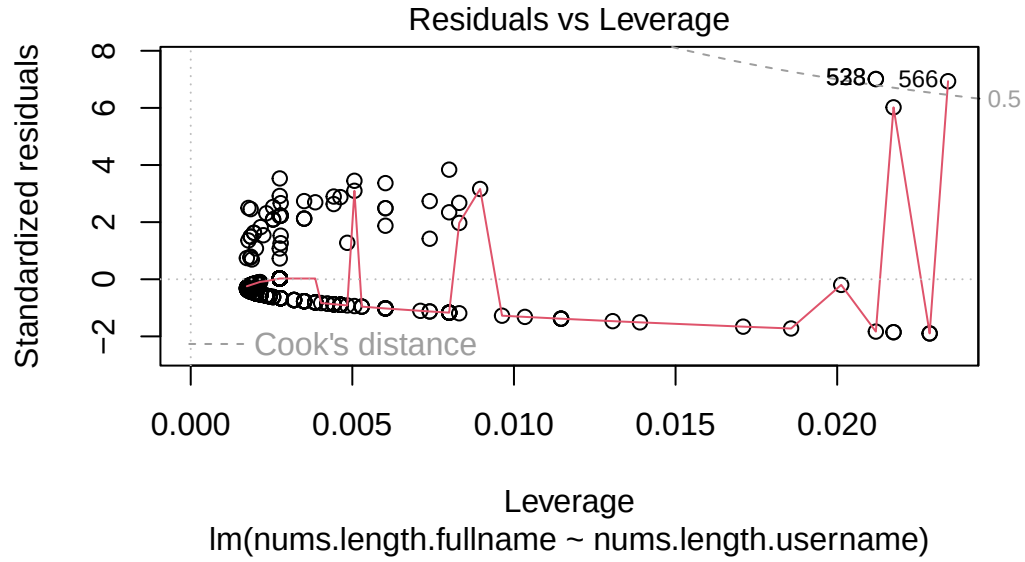
Vamos a visualizar los gráficos que nos proporciona la función plot aplicada al modelo:

```r
plot(modelo)
```









La primera gráfica es simplemente un gráfico entre los residuos y los valores predichos. Vemos que R nos marca algunos valores que están muy lejanos el gráfico, estos son outliers.

El segundo gráfico nos muestra un gráfico nos mostraría una linea reacta si los errores se distribuyesen de manera normal, lo que claramente no es el caso. Los outliers se desvian de dicha línea, y vemos que tenemos muchos valores que se desvian bastante, probablemente los valores de los ejes.

La tercera gráfica también marca outliers.

La cuarta y última muestra la distancia de Cook, que nos indica que puntos tienen una mayor influencia en la regresión. También marca los outliers.

Lo que nos recomienda lo anterior es eliminar esos valores marcados, pero vamos a eliminar todos los valores de los ejes, aunque sean bastantes, para intentar encontrar una relación entre las cuentas que tienen números en el nombre de usuario y nombre real, eliminando los que no tiene números en alguno de los.

Eliminamos outliers:

```
dataset_modif <- dataset %>%
  filter(nums.length.username > 0 & nums.length.fullname > 0)

nrow(dataset_modif)
```

```
[1] 52
```

Nos hemos quedado prácticamente con el 9% de los datos, la mayoría no tienen números en ambos nombres.

Vamos a realizar la predicción ahora:

```
modelo <- lm(dataset_modif$nums.length.fullname ~ dataset_modif$nums.length.username,
             data = dataset_modif)
```

```
summary(modelo)
```

```
Call:
lm(formula = dataset_modif$nums.length.fullname ~ dataset_modif$nums.length.username,
    data = dataset_modif)

Residuals:
     Min       1Q   Median       3Q      Max
-0.56406 -0.04487  0.01122  0.05000  0.23932
```

```
Coefficients:
                                  Estimate Std. Error t value Pr(>|t|)
(Intercept)                        0.02932    0.03803   0.771    0.444
dataset_modif$nums.length.username  0.83109    0.08301  10.012 1.54e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1191 on 50 degrees of freedom
Multiple R-squared:  0.6672,    Adjusted R-squared:  0.6605
F-statistic: 100.2 on 1 and 50 DF,  p-value: 1.544e-13
```

El R^2 ha subido a 0.66, pero recordamos que hemos eliminado el 91% de los datos.

Visualicemos:

```r
data.frame(
  nums.length.username=dataset_modif$nums.length.username,
  nums.length.fullname=dataset_modif$nums.length.fullname,
  prediccion=predict(modelo)
)
```

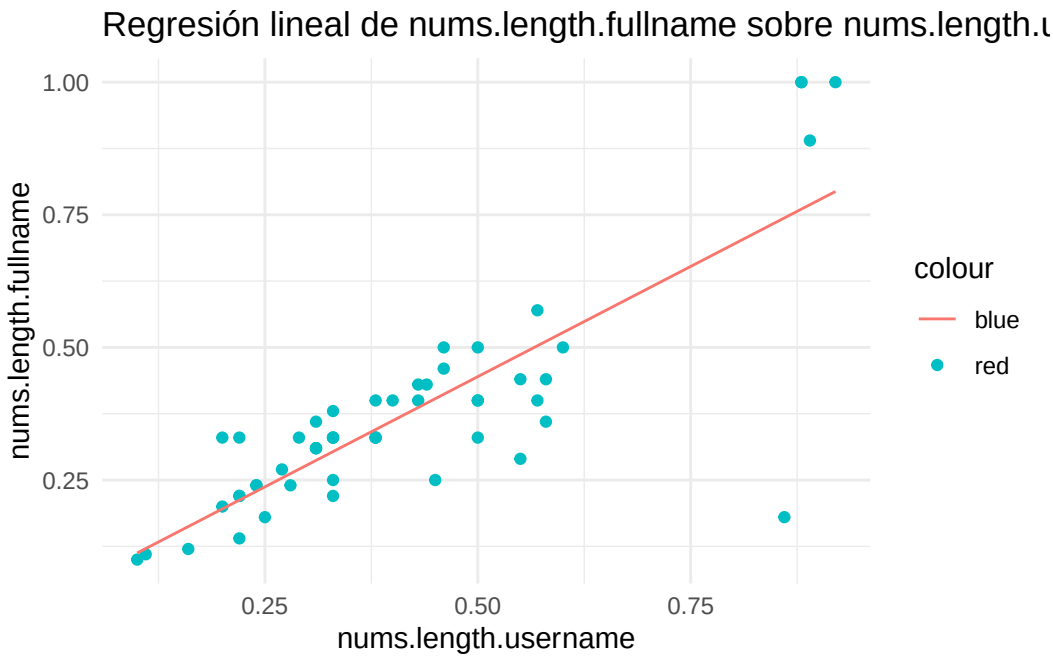
```
   nums.length.username nums.length.fullname prediccion
1                  0.10                 0.10  0.1124314
2                  0.24                 0.24  0.2287844
3                  0.45                 0.25  0.4033138
4                  0.22                 0.14  0.2121625
5                  0.22                 0.22  0.2121625
6                  0.44                 0.43  0.3950029
7                  0.31                 0.36  0.2869609
8                  0.33                 0.33  0.3035827
9                  0.55                 0.29  0.4864231
10                 0.11                 0.11  0.1207423
11                 0.31                 0.31  0.2869609
12                 0.89                 0.89  0.7689946
13                 0.20                 0.20  0.1955407
14                 0.38                 0.40  0.3451374
15                 0.22                 0.22  0.2121625
16                 0.24                 0.24  0.2287844
17                 0.58                 0.36  0.5113559
18                 0.33                 0.38  0.3035827
19                 0.33                 0.33  0.3035827
20                 0.31                 0.31  0.2869609
```

21	0.50	0.33	0.4448685
22	0.33	0.33	0.3035827
23	0.40	0.40	0.3617592
24	0.43	0.40	0.3866920
25	0.57	0.40	0.5030450
26	0.29	0.33	0.2703390
27	0.38	0.33	0.3451374
28	0.33	0.25	0.3035827
29	0.27	0.27	0.2537172
30	0.28	0.24	0.2620281
31	0.20	0.33	0.1955407
32	0.50	0.40	0.4448685
33	0.58	0.44	0.5113559
34	0.22	0.33	0.2121625
35	0.38	0.33	0.3451374
36	0.88	1.00	0.7606837
37	0.88	1.00	0.7606837
38	0.33	0.22	0.3035827
39	0.50	0.50	0.4448685
40	0.25	0.18	0.2370953
41	0.57	0.57	0.5030450
42	0.46	0.46	0.4116248
43	0.60	0.50	0.5279778
44	0.46	0.50	0.4116248
45	0.31	0.31	0.2869609
46	0.43	0.43	0.3866920
47	0.50	0.40	0.4448685
48	0.86	0.18	0.7440619
49	0.16	0.12	0.1622970
50	0.92	1.00	0.7939274
51	0.55	0.44	0.4864231
52	0.38	0.33	0.3451374

Y visualizamos:

```
ggplot(dataset_modif, aes(
  x = nums.length.username,
  y = nums.length.fullname,
  color="red")) +
  geom_point() +
  geom_line(aes(
    x = nums.length.username,
```

```
    y = predict(modelo),
    color ="blue")) +
labs(title = "Regresión lineal de nums.length.fullname sobre nums.length.username",
     x = "nums.length.username",
     y = "nums.length.fullname") +
theme_minimal()
```



El ajuste es mucho mejor que el anterior, y eso que aún tenemos algún outlier como el que se ve abajo derecha.

Ya que hemos probado con estas dos variables estamos más que preparados para pasar a intentar predecir la variable importante, fake.

6.5 Regresión con fake

Ya hemos explorado suficiente, vamos a por lo grande.

Vamos a intentar predecir la variable fake a partir de las demás. Usando ".", podemos ajustar la variable fake respecto a todas las demás. Veamos como queda dicho modelo.

```
modelo2 <- lm(fake ~ ., data = dataset)
```

```
summary(modelo2)
```

```
Call:
lm(formula = fake ~ ., data = dataset)

Residuals:
     Min       1Q   Median       3Q      Max
-0.73096 -0.23729 -0.06653  0.24048  1.01052

Coefficients:
                       Estimate Std. Error t value Pr(>|t|)
(Intercept)           7.931e-01  3.798e-02  20.880  < 2e-16 ***
profile.pic          -4.380e-01  3.345e-02 -13.094  < 2e-16 ***
nums.length.username  8.062e-01  7.522e-02  10.718  < 2e-16 ***
fullname.words       -3.354e-02  1.333e-02  -2.516 0.012142 *
nums.length.fullname -2.775e-02  1.212e-01  -0.229 0.818988
name..username        2.241e-01  7.641e-02   2.933 0.003498 **
description.length   -1.510e-03  4.342e-04  -3.478 0.000544 ***
external.URL         -1.542e-01  4.800e-02  -3.213 0.001390 **
private              -9.459e-03  2.843e-02  -0.333 0.739459
X.posts              -9.094e-05  3.570e-05  -2.547 0.011120 *
X.followers          -9.960e-09  1.539e-08  -0.647 0.517743
X.follows            -1.850e-05  1.499e-05  -1.235 0.217530
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3166 on 564 degrees of freedom
Multiple R-squared:  0.6074,    Adjusted R-squared:  0.5998
F-statistic: 79.33 on 11 and 564 DF,  p-value: < 2.2e-16
```

Nuestro modelo tiene un 0.6 de R^2, un error residual relativamente pequeño y un f-statistic mayor de 79.33. No está nada mal.

Vamos a ver el % de aciertos usando las funciones que creamos al principio.

```
aciertos.binarized(predict(modelo2), fake)
```

```
[1] 89.93056
```

!Casi un 90%¡. Parece que la regresión nos está dando buenos resultados, y eso que aún no hemos intentado mejorar el modelo.

Calculemos el RSE, R^2 Y F-statistic estádisticos para nuestro modelo binarizado:

```
paste("RSE:", round(rse.binarized(modelo2, fake),4))
```

```
[1] "RSE: 0.3179"
```

```
paste("R2:", round(rsquared.binarized(modelo2, fake),4))
```

```
[1] "R2: 0.5972"
```

```
paste("F-statistic:", round(fstatistic.binarized(modelo2, fake),2))
```

```
[1] "F-statistic: 76.03"
```

Los valores de los estádisticos son prácticamente los mismos que para el modelo sin binarizar.

6.6 Mejorando el modelo

Si hacemos summary

```
summary(modelo2)
```

```
Call:
lm(formula = fake ~ ., data = dataset)

Residuals:
     Min       1Q   Median       3Q      Max
-0.73096 -0.23729 -0.06653  0.24048  1.01052

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)           7.931e-01  3.798e-02  20.880  < 2e-16 ***
profile.pic          -4.380e-01  3.345e-02 -13.094  < 2e-16 ***
```

```
nums.length.username  8.062e-01  7.522e-02  10.718  < 2e-16 ***
fullname.words       -3.354e-02  1.333e-02  -2.516 0.012142 *
nums.length.fullname -2.775e-02  1.212e-01  -0.229 0.818988
name..username        2.241e-01  7.641e-02   2.933 0.003498 **
description.length   -1.510e-03  4.342e-04  -3.478 0.000544 ***
external.URL         -1.542e-01  4.800e-02  -3.213 0.001390 **
private              -9.459e-03  2.843e-02  -0.333 0.739459
X.posts              -9.094e-05  3.570e-05  -2.547 0.011120 *
X.followers          -9.960e-09  1.539e-08  -0.647 0.517743
X.follows            -1.850e-05  1.499e-05  -1.235 0.217530
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3166 on 564 degrees of freedom
Multiple R-squared:  0.6074,    Adjusted R-squared:  0.5998
F-statistic: 79.33 on 11 and 564 DF,  p-value: < 2.2e-16
```

Vemos que tenemos varias variables con un p-value bastante alto. Estas variables añaden complejidad al modelo y no aportan nada, vamos a eliminarlas (creando un nuevo modelo sin ellas). Vamos a quedarnos con las variables que tienen un p-value menor a 0.01, es decir, profile.pic, nums.length.username, name..username, description.length y external.URL.

```r
modelo3 <- lm(fake ~ profile.pic + nums.length.username + name..username +  description.le
```

```r
summary(modelo3)
```

```
Call:
lm(formula = fake ~ profile.pic + nums.length.username + name..username +
    description.length + external.URL, data = dataset)

Residuals:
     Min       1Q   Median       3Q      Max
-0.71113 -0.24453 -0.07143  0.24282  0.98317

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)           0.7354681  0.0316443  23.242  < 2e-16 ***
profile.pic          -0.4596554  0.0328023 -14.013  < 2e-16 ***
nums.length.username  0.8445023  0.0687052  12.292  < 2e-16 ***
name..username        0.2314540  0.0733411   3.156 0.001685 **
```

```
description.length   -0.0017381  0.0004285  -4.056 5.68e-05 ***
external.URL         -0.1731966  0.0477099  -3.630 0.000309 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3196 on 570 degrees of freedom
Multiple R-squared:  0.5956,    Adjusted R-squared:  0.592
F-statistic: 167.9 on 5 and 570 DF,  p-value: < 2.2e-16
```

Vemos que el R2 ha bajado un poco pero es prácticamente el mismo, al igual que RSE, que solo ha incrementado un poco, y sin embargo, el fstatistic ha incremetnado bastante.

Hemos conseguido unos resultados prácticamente idénticos, mejorando el fstatistic, y reduciendo considerablemente la complejidad del modelo.

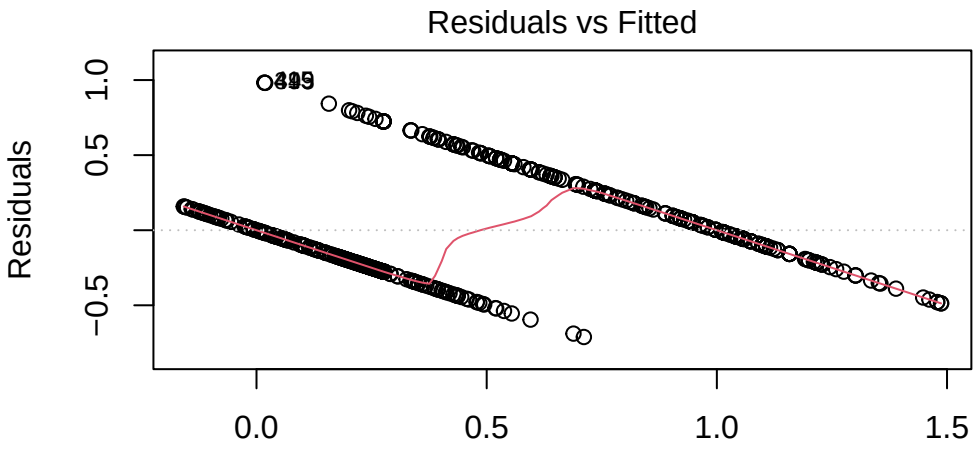
```
aciertos.binarized(predict(modelo3), fake)
```

```
[1] 89.93056
```

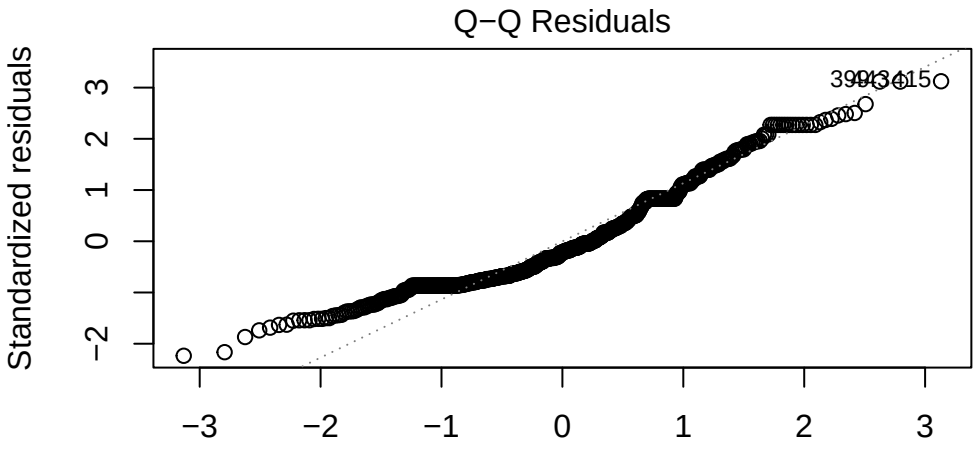
Los aciertos son exactamente los mismos. Todas las variables que henmos eliminado no aportaban prácticamente nada.

Veamos los gráficos que nos proporciona plot.

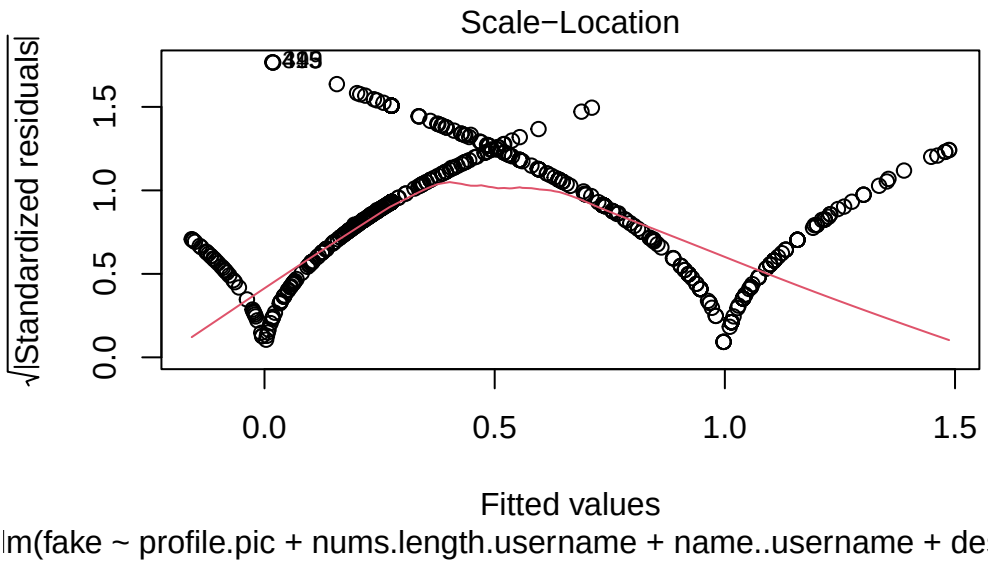
```
plot(modelo3)
```

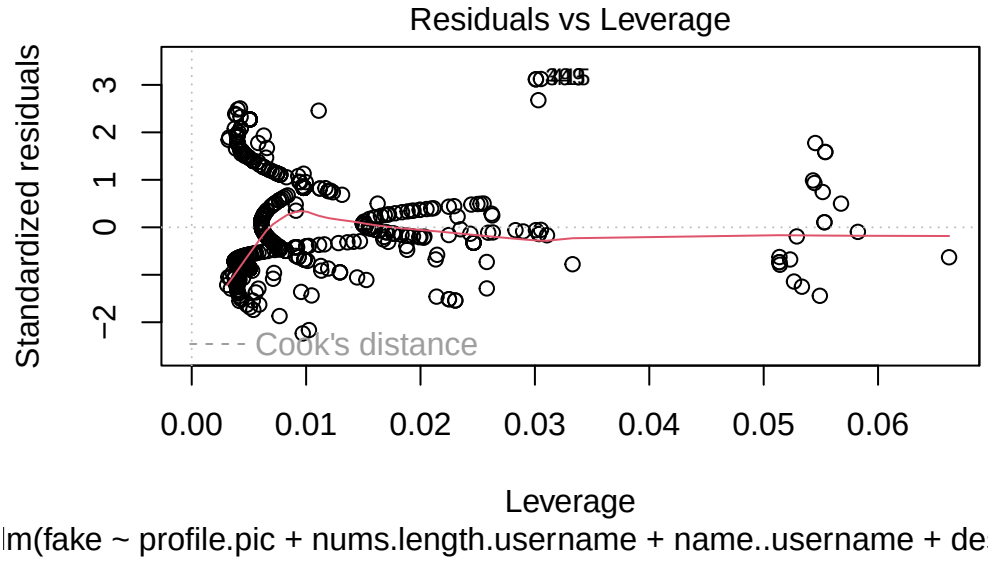


lm(fake ~ profile.pic + nums.length.username + name..username + descript



lm(fake ~ profile.pic + nums.length.username + name..username + descript





Al estar intentando predecir una variable que solo toma dos valores, los gráficos son bastante más distintos de lo normal. Podemos apreciar que hay outliers, aunque no demasiados como

cuando estabamos ajustando las variables nums.length.fullname y nums.length.username.

Ahora, vamos a intentar mejorar nuestro modelo añadiendo términos no lineales. Hay ciertas variables que quizá de la forma en que se nos presenta no aportan mucho a los datos, pero aplicandole alguna transformación, pueden aportar.

Con solo términos lineales estamos ajustando un hiperplano de a nuestra nube de puntos de n dimensiones, pero con términos no lineales formamos figuras mas complejas que un hiper plano.

Vamos a añadir más términos. Para la descripción vamos a añadir términos elevados a 2, 3 y 4. Para los followers, que antes nos salía que no era una variable relevante, vamos a hacerle el logaritmo, ya que los valores crecían exponencialmente (para poder hacerle el logaritmo, tenemos que tener cuidado, pues hay valores con 0, así que simplemente sumamos 1) y por último vamos a hacerle la raíz cuadrada a X.follows.

```
modelo4 <- update(modelo3, . ~ . +
                    I(description.length^2) +
                    I(description.length^3) +
                    I(description.length^4) +
                    I(log(X.followers+1)) +
                    I(log(X.followers+1)^2) +
                    I(log(X.followers+1)^3) +
                    I(log(X.followers+1)^4) +
                    I(log(X.followers+1)^2) +
                    I(X.follows^0.5))
```

Son términos "extraños" que nos hemos inventado, y como vamos a ver ahora, han provocado un ajuste bueno, pero hay que tener cuidado. Podríamos añadir todas las variables que queramos con todas las modificaciones que queramos, pero esto puede provocar un sobreajuste (overfitting).

El sobreajuste ocurre cuando entrenamos el modelo de forma que se adapta demasiado a los datos, sin tener ese grado de generalización que es lo que nos permite que dados nuevos datos, el modelo funcione correctamente.

De hecho, este último modelo, tiene cierto grado de sobreajuste, ya que hemos añadido variables que se adaptan demasiado a los datos.

Vamos a visualizar los estadísticos:

```
summary(modelo4)
```

```
Call:
```

```
lm(formula = fake ~ profile.pic + nums.length.username + name..username +
    description.length + external.URL + I(description.length^2) +
    I(description.length^3) + I(description.length^4) + I(log(X.followers +
    1)) + I(log(X.followers + 1)^2) + I(log(X.followers + 1)^3) +
    I(log(X.followers + 1)^4) + I(X.follows^0.5), data = dataset)

Residuals:
     Min       1Q   Median       3Q      Max
-0.70473 -0.18094 -0.02418  0.14988  0.82024

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                 8.893e-01  5.802e-02  15.327  < 2e-16 ***
profile.pic                -2.612e-01  3.073e-02  -8.498  < 2e-16 ***
nums.length.username        5.798e-01  6.141e-02   9.442  < 2e-16 ***
name..username              1.179e-01  6.220e-02   1.896  0.05847 .
description.length         -1.087e-02  3.774e-03  -2.879  0.00415 **
external.URL               -6.688e-02  4.343e-02  -1.540  0.12412
I(description.length^2)     2.864e-04  1.386e-04   2.067  0.03923 *
I(description.length^3)    -3.118e-06  1.616e-06  -1.929  0.05425 .
I(description.length^4)     1.150e-08  5.832e-09   1.972  0.04916 *
I(log(X.followers + 1))     1.934e-01  4.927e-02   3.925 9.73e-05 ***
I(log(X.followers + 1)^2) -9.358e-02  1.450e-02  -6.455 2.33e-10 ***
I(log(X.followers + 1)^3)  9.707e-03  1.493e-03   6.501 1.76e-10 ***
I(log(X.followers + 1)^4) -2.979e-04  4.920e-05  -6.054 2.59e-09 ***
I(X.follows^0.5)            4.795e-03  1.105e-03   4.341 1.68e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2678 on 562 degrees of freedom
Multiple R-squared:   0.72, Adjusted R-squared:  0.7136
F-statistic: 111.2 on 13 and 562 DF,  p-value: < 2.2e-16
```

Hemos obtenido un error residual de 0.26, un R^2 de 0.72 y un R^2 ajustado de 0.7136 (bastante cercano a 0.72, lo que indica que no tenemos mucha complejidad innecesaria, aunque). Vemos que el F-statistic si ha disminuido, a 111.2, aunque sigue siendo bastante alto.

Veamos el porcentaje de aciertos:

```
aciertos <- aciertos.binarized(predict(modelo4), fake)
paste0("Hemos obtenido un ", round(aciertos, 2), "% de aciertos")
```

```
[1] "Hemos obtenido un 93.92% de aciertos"
```

Casi un 94%, nada mal, pero recordemos que estos son los mismos datos con los que hemos creado el modelo.

Vamos a ver los estadísticos binarizados:

```
paste("RSE:", round(rse.binarized(modelo4, fake),4))
```

```
[1] "RSE: 0.2469"
```

```
paste("R2:", round(rsquared.binarized(modelo4, fake),4))
```

```
[1] "R2: 0.7569"
```

```
paste("F-statistic:", round(fstatistic.binarized(modelo4, fake),2))
```

```
[1] "F-statistic: 134.63"
```

Son incluso mejores, y eso que estos son los reales, ya que como explicamos, nos interesan los datos binarizados, no la predicción que es resultado directo de la regresión. Parece ser que hemos encontrado un buen modelo a pesar de que nuestra predicción es binaria.

Vamos a pasar a la última fase, probar nuestros modelos.

6.7 Modelos finales

A modo de resumen, estos son los 4 modelos con los que nos hemos quedado (que predicen la variable fake) y a los que les haremos pruebas:

- **modelo1**

Modelo de regresión lineal básico que simplemente usaba para predecir fake, la variable binaria profile.pic:

```
summary(modelo1)
```

```
Call:
lm(formula = fake ~ profile.pic, data = dataset)

Residuals:
     Min       1Q   Median       3Q      Max
-0.98837 -0.29208 -0.14023  0.01163  0.70792

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.98837    0.02943   33.58   <2e-16 ***
profile.pic -0.69629    0.03514  -19.81   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.386 on 574 degrees of freedom
Multiple R-squared:  0.4062,    Adjusted R-squared:  0.4051
F-statistic: 392.6 on 1 and 574 DF,  p-value: < 2.2e-16
```

Estadísticos con la binarización de los resultados:

```
paste("RSE:", round(rse.binarized(modelo1, fake),4))
```

```
[1] "RSE: 0.4572"
```

```
paste("R2:", round(rsquared.binarized(modelo1, fake),4))
```

```
[1] "R2: 0.1667"
```

```
paste("F-statistic:", round(fstatistic.binarized(modelo1, fake),2))
```

```
[1] "F-statistic: 114.8"
```

- **modelo2**

Modelo de regresión generalizado usando todas las variables del dataset, al que no se le ha hecho ningún estudio ni ninguna merjora:

```
summary(modelo2)
```

```
Call:
lm(formula = fake ~ ., data = dataset)

Residuals:
     Min       1Q   Median       3Q      Max
-0.73096 -0.23729 -0.06653  0.24048  1.01052

Coefficients:
                       Estimate Std. Error t value Pr(>|t|)
(Intercept)           7.931e-01  3.798e-02  20.880  < 2e-16 ***
profile.pic          -4.380e-01  3.345e-02 -13.094  < 2e-16 ***
nums.length.username  8.062e-01  7.522e-02  10.718  < 2e-16 ***
fullname.words       -3.354e-02  1.333e-02  -2.516 0.012142 *
nums.length.fullname -2.775e-02  1.212e-01  -0.229 0.818988
name..username        2.241e-01  7.641e-02   2.933 0.003498 **
description.length   -1.510e-03  4.342e-04  -3.478 0.000544 ***
external.URL         -1.542e-01  4.800e-02  -3.213 0.001390 **
private              -9.459e-03  2.843e-02  -0.333 0.739459
X.posts              -9.094e-05  3.570e-05  -2.547 0.011120 *
X.followers          -9.960e-09  1.539e-08  -0.647 0.517743
X.follows            -1.850e-05  1.499e-05  -1.235 0.217530
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3166 on 564 degrees of freedom
Multiple R-squared:  0.6074,    Adjusted R-squared:  0.5998
F-statistic: 79.33 on 11 and 564 DF,  p-value: < 2.2e-16
```

Estadísticos con la binarización de los resultados:

```
paste("RSE:", round(rse.binarized(modelo2, fake),4))
```

```
[1] "RSE: 0.3179"
```

```
paste("R2:", round(rsquared.binarized(modelo2, fake),4))
```

```
[1] "R2: 0.5972"
```

```
paste("F-statistic:", round(fstatistic.binarized(modelo2, fake),2))
```

```
[1] "F-statistic: 76.03"
```

- **modelo3**

Es el modelo2 pero eliminando todas las variables que solo aportaban complejidad al modelo y prácticamente nada de información:

```
summary(modelo3)
```

```
Call:
lm(formula = fake ~ profile.pic + nums.length.username + name..username +
    description.length + external.URL, data = dataset)

Residuals:
     Min       1Q   Median       3Q      Max
-0.71113 -0.24453 -0.07143  0.24282  0.98317

Coefficients:
                       Estimate Std. Error t value Pr(>|t|)
(Intercept)           0.7354681  0.0316443  23.242  < 2e-16 ***
profile.pic          -0.4596554  0.0328023 -14.013  < 2e-16 ***
nums.length.username  0.8445023  0.0687052  12.292  < 2e-16 ***
name..username        0.2314540  0.0733411   3.156 0.001685 **
description.length   -0.0017381  0.0004285  -4.056 5.68e-05 ***
external.URL         -0.1731966  0.0477099  -3.630 0.000309 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3196 on 570 degrees of freedom
Multiple R-squared:  0.5956,    Adjusted R-squared:  0.592
F-statistic: 167.9 on 5 and 570 DF,  p-value: < 2.2e-16
```

Estadísticos con la binarización de los resultados:

```
paste("RSE:", round(rse.binarized(modelo3, fake),4))
```

```
[1] "RSE: 0.3179"
```

```
  paste("R2:", round(rsquared.binarized(modelo3, fake),4))
```

```
[1] "R2: 0.5972"
```

```
  paste("F-statistic:", round(fstatistic.binarized(modelo3, fake),2))
```

```
[1] "F-statistic: 169.03"
```

- **modelo4**

Modelo de regresión generalizado no lineal. Resultado de añadir varaibles con logaritmos, exponentes y raices al modelo3.

```
  summary(modelo1)
```

```
Call:
lm(formula = fake ~ profile.pic, data = dataset)

Residuals:
     Min       1Q   Median       3Q      Max
-0.98837 -0.29208 -0.14023  0.01163  0.70792

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.98837    0.02943   33.58   <2e-16 ***
profile.pic -0.69629    0.03514  -19.81   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.386 on 574 degrees of freedom
Multiple R-squared:  0.4062,    Adjusted R-squared:  0.4051
F-statistic: 392.6 on 1 and 574 DF,  p-value: < 2.2e-16
```

Estadísticos con la binarización de los resultados:

```
  paste("RSE:", round(rse.binarized(modelo4, fake),4))
```

```
[1] "RSE: 0.2469"
```

```
paste("R2:", round(rsquared.binarized(modelo4, fake),4))
```

```
[1] "R2: 0.7569"
```

```
paste("F-statistic:", round(fstatistic.binarized(modelo4, fake),2))
```

```
[1] "F-statistic: 134.63"
```

6.8 Pruebas

Hasta ahora, los porcentajes de aciertos que hemos calculado, son sobre nuestro dataset de entrenamiento, es decir, usamos unos datos para crear un modelo y luego lo probamos con los mismos datos. Esto no es realista, necesitamos otros datos para probar realmente nuestro modelo.

Del sitio web de Kaggle, además del dataset "train.csv", se encuentra "test.csv". Vamos a utilizarlo para probar nuestros modelos, ya que probarlo con el mismo train.csv no es real, pues es la misma información que hemos usado para construir los modelos.

Importamos el dataset

```
dataset_test <- read.csv("datasets/test.csv")
```

Este dataset contiene las mismas variables que train.csv:

```
colnames(dataset_test)
```

```
 [1] "profile.pic"          "nums.length.username" "fullname.words"
 [4] "nums.length.fullname" "name..username"       "description.length"
 [7] "external.URL"         "private"              "X.posts"
[10] "X.followers"          "X.follows"            "fake"
```

```
nrow(dataset_test)
```

```
[1] 120
```

Tenemos 120 observaciones, la mitad clasificadas como falsas y la otra mitad como reales. Son observaciones distintas a las del dataset train.csv.

Vamos a calcular el porcentaje de aciertos en el dataset de pruebas usando cada modelo:

- **modelo1**

```
predicciones <- predict(modelo1, dataset_test)

aciertos_test <- aciertos.binarized(predicciones, dataset_test$fake)
aciertos_train <- aciertos.binarized(predict(modelo1, dataset), dataset$fake)

paste0("El modelo1 ha obtenido un ", round(aciertos_test, 2),
       "% de aciertos en el dataset de pruebas")
```

```
[1] "El modelo1 ha obtenido un 74.17% de aciertos en el dataset de pruebas"
```

```
[1] "Mientras que en el dataset de entrenamiento obtuvo un 79.17% de aciertos"
```

Este modelo depende totalmente de la foto de perfil. Es demasiado simple y como vemos con el test de pruebas ha perdido un 5% de aciertos. No es un buen modelo por su simplicidad y dependencia total en una única variable.

- **modelo2**

```
predicciones <- predict(modelo2, dataset_test)

aciertos_test <- aciertos.binarized(predicciones, dataset_test$fake)
aciertos_train <- aciertos.binarized(predict(modelo2, dataset), dataset$fake)

paste0("El modelo2 ha obtenido un ", round(aciertos_test, 2),
       "% de aciertos en el dataset de pruebas")
```

```
[1] "El modelo2 ha obtenido un 89.17% de aciertos en el dataset de pruebas"
```

```
[1] "Mientras que en el dataset de entrenamiento obtuvo un 89.93% de aciertos"
```

Resultados casi iguales. No es un mal modelo pero es mejorable, como vemos con el modelo3, ya que podemos simplificar su complejidad.

- **modelo3**

```r
predicciones <- predict(modelo3, dataset_test)

aciertos_test <- aciertos.binarized(predicciones, dataset_test$fake)
aciertos_train <- aciertos.binarized(predict(modelo3, dataset), dataset$fake)

paste0("El modelo3 ha obtenido un ", round(aciertos_test, 2),
"% de aciertos en el dataset de pruebas")
```

```
[1] "El modelo3 ha obtenido un 89.17% de aciertos en el dataset de pruebas"
```

```
[1] "Mientras que en el dataset de entrenamiento obtuvo un 89.93% de aciertos"
```

Exactamente los mismos resultados que con el modelo2, lo que nos indica que la simplificación que hemos hecho respecto al modelo2 es muy buena.

- **modelo4**

```r
predicciones <- predict(modelo4, dataset_test)

aciertos_test <- aciertos.binarized(predicciones, dataset_test$fake)
aciertos_train <- aciertos.binarized(predict(modelo4, dataset), dataset$fake)

paste0("El modelo4 ha obtenido un ", round(aciertos_test, 2),
       "% de aciertos en el dataset de pruebas")
```

```
[1] "El modelo4 ha obtenido un 93.33% de aciertos en el dataset de pruebas"
```

```
[1] "Mientras que en el dataset de entrenamiento obtuvo un 93.92% de aciertos"
```

Unos resultados muy buenos. Vemos que no tenemos mucho sobreajuste a pesar de que añadimos bastantes variables con exponentes y logaritmos. Es un buen modelo y el mejor de todos los que hemos hecho.

6.9 Conclusiones

Hemos explorado el la regresión como técnica estadística para modelar y predecir la veracidad o falsedad de cuentas de Instagram. A lo largo del análisis, hemos hecho múltiples regresiones.

Aunque la variable fake es binaria, con la regresión hemos podido aproximar su valor con cierto grado de precisión.

Veíamos como el primer modelo, aunque tenía un prácticamente un 80% de aciertos, era demasiado simple y no podíamos guiarnos por algo tan simple. Luego hicimos modelos más complejos y los simplificamos, hasta llegar a uno que nos daba unos valores estadísticos relativamente buenos.

El modelo4 ha resultado ser el mejor al incorporar términos no lineales y transformaciones logarítmicas y cuadráticas, con casi un 94% de aciertos en el dataset de entrenamiento. Aún así, este modelo mostró ciertos signos de sobreajuste (al bajar su f-statistic respecto al modelo3 mientras subía su R^2). Esto destaca la importancia de equilibrar la complejidad del modelo con su capacidad de generalización.

Existen otras técnicas de regresión más complejas, y otros métodos de realizar el entrenamiento, como la validación cruzada, pero en este apartado, nos hemos centrado en intentar conseguir el mejor modelo a partir de lo visto en clase.

7 Series temporales

El análisis de series temporales es una técnica poderosa que se utiliza para comprender y prever comportamientos y tendencias a lo largo del tiempo. Este tipo de análisis es particularmente valioso en contextos donde los datos se recogen en intervalos regulares y presentan patrones que pueden ser aprovechados para hacer predicciones precisas. Las series temporales permiten identificar componentes clave como tendencias a largo plazo, estacionalidades periódicas y fluctuaciones cíclicas, ofreciendo una visión profunda del comportamiento dinámico de los datos.

```
library(ggplot2)
library(forecast)
```

```
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
```

Las técnicas de análisis de series temporales incluyen modelos como por ejemplo ARIMA (Autoregressive Integrated Moving Average) que hemos visto en clase, que es utilizado para modelar y prever series estacionarias y no estacionarias.

```
dataset <- read.csv("datasets/train.csv")
```

Sin embargo, en este libro, al analizar un dataset de cuentas de Instagram, nos encontramos con un problema: no tenemos atributos temporales. Nuestro dataset no contiene marcas de tiempo ni ninguna variable que refleje la evolución cronológica de las métricas. Esto impide la aplicación directa de técnicas de series temporales, ya que no hay un componente temporal que permita rastrear cambios a lo largo del tiempo. Tampoco tenemos ningún tipo de orden en nuestros datos de forma que cada observación se pueda asignar temporalmente ni ordenar de ninguna forma.

Aún así, Vamos a intentar aplicar esta técnica a alguna variable de nuestro dataset, a pesar de que no tenga carácter temporal:

7.1 Serie temporal con description.length

Vamos a usar esta variable para crear nuestras series temporales, como hemos comentado, los datos no tienen ningún orden ni tenemos otras variables que representen tiempo, pero vamos a intentar aplicar algunas cosas de lo aprendido a esta variable para que veamos si se puede sacar algo de informcion.

Creamos la serie temporal, imaginando que empieza en el año 2000 y cada dato representa un mes del año:

```
serie <- ts(dataset$description.length, frequency = 12, start = 2000)
serie
```

```
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2000  53  44   0  82   0  81  50   0  71  40  54  54
2001   0 103  98  46   0  48  63 106  40  35  30  27
2002   0 109   0 132 126 122 138   0  50  35  56   9
2003   0  81 134   0   2   0  23 138  35  93   4   1
2004   4  23  91  57 108  30  82  12  54   0  12   0
2005   3  39   0  68 129  57  64  42  71   0  70  74
2006   8  35   0   0  28  18  28  36   2  11  70  29
2007  24  21  81  34  40  12   0  59  15  54  16  73
2008  24   0  26   0   0   0   0  28  55 140 122 113
2009  38   0  23   0  89  30   0   0   0  12 123   0
2010   0   0  40   0  33   0   5   0  23  35 150  26
2011 149 129   0  18  74   0  59 148   0  15  46   5
2012  98  55  19  71 133 150  43  37  35  87  59   0
2013   0   9  12   0   0  95   0  46 123 117  26   0
2014  58   0  30  62 137 149  14  19 131   0   5   0
2015  11   0  27  10  72   3  51  44   0  73  70  35
2016  13 105  91   0  48  48 126   0  53   8  67  20
2017  26  86  51  26  18  96  17   0  62  86 148   1
2018  39  35 103   0  61  44   0   0   0 112 123  24
2019  34  19   0  42  50  67 134 101   0   0  17   0
2020  32   0  80   2   0 146   0   0   0   0   6   0
2021   0   0  64   0   0   0  49  23 120  34  25   0
2022  12   0   9   1  18  34  23  19 139  13  50  46
2023  30  26   0   0   0  27  37  31  20   7   0   0
2024   0   0   0   0  24   0  43   0   0   0   0   0
2025   0   0   0   0   0   0   0   0   0   0   0   0
2026   0   0   0   0   0   0  43   0   0   0  13   0
2027   0   0   0   0   0   0   0   0   0   0   0   0
```

```
2028   0   0   0   0   0   0   9   0   0   0   0  18
2029   0  10  61   0   0   0   0   0   0   0   0   0
2030  22   0   0   0   2   0 146   0   6  50   0   0
2031   0  39   0   0   0   0   0   5  91   2   0   0
2032   0   0  37   0   0   0   0   0   0   0   0   0
2033   0   0 148   0   0   0   0   0   0   0   0   0
2034   0   0   0   0   0   0 149   0   0   0   0   0
2035  22   0   0   0   0   0   2   0   0   0   0   0
2036   0  20   0   0   0   0   0   0   0   0 148   0
2037  50   0   2   0   0   0   0   0   0   0  34   0
2038   0   0   0   0   0   0  32   0   0   0   0   0
2039   0   0  59   0   6   0   0   0   0   0   0   0
2040   0   0   0   0   0   0   0   0   0   0   0   0
2041   0   1   0   0   0   0   0   0   0   0   0   0
2042   0   0   0   0   0   0   0   0   0   0   0   0
2043   0   0   0   0   0   0   0   0   0   0   0   0
2044   0   0   0   0   0   0   0  19   0   0   0   0
2045   0   0   0   0  43   0   0   0   0   0   0   0
2046   0   0   0   0   0   0   0  33   0   0   0   0
2047  43   0  19   0   5   0  28   0  21   0  11   0
```

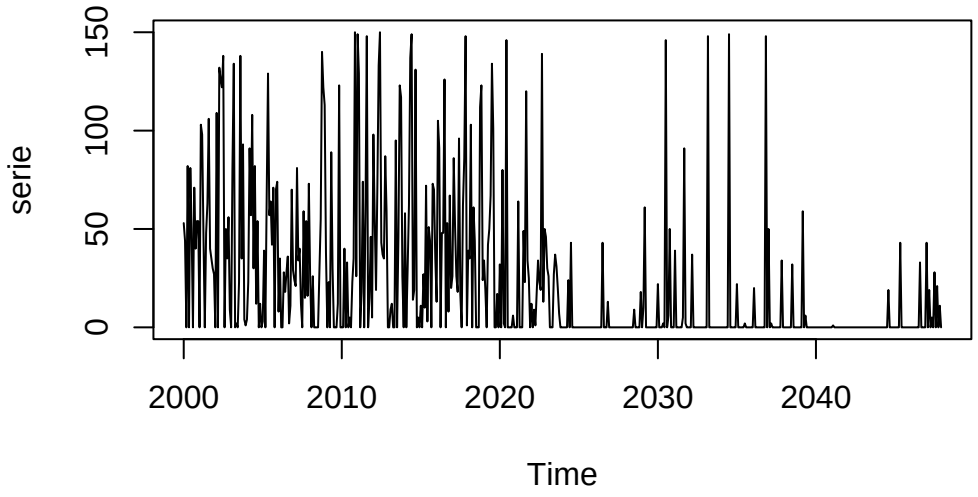
```
length(serie)
```

```
[1] 576
```

Como tenemos 576 datos y hemos simulado que cada uno corresponde a un mes del año, empezando en el año 2000 nos da por resultado que nuestros datos llegan hasta el año 2047.

Vamos a ver nuestra serie dibujado:

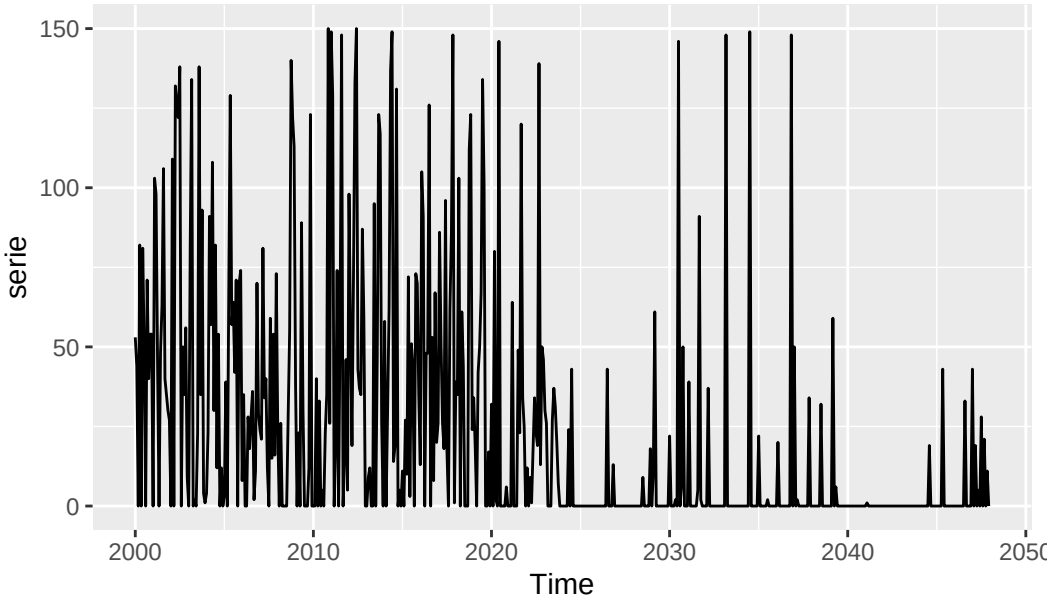
```
plot.ts(serie)
```



Como podemos ver es bastante aleatorio, ya que realmente lo es, se trata de las longitudes de descripción de distintas cuentas de Instagram sin ningún orden ni nada de carácter temporal.

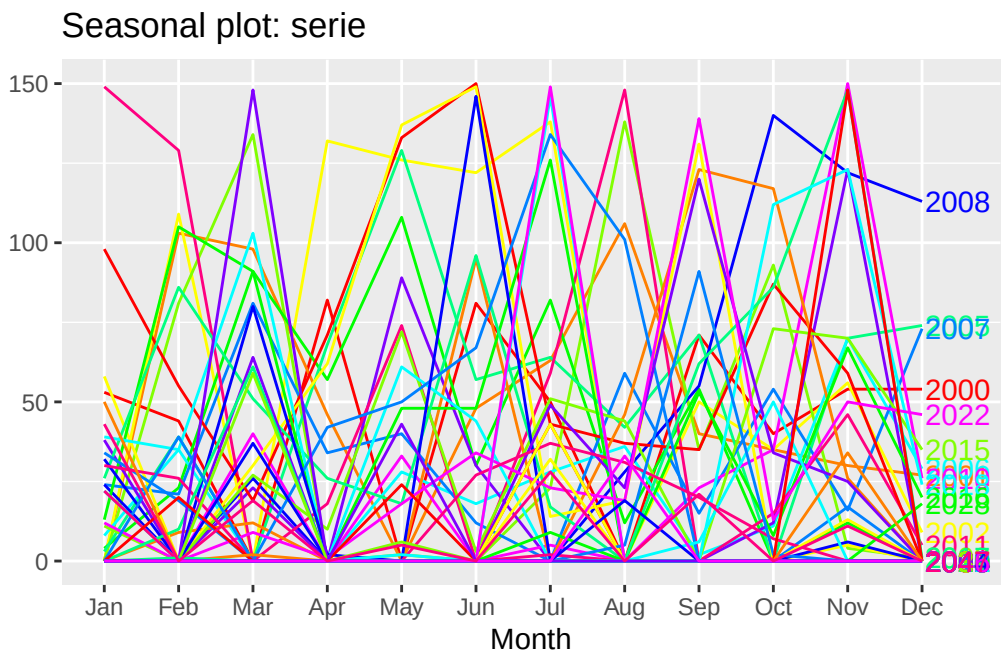
Con autoplot podemos visualizar el mismo gráfico pero con un estilo similar a la filosofia que sigue ggplot2:

```
autoplot(serie)
```



Veamos el gráfico de seasonplot, el cual nos permite observar si nuestra serie temporal es estacional. Este gráfico muestra si cada año se repiten patrones específicos, ayudándonos a identificar y visualizar la estacionalidad en la serie temporal.

```
ggseasonplot(serie, col = rainbow(12), year.labels = TRUE)
```



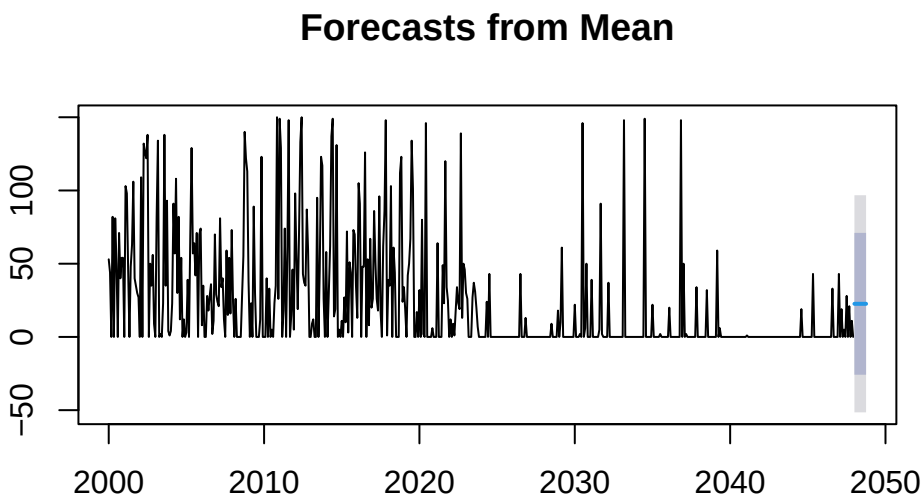
Claramente no muestra signos de estacionalidad, pues es prácticamente aleatorio, no hay ningún tipo de patrón.

7.2 Previsiones

Vamos a probar a realizar previsiones con los métodos básicos que hemos visto. Estos métodos de por si son demasiado sencillos, y para el caso de nuestra serie que venimos diciendo que no es adecuada, sabemos que no va a dar ningún resultado bueno, pero vamos a aplicarlos para que lo veamos. Luego usaremos algunos métodos más complejos:

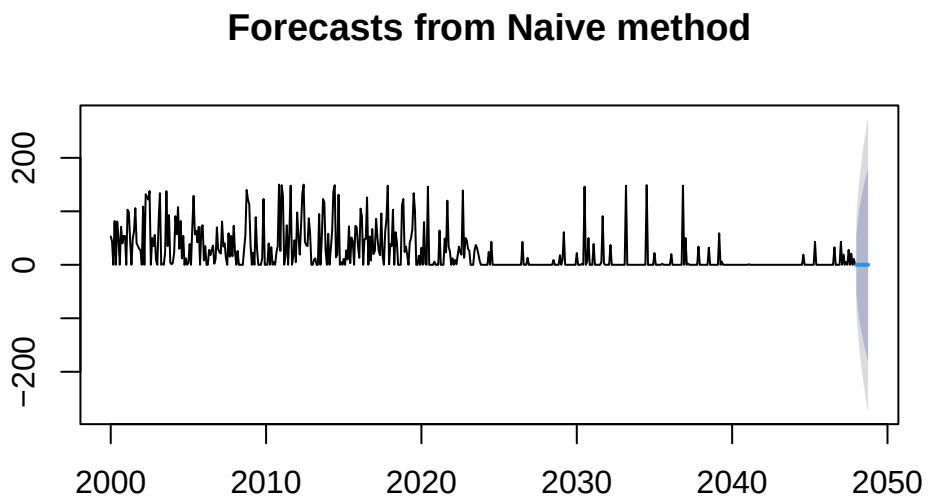
7.2.1 Método de la media

```
avg.serie <- meanf(serie, 10)
plot(avg.serie)
```



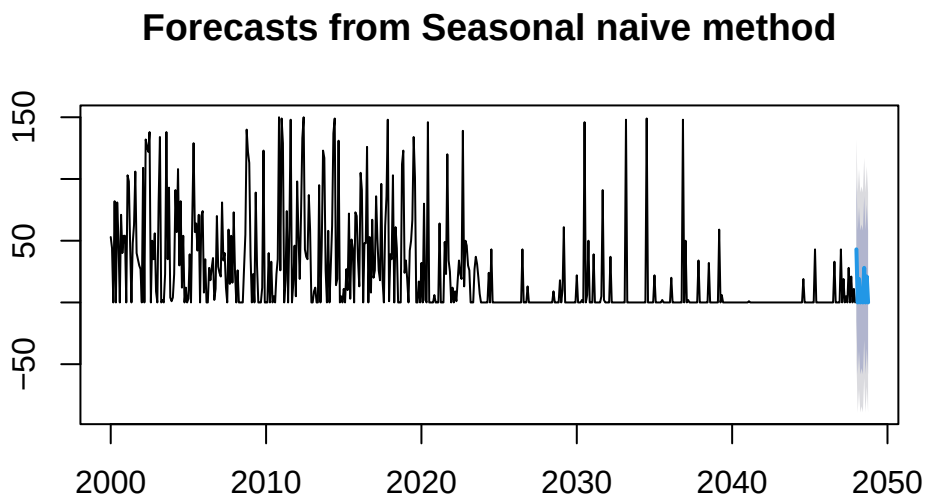
7.2.2 Método *naive*

```
naive.serie <- naive(serie, 10)
plot(naive.serie)
```



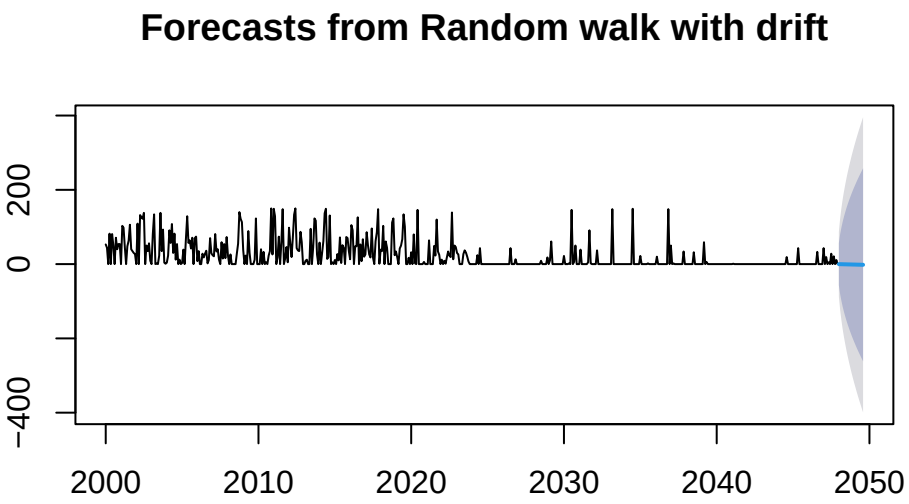
7.2.3 Método *naive* estacional

```
snaive.serie <- snaive(serie, 10)
plot(snaive.serie)
```



7.2.4 Método *drift*

```
rwf.serie <- rwf(serie, 20, drift = TRUE)
plot(rwf.serie)
```



Como hemos comentado, ningún método da una previsión que tenga coherencia, tan solo el método *naive* estacional, aunque ya sabemos que este solo replica la predicción del año anterior.

7.3 Descomposición

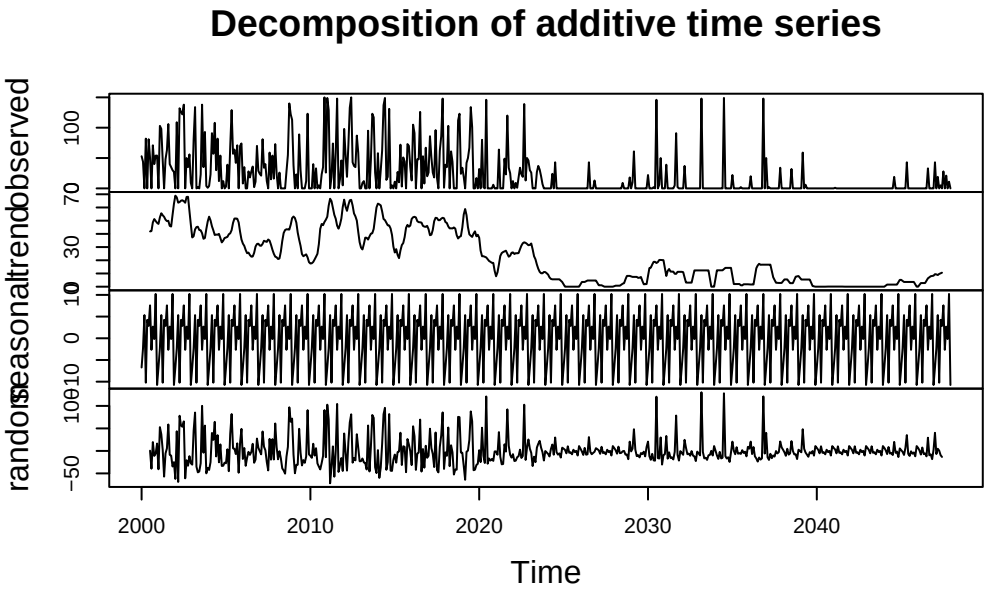
Una serie temporal se puede descomponer en tres componentes: estacional, tendencia y la componente de aleatoriedad.

Vamos a aplicar la descomposición a nuestra serie:

```
decomp <- decompose(serie)
```

Vamos a visualizarlo:

```
plot(decomp)
```



Si nuestra serie fuese realmente una serie válida, estaríamos observando datos interesantes, por ejemplo, podríamos decir que la tendencia global de la longitud de la descripción ha sido bajista, y que tenemos una componente estacionaria en la que hay ciertos momentos del año que hay un pico alto y otros muy bajos.

Pero como el tiempo aquí "nos lo hemos inventando" sacar esas conclusiones es también inventarnoslo.

7.4 Previsiones complejas

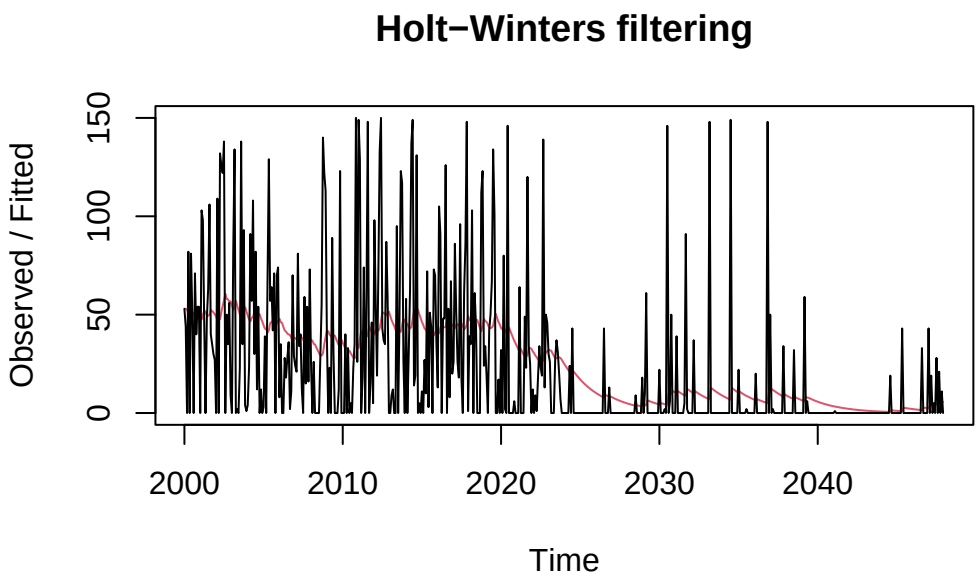
Por útimo, vamos a utilizar métodos más avanzados de previsiones, simplemente para visualizar los resultados.

7.4.1 HoltWinters

El método Holt-Winters es una técnica de suavizamiento exponencial que permite modelar y prever series temporales con tendencia y estacionalidad. Este método tiene tres componentes: el nivel (alpha), la tendencia (beta) y la estacionalidad (gamma). Es bueno usarlo en series temporales con tendencia y estacionalidad o con patrones estacionalides conocidos.

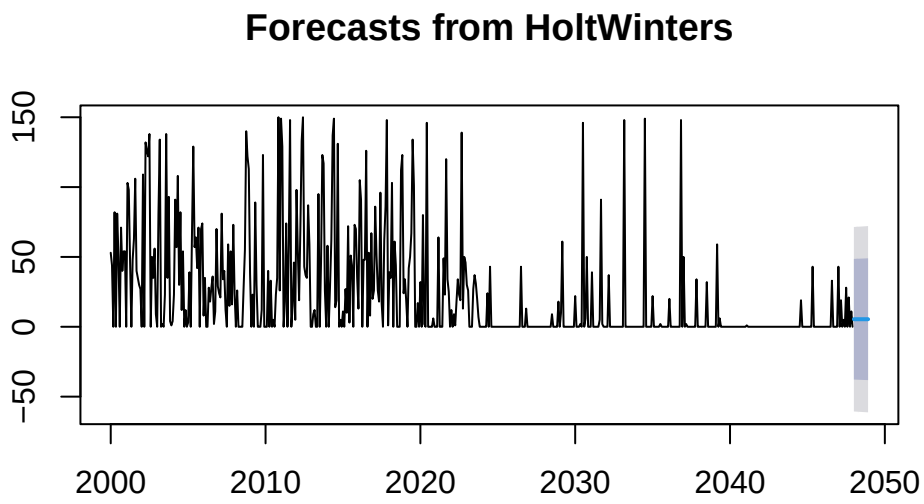
Vamos a aplicarlo a nuestra serie temporal:

```
ts.p1.forecasts <- HoltWinters(serie, beta = FALSE, gamma = FALSE)
plot(ts.p1.forecasts)
```



Visualicemos una predicción de un año:

```
holtwinters_forecast <- forecast(ts.p1.forecasts, h = 12)
plot(holtwinters_forecast)
```

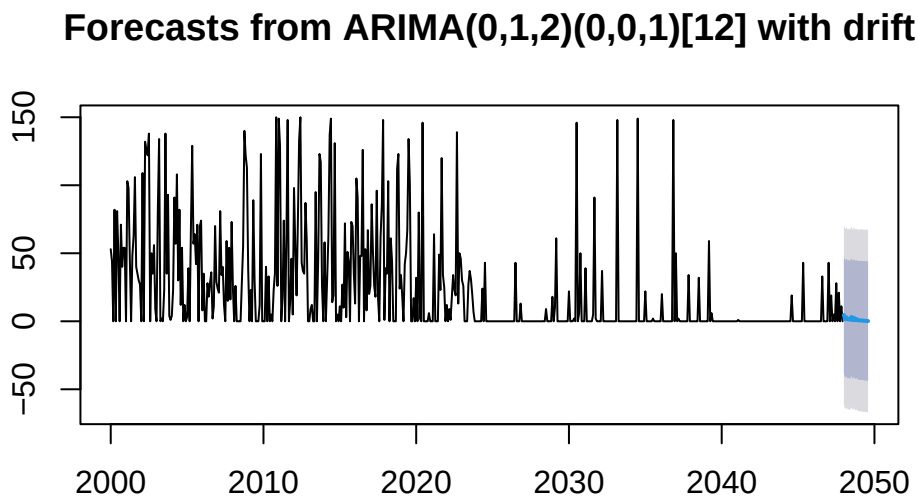


7.4.2 ARIMA y auto.arima

ARIMA (AutoRegressive Integrated Moving Average) es un modelo popular en el análisis de series temporales, utilizado para modelar y prever series que pueden no ser estacionarias. El modelo ARIMA se define por tres parámetros: p (autoregresivo), d (diferenciación) y q (media móvil). El método auto.arima selecciona automáticamente los mejores parámetros para el modelo ARIMA. Es bueno usar ARIMA para series temporales estacionarias.

Aplicamos el modelo auto.arima a nuestra serie temporal:

```
autoarima_model <- auto.arima(serie)
plot(forecast(autoarima_model, h = 20))
```



Queda más que claro que lo que estamos haciendo no conduce a resultados con sentido, hay muchas más cosas que podemos hacer con series temporales, pero sabiendo que no nos conducen a nada, vamos a parar aquí.

7.5 Conclusiones

En este análisis, hemos aplicado varias técnicas de previsión de series temporales, como Holt-Winters y ARIMA, a una variable que en realidad no tiene un carácter temporal. Aunque hemos visto cómo se pueden usar estas técnicas, los resultados no tienen sentido práctico en este caso, ya que la serie de datos es aleatoria y no presenta patrones temporales reales.

Sin embargo, es importante comprender y saber aplicar estas técnicas, ya que son herramientas muy valiosas en el análisis de datos cuando se dispone de series temporales con componentes significativos de tendencia y estacionalidad.

8 Otras técnicas

En este punto hemos realizado: - Extracción información a través de un Análisis Exploratorio de los datos. - Visualización de la distribución y relaciones de distintas variables. - Generación de reglas de asociación con las que hemos conseguido implicaciones y relaciones interesantes. - Generación de conceptos e implicaciones mediante FCA. - Creación de un modelo de regresión que es capaz de predecir con un buen porcentaje de aciertos si una cuenta es falsa o real.

Además, hemos visto como la aplicación de las series temporales no es adecuada para nuestro caso de uso. No tenemos datos de con carácter temporal, ni podemos conseguir nada temporal.

Igual que series temporales, hay más técnicas que hemos visto en la asignatura, que no hemos añadido como apartados de estudio tales como los demás. En este apartado, vamos a explorar estas técnicas y vamos a indigar en las mismas:

8.1 Análisis de redes sociales

Ciertos datos tienen relaciones entre sí, como por ejemplo, los aeropuertos y los vuelos que se pueden realizar entre ellos en determinado país, ciudad, o en el mundo en general. Estos datos se pueden representar mediante grafos, donde los vértices serían los aeropuertos y los arcos, los vuelos.

Con esta representación de datos, se pueden analizar de forma eficiente las relaciones y estructuras subyacentes. Por ejemplo, podemos identificar los aeropuertos más centrales utilizando medidas como el grado de centralidad, que nos indica cuántos vuelos salen o llegan a un aeropuerto en particular. Además, podemos aplicar algoritmos para detectar comunidades dentro de la red, revelando agrupaciones de aeropuertos que tienen más conexiones entre sí que con otros grupos.

Los grafos abren un nuevo mundo de posibilidades y de algoritmos, es un área de estudio que se encuentra en la mayoría de cosas que hacemos cotidianamente.

Lamentablemente, nuestros datos carecen de relaciones, no tenemos ninguna relación entre cada par de observaciones de nuestro dataset, por lo que no podemos modelar de ninguna forma nuestros datos como un grafo, y por tanto no podemos aplicar esta técnica.

Ironicamente el análisis de redes sociales no se puede aplicar a nuestro dataset de Instagram.

8.2 Análisis de componentes principales

El análisis de componentes principales es una técnica de reducción de dimensionalidad que se utiliza para transformar un conjunto de variables posiblemente correlacionadas en un conjunto más pequeño de variables no correlacionadas llamadas componentes principales. Estos componentes capturan la mayor parte de la variabilidad presente en los datos originales, permitiendo simplificar la estructura del dataset mientras se conserva la información esencial.

En nuestro caso, con nuestro dataset de cuentas de Instagram, podríamos considerar aplicar PCA para intentar reducir las 12 variables disponibles a un número menor de componentes que retengan la mayor parte de la información. El problema es que estas 12 variables representan aspectos distintos y son, en general, independientes entre sí. Esto significa que no hay correlación significativa entre todas las variables (aunque si algunas como ya hemos visto), y cada una aporta información única.

Debido a esta independencia, aplicar PCA no resultaría demasiado beneficioso. No lograríamos una reducción sustancial de la dimensionalidad sin perder información importante. Además, la interpretación de los componentes principales sería compleja y poco intuitiva, ya que no habría una estructura subyacente clara que agrupe la varianza en unos pocos componentes. Por tanto, aunque el PCA es una herramienta poderosa en muchos contextos, en este caso específico con solo 12 variables independientes, su aplicación no proporcionaría ventajas significativas en términos de simplificación o comprensión de los datos.

9 Predictor interactivo

Con la regresión, obtuvimos un buen modelo capaz de predecir si una cuenta de Instagram es falsa o no. Nuestro **modelo5** daba muy buenos resultados.

Dado que es un modelo de regresión, lo que estamos haciendo es aplicar una fórmula matemática a nuestros datos, que nos da un resultado númerico. Este luego lo estamos binarizando y en función de si es 0 o 1 sabemos si la cuenta es real o falsa.

Podemos implementar fácilmente dicho modelo usando JavaScript, lo que nos permite incrustar en este apartado de nuestro libro de Quarto un formulario interactivo que nos permita predecir si una cuenta es real o falsa.

- **¿Por qué hacerlo con HTML y JavaScript si tenemos R Shiny?**

R Shiny necesita de un servidor para poder estar ejecutando el código R. Un navegador no ejecuta código R. Como no podemos ejecutar código R, podemos construir una página e implementar la formula de nuestro modelo. Eso es justamente lo que hemos hecho.

En un principio comenzamos a desarrollar este apartado de forma que solo introduciendo el nombre de usuario de Instagram de la persona, te realizaba la predicción, pero lamentablemente, la API para obtener datos de un perfil de usuario de Instagram es limitada actualmente.

A continuación, puedes introducir los datos references a la cuenta de Instagram que desees, y al pulsar el botón **Predecir cuenta** obtendrás una predicción de si la cuenta que has introducido es real o falsa.

10 Resumen

En este libro, hemos abordado el problema de la detección de cuentas falsas de Instagram, mediante las distintas técnicas de análisis de datos que hemos estudiado en la asignatura.

Mediante análisis exploratorio y visualización de datos hemos comprendido como se distribuyen y correlacionan nuestros datos, hemos podido sacar conceptos e implicaciones mediante FCA y las reglas de asociación, y hemos logrado crear un modelo de regresión capaz de predecir si una cuenta es falsa o real con una tasa de aciertos elevada.

Hemos visto otras técnicas que podrían ser interesantes pero que no son aplicables o no nos darían demasiada información para nuestro caso, pero que para trabajos de este tipo con datos algo distintos podrían ser cruciales.

Por último, hemos creado un predictor interactivo que aplica el modelo de regresión que hemos creado a los datos de cualquier cuenta, donde el lector puede introducir los que desee y evaluar la predicción.

10.1 Conclusión final

Con este trabajo, hemos visto como es realmente aplicable esa teoría y práctica que vemos en la asignatura, que es más que simplemente matemáticas que damos por que tenemos que darlo, si no que tienen un uso real. El tema en cuestión de este año ha sido bastante interesante y lograr ver un uso directo y práctico de lo aprendido me parece increíble.

Personalmente, este año he aprendido muchas cosas en la carrera que antes de comenzar el año, si me preguntaban, diría que es imposible que yo hiciese eso, y esta es una de ellas.

Si me llegan a decir en enero, ¿serías capaz si te doy unos datos de cuentas de Instagram y si son falsas o no, crear un código que si le metes una cuetna te diga si es falsa o no? Sinceramente diría que eso ni si quiera se ve en mi carrera, y ahora, en mayo, he conseguido hacerlo simplemente con lo aprendido en la asignatura, estoy muy orgulloso de este trabajo y me ha parecido muy interesante e ideal para esta asignatura.