PR2

Lucía Absalom Bautista

2024-12-29

Descripción del dataset.

El dataset que tenemos contiene datos numéricos y categóricos extraídos de la plataforma airbnb. A continuación haremos una pequeña descripción de las variables que tenemos.

Variables objetivos que planteamos:

- El **título** es una variable interesante pues nos puede aportar el tipo de alojamiento en el que nos encontramos. Si en el nombre pone apartamento, dormitorio, podemos crear una nueva variable que sea tipo de alojamiento y podríamos por ejemplo hacer un método de aprendizzaje supervisado o de clusteriazación que a partir de las demás variables que tengamos pudiera predecir si el alojamiento se trata de un apartamento o una habitación.
- Con el **precio** podemos hacer bastantes estudios, es una variable muy interesante aunque habría que quitarle el símbolo del euro.
- **rating**: es una variable también interesante pues podríamos ver la correlación con el tipo de alojamiento y la correlación con el precio. Junto con rating viene el número de votos y podríamos ajustar el rating como nota a una ponderación entre la nota y el número de reviews.
- **amenities**: es una variable interesante a partir de la cuál podríamos crear nuevas variables agrupándolas, por ejemplo, si el alojamiento cuenta con espacio exterior, si el alojamiento cuenta con vistas, si cuenta con utensilios de aseo, etc.
- **número de registro**: variable muy interesante para dividirla en dos; por una parte el tipo de anfitrión, particular o profesional y por otra parte si el alojamiento tiene licencia.
- **reviews**: sería muy interesantes para analizar usando llms, si tenemos tiempo las usaremos
- **Dormitorio1** y **Dormitorio2** son variables vacías en gran parte de los casos, las tendremos que eliminar
- **location**: variable ficticia relacionada con el emplazamiento

Carga de datos

Comenzamos cargando las diversas librerías que nos harán falta en el estudio y leyendo el csv con la función *read_csv* de la librería *readr*. Nos hemos dado cuenta de que las tildes son un problema en este dataframe, por tanto hemos usado la codificación Latin1 para luego eliminar los acentos con la función *stri_trans_general* de la librería *stringi*.

Resument del dataset

```
glimpse(data)
```

```
## Rows: 107
## Columns: 8
## $ title           <chr> "Hola Hostal Collblanc. Cama en dormitorio 18 pax", "A~
## $ price           <chr> "35 \u0080", "57 \u0080", "75 \u0080", "47 \u0080", "6~
## $ rating          <chr> "4,41 (1124)", "4,96 (147)", "4,77 (1241)", "4,94 (16)~
```

```
## $ amenities       <chr> "['Secador de pelo', 'Lavadora', 'Secadora', 'Servicio~
## $ numero_registro <chr> "['Anfitrión profesional', 'Número de registro: AJ0006~
## $ reviews         <chr> "[{'rating': 'Valoración: 5\\xa0estrellas', 'date': 'V~
## $ `Dormitorio 1`  <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ `Dormitorio 2`  <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
```

```
summary(data)
```

```
##     title              price              rating           amenities
##  Length:107         Length:107         Length:107         Length:107
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##  numero_registro      reviews          Dormitorio 1       Dormitorio 2
##  Length:107         Length:107         Length:107         Length:107
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
```

Vemos variables con información muy interesante y variables cuyo valores es NA casi siempre. A continuación decidiremos qué variables usamos para nuestro estudio, qué variables desechamos y qué variables vamos a modificar.

2. Integración y selección de datos

Seleccionamos las variables relevantes para nuestro análisis.

Comenzamos haciendo la division de rating en dos columnas.

- Una columna llamada rating que sea el valor del rating medio
- Una columna llamada n_rating que sea el número de ratings que ha tenido el alojamiento.

```
df_clean = data %>%
  # Separate rating into two columns, filling missing pieces with NA
  separate(rating, into = c("rating", "n_rating"), sep = " \\(", remove = TRUE,
           fill = "right") %>%
    mutate(
    rating = trimws(rating),
    n_rating = trimws(n_rating)
  ) %>%
    mutate(
    rating = if_else(rating == "Nuevo" | is.na(rating), NA_real_,
                     as.numeric(gsub(",", ".", rating))),
    n_rating = if_else(is.na(n_rating), 0, as.numeric(gsub("\\)", "",
                                                           n_rating)))
  )
```

Tipo de anfitrión

Como a efectos prácticos NA en valores de registro es similar a que no tenga número de registro, imputaremos los valores faltantes de esta variable por el valor Exempt.

```
df_clean = df_clean %>%
  mutate(tiene_registro = ifelse(is.na(n_registro2) | n_registro2 == "Exempt",
                                 0, 1))
```

Por último vamos codificar la variable `tipo_anfitrion` con 1, si el anfitrión es profesional o 0, si es particular.

```
df_clean = df_clean %>%
  mutate(tipo_anfitrion = ifelse(tipo_anfitrion == 'profesional',1,0))
```

Price

A esta columna simplemente, le quitaremos el símbolo del euro (€).

```
df_clean <- df_clean %>%
  mutate(price = as.numeric(gsub(" €", "", price)))
```

Amenities

Esta es una variable que nos aporta mucha información pues en cada fila tenemos una lista con muchos datos. Al ser una variable diversa sería muy interesante poder clasificar cada alojamiento según una serie de características. Se nos ocurren las siguientes variables:

- Acceso a Espacios y Áreas Comunes
- Confort y Climatización
- Mobiliario y Espacios de Relajación
- Cocina y Equipos
- Electrodomésticos y Equipos de Tecnología
- Servicios y Seguridad
- Baño y Higiene
- Niños y Familia
- Actividades y Entretenimiento
- Accesorios y Otros

Y, ¿Cómo clasificar las amenities que son tan diversas entre sí de manera correcta? Echemos un vistazo a los distintos tipos de amenities:

```
# Extraer todos los elementos únicos
unique_amenities <-df_clean %>%
  mutate(amenities = str_remove_all(amenities, "\\[|\\]|'")) %>%
  separate_rows(amenities, sep = ", ") %>%
  distinct(amenities) %>%
  arrange(amenities)
unique_amenities
```

```
## # A tibble: 243 x 1
##    amenities
##    <chr>
##  1 Acceso a la playa
##  2 Acceso al lago
##  3 Acceso al resort
##  4 Acceso compartido a la playa
##  5 Acceso gratuito al complejo
##  6 Acondicionador
##  7 Acondicionador de Marcas francesas
##  8 Admite mascotas
##  9 Agua caliente
## 10 Aire acondicionado
## # ... with 233 more rows
```

Resultaría muy interesante usar, por ejemplo, un llm que pudiera para cada alojamiento nuevo y según las amenities que tenga decir si posee o no cada una de las variables indentificadas anteriormente que tome los valores 0 o 1. Para ellos solo tendríamos que hacer un prompt y llamar al modelo al que tengamos acceso

pasándole los datos respectivos a nuestra fila. Dejamos en código python cómo podríamos hacerlo en el archivo `datos.py`.

Como es algo trabajoso y nuestro conjunto de datos es pequeño hemos decidido exponer la idea pero no implementarla. Así no usaremos la variable amenities.

Location

Como en su día no obtuvimos una variable Localización en la PR1 pero nos parecía interesante usarla teniendo en cuenta la naturaleza del problema que estamos tratando hemos decidido crear una de manera ficticia. Esta variable tendrá 4 categoría según ubicación y serán elegidas teniendo en cuenta la primera letra del título:

```
df_clean$Inicial = toupper(str_sub(df_clean$title, 1, 1))

letras_unicas = sort(unique(df_clean$Inicial))

grupos = split(letras_unicas, cut(seq_along(letras_unicas), 4, labels = FALSE))

grupo_dict = unlist(lapply(seq_along(grupos), function(i){
  setNames(rep(i, length(grupos[[i]])), grupos[[i]])
}))

df_clean$Location= grupo_dict[df_clean$Inicial]

df_clean = df_clean %>% select(-Inicial)
```

Tipo de alojamiento

Queremos poder identificar si un sitio es hacitación, apartamento o cama. Para ello tendremos en cuenta el título y con él intentaremos deducir de qué tipo de alojamiento se trata. Esta variable nos resulta bastante interesante ya que luego, con el resto de los datos intentaremos predecirla.

```
categorize_title <- function(title) {
 # Convert to ASCII (removes accents)
  title <- tolower(title)                              # Convert to lowercase
  if (grepl("cama", title)) {
    return("Cama")
  } else if (grepl("dormitorio|cuarto|habitacion|rooms|room|havitacion", title)) {
    return("Dormitorio")
  } else if (grepl("apartamento|loft|casa|apto|apart|casita|estudio|piso|apt", title)) {
    return("Apartamento")
  } else {
    return("Indefinido") # Default category
  }
}

df_clean$category <- sapply(df_clean$title, categorize_title)


df_clean$category[df_clean$category == 'Indefinido'] <- NA
```

Aunque aquí lo hemos hecho de manera manual, lo ideal sería, como en el caso de las amenities, tener un LLM que categorizase según el título el tipo de alojamiento con el que nos encontramos.

Dormitorio 1 y Dormitorio 2

Debido a la cantidad de NA que tenemos en estas columnas hemos decidido directamente eliminarlas.

```
df_clean = df_clean %>% select(-c(`Dormitorio 1`, `Dormitorio 2`))
```

Nuestro dataset quedaría de la siguiente manera:

```
## # A tibble: 1 x 9
##   title rating n_rating tipo_anfitrion n_registro2 tiene_registro price Location
##   <chr>  <dbl>    <dbl>          <dbl> <chr>                <dbl> <dbl>    <int>
## 1 Hola~   4.41     1124              1 AJ000687                 1    35        2
## # ... with 1 more variable: category <chr>
```

3.2 Conversión de tipos de datos

Las variables que son de tipo categóricas, vamos a modificarlas para que sean de tipo factor.

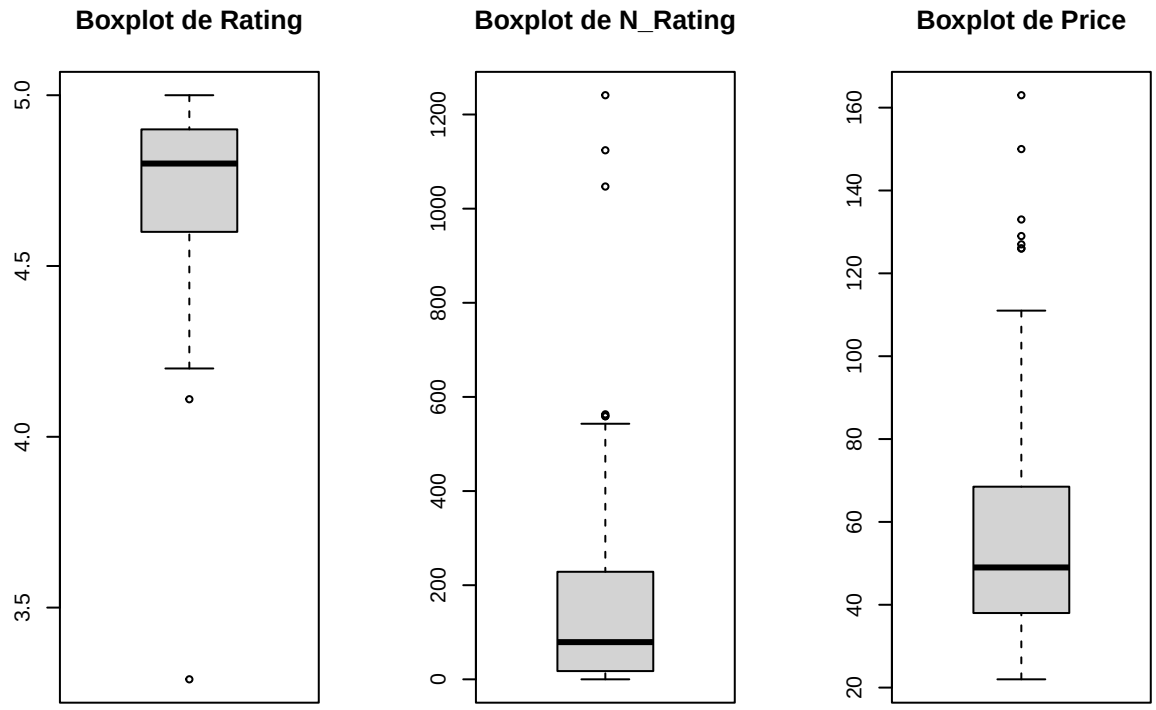
```
r glimpse(df_clean)
## Rows: 107 ## Columns: 11 ## $
title          <chr> "Hola Hostal
Collblanc. Cama en dormitorio 18
pax", "Ar~ ## $ rating
<dbl> 4.41, 4.96, 4.77, 4.94,
4.64, 4.96, 4.91, 5.00, 4.75, 4~
## $ n_rating       <dbl> 1124,
147, 1241, 16, 141, 71, 47, 7, 60,
118, 37, 213, ~ ## $ amenities
<chr> "['Secador de pelo',
'Lavadora', 'Secadora',
'Servicios~ ## $ reviews
<chr> "[{'rating': 'Valoración:
5\\xa0estrellas', 'date': 'Va~ ##
$ tipo_anfitrion <dbl> 1, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 0, 0, 0~ ## $ n_registro2
<chr> "AJ000687", "HUTB-033567",
"ATB-002816", "Exempt", "HUT~ ## $
tiene_registro <dbl> 1, 1, 1, 0,
1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 0, 0, 0~ ## $ price
<dbl> 35, 57, 75, 47, 65, 38, 57,
31, 44, 32, 52, 24, 89, 51,~ ## $
Location       <int> 2, 1, 4, 2,
1, 3, 3, 2, 2, 2, 2, 1, 3, 1, 1,
1, 1, 1, 3~ ## $ category
<chr> "Cama", NA, NA,
"Dormitorio", "Apartamento",
"Apartamen~
```

```
r df_clean <- df_clean %>%
mutate(Location =
as.factor(Location),
tiene_registro  =
as.factor(tiene_registro ),
tipo_anfitrion   =
as.factor(tipo_anfitrion  ) )
```

3.3 Detección y gestión de valores extremos

Buscamos outliers en variables numéricas. Aunque los detectemos, en nuestro caso, como tenemos un conjunto de datos tan pequeño, no los vamos a eliminar pero en conjuntos de datos más grandes si sería conveniente.

```r
# Configurar para mostrar 3 gráficos en una fila
par(mfrow = c(1, 3))

# Crear los boxplots para las tres variables
boxplot(df_clean$rating, main = "Boxplot de Rating")
boxplot(df_clean$n_rating, main = "Boxplot de N_Rating")
boxplot(df_clean$price, main = "Boxplot de Price")
```




```r
# Restaurar la configuración predeterminada de los gráficos
par(mfrow = c(1, 1))
```

```r
# Calcular y mostrar los outliers de 'rating'
bp_stats_rating <- boxplot.stats(df_clean$rating)
```

```
outliers_rating <- bp_stats_rating$out
cat("Outliers de la variable `rating`:\n")
```

```
## Outliers de la variable `rating`:
```

```
kable(outliers_rating)
```

x
4.11
3.29

```
# Calcular y mostrar los ouhottliers de 'n_rating'
bp_stats_n_rating <- boxplot.stats(df_clean$n_rating)
outliers_n_rating <- bp_stats_n_rating$out
cat("\nOutliers de la variable `n_rating`:\n")
```

```
##
## Outliers de la variable `n_rating`:
```

```
kable(outliers_n_rating)
```

x
1124
1241
1047
559
559
563

```
# Calcular y mostrar los outliers de 'price'
bp_stats_price <- boxplot.stats(df_clean$price)
outliers_price <- bp_stats_price$out
cat("\nOutliers de la variable `price`:\n")
```

```
##
## Outliers de la variable `price`:
```

```
kable(outliers_price)
```

x
126
163
133
126
129
150
127

Medidas estadísticas:

```
df_clean = df_clean %>% select(-c(amenities, reviews, n_registro2))
summary(df_clean)
```

```
##     title              rating         n_rating      tipo_anfitrion
##  Length:107         Min.   :3.290   Min.   :   0.0   0:83
##  Class :character   1st Qu.:4.600   1st Qu.:  17.5   1:24
##  Mode  :character   Median :4.800   Median :  79.0
##                     Mean   :4.716   Mean   : 162.1
```

Table 2: Resumen de Media y Desviación Estándar

mean_rating	sd_rating	mean_precio	sd_precio	mean_n_rating	sd_n_rating
4.716225	0.2544795	57.38318	28.90996	162.1402	223.647

```
##                     3rd Qu.:4.900   3rd Qu.: 228.5
##                     Max.   :5.000   Max.   :1241.0
##                     NA's   :9
##  tiene_registro     price        Location   category
##  0:66           Min.   : 22.00   1:40     Length:107
##  1:41           1st Qu.: 38.00   2:42     Class :character
##                 Median : 49.00   3:16     Mode  :character
##                 Mean   : 57.38   4: 9
##                 3rd Qu.: 68.50
##                 Max.   :163.00
##
```

```
# Calcular la media y desviación estándar
summary_stats <- df_clean %>%
  summarise(
    mean_rating = mean(rating, na.rm = TRUE),
    sd_rating = sd(rating, na.rm = TRUE),
    mean_precio = mean(price, na.rm = TRUE),
    sd_precio = sd(price , na.rm = TRUE),
    mean_n_rating = mean(n_rating, na.rm = TRUE),
    sd_n_rating = sd(n_rating , na.rm = TRUE)
  )

# Mostrar el resultado como una tabla bonita con kable
summary_stats %>%
  kable(caption = "Resumen de Media y Desviación Estándar") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed"),
                full_width = FALSE)
```

Vamos con las categorías de tipo cualitativo:

```
# Frecuencia de las categorías de 'tipo_anfitrion'
tipo_anfitrion_freq <- df_clean %>%
  group_by(tipo_anfitrion) %>%
  summarise(frecuencia = n()) %>%
  arrange(desc(frecuencia))

# Mostrar la tabla de frecuencias para 'tipo_anfitrion'
kable(tipo_anfitrion_freq)
```

tipo_anfitrion	frecuencia
0	83
1	24

```
# Frecuencia de las categorías de 'Location'
location_freq <- df_clean %>%
  group_by(Location) %>%
  summarise(frecuencia = n()) %>%
  arrange(desc(frecuencia))
```

```r
# Mostrar la tabla de frecuencias para 'Location'
kable(location_freq)
```

Location	frecuencia
2	42
1	40
3	16
4	9

```r
# Frecuencia de las categorías de 'Location'
location_freq <- df_clean %>%
  group_by(tiene_registro     ) %>%
  summarise(frecuencia = n()) %>%
  arrange(desc(frecuencia))

# Mostrar la tabla de frecuencias para 'Location'
kable(location_freq )
```

tiene_registro	frecuencia
0	66
1	41

```r
# Frecuencia de las categorías de 'Location'
location_freq <- df_clean %>%
  group_by(category     ) %>%
  summarise(frecuencia = n()) %>%
  arrange(desc(frecuencia))
# Mostrar la tabla de frecuencias para 'Location'
kable(location_freq)
```

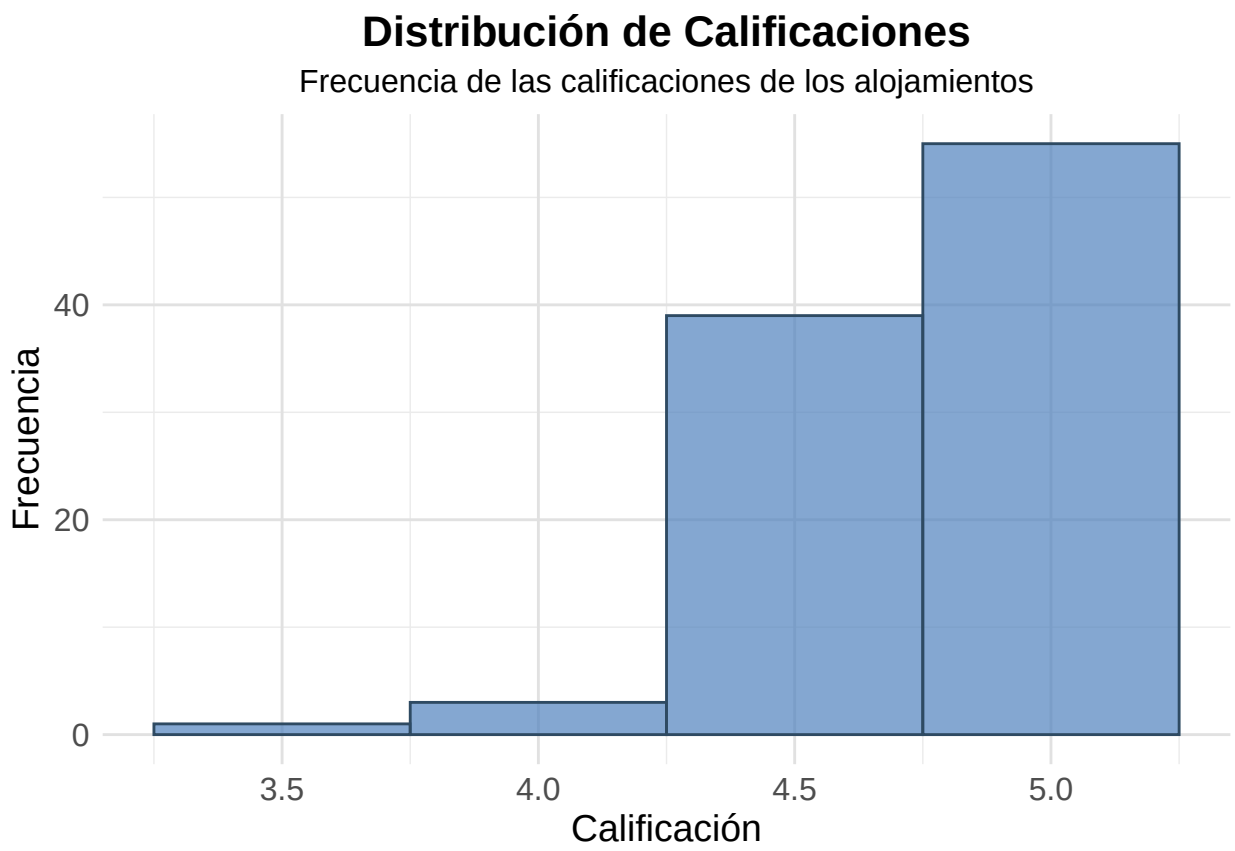
category	frecuencia
Dormitorio	55
NA	24
Apartamento	22
Cama	6

Visualizaciones

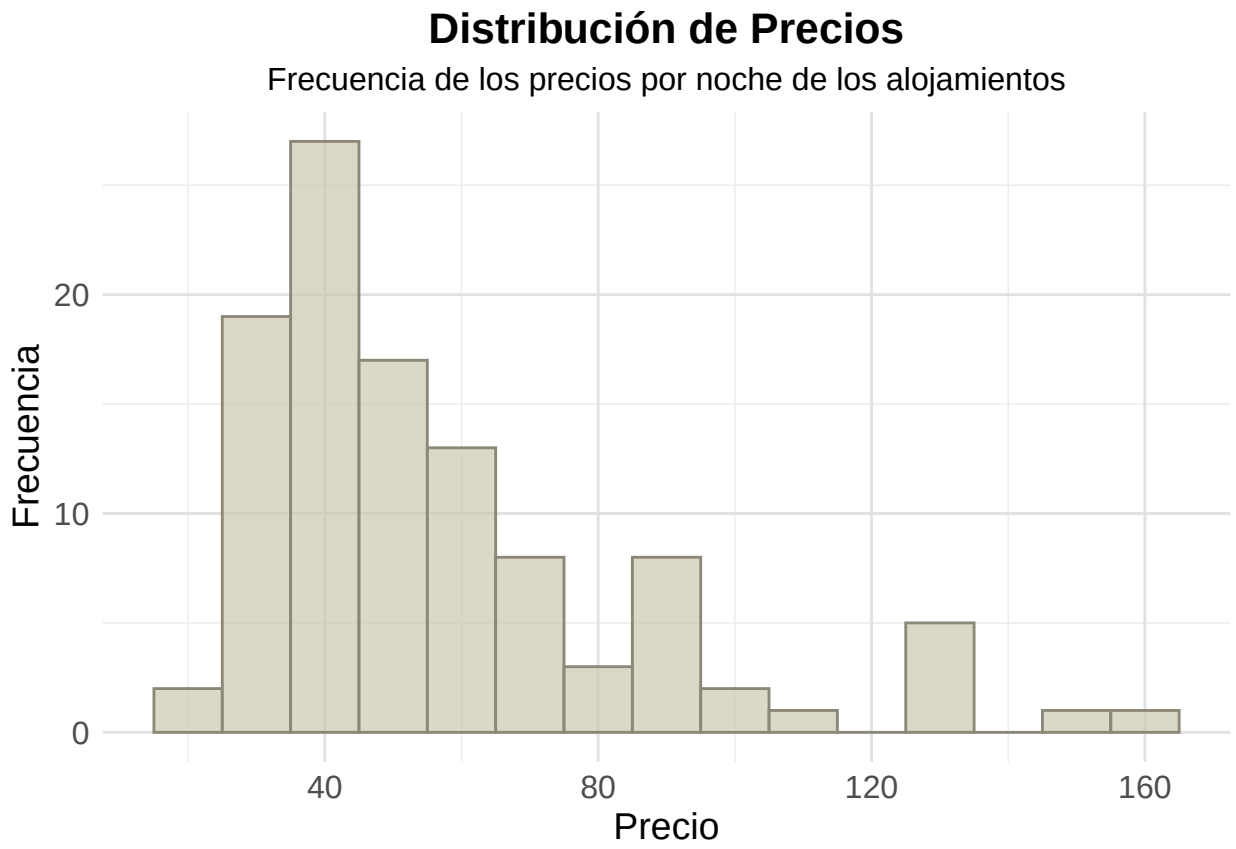
Variables continuas

```r
par(mfrow = c(1, 3))
# Distribución de las calificaciones
ggplot(df_clean, aes(x = rating)) +
  geom_histogram(binwidth = 0.5, fill = "#4F81BD", color = "#2E4A62", alpha = 0.7) +
  labs(
    title = "Distribución de Calificaciones",
    subtitle = "Frecuencia de las calificaciones de los alojamientos",
    x = "Calificación",
    y = "Frecuencia"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    plot.subtitle = element_text(hjust = 0.5, size = 12),
    axis.title = element_text(size = 14),
```

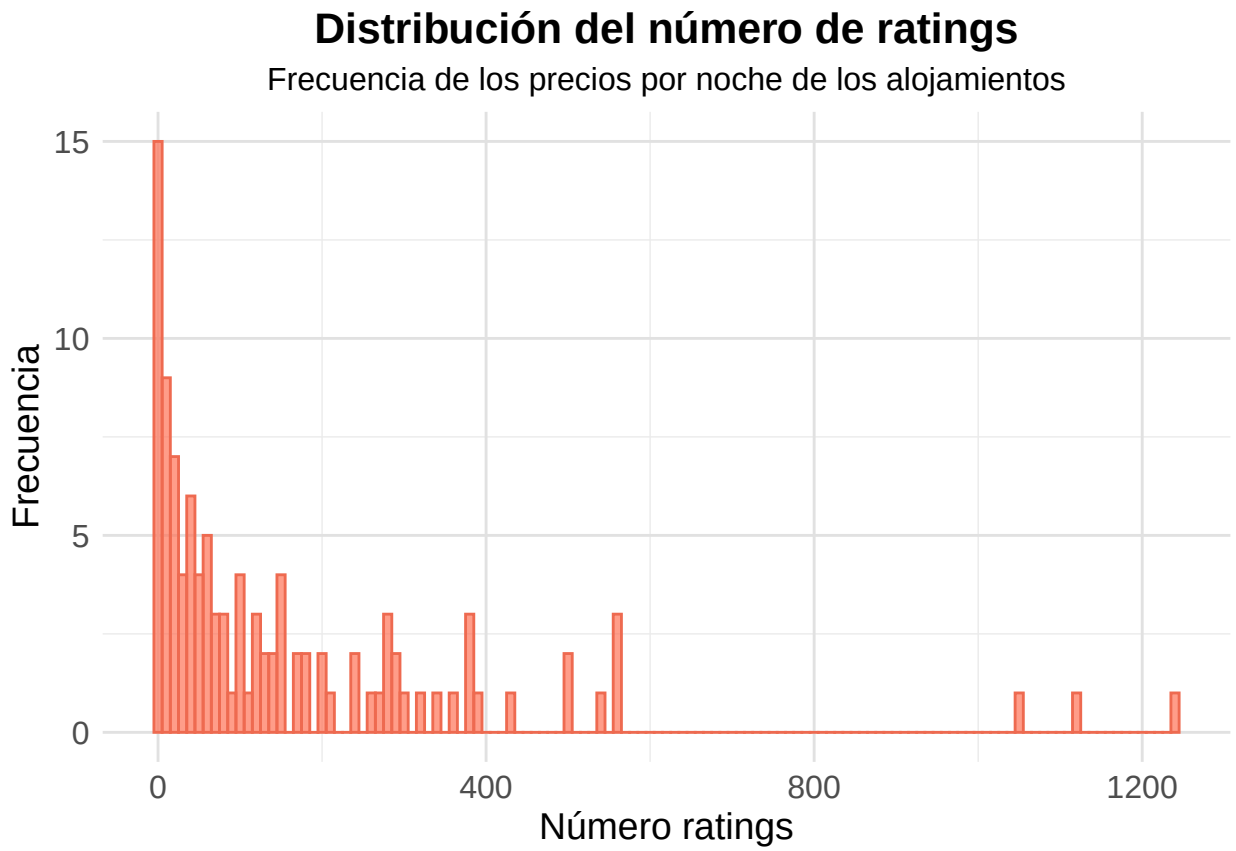
```
    axis.text = element_text(size = 12)
  ) +
  theme(panel.grid.major = element_line(color = "#e1e1e1", size = 0.5))
```




```
# Distribución de los precios (si los tienes)
ggplot(df_clean, aes(x = price)) +
  geom_histogram(binwidth = 10, fill = "#CDC8B1", color = "#8B8878", alpha = 0.7) +
  labs(
    title = "Distribución de Precios",
    subtitle = "Frecuencia de los precios por noche de los alojamientos",
    x = "Precio",
    y = "Frecuencia"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    plot.subtitle = element_text(hjust = 0.5, size = 12),
    axis.title = element_text(size = 14),
    axis.text = element_text(size = 12)
  ) +
  theme(panel.grid.major = element_line(color = "#e1e1e1", size = 0.5))
```

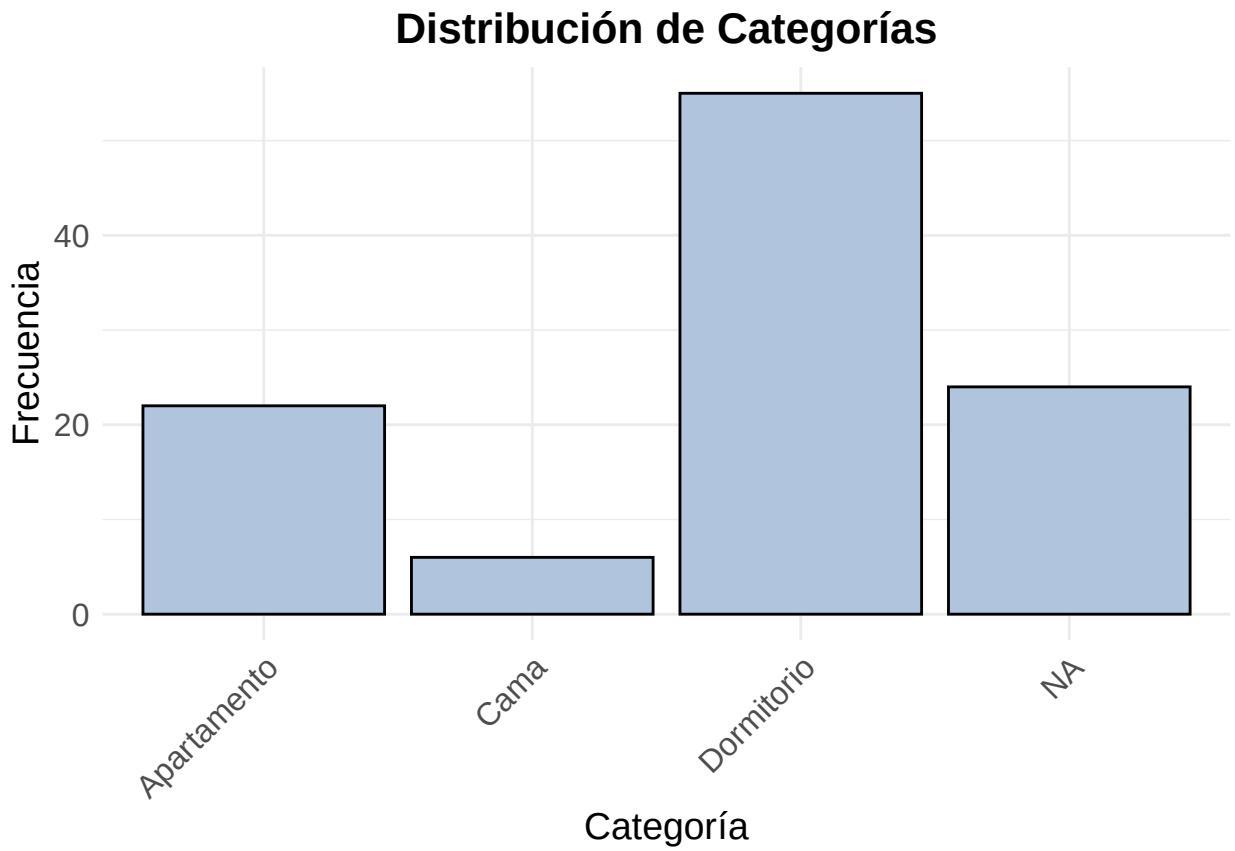



```r
# Distribución de los n_rating (si los tienes)
ggplot(df_clean, aes(x = n_rating)) +
  geom_histogram(binwidth = 10, fill = "#FF7256", color = "#EE6A50", alpha = 0.7) +
  labs(
    title = "Distribución del número de ratings",
    subtitle = "Frecuencia de los precios por noche de los alojamientos",
    x = "Número ratings",
    y = "Frecuencia"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    plot.subtitle = element_text(hjust = 0.5, size = 12),
    axis.title = element_text(size = 14),
    axis.text = element_text(size = 12)
  ) +
  theme(panel.grid.major = element_line(color = "#e1e1e1", size = 0.5))
```

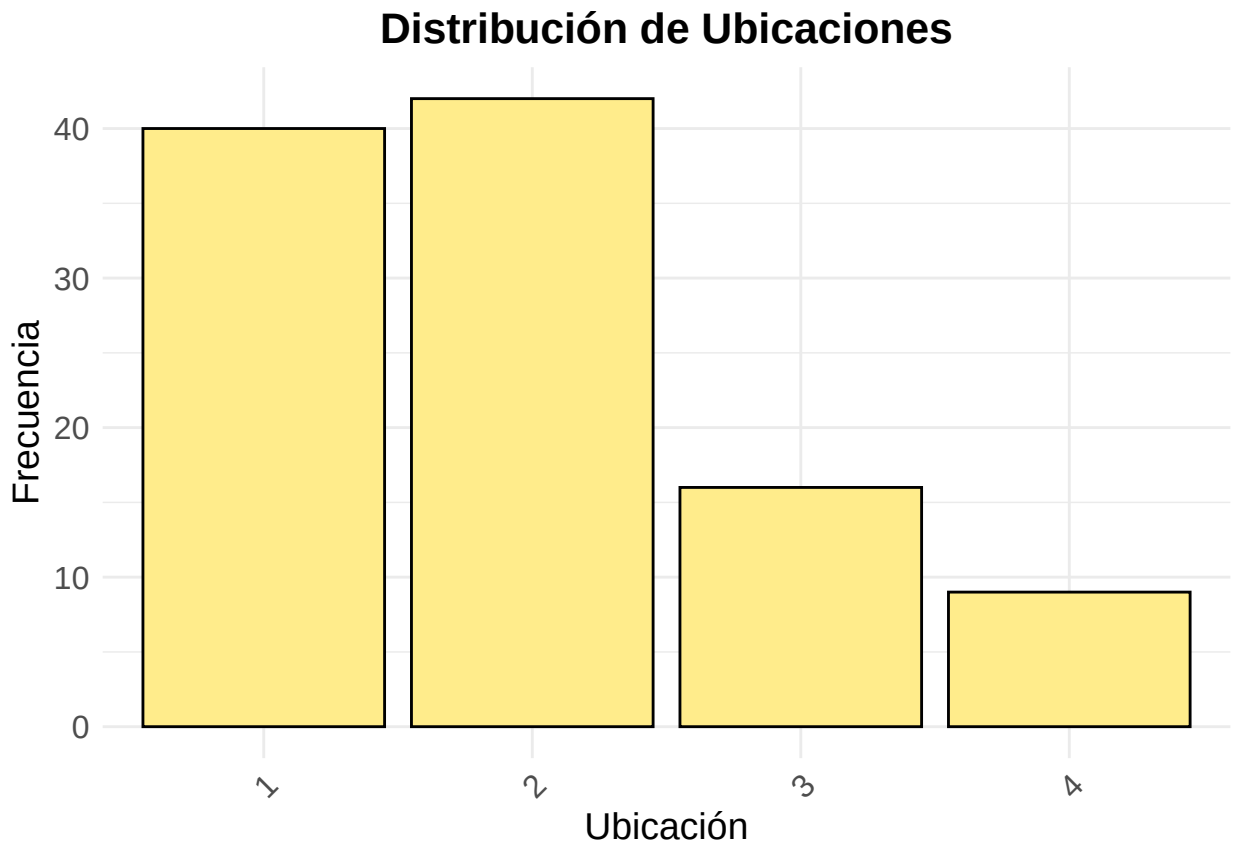



```
par(mfrow = c(1, 1))
```

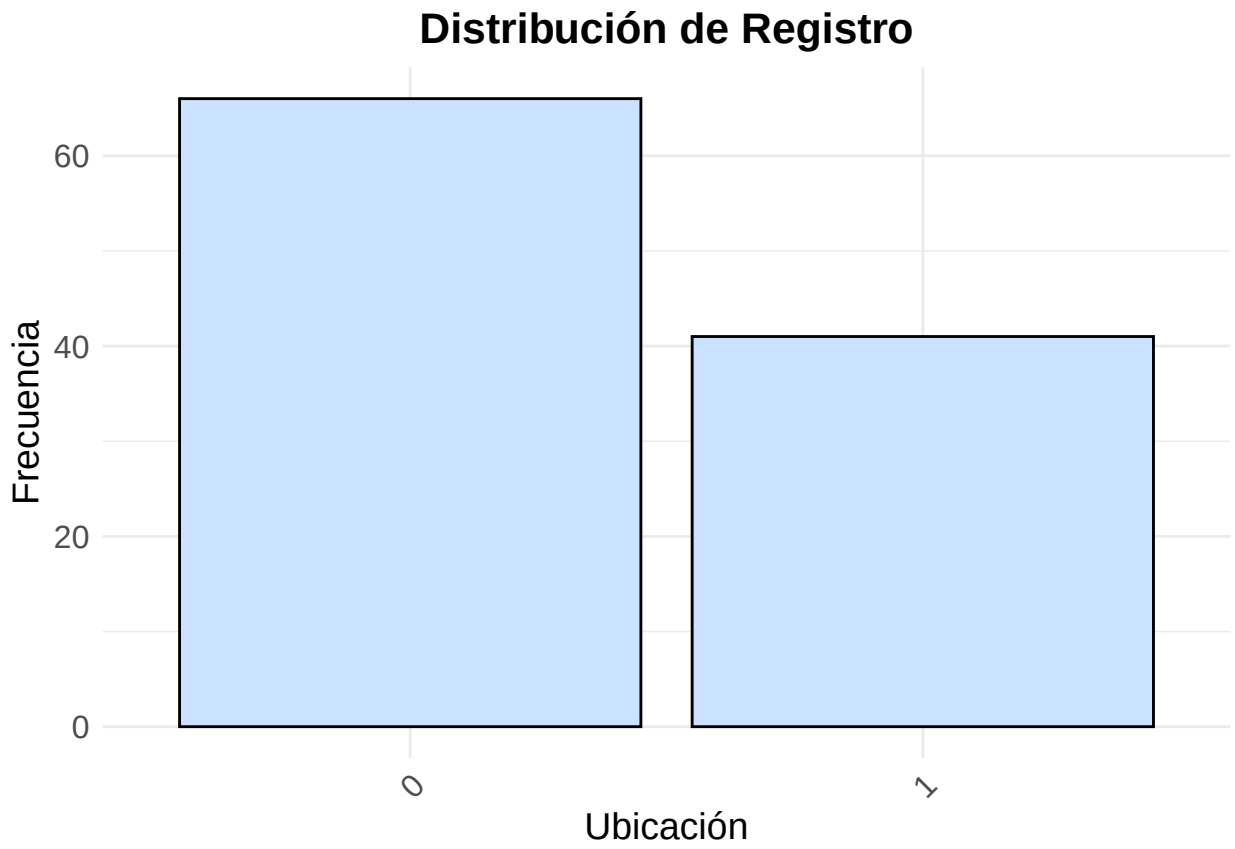
```
# Histograma para la variable 'category'
ggplot(df_clean, aes(x = category)) +
  geom_bar(fill = "#B0C4DE", color = "black") +
  labs(
    title = "Distribución de Categorías",
    x = "Categoría",
    y = "Frecuencia"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 12),
    axis.text.y = element_text(size = 12),
    axis.title = element_text(size = 14)
  )
```



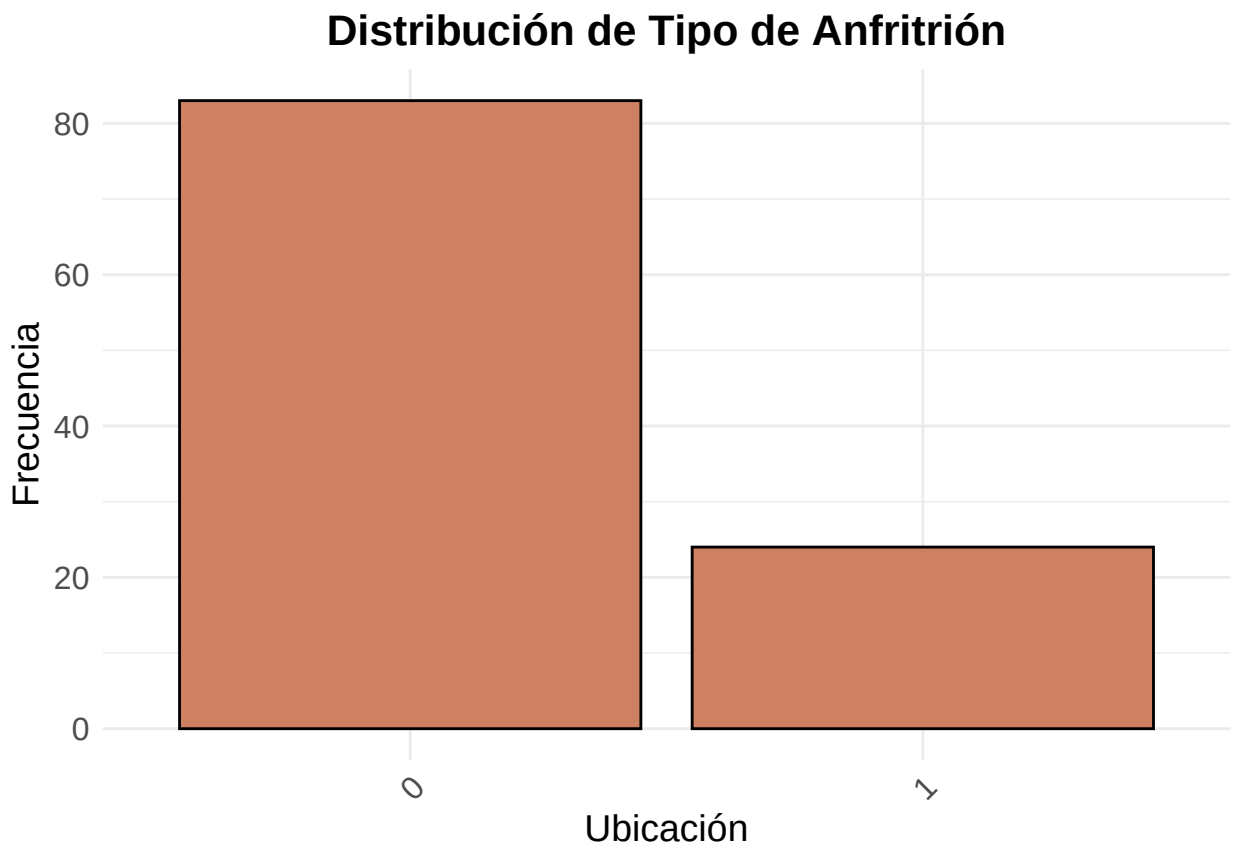

```
ggplot(df_clean, aes(x = as.factor(Location))) +
  geom_bar(fill = "#FFEC8B", color = "black") +
  labs(
    title = "Distribución de Ubicaciones",
    x = "Ubicación",
    y = "Frecuencia"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 12),
    axis.text.y = element_text(size = 12),
    axis.title = element_text(size = 14)
  )
```




```
ggplot(df_clean, aes(x = as.factor(tiene_registro))) +
  geom_bar(fill = "#CAE1FF", color = "black") +
  labs(
    title = "Distribución de Registro",
    x = "Ubicación",
    y = "Frecuencia"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 12),
    axis.text.y = element_text(size = 12),
    axis.title = element_text(size = 14)
  )
```



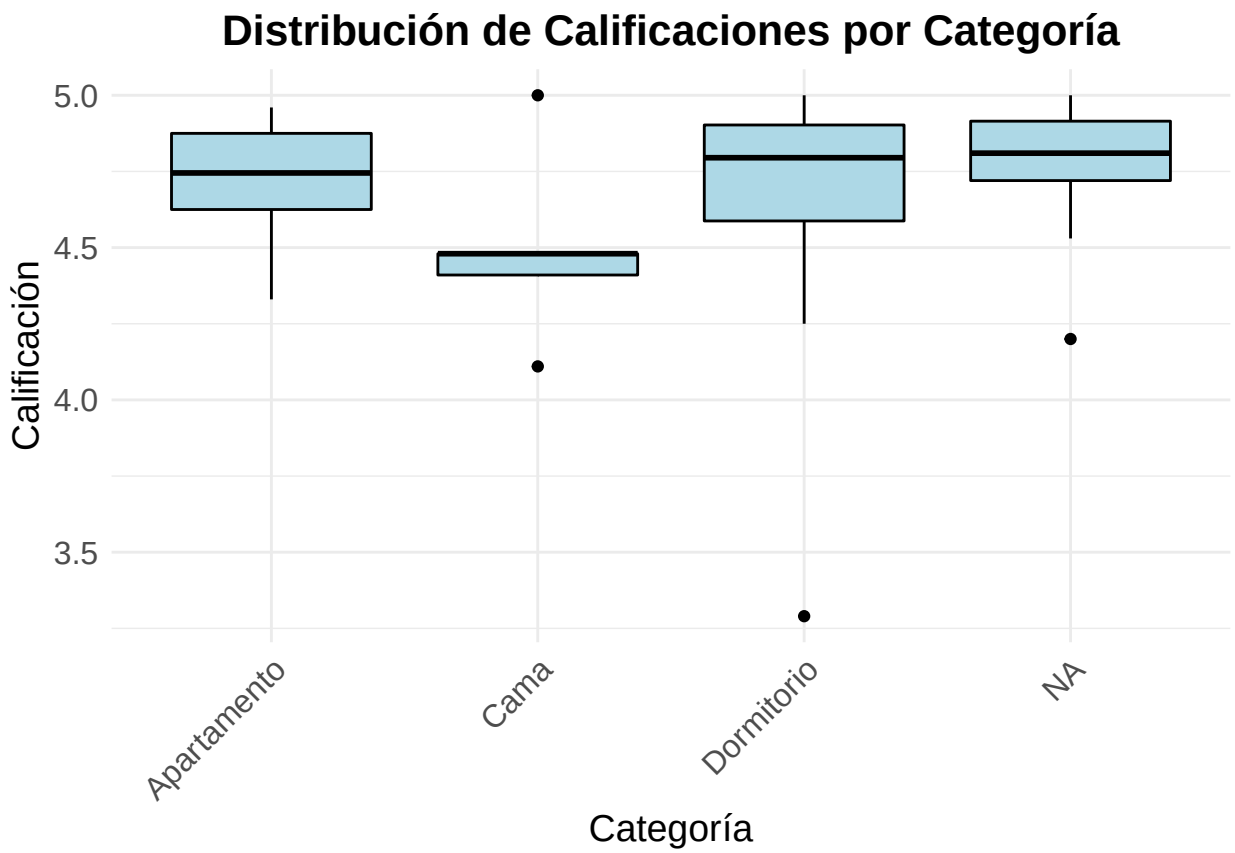

```
ggplot(df_clean, aes(x = as.factor(tipo_anfitrion))) +
  geom_bar(fill = "#CD8162", color = "black") +
  labs(
    title = "Distribución de Tipo de Anfritrión",
    x = "Ubicación",
    y = "Frecuencia"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 12),
    axis.text.y = element_text(size = 12),
    axis.title = element_text(size = 14)
  )
```



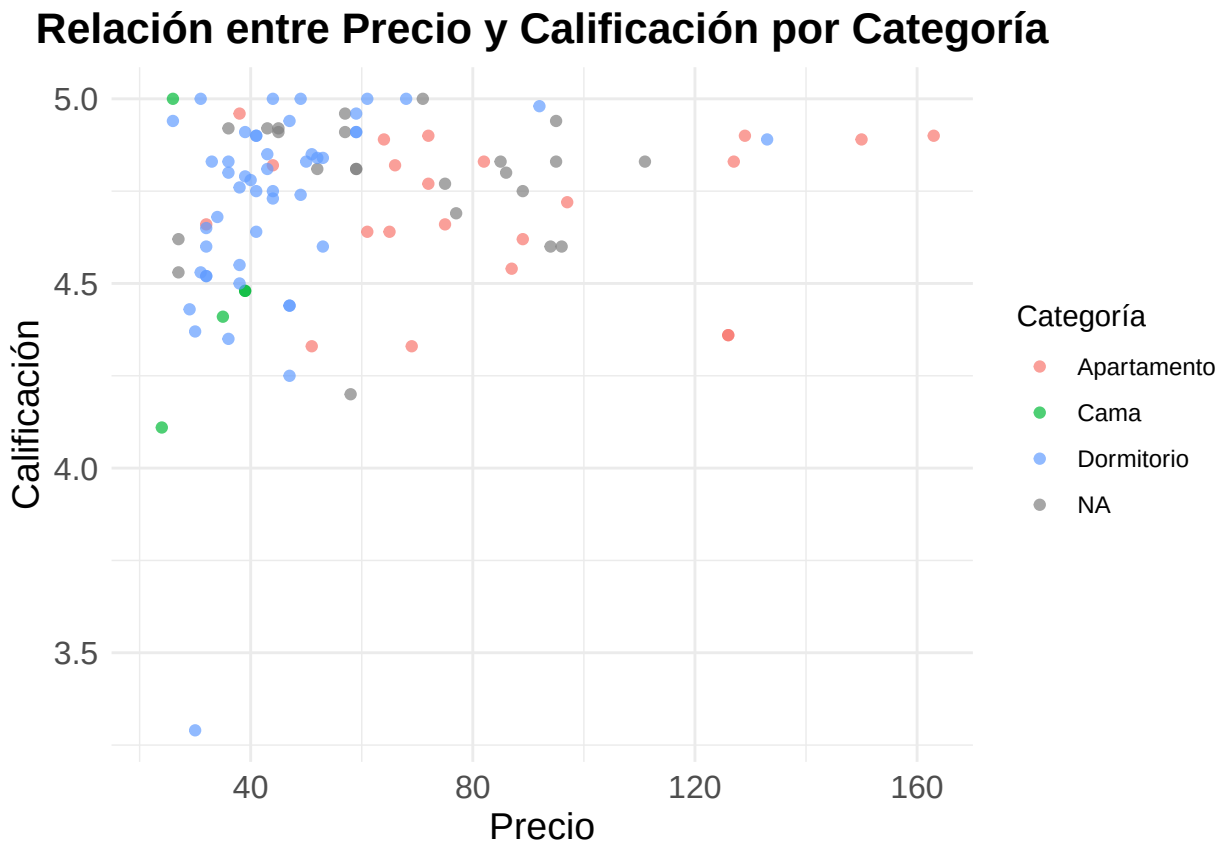
Gráficos varios

Por último vamos a añadir gráficos diversos que pensamos pueden ser interesantes para ayudarnos a entender mejor cómo se distribuyen nuestras variables.

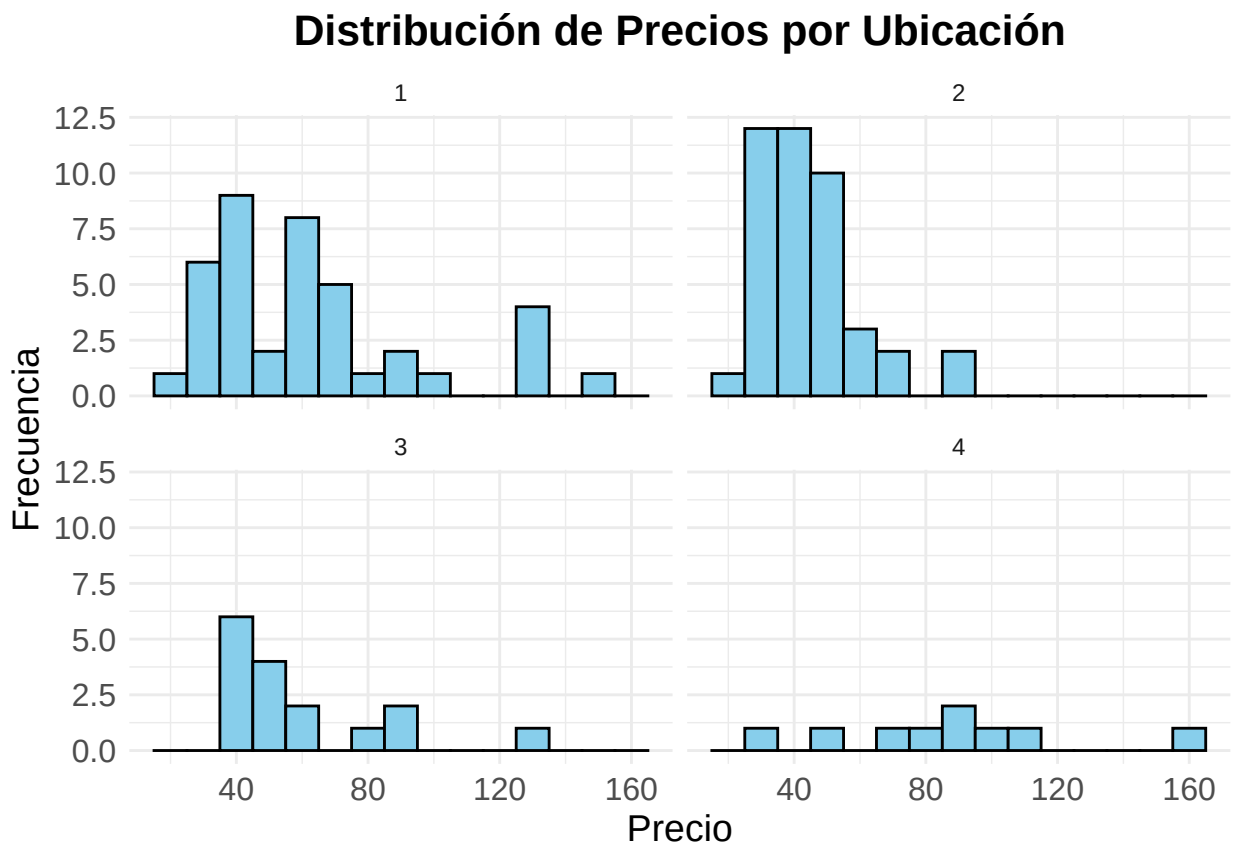
```
# Boxplot de calificaciones por categoría
ggplot(df_clean, aes(x = category, y = rating)) +
  geom_boxplot(fill = "lightblue", color = "black") +
  labs(
    title = "Distribución de Calificaciones por Categoría",
    x = "Categoría",
    y = "Calificación"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 12),
    axis.text.y = element_text(size = 12),
    axis.title = element_text(size = 14)
  )
```



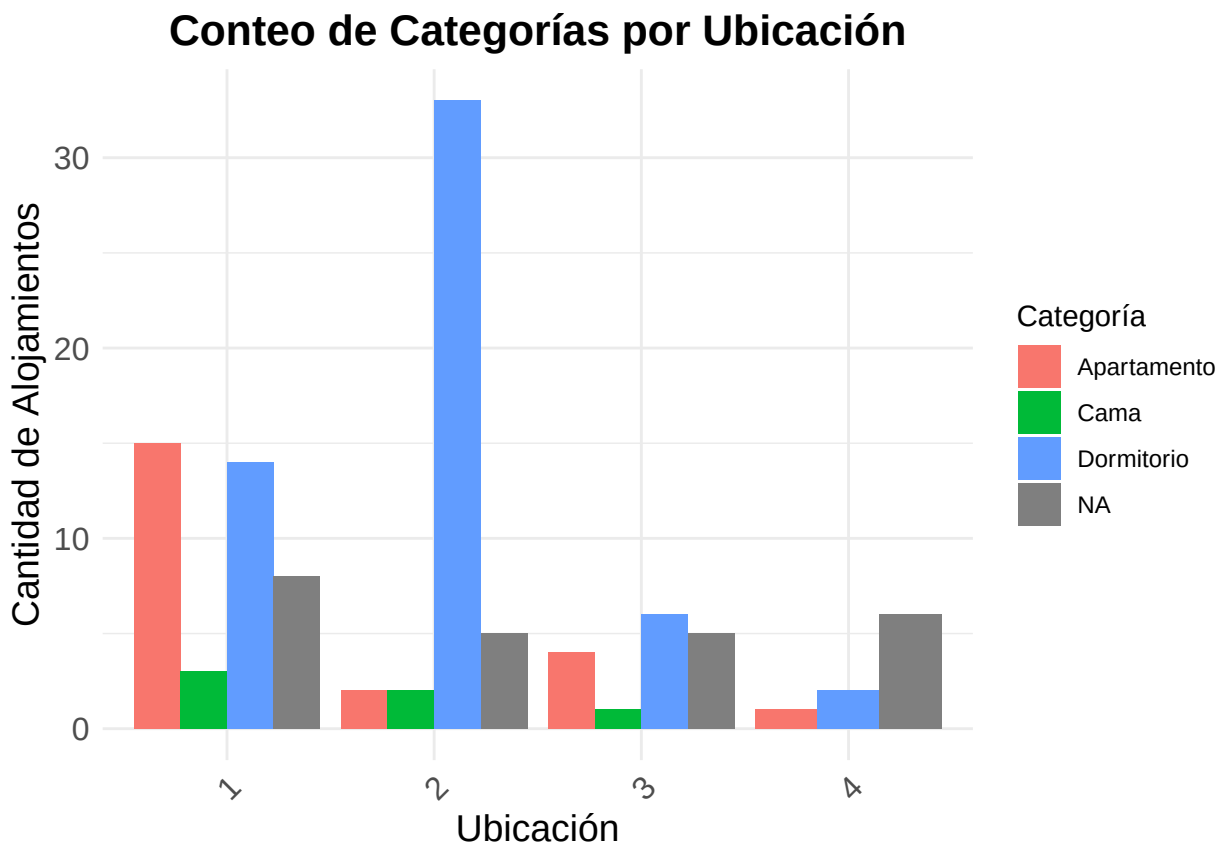

```
# Gráfico de dispersión entre precio y calificación, coloreado por categoría
ggplot(df_clean, aes(x = price, y = rating, color = category)) +
  geom_point(alpha = 0.7) +
  labs(
    title = "Relación entre Precio y Calificación por Categoría",
    x = "Precio",
    y = "Calificación",
    color = "Categoría"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text.x = element_text(size = 12),
    axis.text.y = element_text(size = 12),
    axis.title = element_text(size = 14)
  )
```



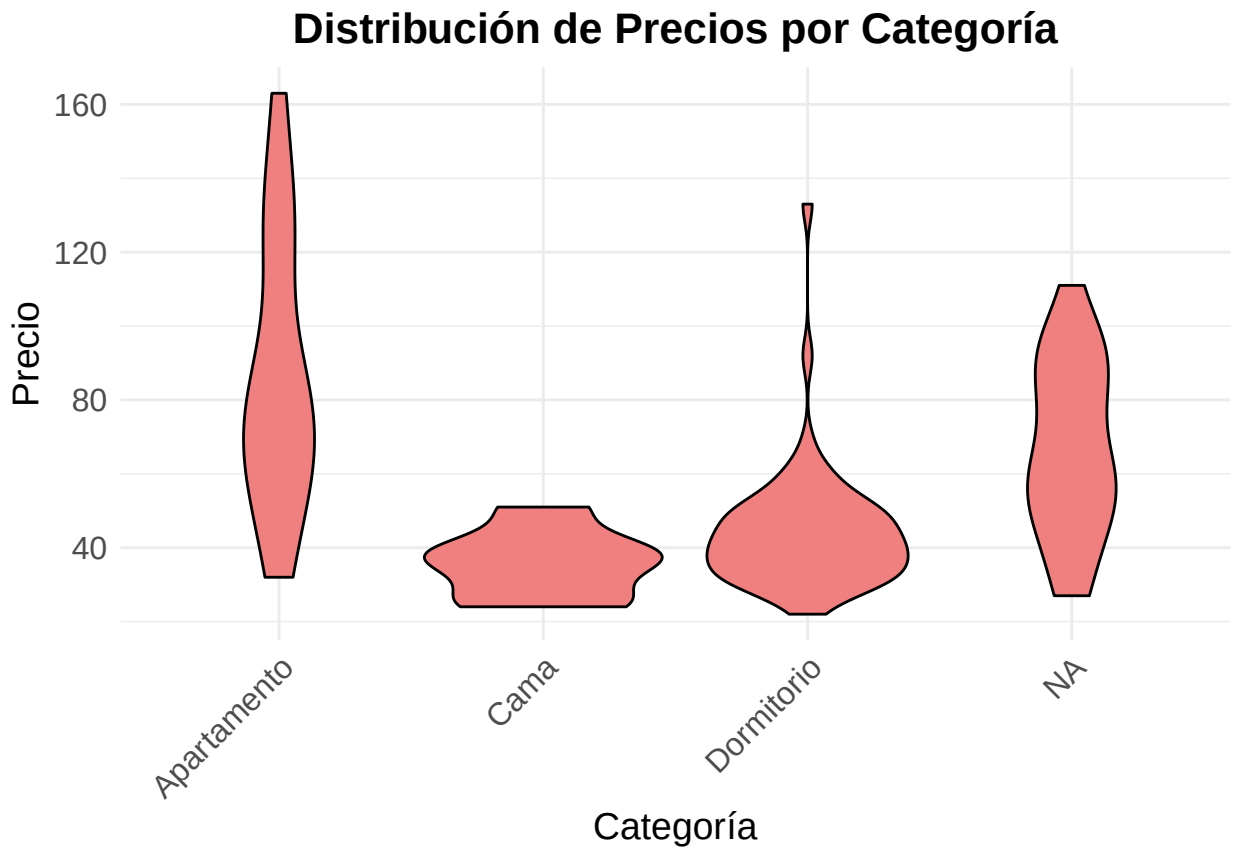

```r
# Histograma faceteado por ubicación
ggplot(df_clean, aes(x = price)) +
  geom_histogram(binwidth = 10, fill = "skyblue", color = "black") +
  facet_wrap(~ Location) +
  labs(
    title = "Distribución de Precios por Ubicación",
    x = "Precio",
    y = "Frecuencia"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text.x = element_text(size = 12),
    axis.text.y = element_text(size = 12),
    axis.title = element_text(size = 14)
  )
```




```
# Gráfico de barras para contar las categorías por ubicación
ggplot(df_clean, aes(x = as.factor(Location), fill = category)) +
  geom_bar(position = "dodge") +
  labs(
    title = "Conteo de Categorías por Ubicación",
    x = "Ubicación",
    y = "Cantidad de Alojamientos",
    fill = "Categoría"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 12),
    axis.text.y = element_text(size = 12),
    axis.title = element_text(size = 14)
  )
```




```
# Violin Plot de precios por categoría
ggplot(df_clean, aes(x = category, y = price)) +
  geom_violin(fill = "lightcoral", color = "black") +
  labs(
    title = "Distribución de Precios por Categoría",
    x = "Categoría",
    y = "Precio"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 12),
    axis.text.y = element_text(size = 12),
    axis.title = element_text(size = 14)
  )
```



4. Análisis de los datos

Codificación de variables

Al tener variables de tipo categórico (Localización y category) sería conveniente seguir una estrategia de one-hot-encoding para poder tratar con ellas de manera efectiva en la modelización de los datos. Como Indefinido realmente significa que no sabemos la naturaleza del alojamiento, lo pondremos como variable NA.

Para poder realizar el proceso de one hot encoding the variables debemos quitar los valores NA que haya en el df

```
df_clean$category <- factor(df_clean$category, levels = c("Cama", "Dormitorio", "Apartamento"))


data_clean_na = na.omit(df_clean)




# Aplicar One Hot Encoding usando model.matrix
dmy <- dummyVars(" ~ category", data = data_clean_na)

trsf <- data.frame(predict(dmy, newdata = data_clean_na))

data_clean_na = cbind(data_clean_na,trsf) %>% select(-category )
```

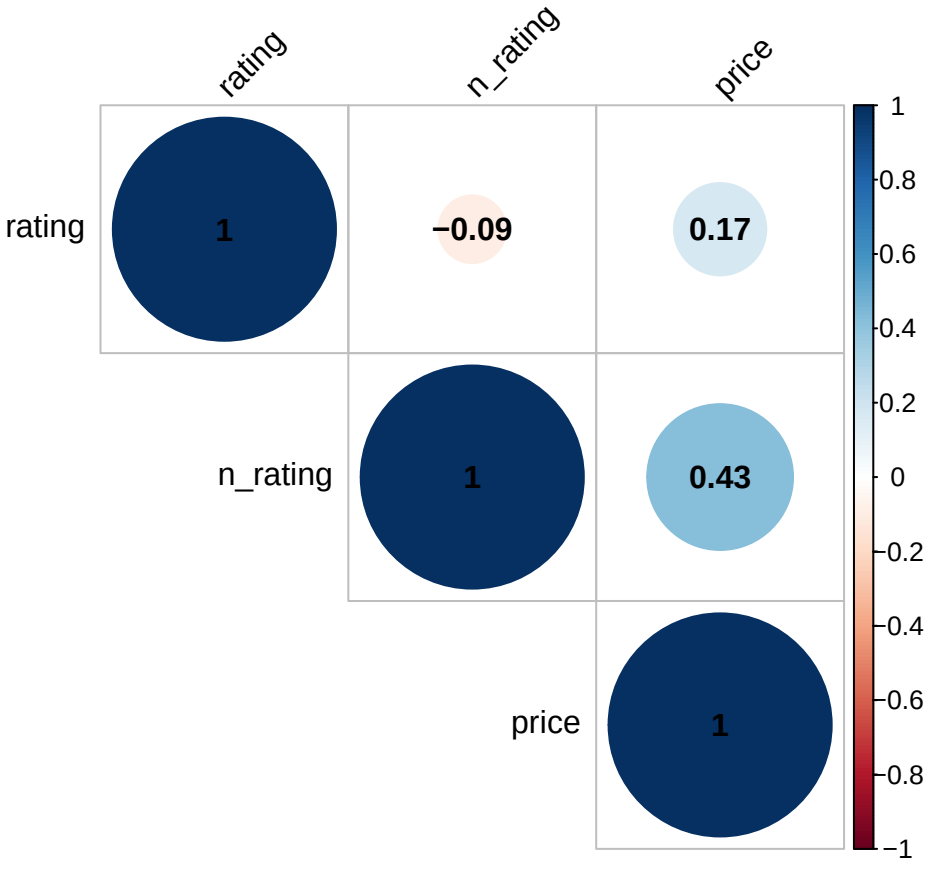
```
data_clean_na$Location <- as.factor(data_clean_na$Location)

dmy <- dummyVars(" ~ Location", data = data_clean_na)
trsf <- data.frame(predict(dmy, newdata = data_clean_na))


data_clean_na = cbind(data_clean_na,trsf) %>% select(-Location )
```

Por último eliminaremos la variable title, que ahora mismo no nos va a aportar nada en el estudio:

```
data_clean_na = data_clean_na %>% select(-title)
data_clean_na2 = data_clean_na
```

Correlaciones entre variables Ploteemos ahora la correlación para las variables numéricas

```
data_clean_na <- na.omit(df_clean %>% select(rating, n_rating, price, category))
data_clean_na = data_clean_na %>%  select(rating, n_rating, price)
cor_matrix = data_clean_na %>%
  cor()
#
corrplot(cor_matrix, method = "circle", type = "upper",
         tl.col = "black", tl.srt = 45, addCoef.col = "black")
```



No observamos una correlación fuerte para ninguna variable

Ahora, obtenedremos también correlación con nuestras variables dummies aunque hay que tener en cuenta varias cosas:

1. Las variables dummy son binarias: Después de aplicar one-hot encoding, las nuevas variables son binarias (0 o 1), lo que significa que no tienen una relación lineal en el mismo sentido que las variables numéricas continuas. Sin embargo, todavía puedes calcular la correlación para explorar si existe alguna relación entre las variables.

2. Interpretación de la correlación:

- Correlación alta: Una correlación alta entre una variable binaria resultante del one-hot encoding y una variable numérica podría indicar que la presencia o ausencia de una categoría tiene una relación fuerte con el valor de la variable numérica.
- Correlación baja o nula: Si la correlación es baja, podría sugerir que no hay una relación lineal entre la variable binaria y la variable numérica.

Modelo no supervisado

En este momento queremos ser capaces de poder crear clústeres con los tipos de alojamientos para, si llega un alojamiento nuevo, poder meterlo en alguno de los grupos

Como algoritmos de clusterización usaremos DBScan o Kmeans pero recordemos que hay muchos otros que podemos usar

Escalamos las variables Esto es un paso fundamental para asegurar el buen funcionamiento de los algoritmos. Escalaremos únicamente las variables númericas

```
# Ensure the columns are numeric and remove rows with NA values
scaled_data <- data_clean_na %>%
  mutate(across(c(price, rating, n_rating), as.numeric)) %>%
  drop_na(price, rating, n_rating)

# Scale the relevant columns
scaled_data[c("price", "rating", "n_rating")] <- scale(scaled_data[c("price", "rating", "n_rating")])

# View the scaled data
scaled_data
```

```
## # A tibble: 75 x 3
##     rating n_rating  price
##      <dbl>    <dbl>  <dbl>
##  1 -1.06      4.49  -0.689
##  2  0.898    -0.723 -0.300
##  3 -0.208    -0.135  0.284
##  4  0.971    -0.464 -0.592
##  5  1.12     -0.765 -0.819
##  6  0.198    -0.516 -0.397
##  7 -0.134    -0.243 -0.786
##  8 -2.16      0.204 -1.05
##  9 -1.35      0.411 -0.170
## 10  0.0870    4.13   1.32
## # ... with 65 more rows
```

DBScan

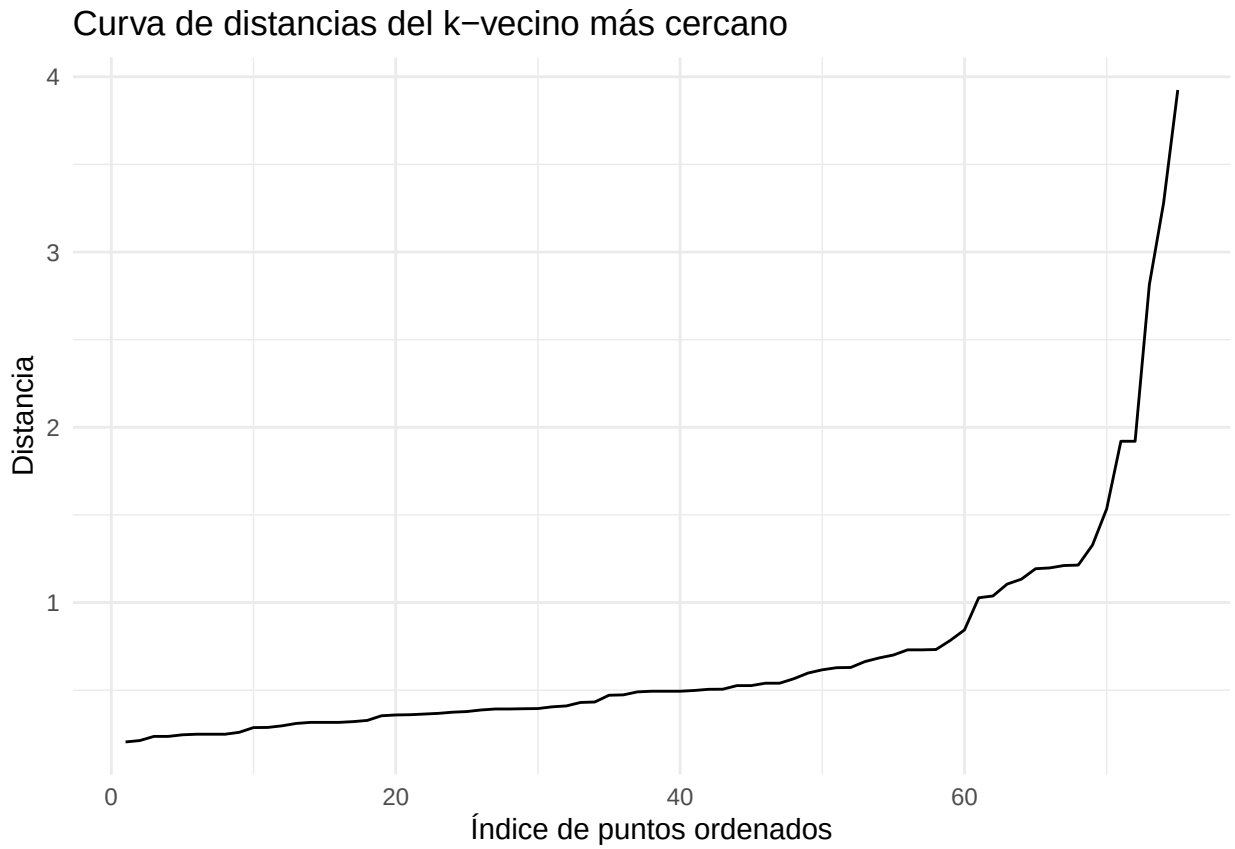
DBSCAN (Density-Based Spatial Clustering of Applications with Noise) es un algoritmo de agrupamiento que clasifica puntos en función de la densidad de puntos cercanos. Forma clusters de alta

densidad y marca como ruido a los puntos que no cumplen con los requisitos de densidad, lo que lo hace eficaz para detectar formas arbitrarias de clusters y manejar datos ruidosos.

Tenemos dos parámetros a estimar, eps y minPts

- eps: Es el radio máximo dentro del cual se consideran los puntos como vecinos. Determina la distancia máxima para buscar puntos cercanos y formar un cluster.
- minPts: Es el número mínimo de puntos requeridos para formar un cluster denso. Un punto es considerado "central" si tiene al menos minPts puntos dentro de su radio eps.

```
# Calcular la distancia a los k-vecinos más cercanos con la función kNN
kNN_dist <- kNN(scaled_data, k = 3)  # Usando 4 vecinos
#
## Graficar la distancia del k-vecino más cercano
ggplot(data.frame(dist = sort(kNN_dist$dist[, 3])), aes(x = seq_along(dist), y = dist)) +
  geom_line() +
  labs(title = "Curva de distancias del k-vecino más cercano", x = "Índice de puntos ordenados", y = "D
  theme_minimal()
```



Podemos intuir que el valor de eps a usar está entre 1 y 2.

Vamos a optimizar parámetros para dbscan

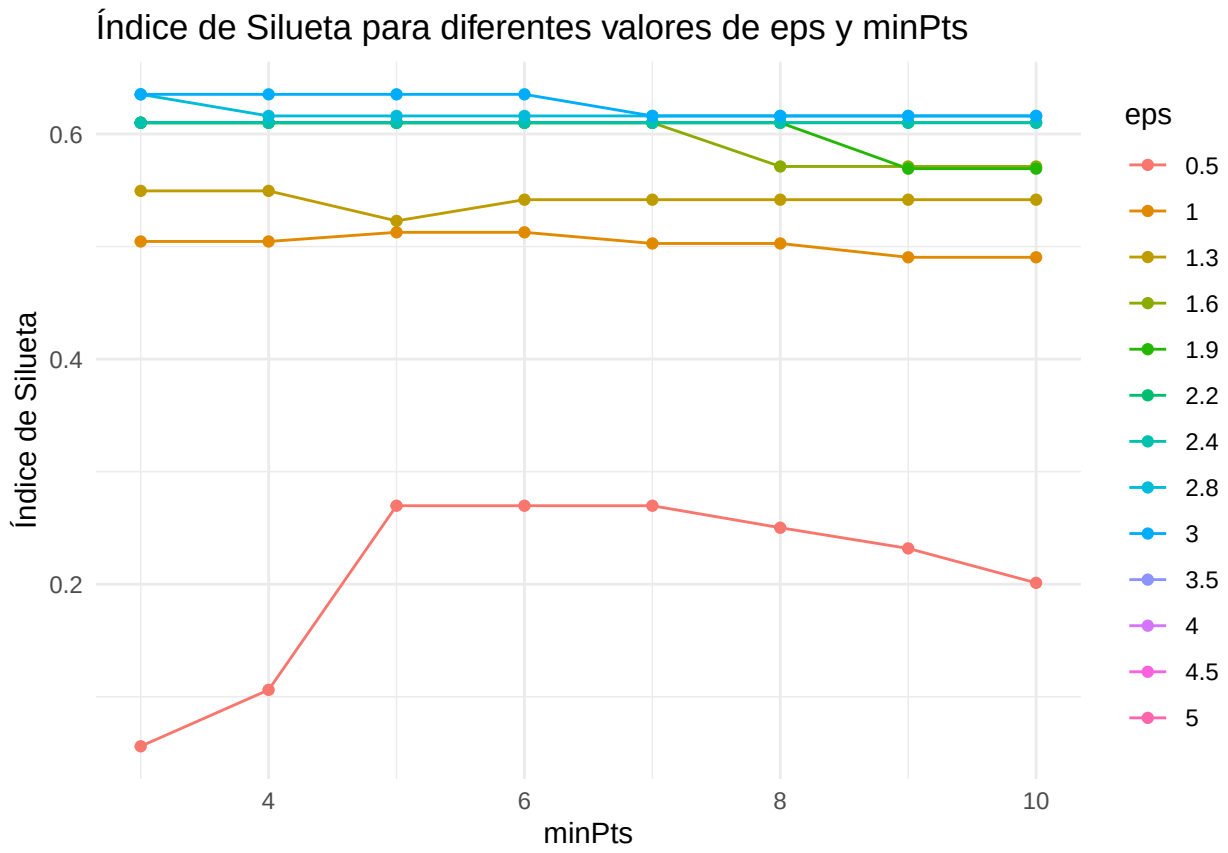
```
library(cluster)

# Rango de valores para eps y minPts
eps_values <- c(0.5, 1, 1.3, 1.6, 1.9, 2.2, 2.4, 2.8, 3, 3.5, 4, 4.5, 5)
minPts_values <- 3:10
```

```r
# Inicializar una lista para almacenar los resultados
results <- data.frame(eps = numeric(), minPts = numeric(), silhouette = numeric())

# Iterar sobre los valores de eps y minPts
for (eps in eps_values) {
  for (minPts in minPts_values) {
    # Ejecutar DBSCAN
    dbscan_result <- dbscan(scaled_data, eps = eps, minPts = minPts)

    # Calcular el índice de silueta si hay más de un cluster
    if (length(unique(dbscan_result$cluster)) > 1) {
      sil <- silhouette(dbscan_result$cluster, dist(scaled_data))
      sil_mean <- mean(sil[, 3])
    } else {
      sil_mean <- NA  # No se puede calcular la silueta si solo hay un cluster
    }

    # Guardar los resultados
    results <- rbind(results, data.frame(eps = eps, minPts = minPts, silhouette = sil_mean))
  }
}

# Filtrar el valor máximo de la silueta
best_result <- results[which.max(results$silhouette), ]

# Convertir el mejor resultado en un data frame bonito para kable
best_result_df <- data.frame(
  "Mejor valor de eps" = best_result$eps,
  "Mejor valor de minPts" = best_result$minPts,
  "Índice de Silueta" = round(best_result$silhouette, 3)
)

# Mostrar el resultado usando kable
kable(best_result_df, caption = "Mejor combinación de eps y minPts con mayor índice de silueta", format
```

Table 3: Mejor combinación de eps y minPts con mayor índice de silueta

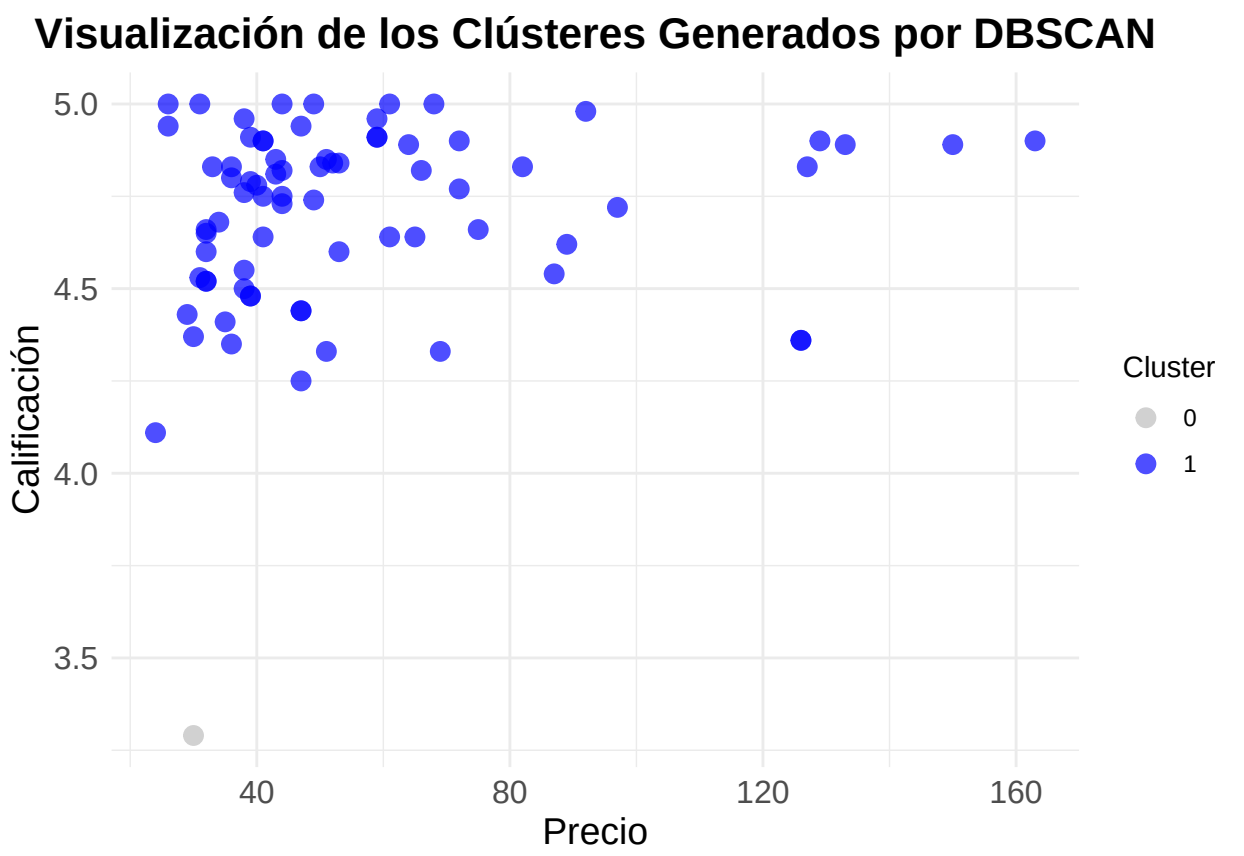
Mejor.valor.de.eps	Mejor.valor.de.minPts	Índice.de.Silueta
2.8	3	0.635

```r
ggplot(results, aes(x = minPts, y = silhouette, color = as.factor(eps))) +
  geom_line() +
  geom_point() +
  labs(
    title = "Índice de Silueta para diferentes valores de eps y minPts",
    x = "minPts",
    y = "Índice de Silueta",
    color = "eps"
  ) +
  theme_minimal()
```



Nos quedamos con eps = 2.8, minPts = 3

```
#library(dbscan)
dbscan_result <- dbscan(scaled_data, eps = 2.8, minPts = 3)
data_clean_na$category_cluster <-as.factor(dbscan_result$cluster)



# Crear un gráfico visual de los clusters
cluster_plot <- ggplot(data_clean_na, aes(x = price, y = rating, color = category_cluster)) +
  geom_point(size = 3, alpha = 0.7) +  # Puntos con color por cluster
  scale_color_manual(values = c("grey", "blue", "red", "green", "purple", "orange", "pink")) +  # Color
  labs(
    title = "Visualización de los Clústeres Generados por DBSCAN",
    x = "Precio",
    y = "Calificación",
    color = "Cluster"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text = element_text(size = 12),
    axis.title = element_text(size = 14)
  )

# Mostrar el gráfico en el HTML usando knitr
print(cluster_plot)
```



Vemos hay únicamente una variable en el clúster 0 aún habiendo hecho DBScan aún habiendo hecho tuning de los parámetros.

Podemos intentar ajustar los parámetros eps y minPts para incluir más puntos en los clusters o probar con un conjunto más pequeño de parámetros. Aún asi, vamos a continuar con el siguiente método.

KMeans

Esta vez vamos a realizar los clústeres tomando únicamente las variables numéricas.

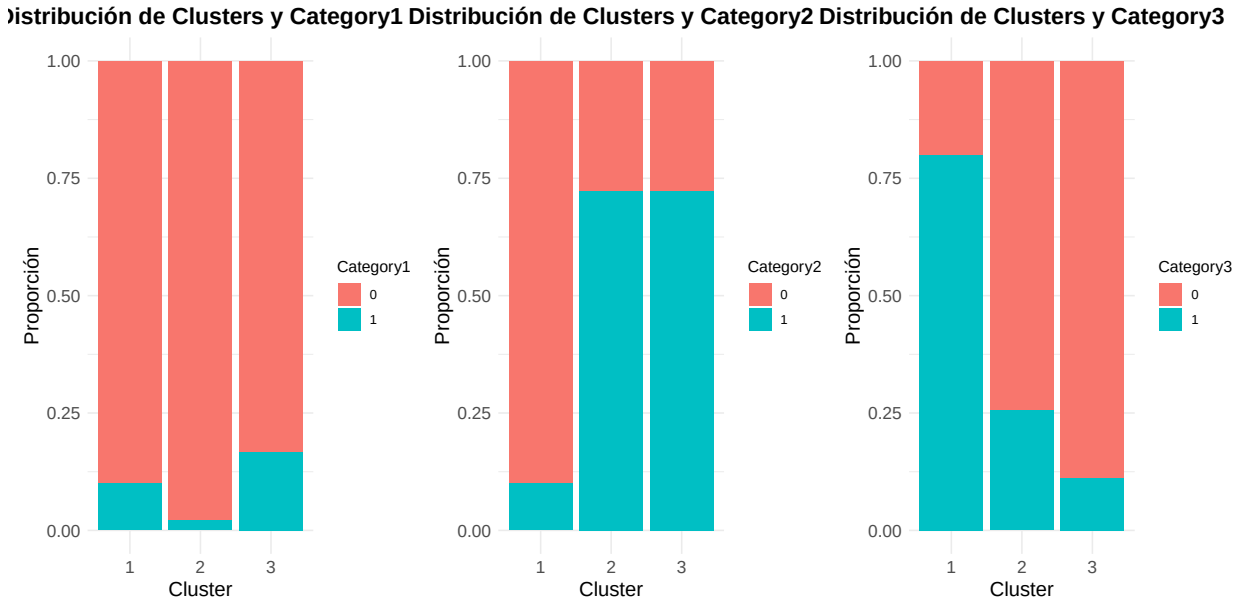
```
set.seed(123)  # Para reproducibilidad
kmeans_result <- kmeans(scaled_data, centers = 3)

# Ver los clusters asignados
table(kmeans_result$cluster)
```

```
##
##  1  2  3
## 10 47 18
```

```
# Agregar los clusters al dataset
data_with_clusters <- data.frame(data_clean_na, cluster_kmeans = kmeans_result$cluster)
```

Veamos la relación entre los clústeres del kmeans y las variables relacionadas con category

```r
data_with_clusters2 = cbind(data_with_clusters, data_clean_na2)
data_with_clusters2 <- data_with_clusters2[, !duplicated(t(data_with_clusters2))]
# Relación entre clusters y 'category1'
plot1 <- ggplot(data_with_clusters2, aes(x = as.factor(cluster_kmeans), fill = as.factor(category.Cama)
  geom_bar(position = "fill") +
  labs(
    title = "Distribución de Clusters y Category1",
    x = "Cluster",
    y = "Proporción",
    fill = "Category1"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text = element_text(size = 12),
    axis.title = element_text(size = 14)
  )

# Relación entre clusters y 'category2'
plot2 <- ggplot(data_with_clusters2, aes(x = as.factor(cluster_kmeans), fill = as.factor(category.Dormi
  geom_bar(position = "fill") +
  labs(
    title = "Distribución de Clusters y Category2",
    x = "Cluster",
    y = "Proporción",
    fill = "Category2"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text = element_text(size = 12),
    axis.title = element_text(size = 14)
  )

# Relación entre clusters y 'category3'
plot3 <- ggplot(data_with_clusters2, aes(x = as.factor(cluster_kmeans), fill = as.factor(category.Aparta
  geom_bar(position = "fill") +
  labs(
    title = "Distribución de Clusters y Category3",
    x = "Cluster",
    y = "Proporción",
    fill = "Category3"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text = element_text(size = 12),
    axis.title = element_text(size = 14)
  )

# Alineación horizontal con patchwork
plot1 + plot2 + plot3
```

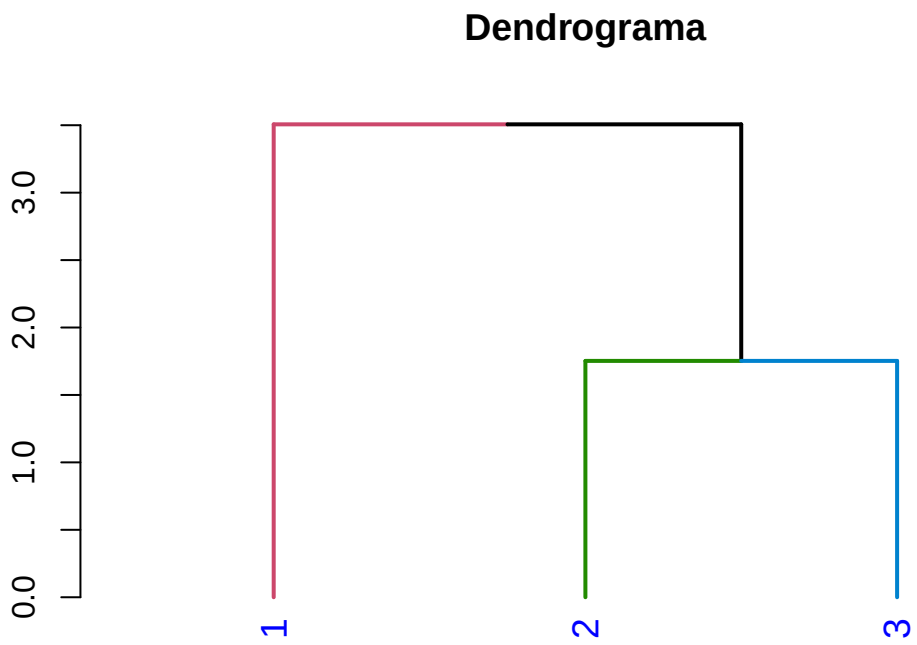


Aunque no de manera 100% correcta, aquí si hemos sido capaces de más o menos distinguir 3 clústeres con la categoría de los datos.

Pensamos que con técnicas aumento de datos como SMOTE, validación cruzada, o simplemente, con más datos podríamos llegar a tener unos clústeres que nos delimiten bien las categorías.

Vamos a representar los clusteres

```
# Obtener las coordenadas de los centros de los clusters
centros_kmeans <- kmeans_result$centers

# Calcular la matriz de distancias entre los centros
distancias <- dist(centros_kmeans)

jerarquia <- hclust(distancias, method = "complete")
dendrograma <- as.dendrogram(jerarquia)
# Mejorar el dendrograma
dendrograma %>%
  set("labels_col", "blue") %>%
  set("branches_k_color", k = 3) %>%
  set("branches_lwd", 2) %>%
  set("labels_cex", 1.2) %>%
  plot(main = "Dendrograma")
```



Modelo supervisado

```
## Convertir `tiene_registro` a factor (variable objetivo binaria)
data_clean_na2$tiene_registro <- as.factor(data_clean_na2$tiene_registro)
#
#
## Dividir en conjunto de entrenamiento y prueba
set.seed(123)
train_idx <- sample(seq_len(nrow(data_clean_na2)), size = 0.8 * nrow(data_clean_na2))
train_data <- data_clean_na2[train_idx, ]
test_data <- data_clean_na2[-train_idx, ]
```

Entrenamos el modelo logístico

Vamos a realize un modelo generalizado lineal (glm) con la variable objetivo `tiene_registro`, una binaria, usando todas las demás columnas (~ .) como predictores. La familia binomial sugiere que es un modelo de regresión logística.

```
## Ajustar el modelo de regresión logística
logistic_model <- glm(tiene_registro ~ ., data = train_data, family = binomial)
#
## Resumen del modelo
summary(logistic_model)
```

```
##
```

```
## Call:
## glm(formula = tiene_registro ~ ., family = binomial, data = train_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8041  -0.4325  -0.3286   0.1817   2.3276
##
## Coefficients: (2 not defined because of singularities)
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)          -2.095e+01  6.523e+03  -0.003    0.997
## rating                7.920e-01  2.558e+00   0.310    0.757
## n_rating              2.739e-03  4.241e-03   0.646    0.518
## tipo_anfitrion1       3.537e+00  1.468e+00   2.409    0.016 *
## price                 7.655e-03  2.088e-02   0.367    0.714
## category.Cama         1.492e+01  3.108e+03   0.005    0.996
## category.Dormitorio  -1.908e+00  1.193e+00  -1.599    0.110
## category.Apartamento         NA         NA      NA       NA
## Location.1            1.634e+01  6.523e+03   0.003    0.998
## Location.2            1.600e+01  6.523e+03   0.002    0.998
## Location.3            1.526e+01  6.523e+03   0.002    0.998
## Location.4                   NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 77.694  on 59  degrees of freedom
## Residual deviance: 37.663  on 50  degrees of freedom
## AIC: 57.663
##
## Number of Fisher Scoring iterations: 17
```

El hecho de que category.Apartamento aparezca como NA por ejemplo tiene relación a que la variable está perfectamente colineal con las demás variables (u otra combinación lineal de ellas). Esto significa que su información ya está capturada por otros predictores.

Simplificamos el modelo quitando la variable Localización y vamos a realizar validación cruzada con 5 pliegues donde los tamaños de los grupos fueron entre 47 y 49.

```
# Convertir `tiene_registro` a factor (variable objetivo binaria)
data_clean_na2$tiene_registro <- as.factor(data_clean_na2$tiene_registro)



# Dividir en conjunto de entrenamiento y prueba
data_clean_na2 = data_clean_na2 %>% select("tiene_registro","rating", "n_rating", "tipo_anfitrion","pri
set.seed(123)
train_idx <- sample(seq_len(nrow(data_clean_na2)), size = 0.8 * nrow(data_clean_na2))
train_data <- data_clean_na2[train_idx, ]
test_data <- data_clean_na2[-train_idx, ]




# Configurar el control para validación cruzada
```

```r
train_control <- trainControl(
  method = "cv",          # Método de validación cruzada
  number = 5,             # Número de folds (en este caso, 5-fold CV)
  savePredictions = TRUE, # Guardar predicciones para evaluar después
  classProbs = TRUE,      # Calcular probabilidades para la clasificación binaria
  summaryFunction = twoClassSummary # Métrica: AUC, sensibilidad, especificidad
)

# Convertir la variable objetivo en formato adecuado para caret
train_data$tiene_registro <- factor(train_data$tiene_registro, levels = c("0", "1"))
# Renombrar los niveles manualmente
train_data$tiene_registro <- factor(train_data$tiene_registro,
                      levels = c("0", "1"),  # Niveles originales
                      labels = c("Class0", "Class1"))  # Nuevos nombres

# Entrenar el modelo con validación cruzada
logistic_cv_model <- train(
  tiene_registro ~ .,              # Fórmula del modelo
  data = train_data,               # Datos de entrenamiento
  method = "glm",                  # Modelo de regresión logística
  family = "binomial",             # Familia binomial
  trControl = train_control,       # Control de validación cruzada
  metric = "ROC"                   # Métrica de evaluación
)

# Resumen del modelo entrenado
print(logistic_cv_model)
```

```
## Generalized Linear Model
##
## 60 samples
##  7 predictor
##  2 classes: 'Class0', 'Class1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 48, 48, 48, 47, 49
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8517857  0.8678571  0.63
```

```r
# Datos
resultados <- data.frame(
  Métrica = c("ROC (Curva Característica Operativa del Receptor)",
              "Sensibilidad",
              "Especificidad"),
  Valor = c(0.85, 0.87, 0.63),
  Descripción = c("Un buen rendimiento general",
                  "El modelo identifica correctamente el 87% de los casos positivos",
                  "El modelo identifica correctamente el 63% de los casos negativos")
)

# Mostrar en formato kable
```

Table 4: Resultados promedio del modelo sobre los pliegues.

Métrica	Valor	Descripción
ROC (Curva Característica Operativa del Receptor)	0.85	Un buen rendimiento general
Sensibilidad	0.87	El modelo identifica correctamente el 87% de los casos posi
Especificidad	0.63	El modelo identifica correctamente el 63% de los casos nega

Table 5: Tabla de Confusión

Predicción	Real	Frecuencia
Class0	0	8
Class1	0	2
Class0	1	2
Class1	1	3

```
kable(resultados, col.names = c("Métrica", "Valor", "Descripción"),
      caption = "Resultados promedio del modelo sobre los pliegues.")
```

```
# Evaluar el modelo en test_data
predicciones_test <- predict(logistic_cv_model, test_data)
# Crear una tabla de contingencia
tabla_confusion <- table(Predicción = predicciones_test,
                         Real = test_data$tiene_registro)

tabla_confusion_df <- as.data.frame(tabla_confusion)

# Mostrar la tabla de confusión con kable
kable(tabla_confusion_df, col.names = c("Predicción", "Real", "Frecuencia"),
      caption = "Tabla de Confusión")
```

Árboles de decisión

Ahora queremos predecir el precio de la vivienda usando todas las variables menos la localización.

```
#
# Convertir la columma 'Class' de caracteres a factores (si no lo está ya)
train_data$tiene_registro <- factor(train_data$tiene_registro,
                                    levels = c("Class0", "Class1"))
test_data$tiene_registro <- factor(test_data$tiene_registro,
                                   levels = c("Class0", "Class1"))
# Convertir los factores en valores binarios (0 y 1)
train_data$tiene_registro <- as.numeric(train_data$tiene_registro) - 1

test_data$tiene_registro <- as.numeric(test_data$tiene_registro) - 1
# Entrenar un modelo de árbol de decisión
modelo_arbol <- rpart(price ~ tiene_registro + rating+n_rating+tipo_anfitrion+
                        category.Cama+category.Dormitorio
+ category.Apartamento, data = train_data, method = "anova")

# Ver el árbol de decisión
print(modelo_arbol)
```

```
## n= 60
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 60 54021.000 56.50000
##   2) category.Apartamento< 0.5 44 14433.730 44.77273
##     4) rating< 4.835 28  1304.107 38.17857 *
##     5) rating>=4.835 16  9781.438 56.31250 *
##   3) category.Apartamento>=0.5 16 16895.000 88.75000 *
```

```
# Hacer predicciones en los datos de prueba
predicciones_arbol <- predict(modelo_arbol, newdata = test_data)

# Calcular el error medio absoluto (MAE) para evaluar el modelo
mae_arbol <- mean(abs(predicciones_arbol - test_data$price))
mape_arbol <- mean(abs((predicciones_arbol - test_data$price) / test_data$price)) * 100
# Crear un data frame con los resultados
resultados <- data.frame(
  Métrica = c("Error Medio Absoluto", "Error Medio Absoluto en Porcentaje"),
  Valor = c(mae_arbol, mape_arbol)
)

# Mostrar los resultados con kable
kable(resultados, col.names = c("Métrica", "Valor"))
```

Métrica	Valor
Error Medio Absoluto	20.55357
Error Medio Absoluto en Porcentaje	41.43866

Pruebas de contraste

No lo vamos a realizar sobre todas las variables pero haremos algunas pruebas a modo demostrativo.

Estudio de la normalidad

```
shapiro_test_rating <- shapiro.test(data_clean_na2$rating)


shapiro_test_n_rating <- shapiro.test(data_clean_na2$n_rating)



shapiro_test_price <- shapiro.test(data_clean_na2$price)


# Crear un data frame con los resultados
resultados_pruebas <- data.frame(
  Prueba = c("Shapiro Test cont1", "Shapiro Test cont2", "Shapiro Test cont3"),
  p_value = c(shapiro_test_rating$p.value, shapiro_test_n_rating$p.value,
              shapiro_test_price$p.value),
  Descripción = c(
    ifelse(shapiro_test_rating$p.value > 0.05, "Normalidad: Aceptar normalidad",
           "Normalidad: Rechazar normalidad"),
```

Table 6: Resultados de las Pruebas de Contraste de Hipótesis

Prueba	p_value	Descripción
Shapiro Test cont1	1e-07	Normalidad: Rechazar normalidad
Shapiro Test cont2	0e+00	Normalidad: Rechazar normalidad
Shapiro Test cont3	0e+00	Normalidad: Rechazar normalidad

```
    ifelse(shapiro_test_n_rating$p.value > 0.05, "Normalidad: Aceptar normalidad",
           "Normalidad: Rechazar normalidad"),
    ifelse(shapiro_test_price$p.value > 0.05, "Normalidad: Aceptar normalidad",
           "Normalidad: Rechazar normalidad")

  )
)

kable(resultados_pruebas,
      caption = "Resultados de las Pruebas de Contraste de Hipótesis")%>%
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE)
```

Verificar la homocedasticidad entre los grupos de las variables categóricas

Vamos a usar la prueba de Levene para una variable continua y una categórica

```
levene_test_tiene_registro_rating <- leveneTest(rating ~ tiene_registro,
                                                data = data_clean_na2)
levene_test_tiene_registro_price <- leveneTest(price ~ tiene_registro,
                                               data = data_clean_na2)

levene_test_tipo_anfitrion_rating <- leveneTest(rating ~ tipo_anfitrion,
                                                data = data_clean_na2)
levene_test_tipo_anfitrion_price <- leveneTest(price ~ tipo_anfitrion,
                                               data = data_clean_na2)


# Crear un data frame con los resultados
resultados_pruebas <- data.frame(
  Prueba = c(
             "Levene Test rating ~ tiene_registro",
             "Levene Test price ~ tiene_registro",
             "Levene Test rating ~ tipo_anfitrion",
             "Levene Test price ~ tipo_anfitrion"
         ),
  p_value = c(
              levene_test_tiene_registro_rating$`Pr(>F)`[1],
              levene_test_tiene_registro_price$`Pr(>F)`[1],
              levene_test_tipo_anfitrion_rating$`Pr(>F)`[1],
              levene_test_tipo_anfitrion_price$`Pr(>F)`[1]
              ),
  Descripción = c(

    ifelse(levene_test_tiene_registro_rating$`Pr(>F)`[1] > 0.05,
           "Homocedasticidad: Aceptar", "Homocedasticidad: Rechazar"),
    ifelse(levene_test_tiene_registro_price$`Pr(>F)`[1] > 0.05,
```

Table 7: Resultados de las Pruebas de Contraste de Hipótesis

Prueba	p_value	Descripción
Levene Test rating ~ tiene_registro	0.4220975	Homocedasticidad: Aceptar
Levene Test price ~ tiene_registro	0.0000253	Homocedasticidad: Rechazar
Levene Test rating ~ tipo_anfitrion	0.4551823	Homocedasticidad: Aceptar
Levene Test price ~ tipo_anfitrion	0.0133969	Homocedasticidad: Rechazar

```
          "Homocedasticidad: Aceptar", "Homocedasticidad: Rechazar"),
        ifelse(levene_test_tipo_anfitrion_rating$`Pr(>F)`[1] > 0.05,
               "Homocedasticidad: Aceptar", "Homocedasticidad: Rechazar"),
    ifelse(levene_test_tipo_anfitrion_price$`Pr(>F)`[1] > 0.05,
           "Homocedasticidad: Aceptar", "Homocedasticidad: Rechazar")

  )
)
kable(resultados_pruebas,
      caption = "Resultados de las Pruebas de Contraste de Hipótesis")%>%
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE)
```

Contrastar las medias de las variables continuas entre los grupos categóricos

Como las variables no son normales vamos a usar Kruskal-Wallis

```
kruskal_result_rating_tiene_registro <- kruskal.test(rating ~ tiene_registro,
                                                     data =data_clean_na2)
kruskal_rating_tipo_anfitrion <- kruskal.test(rating ~ tipo_anfitrion,
                                              data = data_clean_na2)




resultados_pruebas <- data.frame(
  Prueba = c(
             "Kruskal-Wallis rating ~ tiene_registro",
             "Kruskal-Wallis rating ~ tipo_anfitrion"),
  p_value = c(
              kruskal_result_rating_tiene_registro$p.value,
              kruskal_rating_tipo_anfitrion$p.value),
  Descripción = c(
    ifelse(kruskal_result_rating_tiene_registro$p.value > 0.05,
           "Medias iguales (Kruskal-Wallis)", "Medias diferentes (Kruskal-Wallis)"),
    ifelse(kruskal_rating_tipo_anfitrion$p.value > 0.05,
           "Medias iguales (Kruskal-Wallis)", "Medias diferentes (Kruskal-Wallis)")
  )
)

kable(resultados_pruebas,
      caption = "Resultados de las Pruebas de Contraste de Hipótesis") %>%
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE)
```

Table 8: Resultados de las Pruebas de Contraste de Hipótesis

Prueba	p_value	Descripción
Kruskal-Wallis rating ~ tiene_registro	0.1943290	Medias iguales (Kruskal-Wallis)
Kruskal-Wallis rating ~ tipo_anfitrion	0.0428213	Medias diferentes (Kruskal-Wallis)