Práctica 2: Booking Data Cleaning

**Gerard Alcalde and Guillem Rochina**

2022-12-24

## 1. DESCRIPCIÓN DEL DATASET

### ¿Por qué es importante y qué pregunta/problema pretende responder?

#### 1.1. Descripción general

El conjunto de datos a tratar es el obtenido como resultado de la práctica anterior de Web Scraping. El dataset presenta los datos e indicadores más relevantes de cada uno de los hoteles encontrados en función de determinados criterios de búsqueda (ciudad, fecha de check-in, fecha de check-out, número de adultos, niños y habitaciones). En este dataset en concreto presentamos datos para distintas fechas (diciembre, marzo y junio), ciudades (barcelona, madrid, valencia), número de adultos, niños y habitaciones a fin de tener una muestra más amplia e informativa que la que nos daría una búsqueda con tan sólo unos parámetros fijos.

A continuación, se realiza una breve descripción de cada una de las variables disponibles antes de realizar la limpieza:

* **Name (str)**: Nombre del hotel.
* **City (str)**: Nombre de la ciudad donde se realiza la búsqueda.
* **Check-in (str)**: Fecha de entrada al hotel.
* **Check-out (str)**: Fecha de salida del hotel.
* **Adults (int)**: Nº de adultos para los que se realiza la reserva.
* **Num\_rooms (int)**: Nº de habitaciones reservadas.
* **Address (str)**: Dirección postal del hotel, nombre de calle, barrio de la ciudad, código postal, etc.
* **Hotel\_coordinates (str)**: Latitud y longitud de la ubicación del hotel.
* **Hotel\_score (int)**: Nota general que recibe el hotel por los usuarios.
* **Hotel\_scores (dict)**: Puntuaciones de cada una de las dimensiones que puede valorar un usuario.
* **Hotel description (list)**: Descripción aportada por el propietario del hotel
* **Features (list)**: Lista de servicios añadidos del hotel.
* **Room\_data (dict)**: Conjunto de diccionarios que recogen las características de cada una de las habitaciones disponibles en el hotel.
* **Page\_count (int)**: Posición en la que ha aparecido el hotel en el buscador.
* **Current\_page (int)**: Página en la que ha aparecido el hotel en el buscador.

#### 1.2. Importancia y problema a responder

Aparecer en los primeros resultados de una búsqueda de un usuario por internet se ha convertido en los últimos años en una de las preocupaciones principales de cualquier negocio que busca hacerse un hueco en el mundo digital. Este fenómeno explica la proliferación de herramientas como el SEO y el SEM, mediante las cuales se intenta innovar (más allá del pago de tarifas para aparecer en las primeras páginas de búsqueda) a fin de lograr un mejor “posicionamiento” en buscadores como Google o Bing. El negocio hotelero no es una excepción en este caso.

No obstante, el uso de dichas herramientas no resulta útil si nos enfrentamos a un buscador dentro de una página web, como el caso que nos ocupa con Booking.com. ¿Qué hacer entonces? ¿Cómo mejorar el posicionamiento dentro de la web?

Asimismo, la fijación de precios o “pricing” puede ser una tardea ardua en los últimos años, pues la aparición de apartamentos vacacionales a través de plataformas como AirBnB ha incrementado de manera significativa la competencia a la que se deben enfrentar los propietarios de los hoteles, lo que influye en última instancia los precios a establecer por sus servicios. Dicho esto, ¿Qué precios se deben fijar dadas las características de un hotel? ¿Añadir un servicio adicional permitiría a un hotel elevar los precios de sus habitaciones?.

Ante las preguntas presentadas en los párrafos anteriores, el objetivo de este proyecto es dual. Por un lado se busca obtener las características que hacen que un hotel esté mejor posicionado que otro en el metabuscador de Booking.com y, por el otro, se persigue desarrollar un modelo que indique qué precios se deberían establecer dadas las características de una habitación y qué elementos pueden incrementar el precio de forma relevante.

En conclusión, el presente proyecto pretende dar continuidad a la Práctica 1, explotando los datos que se obtuvieron mediante web scrapping para ayudar a superar dos de los desafíos más relevantes para el sector hotelero.

## 2. LIMPIEZA DE LOS DATOS

A fin de iniciar el proceso de limpieza de datos, en primer lugar, se debe realizar la lectura del fichero csv que obtuvimos en la práctica anterior. Como resultado de la función de R-base “read.csv” obtenemos un objeto data.frame, que manipularemos en la presente sección.

# Lectura y breve descripción de los datos  
  
booking <- read.csv("hotels\_data.csv", header = TRUE)  
summary(booking)

Tanto con la función “summary” como con la función “class” nos muestran que a cada variable se le ha asignado o bien la clase de entero o la de string. En concreto, muchas de las variables interpretadas como string se tratan de diccionarios y listas, factor que deberemos tener en cuenta a la hora de trabajar con la limpieza de estas variables.

# Tipo de dato asignado a cada variable  
  
res <- sapply(booking, function(x) class(x))  
kable(data.frame(var=names(res), clase=as.vector(res)))

| **var** | **clase** |
| --- | --- |
| name | character |
| city | character |
| check.in | character |
| adults | integer |
| children | integer |
| check.out | character |
| num\_rooms | integer |
| address | character |
| hotel\_coordinates | character |
| hotel\_score | numeric |
| hotel\_scores | character |
| hotel\_description | character |
| features | character |
| room\_data | character |
| page\_count | integer |
| current\_page | integer |
| in\_page\_count | integer |
| search\_date | character |

#### 2.1. Transformación de las variables en el formato adecuado

Como se ha comentado, existen algunas variables que presentan todavía una estructura no válida para tratarlos estadísticamente o introducirlos en un modelo de data mining. En esta sección nos encargamos de tratar dichas columnas a fin de obtener nuevas variables con un formato adecuado, eliminando a su vez cualquier información que no nos sea de utilidad.

En primer lugar, cabe destacar que no todas las columnas que tenemos disponibles en el dataset nos ofrecen información relevante para nuestro problema. Por tanto, a fin de reducir la dimensionalidad de este, nos encargamos de eliminar las columnas con información redundante (name y search\_date).

# Seleccionamos con la función "select" de dplyr todas las columnas excepto name y search\_date, que nos van a ser de poca utilidad.  
  
booking <- booking %>%  
 dplyr::select(-c("name", "search\_date"))

La columna hotel\_coordinates incluye tanto los valores de latitud como los de longitud, separados por una coma. Dado que ambos dos valores nos indican información distinta, pues la latitud nos proporcionar información de la posición en dirección norte o sur del ecuador y la longitud información de la posición en dirección este u oeste, en este caso hemos optado por separarlos en dos columnas distintas.

# Separamos los valores de la variable hotel coordinates en dos nuevas columnas, latitud y longitud  
# La columna hotel\_coordinates se elimina en el proceso  
  
booking <- booking %>%  
 separate(hotel\_coordinates, c("latitude", "longitude"), sep = ",")

booking$latitude <- as.numeric(booking$latitude)  
booking$longitude <- as.numeric(booking$longitude)

Por su parte, hotel\_scores alberga diccionarios en cada uno de los registros. Cada diccionario consta del nombre de cada dimensión (característica a valorar) como clave y la valoración que recibe como valor. Dado que R lo ha interpretado como un string, nuestro enfoque en su limpieza se ha basado en separar la información con comas y, una vez obtenidas todas las columnas resultantes, extraer los datos numéricos de cada registro para quedarnos solo con los scores de cada dimensión.

# Definimos un vector con los nombres de las columnas a crear  
  
columns <- c("staff\_score", "facilities\_score", "cleanliness\_score", "comfort\_score", "value\_for\_money\_score", "location\_score", "free\_wifi\_score")  
  
# Separamos los strings (diccionarios) en función de las comas y creamos las columnas correspondientes. La primera columna creada es una NA porque todos los primeros registros en cada diccionario están vacíos (por un fallo en la extracción de datos). Estableciendo la columna como NA, indicamos a la función separate que ignore dichos datos y no cree columna alguna para ellos.  
  
booking <- booking %>%  
 separate(hotel\_scores, c(NA, "staff\_score", "facilities\_score", "cleanliness\_score", "comfort\_score", "value\_for\_money\_score", "location\_score", "free\_wifi\_score"), sep = ",")

# Extraemos los valores numéricos de cada columna con la función del paquete readr, "parse\_number"  
  
booking[columns] <- apply(booking[columns], 2, readr::parse\_number)

Dado que de las fechas tan sólo nos interesa el mes en el que se realiza la reserva (un alojamiento con unas características determinadas podría estar mejor posicionado en un determinado mes que en otro o tener un precio distinto) procedemos a extraer dicha información de las variables check-in y check-out, eliminando estas últimas en el proceso.

booking <- booking %>%  
 # Separamos el nombre del mes del resto de elementos del registro para check\_in y check\_out. Eliminamos las columnas en el proceso.  
 separate(check.in, c(NA, "month\_in", NA), sep = "-") %>%  
 separate(check.out, c(NA, "month\_out", NA), sep = "-")

De la columna address tan sólo nos interesa el código postal (el barrio no se incluye en todos los registros a diferencia del CP, así que hemos optado por quedarnos con estos valores), por lo que extraemos dicha información para crear una nueva columna llamada postal\_code, eliminando la información redundante en el proceso.

# Extraemos los valores de codigo postal de la columna addres utilizando expressiones regulares  
booking <- booking %>%  
 extract(address, c("postal\_code"), regex = "( [0-9]{5} )")

Los features, entendidos como servicios adicionales más allá de la propia habitación, fueron guardados como una lista. No obstante, de nuevo nos encontramos como R lo ha identificado como string. Es por ello por lo que realizamos un tratamiento similar al de la columna de scores. En este caso creamos tantas variables dummy (columnas dicotómicas con valores de 1, en caso de que el feature exista en dicho registro, o 0, en caso de que no exista) como servicios adicionales de hotel consideramos interesantes.

# Definimos las features que consideramos que son interesantes. Utilizamos este formato para filtrar por expresión regular.  
feature\_list <- c("('Free Wifi')", "('Air conditioning')", "('24-hour front desk')", "('Safe')", "('Heating')", "('Elevator')", "('Private Bathroom')", "('Non-smoking rooms')", "('Aparments')", "('City view')","('Kitchen')", "('Pet Friendly')", "('Swimming pool')", "('Balcony')")  
  
# Creamos un bucle en el que se recorre cada elemento de la lista de features  
for (feature in feature\_list){  
 # Se eliminan los parentésis y comillas de la variable local feature para crear el nombre de la columna.  
 col\_name <- str\_replace(str\_replace(str\_to\_lower(str\_extract(feature, "([A-Z][a-z]\*( |-)?[A-Z]?[a-z]\* ? ?[a-z]\*)")), " ", "\_"), "-", "\_")  
 booking <- booking %>%  
 # Extraemos el nombre del feature de cada uno de los registros (strings). En caso de que no encuentre ningún valor devuelve un NA.  
 extract(features, c(col\_name), regex = feature, remove = FALSE) %>%  
 # Transformamos la columna que acaba de ser creada para que indique con un 1 si el registro tenia dicho servicio y 0 si el valor era NA (no tenida dicho servicio)  
 mutate\_(.dots = setNames(list(paste0("as.integer(!is.na(",col\_name,"))")), col\_name))  
}

# Eliminamos la columna de features para finalizar  
booking <- booking %>%  
 dplyr::select(-c("features"))

Por otro lado, nos encontramos con la columna que incluye la descripción del hotel, esta descripción es muy amplia y un análisis profundo de ella requeriría de técnicas de NLP que no son el objetivo de esta práctica. No obstante, consideramos que la longitud de la descripción del hotel sí que puede tener una relación con la posición en el buscador, por lo que transformaremos esta variable a una nueva variable que contenga el número total de palabras que contiene la descripción, eliminando de nuevo dicha columna en el proceso.

# Evaluamos la longitud de la descripción del hotel  
booking$length\_description <- lengths(gregexpr("\\W+", booking$hotel\_description)) + 1  
  
# Eliminamos la columna hotel\_description  
booking <- booking %>%  
 dplyr::select(-c("hotel\_description"))

Finalmente, en la columna Room\_data, se incluye mucha información referente a los tipos de habitaciones disponibles, precios, características, etc. Esta se había extraído de esta forma con el objetivo de procesarla mediante un diccionario de Python. No obstante, al haber optado por un procesado con R este se complicará un poco más. En esta columna tenemos distintos datos que pueden ser muy relevantes, de los cuales extraeremos los siguientes:

* **min\_price**: Precio de la habitación más económica.
* **mean\_price**: Precio medio de las habitaciones del hotel.
* **suite\_available**: Una de las habitaciones es de tipo suite.
* **apartment\_available**: Una de las habitaciones es de tipo apartamento.
* **free\_cancelation\_available**: Alguna de las habitaciones tiene cancelación gratuita.

# Función que encuentra el precio mínimo de la habitación dado un string con la información de las habitaciones  
find\_min\_price <- function(text) {  
 if (text=="{}"){  
 return(NA)  
 }  
 a <- gregexpr("([0-9]\*', 'room\_capacity)", text)  
   
 min\_price <- NA  
 for (value in a[[1]]){  
 substring <- substr(text, value, value+20)  
 price <- as.numeric(strsplit(substring, "'")[[1]][1])  
 if (is.na(min\_price)) {  
 min\_price <- price  
 }  
   
 if (price < min\_price) {  
 min\_price <- price  
 }   
 }  
 if (min\_price == 0){  
 return(NA)  
 }  
 else {  
 return(min\_price)  
 }  
}  
  
# Aplicamos la función a todos los valores  
min\_price\_vector <- c()  
for (i in seq(1, length(booking$room\_data))) {  
 min\_price\_vector <- append(min\_price\_vector, find\_min\_price(booking$room\_data[i]))  
}  
  
# Guardamos los resultados en una nueva columna.  
booking$min\_price <- min\_price\_vector  
  
# Función que encuentra el precio medio del hotel dado un string con la información de las habitaciones  
find\_mean\_price <- function(text) {  
 if (text=="{}"){  
 return(NA)  
 }  
 a <- gregexpr("([0-9]\*', 'room\_capacity)", text)  
 price\_vec <- c()  
 for (value in a[[1]]){  
 substring <- substr(text, value, value+20)  
 price <- as.numeric(strsplit(substring, "'")[[1]][1])  
 price\_vec <- append(price\_vec, price)  
 }  
 if (mean(price\_vec, na.rm = TRUE) == 0){  
 return(NA)  
 }  
 {  
 return(round(mean(price\_vec, na.rm = TRUE), 2))  
 }  
}  
  
# Aplicamos la función a todos los valores  
  
mean\_price\_vector <- c()  
for (i in seq(1, length(booking$room\_data))) {  
 mean\_price\_vector <- append(mean\_price\_vector, find\_mean\_price(booking$room\_data[i]))  
}  
  
# Guardamos los resultados en una nueva columna.  
booking$mean\_price <- mean\_price\_vector  
  
# Buscamos si hay una habitación en suite  
is\_suite\_vector <- c()  
for (i in seq(1, length(booking$room\_data))) {  
 is\_suite <- gregexpr("(suite)", booking$room\_data[i])[[1]][1]  
   
 if (is\_suite==-1) {  
 is\_suite\_vector <- append(is\_suite\_vector, 0)  
 } else {  
 is\_suite\_vector <- append(is\_suite\_vector, 1)  
 }  
}  
  
# Guardamos los resultados en una nueva columna.  
booking$is\_suite <- is\_suite\_vector  
  
# Buscamos si hay opción de apartamento  
is\_apartment\_vector <- c()  
for (i in seq(1, length(booking$room\_data))) {  
 is\_apartment <- gregexpr("(Apartment)", booking$room\_data[i])[[1]][1]  
   
 if (is\_apartment==-1) {  
 is\_apartment\_vector <- append(is\_apartment\_vector, 0)  
 } else {  
 is\_apartment\_vector <- append(is\_apartment\_vector, 1)  
 }  
}  
  
# Guardamos los resultados en una nueva columna.  
booking$is\_apartment <- is\_apartment\_vector  
  
# Buscamos si tiene cancelación gratuita  
free\_cancelation\_vector <- c()  
for (i in seq(1, length(booking$room\_data))) {  
 free\_cancelation <- gregexpr("(Free cancellation)", booking$room\_data[i])[[1]][1]  
   
 if (free\_cancelation==-1) {  
 free\_cancelation\_vector <- append(free\_cancelation\_vector, 0)  
 } else {  
 free\_cancelation\_vector <- append(free\_cancelation\_vector, 1)  
 }  
}  
  
# Guardamos los resultados en una nueva columna.  
booking$has\_free\_cancelation <- free\_cancelation\_vector  
  
# Eliminamos la columna correspondiente a room\_data  
booking <- booking %>%  
 dplyr::select(-c("room\_data"))

#### 2.2. Ceros y elementos vacíos

Dando paso a el tratamiento de ceros y elementos vacios, lo primero que debemos hacer es identificar aquellas variables en las que encontramos valores faltantes. Como observamos, "postal\_code" (19), "longitude" (16), "hotel\_score" (8), min y mean\_price y el resto de variables relacionadas con los scores (60) presentan valores faltantes, destacando free\_wifi\_score con 423 registros faltantes, consecuencia con toda seguridad de la ausencia de este servicio en dichos hoteles.

sapply(booking, function(y) sum(length(which(is.na(y)))))

## month\_out num\_rooms postal\_code   
## 0 0 19   
## latitude longitude hotel\_score   
## 16 16 8   
## staff\_score facilities\_score cleanliness\_score   
## 60 60 60   
## comfort\_score value\_for\_money\_score location\_score   
## 60 60 60   
## free\_wifi\_score balcony swimming\_pool   
## 423 0 0

Por lo que respecta a la variable postal\_code encontramos que en todos aquellos registros donde se encuentan NAs, también hallamos datos faltantes en la mayoría del resto de columnas. Por tanto, dado que dichos registros no nos aportan información de valor, procedemos a eliminarlos del dataset.

# Comprobamos algunos de los registros donde hay datos faltantes de postal\_code  
booking[is.na(booking$postal\_code),] %>%  
 dplyr::select(c("city", "month\_in", "postal\_code", "longitude", "hotel\_score", "staff\_score", "location\_score")) %>%  
 head(5)

## city month\_in postal\_code longitude hotel\_score staff\_score  
## 202 Barcelona December <NA> NA -1 NA  
## 203 Barcelona December <NA> NA -1 NA  
## 223 Barcelona December <NA> NA -1 NA  
## 224 Barcelona December <NA> NA -1 NA  
## 225 Barcelona December <NA> NA -1 NA

# Filtramos los datos para eliminar aquellos registros que tienen NAs en postal\_code  
booking <- booking[!is.na(booking$postal\_code),]

Si comprobamos de nuevo que columnas presentan todavía datos faltantes, observamos que en algunos casos (como longitude) hemos logrado eliminar todos los datos faltantes, mientras que en el resto se han reducido en 16 unidades el número de registros con NAs, pues como se había comentado todos aquellos registros con ausencia de postal\_code, también carecía del resto de datos.

sapply(booking, function(y) sum(is.na(y)))

## latitude longitude hotel\_score   
## 0 0 5   
## staff\_score facilities\_score cleanliness\_score   
## 44 44 44   
## comfort\_score value\_for\_money\_score location\_score   
## 44 44 44   
## free\_wifi\_score balcony swimming\_pool   
## 407 0 0

A continuación, analizamos los datos faltantes de hotel\_score. En este caso, encontramos que la ausencia de datos se registra de dos formas distintas, o bien con la introducción de NAs o bien con el valor -1. Por tanto, dado que el tratamiento en ambos casos va a ser el mismo, transformamos todos los valores -1 en NAs para poder llevar a cabo el siguiente proceso de inputación.

# Comprobamos algunos de los registros donde hay datos faltantes de hotel\_score  
booking[is.na(booking$hotel\_score),] %>%  
 dplyr::select(c("city", "month\_in", "hotel\_score", "staff\_score", "location\_score", "longitude")) %>%  
 head(5)

## city month\_in hotel\_score staff\_score location\_score longitude  
## 310 Barcelona December NA 9.4 9.4 2.1787906  
## 844 Valencia December NA 7.7 9.5 -0.3775288  
## 958 Valencia December NA 9.2 9.3 -0.3247243  
## 1210 Barcelona March NA 8.1 8.6 2.1429321  
## 2708 Barcelona June NA 8.1 8.6 2.1429321

# Comprobamos algunos de los registros donde hay datos faltantes de hotel\_score  
booking[booking$hotel\_score == "-1",] %>%  
 dplyr::select(c("city", "month\_in", "hotel\_score", "staff\_score", "location\_score", "longitude")) %>%  
 tail(5)

## city month\_in hotel\_score staff\_score location\_score longitude  
## 3690 Madrid June -1 NA NA -3.699663  
## 3736 Madrid March -1 NA NA -3.709937  
## 3855 Madrid March -1 NA NA -3.700559  
## 3877 Madrid March -1 NA NA -3.699663  
## 3909 Madrid March -1 NA NA -3.699676

booking[booking$hotel\_score == "-1",] %>%  
 dplyr::select(c("city", "month\_in", "hotel\_score", "staff\_score", "free\_wifi\_score", "free\_wifi")) %>%  
 tail(10)

## city month\_in hotel\_score staff\_score free\_wifi\_score free\_wifi  
## 3511 Madrid June -1 NA NA 0  
## 3544 Madrid June -1 NA NA 0  
## 3625 Madrid June -1 NA NA 0  
## 3667 Madrid June -1 NA NA 0  
## 3668 Madrid June -1 NA NA 0

# Tranformamos los valores -1 en NAs para poder llevar a cabo el proceso de inputación correctamente.  
booking$hotel\_score <- ifelse(booking$hotel\_score == "-1", NA, booking$hotel\_score)

Dado que en este caso tan sólo encontramos datos faltantes en las columnas de score no consideramos eliminar estos registros, sino llevar a cabo una imputación. Aunque se podría haber utilizado una metología más simple como el uso de la media o la mediana, hemos optado por hacer uso de un método más complejo, basado en la similitud entre registros, en concreto llevar a cabo una imputación mediante el algoritmo de K vecinos más próximos. En este caso, la función se encarga de devolver los mismos valores en caso de que ya existan en el dataset y el valor de los registros más “similares” en el caso de un registro con NAs. No obstante, en el caso de “free\_wifi\_score” tan sólo introduciremos el valor de 0 en aquellos datos con ausencia de valores, pues tras analizar más en detenimiento los registros, aquellos con datos faltantes de free\_wifi\_score tienen un valor igual a 0 en la columna free\_wifi, lo que nos indica ausencia de este servicio en el hotel.

# Imputamos los datos con K Neirest Neighbours  
booking$hotel\_score <- kNN(booking)$hotel\_score  
booking$staff\_score <- kNN(booking)$staff\_score   
booking$facilities\_score <- kNN(booking)$facilities\_score  
booking$cleanliness\_score <- kNN(booking)$cleanliness\_score  
booking$comfort\_score <- kNN(booking)$comfort\_score  
booking$value\_for\_money\_score <- kNN(booking)$value\_for\_money\_score  
booking$location\_score <- kNN(booking)$location\_score  
# Imputamos valores de 0 a la columan free\_wifi\_score  
booking$free\_wifi\_score <- ifelse(is.na(booking$free\_wifi\_score), 0, booking$free\_wifi\_score)

En el caso de los precios, encontramos NAs en aquellos registros que presentaban valores iguales a 0 (no son valores lógicos). En este caso también encontramos que los únicos valores faltantes en estos registros corresponden a la columna de precio, por lo que procedemos de la misma forma que con los scores.

booking[is.na(booking$min\_price),] %>%  
 dplyr::select(c("city", "month\_in", "hotel\_score", "mean\_price", "min\_price")) %>%  
 tail(10)

## city month\_in hotel\_score mean\_price min\_price  
## 2767 Barcelona June 8.8 613.00 NA  
## 3219 Valencia March 9.6 575.00 NA  
## 3369 Valencia March 8.7 571.26 NA  
## 3467 Madrid June 8.9 517.71 NA  
## 3558 Madrid June 8.0 482.90 NA  
## 3620 Madrid June 9.5 398.00 NA  
## 3698 Madrid June 8.2 460.10 NA  
## 3846 Madrid March 8.8 474.82 NA  
## 3847 Madrid March 8.6 399.72 NA  
## 3849 Madrid March 8.9 517.71 NA

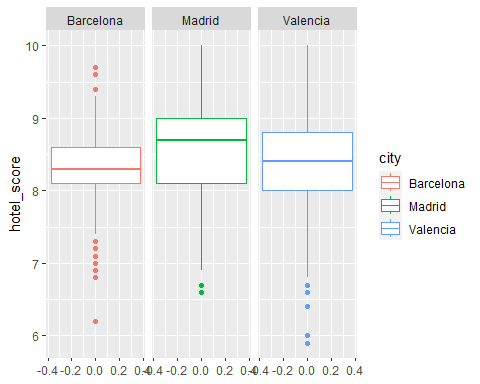
# Imputamos los datos con K Neirest Neighbours  
booking$mean\_price <- kNN(booking)$mean\_price  
booking$min\_price <- kNN(booking)$min\_price

Una última comprobación nos revela que el dataset ya no cuenta con datos faltantes, con independencia del formato que pudieran presenta (strings vacías, valores de -1 en integers, o simplemente NAs).

#### 2.3. Valores extremos

Los outliers son aquellos datos extremos que dada su distancia con respecto al grueso de la distribución a priori resultan no ser congruentes si los comparamos con la población/muestra analizada. Existen varias vías para detectarlos, no obstante, el método más rápido y común se basa en el uso de boxplots, donde se detectan aquellos valores que se sitúan 1.5 veces más allá del rango intercuartílico con respecto a la mediana. Asimismo, en este trabajo también haremos uso de la función “boxplots.stats()” a fin de detectar, con un output más allá del visual, los outliers que encontramos usando boxplots. Dado que muchas de las variables que hemos generado en la fase de limpieza son dicotómicas, tan solo nos encargaremos de comprobar si existen valores extremos en las columnas de **num\_rooms**, **latitude**, **longitude**, **hotel\_score** y el resto de **scores del hotel**, **length\_description**, **min\_price** y **mean\_price**. A continuación, se muestra tan sólo la comprobación de una variable a fin de ajustarnos al límite de páginas:

ggplot(booking, aes(y = hotel\_score, color = city))+  
 geom\_boxplot() +  
 facet\_grid(~city)



boxplot.stats(booking$hotel\_score)$out

## [1] 6.9 6.9 6.9 6.9 6.9 10.0 6.6 6.6 5.9 7.0 6.6 6.0 6.6 5.9 6.7  
## [16] 10.0 5.9 7.0 6.8 6.6 6.7 6.9 6.0 6.9 6.4 10.0 6.6 6.6 10.0 7.0  
## [31] 6.9 6.9 6.9 10.0 7.0 6.6 7.0 6.6 6.7 7.0 6.4 7.0 6.9 6.8 10.0  
## [46] 6.9 6.9 7.0 6.2 6.8 7.0 6.6 6.7 7.0 6.9 6.9 6.9 6.9 7.0

booking %>%  
 filter(hotel\_score <= 7.0) %>%  
 dplyr::select(c("hotel\_score", "facilities\_score", "cleanliness\_score", "location\_score")) %>%  
 sample\_n(5)

## hotel\_score facilities\_score cleanliness\_score location\_score  
## 1 6.9 6.8 7.5 8.2  
## 2 7.0 6.8 7.2 7.6  
## 3 6.7 6.3 6.8 9.0  
## 4 7.0 6.8 7.3 8.8  
## 5 6.6 6.5 7.0 8.9

#### 2.4. Exportación de los datos postprocesados

#write.csv(booking, "hotel\_data\_processed.csv")

## 3. INTEGRACIÓN Y SELECCIÓN

A fin de responder este apartado hemos optado por añadir datos de vuelos por varias razones:

* Un mayor número de vuelos en un mes podría aumentar las búsquedas de hoteles cerca del aeropuerto.
* Un mayor número de vuelos en un mes indica más turismo internacional (en lugar de sólo turismo nacional). Un tipo de turismo que podría estar buscando hoteles con características diferentes a los turistas nacionales.
* Un mayor número de vuelos en un mes indica más turismo general, lo que puede influir en las estrategias de promoción y ofertas de determinados hoteles, influyendo en última instancia en su posición en el buscador Booking.

# Leemos los datos de vuelos  
flights <- read.csv("avia\_tf\_apal\_linear.csv.gz")  
summary(flights)

## DATAFLOW LAST.UPDATE freq unit   
## Length:1502955 Length:1502955 Length:1502955 Length:1502955   
## Class :character Class :character Class :character Class :character   
## Mode :character Mode :character Mode :character Mode :character   
##   
##   
##   
## tra\_meas airline rep\_airp TIME\_PERIOD   
## Length:1502955 Length:1502955 Length:1502955 Length:1502955   
## Class :character Class :character Class :character Class :character   
## Mode :character Mode :character Mode :character Mode :character   
##   
##   
##   
## OBS\_VALUE OBS\_FLAG   
## Min. : 0 Mode:logical   
## 1st Qu.: 0 NA's:1502955   
## Median : 264   
## Mean : 75333   
## 3rd Qu.: 3166   
## Max. :289204357

Para añadir dicha información hacemos un tratamiento de los datos obtenidos en Eurostat a fin de obtener una media de vuelos por mes para los últimos 5 años. Una vez obtenidos, tan solo debemos hacer un merge con nuestro dataset original.

proc\_flights <- flights %>%   
 # Filtramos para quedarnos solo con los aeropuertos de interés  
 # Código OACI/ICAO: Barcelona --> ES\_LEBL; Valencia --> ES\_LEVC; Madrid: ES\_LEMD  
 filter(rep\_airp %in% c("ES\_LEBL", "ES\_LEVC", "ES\_LEMD")) %>%   
 # Filtramos para quedarnos solo con los datos de carga de pasajeros  
 filter(tra\_meas == "PAS\_CRD") %>%  
 # Nos interesa solo los datos mensuales, así que filtramos por ellos  
 filter(freq == "M") %>%  
 separate(TIME\_PERIOD, c("YEAR", "MONTH"), sep = "-") %>%  
 # Nos quedamos solo con los últimos 5 años y los meses de Marzo, Junio y Diciembre  
 filter(YEAR %in% c("2022", "2021", "2020", "2019", "2018") & MONTH %in% c("03", "06", "12")) %>%  
 # No nos interesa los datos por aerolínea, solo los generales  
 filter(airline == "TOTAL") %>%  
 # Modificamos la columna de MONTH para tener los nombres del mes y cambiamos el código del aeropuerto por el nombre de la ciudad  
 mutate(month\_name = ifelse(MONTH == "03", "March", ifelse(MONTH == "06", "June", "December")),  
 city\_airp = ifelse(rep\_airp == "ES\_LEBL", "Barcelona", ifelse(rep\_airp == "ES\_LEVC", "Valencia", "Madrid"))) %>%  
 # Agrupamos por ciudad y mes y calculamos la media de vuelos.  
 group\_by(rep\_airp, month\_name) %>%  
 mutate(mean\_fligths = mean(OBS\_VALUE)) %>%  
 ungroup() %>%  
 dplyr::select(c(city\_airp, month\_name, mean\_fligths))  
  
# Nos quedamos con los registros únicos  
final\_flights <- unique(proc\_flights)  
  
# Hacemos un merge con bookings  
booking <- booking %>%  
 merge(final\_flights, by.x = c("city", "month\_in"), by.y = c("city\_airp", "month\_name"))

## 4. ANÁLISIS DE LOS DATOS

A continuación, se evaluará la influencia de las distintas variables sobre la posición de un hotel en el buscador, para ello se emplea una matriz de correlación. Una vez obtenidas las variables con mayor correlación con la variable objetivo generaremos una selección de los variables a analizar, un grupo de variables distinto por cada problema a tratar.

Una vez generados los distintos grupos procederemos a evaluar la normalidad y la homogeneidad de la varianza para así poder determinar en el siguiente punto que pruebas estadísticas se pueden aplicar.

Finalmente aplicaremos distintas pruebas estadísticas adicionales a la correlación hallada inicialmente. Estas pruebas estadísticas incluirán contrastes de hipótesis y regresiones lineales.

#### 4.1 Correlación con la variable objetivo y selección de los grupos de datos que se quieren analizar

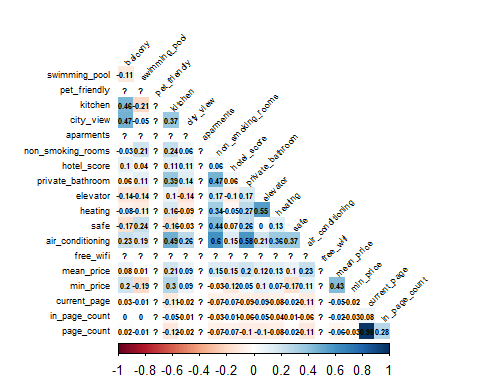
Comenzamos evaluando la correlación (de spearman, dado que no podemos asumir normalidad y utilizamos variables categóricas también) entre las distintas variables con nuestra variable objetivo, para ello creamos una matriz de correlación y evaluamos los resultados obtenidos.

# Transformamos las variables a numérico  
booking$postal\_code <- as.numeric(booking$postal\_code)  
booking$latitude <- as.numeric(booking$latitude)  
booking$longitude <- as.numeric(booking$longitude)  
booking$free\_wifi <- as.numeric(booking$free\_wifi)  
#booking$apartments <- as.numeric(booking$apartments)  
#booking$pet\_friendly <- as.numeric(booking$pet\_friendly)  
  
# Seleccionamos solo las variables numéricas  
booking\_numeric <- booking %>%  
 dplyr::select(-c("city", "month\_in", "check\_in", "month\_out", "check\_out", "as.integer(!is.na(NA))"))  
  
mattmp <- booking\_numeric %>%  
 dplyr::select(c("adults", "children", "num\_rooms", "postal\_code", "latitude", "longitude", "hotel\_score", "length\_description",   
 "is\_suite", "is\_apartment", "has\_free\_cancelation", "min\_price", "current\_page", "in\_page\_count", "page\_count"))

# Creamos una matriz de correlación del dataset  
cor\_mat <- cor(mattmp, method = "spearman")  
plot.new()  
plot.window(xlim=c(-2,2), ylim=c(5,10))  
corrplot(cor\_mat, method="color", type = "lower", tl.col = "black", tl.srt = 45, diag = FALSE, tl.cex = 0.55, addCoef.col = 'black', number.cex = 0.5)

Observamos que variables como las distintas puntuaciones están correlacionadas entre ellas, por lo que un hotel que haya obtenido una puntuación elevada en un aspecto generalmente también lo obtendrá en el resto. También encontramos que los hoteles con distintos atributos como aire acondicionado o baño privado suelen tener el resto de atributos.

Chart

Description automatically generatedChart

Description automatically generated with low confidence

# Filtramos para obtener cuales son las variables más correlacionadas con nuestra variable objetivo, current\_page  
cor\_mat <- cor(booking\_numeric, method = "spearman")

cor\_mat\_df <- as.data.frame(cor\_mat)  
cor\_mat\_df %>%  
 filter(abs(page\_count) > 0.065) %>%  
 filter(page\_count < 0.95) %>%  
 arrange(page\_count) %>%  
 dplyr::select(page\_count)

## page\_count  
## kitchen -0.12023093  
## is\_apartment -0.12015885  
## adults -0.11580745  
## air\_conditioning -0.11400432  
## elevator -0.09860877  
## private\_bathroom -0.09661382  
## num\_rooms -0.08662662  
## value\_for\_money\_score -0.08508046  
## heating -0.08399204  
## non\_smoking\_rooms -0.07424905  
## mean\_flights -0.06952514  
## comfort\_score -0.06710461  
## hotel\_score -0.06593433  
## longitude 0.07601044

Cuando evaluamos la posición en la página web observamos que no hay una correlación elevada con ninguna variable, encontrando valores por debajo de 0.12 (en valor absoluto) para todos los casos. No obstante, para el caso de page\_count, hemos optado por someter a estudio aquellas variables que presentan mayor correlación de spearman con la variable dependiente (pues evaluando con distintas métricas que buscan determinar el nivel de dependencia no-lineal hemos obtenido resultados similares). Las variables seleccionadas en cuestión son: **kitchen, is\_apartment**, **adults, air\_conditioning, elevator, private\_bathroom,num\_rooms, value\_for\_money\_score, heating, non\_smoking\_rooms, comfort\_score, hotel\_score, longitude, has\_free\_cancelation**, mientras que la variable dependiente será **page\_count**

X\_pcount <- booking\_numeric %>%  
 dplyr::select(c("kitchen", "is\_apartment", "adults", "air\_conditioning", "elevator", "private\_bathroom",  
 "num\_rooms", "value\_for\_money\_score", "heating", "non\_smoking\_rooms", "comfort\_score", "hotel\_score",  
 "longitude", "has\_free\_cancelation"))  
y\_pcount <- booking\_numeric %>%  
 dplyr::select("page\_count")

En el caso del problema a resolver relacionado con el precio, al analizar las correlaciones sí que encontramos valores entre 0.15 y 0.20. Aunque estas correlaciones no sean muy elevadas nos ofrecen un escenario más prometedor que el anteriormente presentado. A continuación, procedemos a seguir con el mismo procedimiento que con page\_count, esta vez seleccionando las siguientes variables, **postal\_code, elevator, heating, cleanliness\_score, latitude, mean\_flights, location\_score, hotel\_score, facilities\_score, non\_smoking\_rooms, comfort\_score, is\_apartment, private\_bathroom, kitchen, air\_conditioning.**

# Filtramos para obtener cuales son las variables más correlacionadas con nuestra variable objetivo, mean\_price  
cor\_mat <- cor(booking\_numeric, method = "spearman")

## Warning in cor(booking\_numeric, method = "spearman"): the standard deviation is  
## zero

cor\_mat\_df <- as.data.frame(cor\_mat)  
cor\_mat\_df %>%  
 filter(abs(mean\_price) > 0.1) %>%  
 filter(mean\_price < 0.95) %>%  
 arrange(mean\_price) %>%  
 dplyr::select(mean\_price, min\_price)

## mean\_price min\_price  
## postal\_code -0.1192213 -0.04142959  
## elevator 0.1203196 0.10057692  
## heating 0.1266339 0.07064159  
## cleanliness\_score 0.1288331 -0.15597670  
## latitude 0.1307897 0.04080835  
## mean\_flights 0.1316304 -0.06590563  
## location\_score 0.1382029 -0.09181542  
## hotel\_score 0.1460874 -0.12460077  
## non\_smoking\_rooms 0.1468123 -0.02672426  
## facilities\_score 0.1489429 -0.11356144  
## comfort\_score 0.1521425 -0.17078260  
## is\_apartment 0.2002154 0.28503850  
## private\_bathroom 0.2045901 0.05453184  
## kitchen 0.2075148 0.29653148  
## air\_conditioning 0.2319272 0.10878705

X\_price <- booking\_numeric %>%  
 dplyr::select(c("postal\_code", "elevator", "heating", "cleanliness\_score", "latitude", "mean\_flights",  
 "location\_score", "hotel\_score", "facilities\_score", "non\_smoking\_rooms", "comfort\_score", "is\_apartment",  
 "private\_bathroom", "kitchen", "air\_conditioning"))  
y\_price <- booking\_numeric %>%  
 dplyr::select("mean\_price")

#### 4.2 Comprobación de la normalidad y homogeneidad de la varianza

Para comprobar si los valores de las variables de nuestro dataset se distribuyen o se aproximan a una población distribuida normalmente haremos un análisis visual, con QQ-Plots y Histogramas, y un análisis mediante el test Shapiro-Wilk. Concretamente solo comprobaremos la normalidad para las variables seleccionadas en el apartado anterior.

norm\_cols <- c("hotel\_score", "length\_description", "hotel\_score", "staff\_score", "facilities\_score", "cleanliness\_score", "comfort\_score", "value\_for\_money\_score", "location\_score", "free\_wifi\_score", "min\_price")  
  
booking\_norm <- booking\_numeric %>%  
 dplyr::select(norm\_cols)

par(mfrow=c(2,2))  
for(i in 1:ncol(booking\_norm)){  
 qqnorm(booking\_norm[,i], main = paste("QQ-Plot de ",colnames(booking\_norm)[i]), col = "navy")  
 qqline(booking\_norm[,i], col = "red")  
 hist(booking\_norm[,i], main = paste("Histograma de ", colnames(booking\_norm)[i]), xlab=colnames(booking\_norm)[i], freq = FALSE)  
}

Chart, histogram

Description automatically generatedLos QQ-plots y los histogramas nos muestran una distribuciones con una evidente asimetría negativa en la mayoría de los casos (excepto variables como length\_description que muestran una asimetría más bien positiva, aunque podría ser influencia de un dato extremo que no ha sido considerado outlier), así como un evidente problema con las colas debido a esta, pues en la mayoría de los casos los puntos se alejan de la línea roja, la cual corresponde a la hipotética distribución normal. Por tanto, a priori podríamos rechazar la hipótesis de normalidad de las variables.

No obstante, tras ejecutar el test Shapiro-Wilk confirmamos de forma más fiable nuestras primeras conclusiones, pues todos los p-valores muestran valores muy inferiores a 0.05, rechazando en todos los casos la hipótesis nula sobre la normalidad de los datos.

lapply(booking\_norm, shapiro.test)  
##   
##   
## $hotel\_score  
##   
## Shapiro-Wilk normality test  
##   
## data: X[[i]]  
## W = 0.98372, p-value < 2.2e-16  
##   
##   
## $cleanliness\_score  
##   
## Shapiro-Wilk normality test  
##   
## data: X[[i]]  
## W = 0.97377, p-value < 2.2e-16

A fin de aproximar más los datos a una distribución normal, se propone hacer una transformación de box-cox y comprobar si los datos efectivamente se aproximan más a una normal. Tras llevar a cabo este proceso, si bien los histogramas muestran en algunos casos una distribución “visualmente más normales” y los QQ-plots parecen mostrar que la transformación ha aliviado ligeramente el problema con las colas, los resultados de la transformación no han sido suficientes como para poder considerar que los datos se distribuyen como una normal. No obstante, no debemos preocuparnos demasiado por este problema, pues tenemos suficientes registros como para poder apoyarnos en el Teorema del Límite Central. Por ejemplo, el t-test asume que las medias de las diferentes muestras se distribuyen normalmente (siendo una muestra de más de 30 registros suficiente, como rule of thumb, para que el t-test sea válido aunque los datos no se distribuyan normalmente).

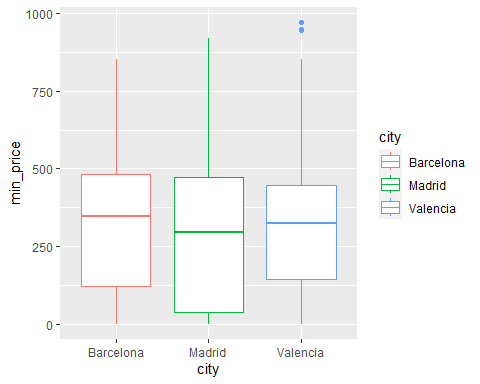
Graphical user interface, chart, line chart

Description automatically generated

norm\_cols <- c("hotel\_score", "length\_description", "hotel\_score", "staff\_score", "facilities\_score", "cleanliness\_score", "comfort\_score", "value\_for\_money\_score", "location\_score", "free\_wifi\_score", "min\_price")  
  
booking\_norm$min\_price <- ifelse(booking\_norm$min\_price == 0, 1, booking\_norm$min\_price)  
booking\_norm$free\_wifi\_score <- ifelse(booking\_norm$free\_wifi\_score == 0, 1, booking\_norm$free\_wifi\_score)  
  
par(mfrow=c(2,3))  
for(i in 1:ncol(booking\_norm)){  
 b <- boxcox(lm(booking\_norm[,i] ~ 1))  
 lambda <- b$x[which.max(b$y)]  
 qqnorm((booking\_norm[,i]^lambda - 1)/lambda, main = paste("QQ-Plot de ",colnames(booking\_norm)[i]), col = "navy")  
 qqline((booking\_norm[,i]^lambda - 1)/lambda, col = "red")  
 hist((booking\_norm[,i]^lambda - 1)/lambda, main = paste("Histograma de ", colnames(booking\_norm)[i]), xlab=colnames(booking\_norm)[i], freq = FALSE)  
}

Seguidamente, analizaremos la homocedasticidad mediante boxplots y varios test de homocedasticidad, o de homogeneidad de varianzas. En este caso, no podemos utilizar el test de Levene ni el de Barlett ya que son muy sensibles ante datos no normales y, como acabamos de presentar, los datos analizados no se aproximan en ningún caso a una normal.

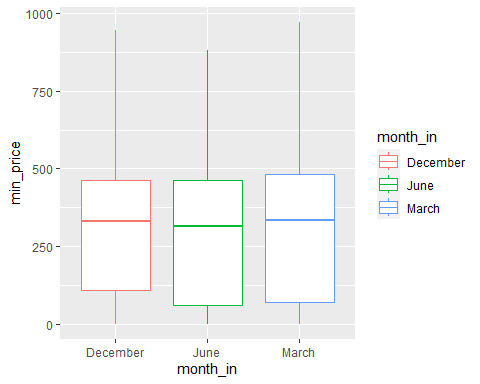
ggplot(booking, aes(x = city, y = min\_price, color = city)) +   
 geom\_boxplot()



fligner.test(min\_price ~ city, data = booking)

##   
## Fligner-Killeen test of homogeneity of variances  
##   
## data: min\_price by city  
## Fligner-Killeen:med chi-squared = 54.089, df = 2, p-value = 1.798e-12

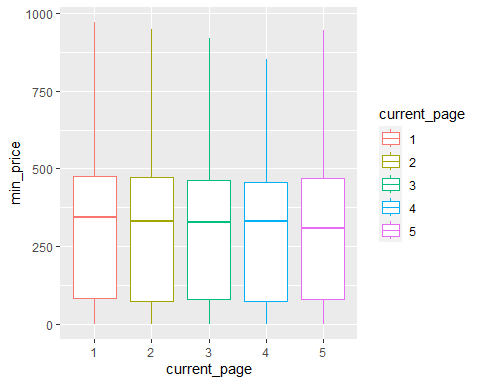
ggplot(booking, aes(x = month\_in, y = min\_price, color = month\_in)) +   
 geom\_boxplot()



fligner.test(min\_price ~ month\_in, data = booking)

##   
## Fligner-Killeen test of homogeneity of variances  
##   
## data: min\_price by month\_in  
## Fligner-Killeen:med chi-squared = 16.414, df = 2, p-value = 0.0002727

booking$current\_page <- as.factor(booking$current\_page)  
ggplot(booking, aes(x = current\_page, y = min\_price, color = current\_page)) +   
 geom\_boxplot()



fligner.test(min\_price ~ current\_page, data = booking)

##   
## Fligner-Killeen test of homogeneity of variances  
##   
## data: min\_price by current\_page  
## Fligner-Killeen:med chi-squared = 4.462, df = 4, p-value = 0.3471

#### 4.3 Aplicación de pruebas estadísticas para comparar los grupos de datos

#### 4.4 Aplicación de funciones lineales para resolución del problema

## 5. REPRESENTACIÓN DE LOS RESULTADOS

A lo largo del documento se han presentado representaciones gráficas de los análisis realizados y de los resultados obtenidos.

## 6. RESOLUCIÓN DEL PROBLEMA