

# Resumen de R

Tratamiento de Datos. Grado en Ciencia de Datos - UV

Carlos Santiago Martinez Torres

2025-04-22

## Índice

<b>Antes de empezar</b>	<b>3</b>
<b>Cleaning Data in R</b>	<b>3</b>
<b>CAPITULO 1</b>	<b>3</b>
Chequeando el tipo de dato . . . . .	4
Por que es importante? . . . . .	4
Conversion de texto a numero . . . . .	4
Conversion de factor a numero . . . . .	5
Ejercicios . . . . .	5
Hacer recortes de una cadena de texto . . . . .	5
Rango de los valores - limitaciones de los rangos -. . . . .	5
Encontrando valores fuera de rango . . . . .	6
Manipulado out of range values . . . . .	6
Tratando valores como NA . . . . .	6
Limitaciones con tipo de dato FECHA . . . . .	7
Ejercicios . . . . .	7
Convertir a tipo fechay filtrar por fechas anteriores a la fecha actual . . . . .	7
Duplicados . . . . .	8
Eliminando duplicados . . . . .	8
Encontrado valores parcialmente duplicados . . . . .	8
Eliminando los duplicados parciales . . . . .	8
Operar con los valores duplicados . . . . .	9
Ejercicios . . . . .	9
Full duplicados . . . . .	9
Duplicados parciales . . . . .	9
Agregacion para parciales . . . . .	9

<b>CAPITULO 2</b>	<b>10</b>
Datos categoricos . . . . .	10
Filtrando con joins . . . . .	10
Ejercicios . . . . .	10
Variables categoricas . . . . .	10
Problemas con datos categoricos . . . . .	11
Ejercicios . . . . .	12
Identificando inconsistencias . . . . .	12
Corrigiendo las inconsistencias . . . . .	12
Colapsando las categorias . . . . .	12
Cleaning text data . . . . .	12
Expresiones regulares . . . . .	13
Ejercicios . . . . .	14
Remover caracteres de una expresion . . . . .	14
Reemplazar valores que no cumplen con una condicion . . . . .	14
<b>CAPITULO 3</b>	<b>14</b>
Uniformidad . . . . .	14
Encontrando valores . . . . .	15
Uniformidad con fechas . . . . .	15
Ejercicios . . . . .	15
Uniformidad con fechas . . . . .	15
Corrigiendo valor con condicional . . . . .	16
Cross field validation . . . . .	16
Ejercicios . . . . .	16
Validando totales . . . . .	16
Validando fechas . . . . .	17
Datos incompletos . . . . .	17
Analizando los valores faltantes . . . . .	18
Tipos de datos faltantes . . . . .	19
Missing Completely at Random (MCAR) . . . . .	19
Missing at Random (MAR) . . . . .	19
Missing Not at Random (MNAR) . . . . .	20
Tratando con valores faltantes . . . . .	20
Eliminando valores faltantes . . . . .	20
Reemplazado valores faltantes . . . . .	20
Ejercicios . . . . .	20

Comparando strings . . . . .	21
Metodos para editar distancias . . . . .	21
Comparando valores para no hacer el map de la unidad 2 . . . . .	22
Ejercicios . . . . .	22
Generando y comparando . . . . .	22

## Antes de empezar

Siempre usar pacman para cargar las librerias

```
# Asegurarse de que el paquete "pacman" está instalado
if (!require("pacman")) install.packages("pacman")
```

Listado de librerias que se han implementado a lo largo del curso

```
pacman::p_load(tidyverse, readxl, lubridate, visdat, stringdist, fuzzyjoin)

# Con tidyverse: ggplot2, dplyr, tidyr, readr, purrr, tibble, stringr, forcats
# readxl, libreria que requiere una carga aparte explícita
# lubridate, se utiliza con frecuencia con paquetes de tidyverse, pero requiere una carga aparte
# visdat, exploracion visual de los datos
# stringdist: Approximate String Matching, Fuzzy Text Search, and String Distance Functions
# Fuzzyjoin, para unir df's por medio de string distances
```

## Cleaning Data in R

### CAPITULO 1

```
# library(dplyr)
# library(stringr)
```

Problemas comunes que se pueden tener con los datos cuando no están limpios. Por que limpiarlos? Porque el error aparece al inicio de los datos. Es decir:

Error humano, acceso a datos, exploración de los datos, extracción de datos y reporte.

Si no se soluciona, todo el código ira con el error humano.

Limitaciones de tipo de dato:

- Texto, que es el 'character'
- Entero, 'integer'
- Decimal, 'numeric'
- Binario, 'logical'
- Categórico, 'factor'
- Fecha, 'date'

## Chequeando el tipo de dato

- `sales <- read.csv('sales.csv')`
- `head(sales)`

Para saber el tipo de dato que estamos usando:

- `Columna <- sales$revenue`
- `Is.numeric(columna)`

Si no, con la librería `assertive`:

- `Library(assertive)`
- `Assert_is_numeric(columna)`

Cuando usamos expresiones como:

- `Is.numeric()`, `is.character()`, `is.logical()`, etc. . .

Estamos evaluando expresiones que nos devuelven un `return TRUE/FALSE`.

Cuando usamos la expresión:

- `Assert_is_character()`, `assert_is_numeric()`, etc. . .

Devuelve error cuando es `FALSE`.

## Por que es importante?

Podemos revisar el tipo de dato con la función:

- `Class(columna)`
- Tambien usando `glimpse(dataframe)`, de la libreria `dplyr`.

Es importante para evaluar la expresion. No podemos hacer la media de una columna que sea de tipo `character` . . .

## Conversion de texto a numero

- `Library(stringr)`
- `str_remove(columna, ',')`

El primer argumento es la variable a la que queremos quitarle el string. el segundo, es lo que queremos quitar.

Si lo queremos dejar en el dataframe original, podemos:

- `sales %>% mutate (revenue = as.numeri(str_remove(revenue, ',')))`

## Conversion de factor a numero

- `as.numeric(as.character(columna_factor))`

## Ejercicios

```
# # Glimpse at bike_share_rides
# glimpse(bike_share_rides)
#
# # Summary of user_birth_year
# summary(bike_share_rides$user_birth_year)
#
# # Convert user_birth_year to factor: user_birth_year_fct
# bike_share_rides <- bike_share_rides %>%
#   mutate(user_birth_year_fct = as.factor(user_birth_year))
#
# # Assert user_birth_year_fct is a factor
# assert_is_factor(bike_share_rides$user_birth_year_fct)
#
# # Summary of user_birth_year_fct
# summary(bike_share_rides$user_birth_year_fct)
```

## Hacer recortes de una cadena de texto

Sirve para eliminar un patto de una cadena de texto.

```
# bike_share_rides <- bike_share_rides %>%
#   # Remove 'minutes' from duration: duration_trimmed
#   mutate(duration_trimmed = str_remove(string= duration, pattern='minutes'),
#           # Convert duration_trimmed to numeric: duration_mins
#           duration_mins = as.numeric(duration_trimmed))
#
# # Glimpse at bike_share_rides
# glimpse(bike_share_rides)
#
# # Assert duration_mins is numeric
# assert_is_numeric(bike_share_rides$duration_mins)
#
# # Calculate mean duration
# mean(bike_share_rides$duration_mins)
```

Para saber en que libreria se encuentra un dataset, podemos usar la funcion

```
# ??iris
```

## Rango de los valores - limitaciones de los rangos -.

```
# library(ggplot2)
# library(ggplot2movies)
# install.packages('ggplot2movies')
```

Para ver todos los datasets disponibles podemos emplear:

```
# data()
```

## Encontrando valores fuera de rango

Podemos crear un histograma

```
# # Creacion de un factor llamado breaks
# breaks <- c(min(movies$avg), 0, 5, max(movies$avg))
# # Contiene el rating minimo, luego cero como la parte inicial (de abajo), 5 que es el top de lo esperado
#
# ggplot(data = movies, aes(avg)) +
#   geom_histogram(breaks = breaks)
```

Con la libreria assertive, podemos ver los valores que estan fuera del rango

```
# library(assertive)
# assert_all_are_in_closed_range(movies$avg, lower=0, upper=5)
```

## Manipulado out of range values

- Remove rows
- Treat as missing NA
- Replace with range limit
- Replace with other value based on domain knowledge and/or knowledge of dataset

```
# # Si queremos eliminar
# movies %>%
#   filter(avg>=0, avg<=5) %>%
#
#   ggplot(data = movies, aes(avg)) +
#     geom_histogram(breaks = c(min(movies$avg), 0, 5, max(movies$avg)))
#
# # Va a tener un total de 3 intervalos.
# # 1 - Del minimo al 0
# # 2 - Del 0 al 5
# # 3 - Del 5 al maximo
#
# # Este tipo de agrupaciones sirve para cuando se quiere agrupar los datos por categorias especificas
```

## Tratando valores como NA

Usando la funcion replace

```
# replace(col, condicion, replacement)
#
# movies %>%
#   mutate(rating_miss = replace(avg, avg>5, NA))
```

## Limitaciones con tipo de dato FECHA

```
# library(lubridate)
```

```
# assert_all_are_in_past(movies$date_recorded) # Nos dara error si hay algun valor que no este antes de
```

```
# movies %>%  
#   filter(date_recorded > today())
```

## Ejercicios

- Create a three-bin histogram of the duration\_min column of bike\_share\_rides using ggplot2 to identify if there is out-of-range data.

```
# # Create breaks  
# breaks <- c(min(bike_share_rides$duration_min), 0, 1440, max(bike_share_rides$duration_min))  
#  
# # Create a histogram of duration_min  
# ggplot(data = bike_share_rides, aes(duration_min)) +  
#   geom_histogram(breaks = breaks)
```

- Replace the values of duration\_min that are greater than 1440 minutes (24 hours) with 1440. Add this to bike\_share\_rides as a new column called duration\_min\_const.
- Assert that all values of duration\_min\_const are between 0 and 1440.

```
# duration_min_const: replace vals of duration_min > 1440 with 1440  
# bike_share_rides <- bike_share_rides %>%  
#   mutate(duration_min_const = replace(duration_min, duration_min>1440, 1440))  
#  
# # Make sure all values of duration_min_const are between 0 and 1440  
# assert_all_are_in_closed_range(bike_share_rides$duration_min_const, lower = 0, upper = 1440)
```

## Convertir a tipo fecha y filtrar por fechas anteriores a la fecha actual

```
# library(lubridate)  
#  
# # Convert date to Date type  
# bike_share_rides <- bike_share_rides %>%  
#   mutate(date = as.Date(date, format= '%Y-%m-%d'))  
#   # as.Date si solo necesitamos la fecha.  
#  
# # Make sure all dates are in the past  
# assert_all_are_in_past(bike_share_rides$date)  
#  
# # Filter for rides that occurred before or on today's date  
# bike_share_rides_past <- bike_share_rides %>%  
#   filter(date <= today())
```

```
#
# # Make sure all dates from bike_share_rides_past are in the past
# assert_all_are_in_past(bike_share_rides_past$date)
```

## Duplicados

- Error humano
- Uniones
- Bugs

En este caso, el dataframe es 'credit\_scores', que tambien contiene una columna de tipo numerica llamanda 'credit\_scores'.

```
# duplicated(x = dataframe)
# # Nos devuelve valores logicos
#
# sum(duplicated(x = dataframe))
# # El total de valores duplicados
#
# filter(dataframe, duplicated(dataframe))
```

## Eliminando duplicados

```
# credit_scores_unique <- distinct(dataframe)
# sum(duplicated(x = credit_scores_unique))
# # Esta suma deberia dar 0
```

## Encontrado valores parcialmente duplicados

```
# # Ver los valores que estan repetidos por al menos dos de sus variables
# duplicados_ids <- credit_scores %>%
#   count(first_name, last_name) %>% # contamos las veces que aparece este par de elementos por registr
#   filter(n > 1) # Filtra por los valores que estan duplicados. Es decir, donde aparecen mas de una ve
#
# credit_scores %>% # Lista de los registros parcialmente duplicados
#   filter(first_name %in% duplicados_ids$firs_name, last_name %in% duplicados_ids$last_name)
```

## Eliminando los duplicados parciales

```
# credit_scores %>%
#   distinct(first_name, last_name, .keep_all = TRUE)
```



## Operar con los valores duplicados

Podemos usar la media de los valores, agrupando por nombre o apellido, por ejemplo.

```
# credit_scores %>%  
#   group_by(first_name, last_name) %>%  
#   mutate(mean_credit_score = mean(credit_score))
```

## Ejercicios

### Full duplicados

Cuando son full duplicados, debemos usar el `distinct` sobre todo el dataframe

```
# # Count the number of full duplicates  
# sum(duplicated(bike_share_rides))  
#  
# # Remove duplicates  
# bike_share_rides_unique <- distinct(bike_share_rides)  
#  
# # Count the full duplicates in bike_share_rides_unique  
# sum(duplicated(bike_share_rides_unique))
```

### Duplicados parciales

Para cuando queremos evaluar duplicados parciales

```
# # Find duplicated ride_ids  
# bike_share_rides %>%  
#   count(ride_id) %>%  
#   filter(n > 1)  
#  
# # Remove full and partial duplicates  
# bike_share_rides_unique <- bike_share_rides %>%  
#   # Only based on ride_id instead of all cols  
#   distinct(ride_id, .keep_all = TRUE)  
#  
# # Find duplicated ride_ids in bike_share_rides_unique  
# bike_share_rides_unique %>%  
#   # Count the number of occurrences of each ride_id  
#   count(ride_id) %>%  
#   # Filter for rows with a count > 1  
#   filter(n>1)
```

### Agregacion para parciales

Agregaciones para duplicados parciales

```
# bike_share_rides %>%
# # Group by ride_id and date
# group_by(ride_id, date) %>%
# # Add duration_min_avg column
# mutate(duration_min_avg = mean(duration_min) ) %>%
# # Remove duplicates based on ride_id and date, keep all cols
# distinct(ride_id, date, .keep_all=TRUE) %>%
# # Remove duration_min column
# select(-duration_min)
```

## CAPITULO 2

### Datos categoricos

- Los datos categoricos tienen valores conocidos
- Los factores, son un tipo de numero que representa cada categoria
- Factores, tiene los valores que puede recibir la columna y los niveles, que representa

### Filtrando con joins

- Semi-join, las observaciones de x que estan en y
- Anti-join, las observaciones de x que NO estan en y

```
# dataset <- study_data
# dataset2 <- blood_types

# # sera el valor diferente
# study_data %>%
#   anti_join(blood_types, by='blood_type')
#
# # Para ver los valores 'correctos'
# study_data %>%
#   semi_join(blood_types, by='blood_type')
```

## Ejercicios

### Variables categoricas

Para contar las veces en que aparece una variable categórica

```
# # Count the number of occurrences of dest_size
# sfo_survey %>%
#   count(dest_size)
```

Para ver los valores incorrectos, usamos el anti join

```
# # Find bad dest_size rows
# sfo_survey %>%
#   # Join with dest_sizes data frame to get bad dest_size rows
#   anti_join(dest_sizes, by='dest_size') %>%
#   # Select id, airline, destination, and dest_size cols
#   select(id, airline, destination, dest_size)
```

Para filtrar por los valores correctos

```
# # Remove bad dest_size rows
# sfo_survey %>%
#   # Join with dest_sizes
#   inner_join(dest_sizes, by='dest_size') %>%
#   # Count the number of each dest_size
#   count(dest_size)
```

## Problemas con datos categoricos

- Letras mayusculas o minusculas
- Demasiadas categorias para la misma ‘especie’
- Espacios por cada texto

El primer filtro, puede ser convertir todo a minuscula o minuscula

```
# # Usando la libreria
# library(stringr)
#
# animals %>%
#   mutate(type_lower = str_to_lower(type))
```

Segundo, para eliminar espacios al inicio y al final de la cadena, podemos usar la funcion

```
# animals %>%
#   mutate(type_trimmed = str_trim(type_lower))
```

Para ver el ‘resultado’ final, podemos contar el tipo de dato y ordenarlo

```
# animals %>%
#   count(type_trimmed, sort=TRUE)
```

Si quiero evaluar todas las categorias de un df, y cambiar su valor

```
# # Crear una lista con las categorias que quiero cambiar, o las que quiero asignar un nuevo valor
#
# other_cats = c('a', 'b', 'c', etc...)
#
# # Importar la libreria para colapsar todo
# library(forcats)
#
# # Crea una nueva columna, que evalua la columna con las categorias actuales. Si la categoria actual, c
#
# animals %>%
#   mutate(type_collapsed = fct_collapse(type_trimmed, other = other_cats))
```

## Ejercicios

### Identificando inconsistencias

Podemos analizar si el tipo de inconsistencia es por espacios, mayusculas, minusculas, etc

```
# # Count dest_size
# sfo_survey %>%
#   count(dest_size)
```

### Corrigiendo las inconsistencias

```
# # Add new columns to sfo_survey
# sfo_survey <- sfo_survey %>%
#   # dest_size_trimmed: dest_size without whitespace
#   mutate(dest_size_trimmed = str_trim(dest_size),
#           # cleanliness_lower: cleanliness converted to lowercase
#           cleanliness_lower = str_to_lower(cleanliness))
#
# # Count values of dest_size_trimmed
# sfo_survey %>%
#   count(dest_size_trimmed)
#
# # Count values of cleanliness_lower
# sfo_survey %>%
#   count(cleanliness_lower)
```

### Colapsando las categorias

Colapsar categorias no es mas que decirle que analice una columna, y si tiene ese valor, que lo cambie por uno normalizado u otro cualquiera.

```
# # Count categories of dest_region
# sfo_survey %>%
#   count(dest_region)
#
# # Categories to map to Europe
# europe_categories <- c('EU', 'Europ', 'eur')
#
# # Add a new col dest_region_collapsed
# sfo_survey %>%
#   # Map all categories in europe_categories to Europe
#   mutate(dest_region_collapsed = fct_collapse(dest_region,
#                                                Europe = europe_categories)) %>%
#   # Count categories of dest_region_collapsed
#   count(dest_region_collapsed)
```

## Cleaning text data

Son todos los datos de tipo character

- Problemas con formateo y estandarización
- Espacios entre números

Para detectar un patrón, podemos usar

```
# # Devuelve valores lógicos
# str_detect(columna, 'valor a detectar')
```

Podemos ver en el dataframe, los valores que siguen ese patrón, de la forma

```
# # Detectar patrón de la forma 0000-0000-0000-0000
# customer %>% # DF
# filter(str_detect(credit_card, '-'))
```

Se pueden reemplazar los valores que sigan ese patrón por otro. Por ejemplo, cambiar el '-' por un espacio ' '

```
# # crea una col, al que se le pasa la función str_replace_all, que cambia todos los '-' por un espacio
# customer %>%
# mutate(credit_card_spaces = str_replace_all(credit_card, '-', ' '))
```

Si se quisiera normalizar, para no tener espacios ni barras, se puede usar

```
# # A la columna, a cada valor de esa columna, quita los caracteres - y ' '.
# credit_card_clean <- customer$credit_card %>%
# str_remove_all('-') %>%
# str_remove_all(' ')
#
# # Una vez los quitas de la columna, esa columna limpia, se reemplaza en el df original
#
# customers %>%
# mutate(credit_card = credit_card_clean)
```

Si queremos ver valores que no son correctos, sabiendo que tienen cierta longitud determinada

```
# # De cada valor de la columna, obtiene su longitud
# str_length(customer$credit_card)
#
# # Luego muestra cuáles no cumplen con la condición
# customer %>%
# filter(str_length(credit_card) != 16)
```

Para dejar limpio el dataframe, con los valores que si cumplen la condición, se puede emplear

```
# customer %>%
# filter(str_length(credit_card) == 16)
```

## Expresiones regulares

So útiles para encontrar patrones dentro de cada texto. Disponible en otro curso :)

## Ejercicios

```
# # Filter for rows with "-" in the phone column
# sfo_survey %>%
#   filter(str_detect(phone, '-'))
```

IMPORTANTE: Remember to use `fixed()` when searching for parentheses.

```
# # Filter for rows with "(" or ")" in the phone column
# sfo_survey %>%
#   filter(str_detect(phone, fixed('(')) | str_detect(phone, fixed(')')))
```

### Remover caracteres de una expresion

```
# # Remove parentheses from phone column
# phone_no_parens <- sfo_survey$phone %>%
#   # Remove "("s
#   str_remove_all(fixed("(")) %>%
#   # Remove ")"s
#   str_remove_all(fixed(")"))
#
# # Add phone_no_parens as column
# sfo_survey %>%
#   mutate(phone_no_parens = phone_no_parens,
#   # Replace all hyphens in phone_no_parens with spaces
#   phone_clean = str_replace_all(phone_no_parens, '-', ' '))
```

### Reemplazar valores que no cumplen con una condicion

```
# # Check out the invalid numbers. It must have 12
# sfo_survey %>%
#   filter(str_length(phone) != 12)
#
# # Remove rows with invalid numbers
# sfo_survey %>%
#   filter(str_length(phone) == 12)
```

## CAPITULO 3

### Uniformidad

- Para diferentes unidades o formatos
- Temperatura, pesos, tasas de cambio

## Encontrando valores

Para convertir los valores, podemos usar una formula de conversion, cuando el ejercicio lo permita.

Implementando la codicion ifelse, tenemos:

```
# ifelse(condition, value_if_true, value_if_false)
#
# # Si la temperatura es mayor a 50
# # Aplica la formula para pasar de F a Celsius
# # si no, deja la temperatura igual
#
# nyc_temps %>%
#   mutate(temp_c = ifelse(temp>50, (temp-32)*5/9, temp))
```

## Uniformidad con fechas

Para ver los formatos de las fechas

```
# ?strptime
```

Para analizar los multiples formatos

```
# # Orders contiene los formatos diferentes
#
# library(lubridate)
# parse_date_time(nyc_temps$date,
#                 orders = c('%Y-%m-%d', '%m/%d/%y', '%B %d %Y'))
#
# # Esto lo devolvera en un formato yy-mm-dd
```

## Ejercicios

Para considerar, By default, as.Date() can't convert "Month DD, YYYY" formats.

## Uniformidad con fechas

```
# # Check out the accounts data frame
# head(accounts)
#
# # Formato yyyy-mm-dd
# # Formato Mes dd, yyyy
#
# # Define the date formats
# formats <- c("%Y-%m-%d", "%B %d, %Y")
#
# # Convert dates to the same format
# accounts %>%
#   mutate(date_opened_clean = parse_date_time(date_opened, orders = formats))
```

## Corrigiendo valor con condicional

```
# # Scatter plot of opening date and total amount
# accounts %>%
#   ggplot(aes(x = date_opened, y = total)) +
#   geom_point()
#
# # Left join accounts to account_offices by id
# accounts %>%
#   left_join(account_offices, by = "id") %>%
#   # Convert totals from the Tokyo office to USD
#   mutate(total_usd = ifelse(office == "Tokyo", total / 104, total)) %>%
#   # Scatter plot of opening date vs total_usd
#   ggplot(aes(x = date_opened, y = total_usd)) +
#   geom_point()
```

## Cross field validation

- Validacion con numeros

Sumar todos los numeros y comprobar con la columna total

- Validacion con fechas

Podemos obtener la diferencia de fechas con

```
# library(lubridate)
#
# date_difference <- as.Date('2015-09-04') %--% today()
# date_difference
#
# # Para obtener un valor adecuado
#
# as.numeric(date_difference, 'years')
#
# floor(as.numeric(date_difference, 'years'))
#
# # Podemos hacer la validacion en el dataframe con
#
# credit_cards %>%
#   mutate(theor_age = floor(as.numeric(date_opened %--% today(), 'years')) %>%
#   filter(theor_age != acct_age)
```

- Se puede eliminar
- Input datos

## Ejercicios

Validando totales



```
# # Find invalid totals
# accounts %>%
#   # theoretical_total: sum of the three funds
#   mutate(theoretical_total = rowSums(., 4:6))) %>%
#   # Find accounts where total doesn't match theoretical_total
#   filter(theoretical_total != total)
#
# #accounts %>%
# #   select(starts_with('fund')) %>%
# #   mutate(theoretical_total = rowSums(.))
```

## Validando fechas

```
# # Find invalid acct_age
# accounts %>%
#   # theoretical_age: age of acct based on date_opened
#   mutate(theoretical_age = floor(as.numeric(date_opened %--% today(), 'years')))) %>%
#   # Filter for rows where acct_age is different from theoretical_age
#   filter(theoretical_age != acct_age)
```

## Datos incompletos

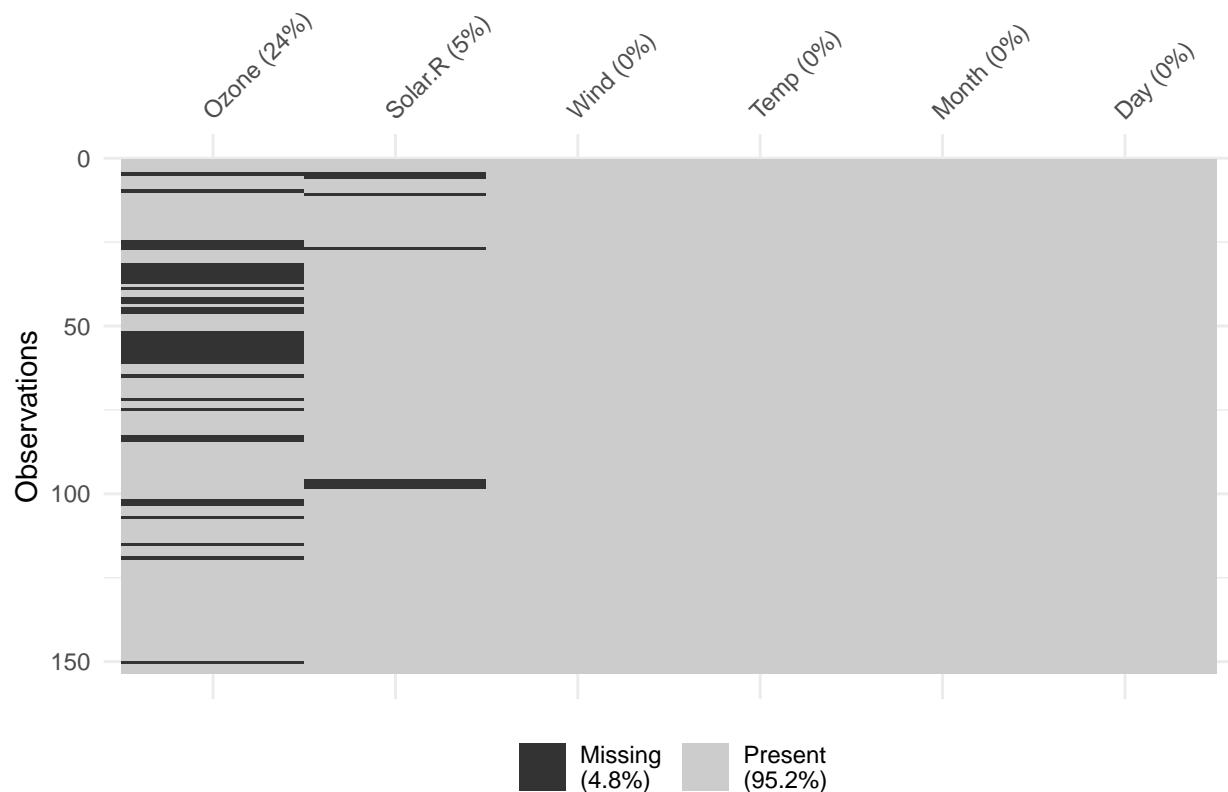
- NA, nan, 0, 99

Para encontrar valores faltantes, podemos usar

```
# # Devuelve valores logicos
# is.na(airquality)
#
# # Para obtener el total de valores se puede hacer la suma
# sum(is.na(airquality))
```

Para visualizar los valores faltantes, existe la libreria siguiente

```
# library(dplyr)
# library(visdat)
vis_miss(airquality)
```



## Analizando los valores faltantes

```
airquality %>%

  # Crear una columna que indica si otra columna tiene valor nulo o no
  mutate(miss_ozone = is.na(Ozone)) %>%

  # Agrupa los datos por la columna creada, entre TRUE y FALSE
  group_by(miss_ozone) %>%

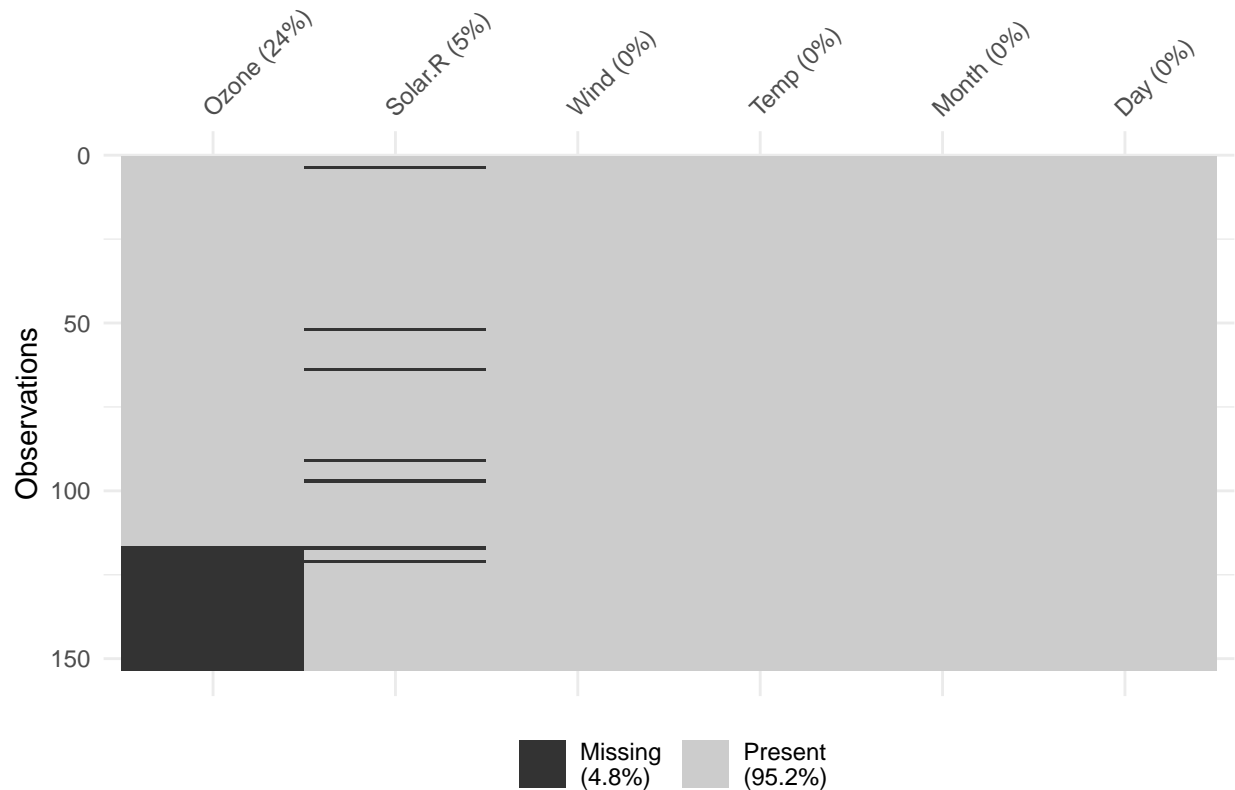
  # Por cada valor, TRUE o FALSE, halla la media de cada columna
  summarise(across(everything(), median, na.rm=TRUE))

## Warning: There was 1 warning in 'summarise()'.
## i In argument: 'across(everything(), median, na.rm = TRUE)'.
## i In group 1: 'miss_ozone = FALSE'.
## Caused by warning:
## ! The '...' argument of 'across()' is deprecated as of dplyr 1.1.0.
## Supply arguments directly to '.fns' through an anonymous function instead.
##
## # Previously
## across(a:b, mean, na.rm = TRUE)
##
```

```
## # Now
## across(a:b, \(x) mean(x, na.rm = TRUE))
```

Para interpretar mejor los valores faltantes, los podemos organizar por una variable

```
airquality %>%
  arrange(Ozone) %>%
  vis_miss()
```



## Tipos de datos faltantes

### Missing Completely at Random (MCAR)

No hay una relacion entre los datos perdidos y los otros valores.

- Errores en la entrada de datos

### Missing at Random (MAR)

Relacion sistematica entre los valores faltantes y otros valores observados

- Los datos perdidos de 'ozone' para temperaturas altas en el ejercicio anterior

## Missing Not at Random (MNAR)

Relacion directa entre los valores faltantes y los otros valores no observados

- Datos perdidos para temperaturas muy altas

## Tratando con valores faltantes

- Eliminar
- Imputar valores estadísticos

## Eliminando valores faltantes

```
airquality %>%  
  filter(!is.na(Ozone), !is.na(Solar.R)) %>%  
  head(5)
```

## Reemplazado valores faltantes

```
airquality %>%  
  # Crea una columna para calcular valores faltantes  
  # ifelse(condition, value_if_true, value_if_false)  
  mutate(ozone_filled = ifelse(is.na(Ozone), mean(Ozone, na.rm=TRUE), Ozone)) %>%  
  
  mutate(ozone_filled = round(x=ozone_filled, digits = 2)) %>%  
  
  head(5)
```

## Ejercicios

- Visualizar los valores faltantes de un conjunto
- Crear una columna con valores lógicos para saber si una columna tiene valores falsos o no
- Agrupar por una columna para aplicar una función de agregación

```
# # Visualize the missing values by column  
# vis_miss(accounts)  
#  
# accounts %>%  
#   # missing_inv: Is inv_amount missing?  
#   mutate(missing_inv = is.na(inv_amount)) %>%  
#   # Group by missing_inv  
#   group_by(missing_inv) %>%  
#   # Calculate mean age for each missing_inv group  
#   summarize(avg_age = mean(age))  
#  
# # Sort by age and visualize missing vals  
# accounts %>%  
#   arrange(age) %>%  
#   vis_miss()
```

- Dejar un set de datos limpios, con valores NA reemplazados con valores estadísticos
- Verificar/Assert que los valores de una columna contienen nulos

```
# # Create accounts_clean
# accounts_clean <- accounts %>%
#   # Filter to remove rows with missing cust_id
#   filter(!is.na(cust_id)) %>%
#   # Add new col acct_amount_filled with replaced NAs
#   mutate(acct_amount_filled = ifelse(is.na(acct_amount), inv_amount * 5, acct_amount))
#
# # Assert that cust_id has no missing vals
# assert_all_are_not_na(accounts_clean$cust_id)
#
# # Assert that acct_amount_filled has no missing vals
# assert_all_are_not_na(accounts_clean$acct_amount_filled)
```

## Comparando strings

- Cambiar los valores de una palabra. Eliminar, insertar, cambiar o intercambiar.

## Metodos para editar distancias

- Damerau-Levenshtein: insertar, eliminar, sustituir, trasponer (cambiar posición) - ORDEN CORRECTO
- Levenshtein: solo sustituir, insertar y eliminar
- LCS (Longest common Subsequence): solo inserta y elimina . Others

```
# library(stringdist)
# Damerau-Levenshtein
stringdist(a='baboon', b='typhoon', method='dl')
```

```
## [1] 4
```

```
# Inserta h          (babhoon)
# Sustituye b por t  (tabhoon)
# Sustituye a por y  (tybhoon)
# Sustituye b por p  (typhoon)
```

Ejemplos con otros metodos

```
# LCS
stringdist(a='baboon', b='typhoon', method='lcs')
```

```
## [1] 7
```

```
# Jaccard # Escala de 0 a 1
stringdist(a='baboon', b='typhoon', method='jaccard')
```

```
## [1] 0.75
```

## Comparando valores para no hacer el map de la unidad 2

```
# df survey
# city, move_score

# df citiies
# city
```

Usando la libreria fuzzyjoin, se podra hacer la union de los dos dataframe, por medio de la distancia de los strings

```
# # library(fuzzyjoin)
# stringdist_left_join(survey, citiies, by='city', method='dl')
#
# # Se puede pasar el argumento max_dist para ser mas precisos con el match
# stringdist_left_join(survey, citiies, by='city', method='dl', max_dist=1)
```

## Ejercicios

```
# f1 <- 'CleaningData/zagat.rds'
# zagat <- readRDS(file = f1)
```

- Contar los valores de cada ciudad. Variaciones de los nombres
- Hacer un join con string distance

```
# # Count the number of each city variation
# zagat %>%
#   count(city)
#
# # Join zagat and cities and look at results
# zagat %>%
#   # Left join based on stringdist using city and city_actual cols
#   stringdist_left_join(cities, by = c('city' = 'city_actual')) %>%
#   # Select the name, city, and city_actual cols
#   select(name, city, city_actual)
```

## Generando y comparando