Anexo 1: Taller de Aplicación en Manejo e Imputación de Datos Perdidos con R. Instituto Nacional de Estadísticas

Departamento de Metodologías e Innovación Estadística
Subdepartamento de Investigación Estadística
Instituto Nacional de Estadística

Índice

1 Manejo de datos perdidos

El siguiente anexo se enfoca en resaltar la trascendencia de la gestión y análisis de datos faltantes, una constante en cualquier estudio estadístico del mundo real. La aparición de valores ausentes puede derivar de diversas circunstancias, que abarcan desde inconvenientes en la toma de medidas hasta cuestiones vinculadas a la integridad de los datos. Esta presencia puede significativamente complicar la comprensión y la interpretación de los análisis.

En este contexto, el siguiente anexo, tiene un enfoque tipo taller, que procura dotar a los participantes con las competencias y herramientas indispensables para gestionar la ausencia de datos de manera eficiente, promoviendo la mejora de la calidad de los análisis. Se muestra el empleo del paquete `naniar` de R, una herramienta de gran utilidad para explorar, visualizar y gestionar valores faltantes.

El anexo se compone de distintas secciones, cada una orientada hacia un aspecto específico de la gestión de datos faltantes. La sección inicial abarca la detección, el conteo y la síntesis de los datos faltantes, haciendo uso de técnicas tales como la función `is.na()` y las funciones de resumen, como `sum()`, `mean()`, entre otras. Además, se instruye sobre la exploración de patrones asociados con los datos faltantes en diversas variables y casos, así como la identificación de sesgos y patrones subyacentes en los datos.

En las secciones subsiguientes, se aborda la imputación de valores faltantes, es decir, el procedimiento para rellenar las lagunas en los datos mediante diversas técnicas de imputación, como la imputación por media, la imputación múltiple y la regresión de valores faltantes, entre otras. También se proporciona orientación para evaluar la calidad de los datos imputados y tomar decisiones basadas en conjuntos de datos imputados.

El anexo incluye, además, una sección dedicada a la visualización de datos faltantes, en la cual se instruye sobre la generación de visualizaciones abarcadoras tanto para el conjunto de datos en su totalidad como para variables, casos y otros resúmenes, así como la exploración de estos elementos en sus respectivos grupos.

En resumen, el anexo provee de tips y herramientas necesarias para gestionar, explorar y analizar datos faltantes de forma eficiente, con el propósito de elevar la calidad de los análisis estadísticos.

1.1 Introducción a los datos perdidos

Para este anexo se utilizarán herramientas como `tidyverse` y el paquete R de `naniar` para enseñar a manejar y analizar datos faltantes de manera efectiva. El paquete `naniar` es una herramienta muy útil para explorar, visualizar y manejar valores faltantes en R.

Gertrude Mary Cox, destacada figura en el campo de la estadística, expresó en una ocasión: “La mejor estrategia frente a la ausencia de datos es simplemente no tener ninguno”. Aunque esta afirmación es cierta, no se ajusta a la realidad en la que vivimos. En el mundo de los análisis de datos con aplicación en situaciones reales, la presencia de datos faltantes es una constante. Para sobresalir como analista, es imperativo dominar la habilidad de gestionar estos valores ausentes.

Comprender el funcionamiento de los datos faltantes reviste una importancia fundamental, dado que pueden influir de manera inesperada en tus análisis. Por ejemplo, al aplicar un modelo lineal a conjuntos de datos con valores faltantes, se produce una pérdida de fragmentos de información, lo que a su vez implica que tus decisiones carecerán de una base de evidencia sólida. La sustitución de valores faltantes, conocida como imputación, debe realizarse con sumo cuidado, ya que la simple inserción de la media arroja estimaciones y decisiones de baja calidad.

En este documento aprenderás sobre qué son los valores faltantes, cómo encontrar datos faltantes, cómo manipular y limpiar datos faltantes, por qué faltan datos y cómo imputar valores faltantes.

Por lo tanto, asumiremos que tienes experiencia básica a intermedia con R, experiencia en la creación de gráficos utilizando `ggplot2`, experiencia en el uso de `dplyr` para manipular datos y experiencia en ajustar modelos lineales en R. En este primer capítulo, presentamos los valores faltantes y cómo verificarlos y contarlos.

1.2 Algunas definiciones iniciales

¿Qué son los valores faltantes? Antes de comenzar, debemos definir los valores faltantes. Los valores faltantes son valores que deberían haberse registrado, pero no lo fueron. Piensa en esto de esta manera: puedes no haber registrado accidentalmente que viste un pájaro, esto es un valor faltante. Esto es diferente a registrar que no se observaron pájaros. R almacena los valores faltantes como `NA`, que significa no disponible.

¿Cómo podemos verificar si tengo valores faltantes? Los valores faltantes no saltan y gritan "¡Estoy aquí!". Por lo general, están ocultos, como una aguja en un pajar. Para detectar valores faltantes, usa `any_na`, que devuelve `TRUE` si hay valores faltantes y `FALSE` si no los hay. `are_na` pregunta "¿son estos `NA`?" y devuelve `TRUE/FALSE` para cada valor. `are_na` nos muestra 3 valores `TRUE`, que corresponden a 3 valores faltantes. Para evitar contar cada `TRUE` manualmente, `n_miss` cuenta el número de valores faltantes. Y `prop_miss` entrega la proporción de valores faltantes, lo que nos da un contexto importante: ¡el 50% de los datos está faltando!

¿Qué sucede cuando mezclamos valores faltantes con nuestros cálculos? Necesitamos saber qué sucede para poder estar preparados para encontrar estos casos. La regla general es: Los cálculos con `NA` devuelven `NA`. Digamos que tienes la altura de tres amigos: Sofia, Juan y José. La suma de sus alturas devuelve `NA`, esto se debe a que no conocemos la suma de un número y `NA`.

Tips con datos faltantes Hay algunos "tips" que debes tener en cuenta al trabajar con datos faltantes: Por ejemplo, `NaN` significa "Not a Number" (No es un número) y se obtiene de operaciones como la raíz cuadrada de -1. R interpreta `NaN` como un valor faltante. `NULL` es un valor vacío pero no es faltante. Esto es sutilmente diferente de los valores faltantes: un cubo vacío no tiene agua faltante. `Inf` es un valor infinito, y se obtiene de ecuaciones como 10 dividido por 0 y no es faltante.

Por último, debemos tener cuidado con las declaraciones condicionales con valores faltantes. Por ejemplo, `NA` o `TRUE` es `TRUE`. `NA` o `FALSE` es `NA`. `NA + NaN` es `NA`. `NaN + NA` es `NaN`.

1.3 Carga de librerias

```
library(naniar)
library(tidyverse)
library(readxl)
library(dplyr)
library(tidyr)
library(ggplot2)
library(readxl)
```

1.4 Usando y encontrando valores faltantes

Al trabajar con datos faltantes, hay algunos comandos con los que deberiamos estar familiarizados - en primer lugar, debemos poder identificar si hay valores faltantes y dónde se encuentran.

Usando las herramientas `any_na()` y `are_na()`, identifica qué valores faltan.

```
# Crea x, un vector, con valores NA, NaN, Inf, ".", y "missing"
x <- c(NA, NaN,Inf, ".", "missing")

# Usa any_na() y are_na() sobre x para explorar los missings
any_na(x)
```

```
## [1] TRUE
```

```
are_na(x)
```

```
## [1]  TRUE FALSE FALSE FALSE FALSE
```

1.4.1 ¿Cuántos valores faltantes hay?

Una de las primeras cosas que deseamos comprobar en un nuevo conjunto de datos es si existen valores faltantes y cuántos hay.

Podríamos usar `are_na()` y contar los valores faltantes, pero la forma más eficiente de contarlos es usar la función `n_miss()`. Esto te dirá el número total de valores faltantes en los datos.

Luego puedes encontrar el porcentaje de valores faltantes en los datos con la función `pct_miss`. Esto te dirá el porcentaje de valores faltantes en los datos.

También puedes encontrar el complemento de estos valores, cuántos valores completos hay, usando `n_complete` y `pct_complete`.

```
# Usando un dataframe de ejemplo de (alturas y pesos) heights y weights dat_hw
dat_hw <- read.table("dealing/dat_hw.txt", h=T,  dec=".")
```

```
# Usa n_miss() para contar el numero total de valores missing en dat_hw
naniar::n_miss(dat_hw)
```

```
## [1] 30
```

```
# Usa n_miss() sobre dat_hw$weight para contar el numero total de valores missing
naniar::n_miss(dat_hw$weight)
```

```
## [1] 15
```

```
# Usa n_complete() sobre dat_hw para contar el numero total de valores completos
n_complete(dat_hw)
```

```
## [1] 170
```

```
# Utiliza n_complete() en dat_hw$weight para contar el número total de valores completos
n_complete(dat_hw$weight)
```

```
## [1] 85
```

```
# Utiliza prop_miss() y prop_complete() en dat_hw para contar el número total de valores
# faltantes y completos respectivamente.
prop_miss(dat_hw)
```

```
## [1] 0.15
```

```
prop_complete(dat_hw)
```

```
## [1] 0.85
```

1.5 Como resumir valores perdidos

1.5.1 Resumiendo la ausencia de datos

Ahora que comprendemos el comportamiento de los valores faltantes en R y cómo contarlos, escalaremos nuestros resúmenes para casos (filas) y variables, utilizando `miss_var_summary()` y `miss_case_summary()`, y también exploraremos cómo se pueden aplicar a grupos en un dataframe utilizando la función `group_by` de `dplyr`.

```
# Resumen de la ausencia de valores en cada variable del conjunto de datos `airquality`
miss_var_summary(airquality)
```

```
## # A tibble: 6 x 3
##   variable n_miss pct_miss
##   <chr>     <int>    <dbl>
```

```
## 1 Ozone        37    24.2
## 2 Solar.R       7     4.58
## 3 Wind          0     0
## 4 Temp          0     0
## 5 Month         0     0
## 6 Day           0     0
```

```
# Resumen de la ausencia de valores en cada caso del conjunto de datos `airquality`
miss_case_summary(airquality)
```

```
## # A tibble: 153 x 3
##     case n_miss pct_miss
##    <int>  <int>    <dbl>
##  1     5      2     33.3
##  2    27      2     33.3
##  3     6      1     16.7
##  4    10      1     16.7
##  5    11      1     16.7
##  6    25      1     16.7
##  7    26      1     16.7
##  8    32      1     16.7
##  9    33      1     16.7
## 10    34      1     16.7
## # i 143 more rows
```

```
# Muestra el resumen de la ausencia de valores en cada variable, agrupados por Mes,
# en el conjunto de datos `airquality`
airquality %>% group_by(Month) %>% miss_var_summary()
```

```
## # A tibble: 25 x 4
## # Groups:   Month [5]
##    Month variable n_miss pct_miss
##    <int> <chr>     <int>    <dbl>
##  1     5 Ozone         5     16.1
##  2     5 Solar.R       4     12.9
##  3     5 Wind          0      0
##  4     5 Temp          0      0
##  5     5 Day           0      0
##  6     6 Ozone        21     70
##  7     6 Solar.R       0      0
##  8     6 Wind          0      0
##  9     6 Temp          0      0
## 10     6 Day           0      0
## # i 15 more rows
```

```
# Muestra el resumen de la ausencia de valores en cada caso, agrupados por Mes,
# en el conjunto de datos `airquality`
airquality %>% group_by(Month) %>% miss_case_summary()
```

```
## # A tibble: 153 x 4
## # Groups:   Month [5]
##    Month  case n_miss pct_miss
##    <int> <int>  <int>    <dbl>
##  1     5     5      2       40
##  2     5    27      2       40
##  3     5     6      1       20
```

```
##  4     5    10      1       20
##  5     5    11      1       20
##  6     5    25      1       20
##  7     5    26      1       20
##  8     5     1      0        0
##  9     5     2      0        0
## 10     5     3      0        0
## # i 143 more rows
```

1.5.2 Tabulando los valores faltantes

Loss resumenes de los valores faltantes que acabamos de calcular nos dan el número y el porcentaje de observaciones faltantes para los casos y variables.

Otra manera de resumir los valores faltantes es mediante la tabulación del número de veces que hay 0, 1, 2, 3, valores faltantes en una variable o en un caso.

En este ejercicio, vamos a tabular el número de valores faltantes en cada caso y variable utilizando `miss_var_table()` y `miss_case_table()`, y también combinaremos estos resúmenes con el operador `group_by` de `dplyr` para explorar los resúmenes sobre una variable de agrupación en el conjunto de datos.

```
# Tabula la ausencia de valores en cada variable y caso del conjunto
# de datos `airquality`
miss_case_table(airquality)
```

```
## # A tibble: 3 x 3
##   n_miss_in_case n_cases pct_cases
##            <int>   <int>     <dbl>
## 1              0     111     72.5
## 2              1      40     26.1
## 3              2       2      1.31
```

```
miss_var_table(airquality)
```

```
## # A tibble: 3 x 3
##   n_miss_in_var n_vars pct_vars
##           <int>  <int>    <dbl>
## 1             0      4     66.7
## 2             7      1     16.7
## 3            37      1     16.7
```

```
# Tabula la ausencia de valores en cada variable, agrupados por Mes,
# en el conjunto de datos `airquality`
airquality %>% group_by(Month) %>% miss_var_table()
```

```
## # A tibble: 12 x 4
## # Groups:   Month [5]
##    Month n_miss_in_var n_vars pct_vars
##    <int>         <int>  <int>    <dbl>
##  1     5             0      3       60
##  2     5             4      1       20
##  3     5             5      1       20
##  4     6             0      4       80
##  5     6            21      1       20
##  6     7             0      4       80
##  7     7             5      1       20
##  8     8             0      3       60
##  9     8             3      1       20
```

```
## 10     8             5      1       20
## 11     9             0      4       80
## 12     9             1      1       20
```

```
# Tabula la ausencia de valores en cada caso, agrupados por Mes,
# en el conjunto de datos `airquality`
airquality %>% group_by(Month) %>% miss_case_table()
```

```
## # A tibble: 11 x 4
## # Groups:   Month [5]
##    Month n_miss_in_case n_cases pct_cases
##    <int>          <int>   <int>     <dbl>
##  1     5              0      24     77.4
##  2     5              1       5     16.1
##  3     5              2       2      6.45
##  4     6              0       9     30
##  5     6              1      21     70
##  6     7              0      26     83.9
##  7     7              1       5     16.1
##  8     8              0      23     74.2
##  9     8              1       8     25.8
## 10     9              0      29     96.7
## 11     9              1       1      3.33
```

1.5.3 Otros resúmenes de valores faltantes

Algunos resúmenes de valores faltantes son particularmente útiles para diferentes tipos de datos. Por ejemplo, `miss_var_span()` y `miss_var_run()`.

`miss_var_span()` calcula el número de valores faltantes en una variable especificada para un intervalo repetido. Esto es muy útil en datos de series de tiempo, para buscar patrones de valores faltantes semanales (de 7 días).

`miss_var_run()` calcula el número de "rachas" o "cadenas" de valores faltantes. Esto es útil para encontrar patrones inusuales de valores faltantes, por ejemplo, podría encontrar un patrón repetitivo de 5 completos y 5 faltantes.

Tanto `miss_var_span()` como `miss_var_run()` funcionan con el operador `group_by` de `dplyr`.

```
# Calcula los resúmenes para cada secuencia de ausencia de valores
# para la variable hourly_counts
miss_var_run(pedestrian, var = hourly_counts)
```

```
## # A tibble: 35 x 2
##    run_length is_na
##         <int> <chr>
##  1       6628 complete
##  2          1 missing
##  3       5250 complete
##  4        624 missing
##  5       3652 complete
##  6          1 missing
##  7       1290 complete
##  8        744 missing
##  9       7420 complete
## 10          1 missing
## # i 25 more rows
```

```
# Calcula los resúmenes para cada intervalo de ausencia de valores,
# con un intervalo de 4000, para la variable hourly_counts
miss_var_span(pedestrian, var = hourly_counts, span_every = 4000)
```

```
## # A tibble: 10 x 6
##    span_counter n_miss n_complete prop_miss prop_complete n_in_span
##           <int>  <int>      <int>     <dbl>         <dbl>     <int>
##  1            1      0       4000   0               1          4000
##  2            2      1       3999   0.00025         1.00       4000
##  3            3    121       3879   0.0302          0.970      4000
##  4            4    503       3497   0.126           0.874      4000
##  5            5    745       3255   0.186           0.814      4000
##  6            6      0       4000   0               1          4000
##  7            7      1       3999   0.00025         1.00       4000
##  8            8      0       4000   0               1          4000
##  9            9    745       3255   0.186           0.814      4000
## 10           10    432       1268   0.254           0.746      1700
```

```
# Para cada variable `month`, calcular la secuencia de ausencia de
# valores para hourly_counts
pedestrian %>% group_by(month) %>% miss_var_run(var = hourly_counts)
```

```
## # A tibble: 51 x 3
## # Groups:   month [12]
##    month    run_length is_na
##    <ord>         <int> <chr>
##  1 January        2976 complete
##  2 February       2784 complete
##  3 March          2976 complete
##  4 April           888 complete
##  5 April           552 missing
##  6 April          1440 complete
##  7 May             744 complete
##  8 May              72 missing
##  9 May            2160 complete
## 10 June           2880 complete
## # i 41 more rows
```

```
# Para cada variable `month`, calcular el intervalo de ausencia de
# valores con un intervalo de 2000, para la variable hourly_counts
pedestrian %>% group_by(month) %>% miss_var_span(var = hourly_counts, span_every = 2000)
```

```
## # A tibble: 25 x 7
## # Groups:   month [12]
##    month    span_counter n_miss n_complete prop_miss prop_complete n_in_span
##    <ord>           <int>  <int>      <int>     <dbl>         <dbl>     <int>
##  1 January             1      0       2000     0             1          2000
##  2 January             2      0        976     0             1           976
##  3 February            1      0       2000     0             1          2000
##  4 February            2      0        784     0             1           784
##  5 March               1      0       2000     0             1          2000
##  6 March               2      0        976     0             1           976
##  7 April               1    552       1448     0.276         0.724      2000
##  8 April               2      0        880     0             1           880
##  9 May                 1     72       1928     0.036         0.964      2000
```

```
## 10 May                2      0        976      0             1           976
## # i 15 more rows
```

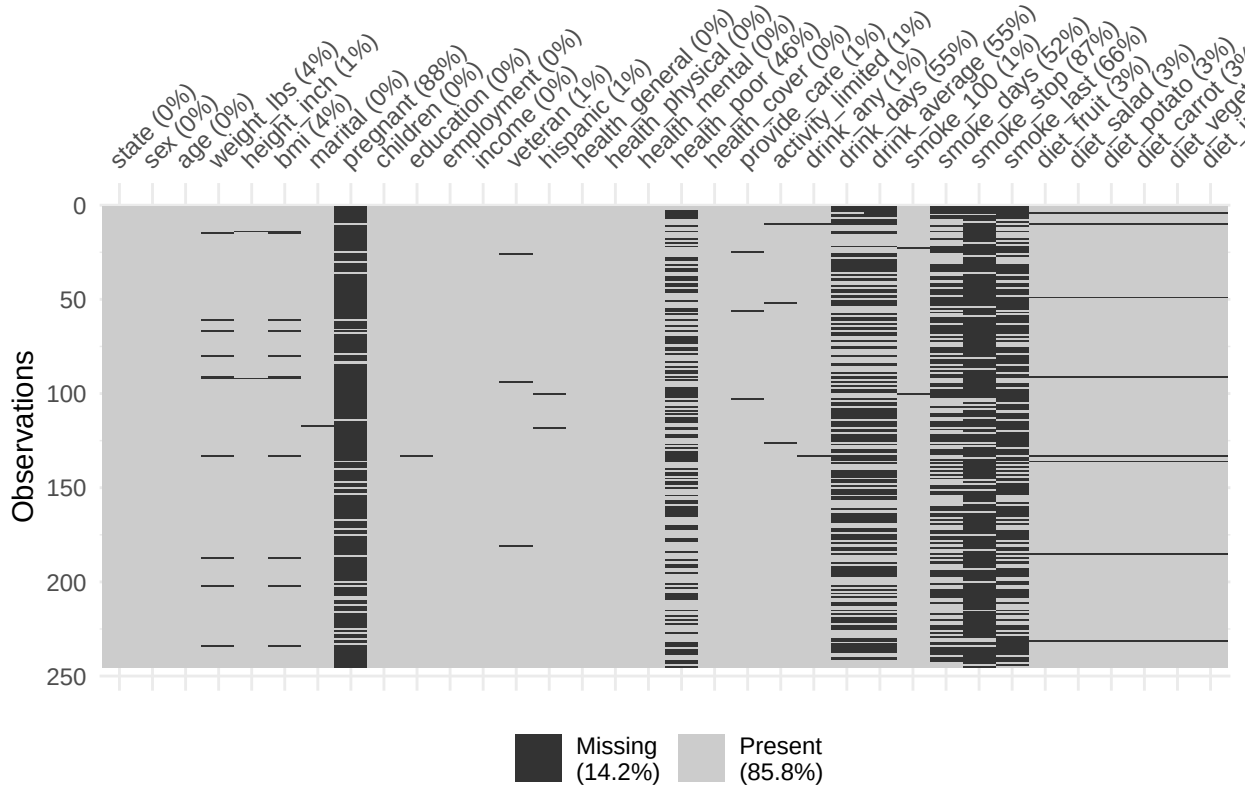
1.6 ¿Como visualizar los valores perdidos?

1.6.1 Primeras visualizaciones de datos faltantes

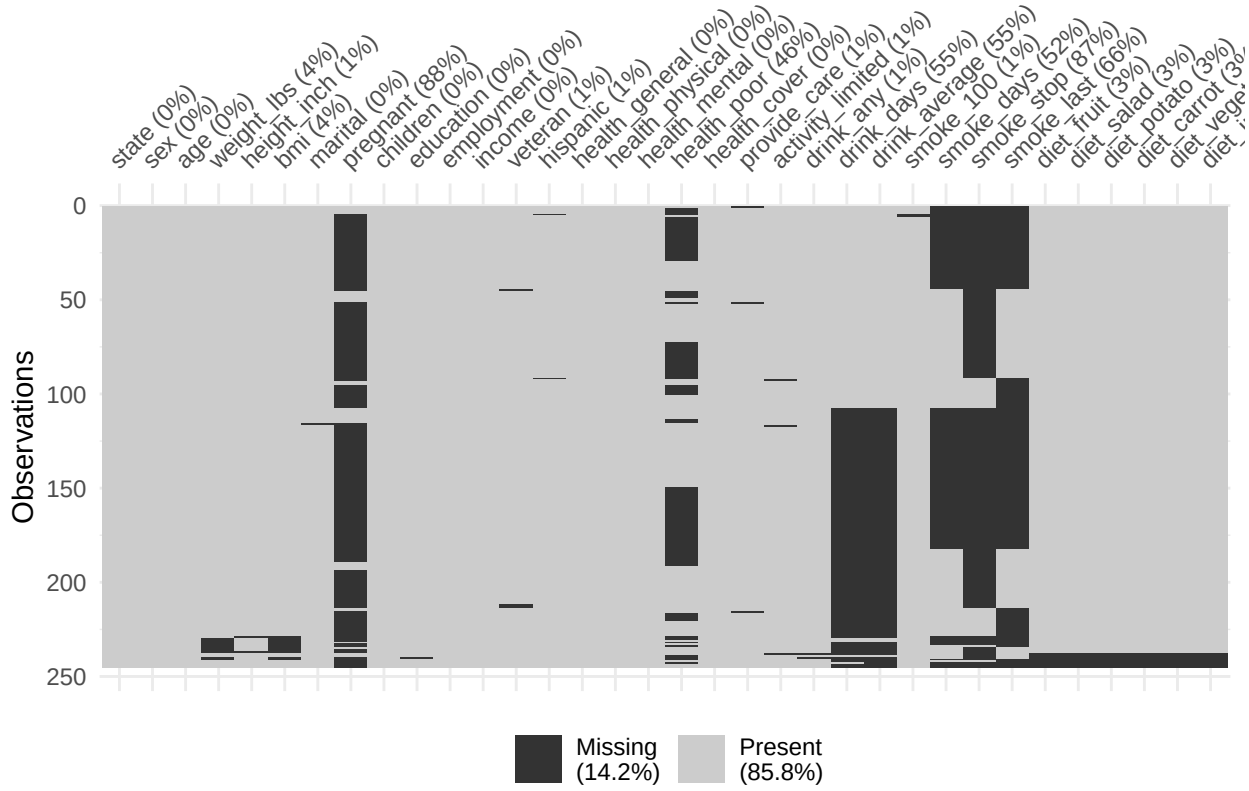
Puede resultar difícil determinar dónde están los valores faltantes en sus datos, y aquí es donde la visualización realmente puede ayudar.

La función `vis_miss()` crea una visualización general de la falta de datos en los datos. También tiene opciones para agrupar filas según la falta de datos, usando `cluster = TRUE`; así como opciones para ordenar las columnas, desde las más faltantes hasta las menos faltantes (`sort_miss = TRUE`).

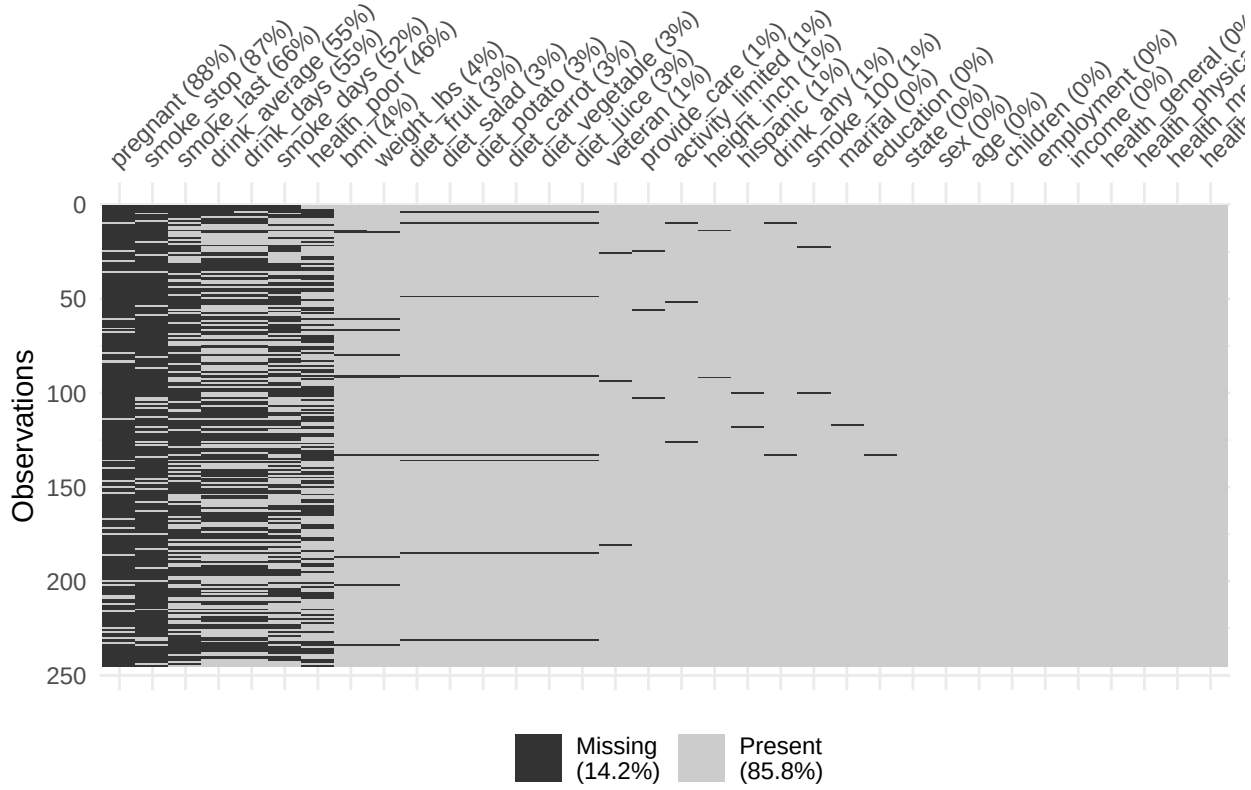
```
# Visualizar la ausencia de valores en el conjunto de datos 'riskfactors'
vis_miss(riskfactors)
```




```
# Visualizar y clusterizar la ausencia de valores en el conjunto de datos 'riskfactors'
vis_miss(riskfactors, cluster = TRUE)
```




```
# Visualizar y ordenar las columnas por la ausencia de valores en el conjunto de datos 'riskfactors'
vis_miss(riskfactors, sort_miss = TRUE)
```

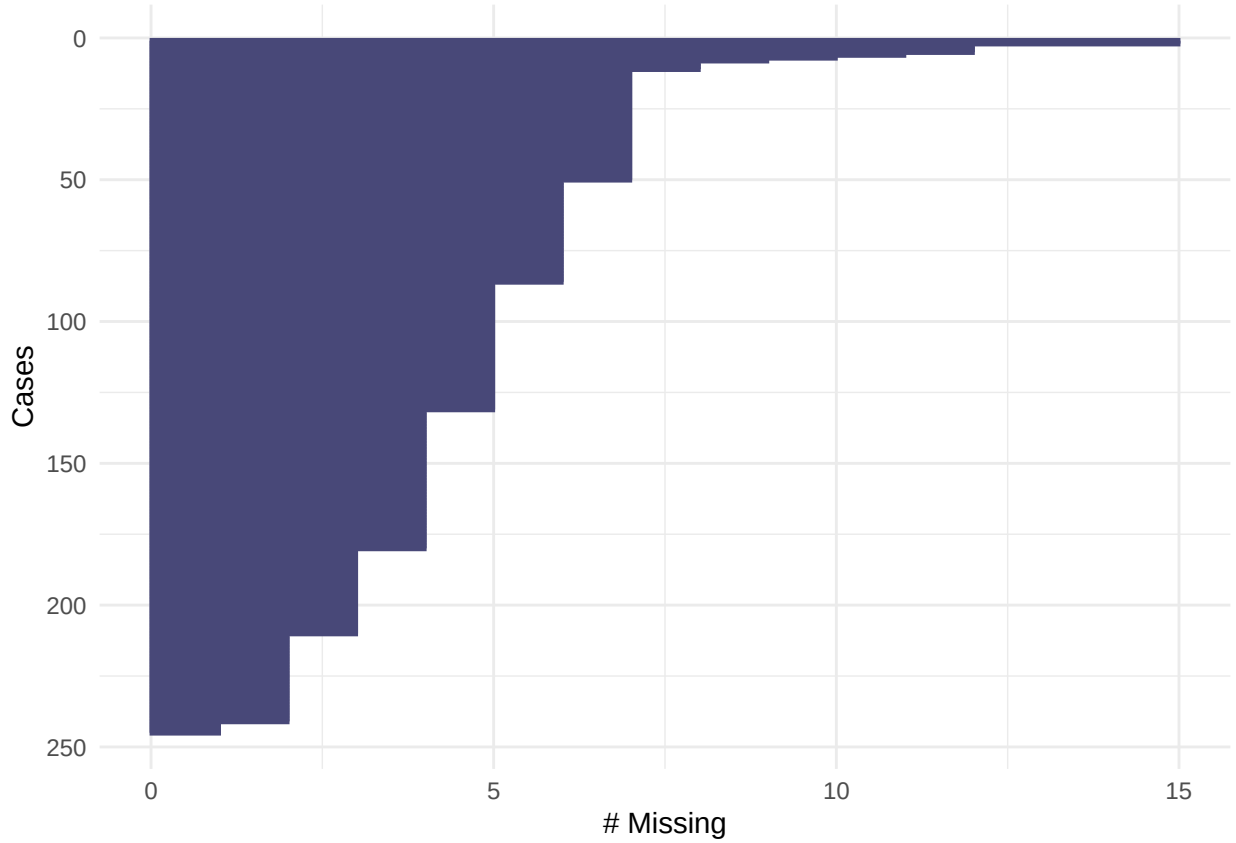


1.6.2 Visualización de casos y variables faltantes

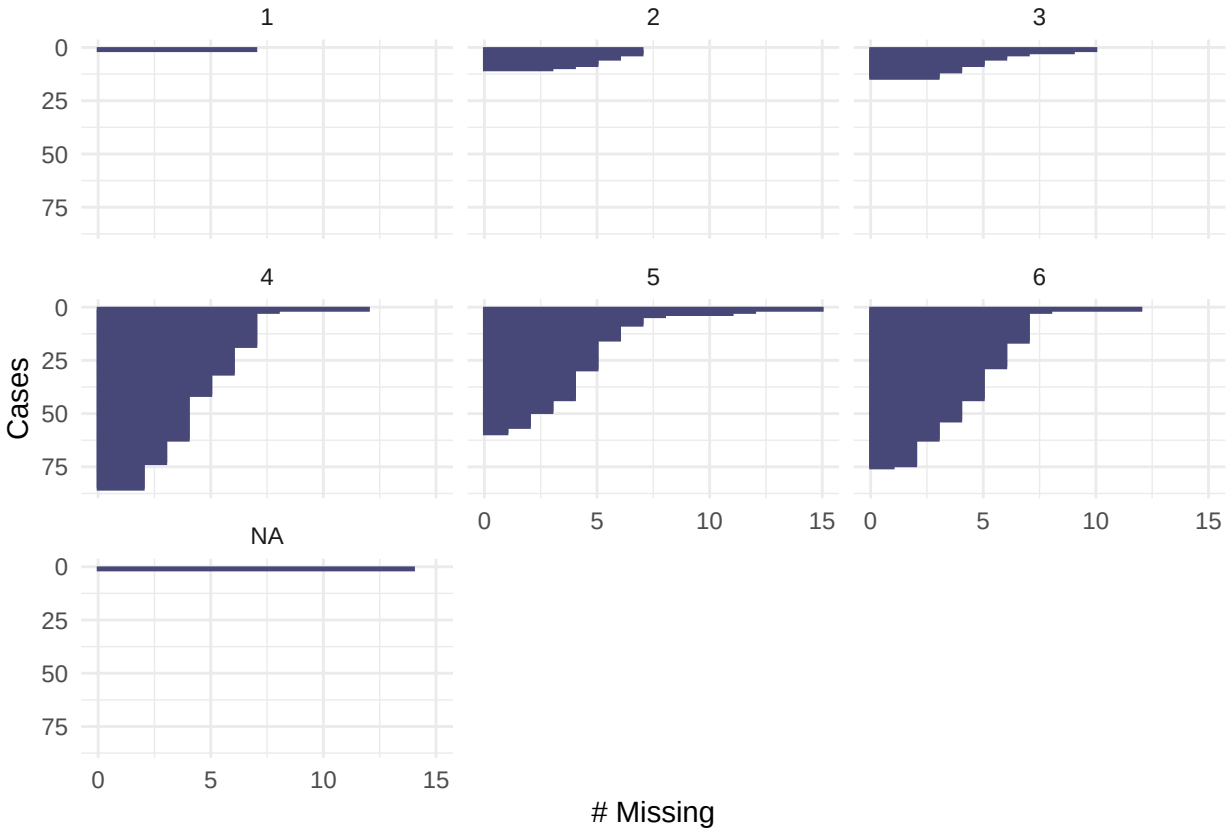
Para obtener una imagen clara de los valores faltantes en las variables y casos, utiliza `gg_miss_var()` y `gg_miss_case()`. Estas son las versiones visuales de `miss_var_summary()` y `miss_case_summary()`.

Estos pueden dividirse en múltiples gráficos, uno para cada categoría, eligiendo una variable para segmentar por ella.

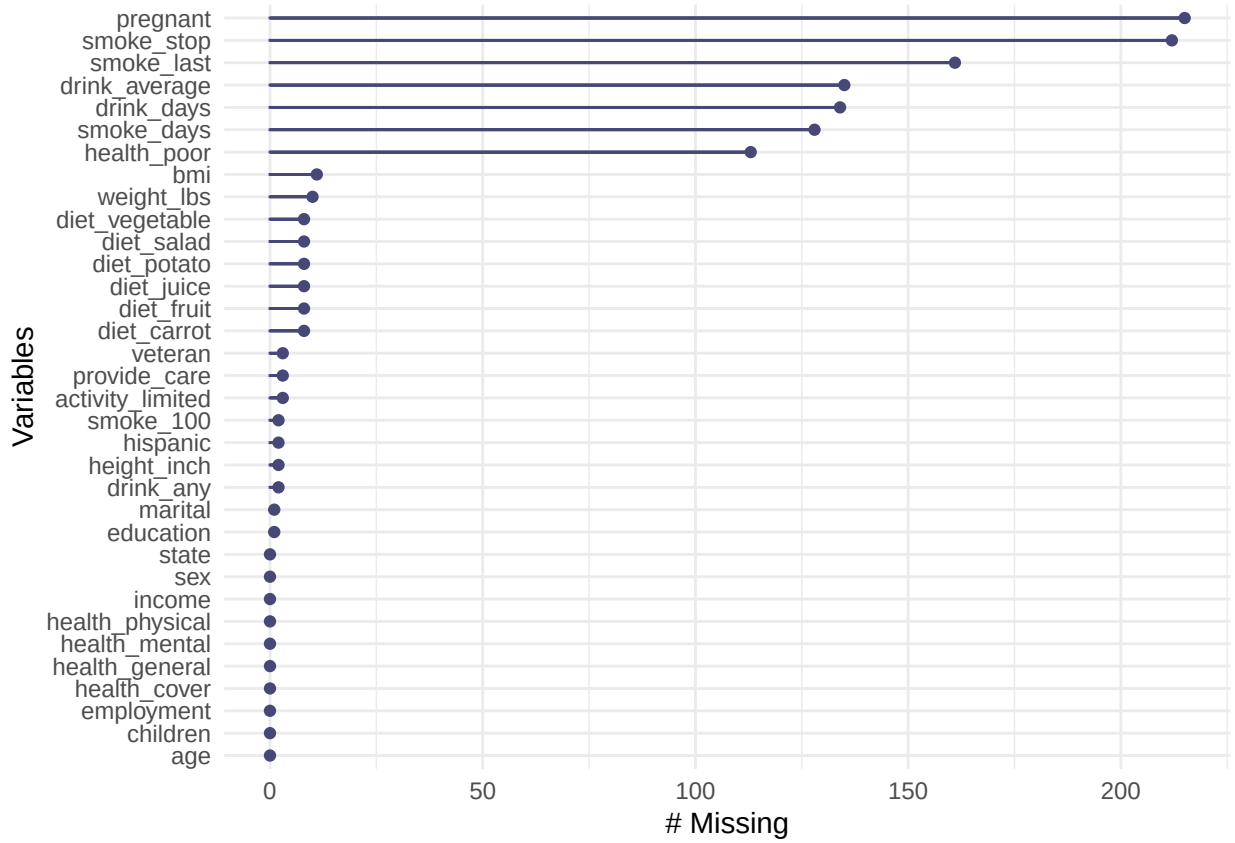
```
# Visualizar el número de valores faltantes en casos utilizando gg_miss_case()
gg_miss_case(riskfactors)
```



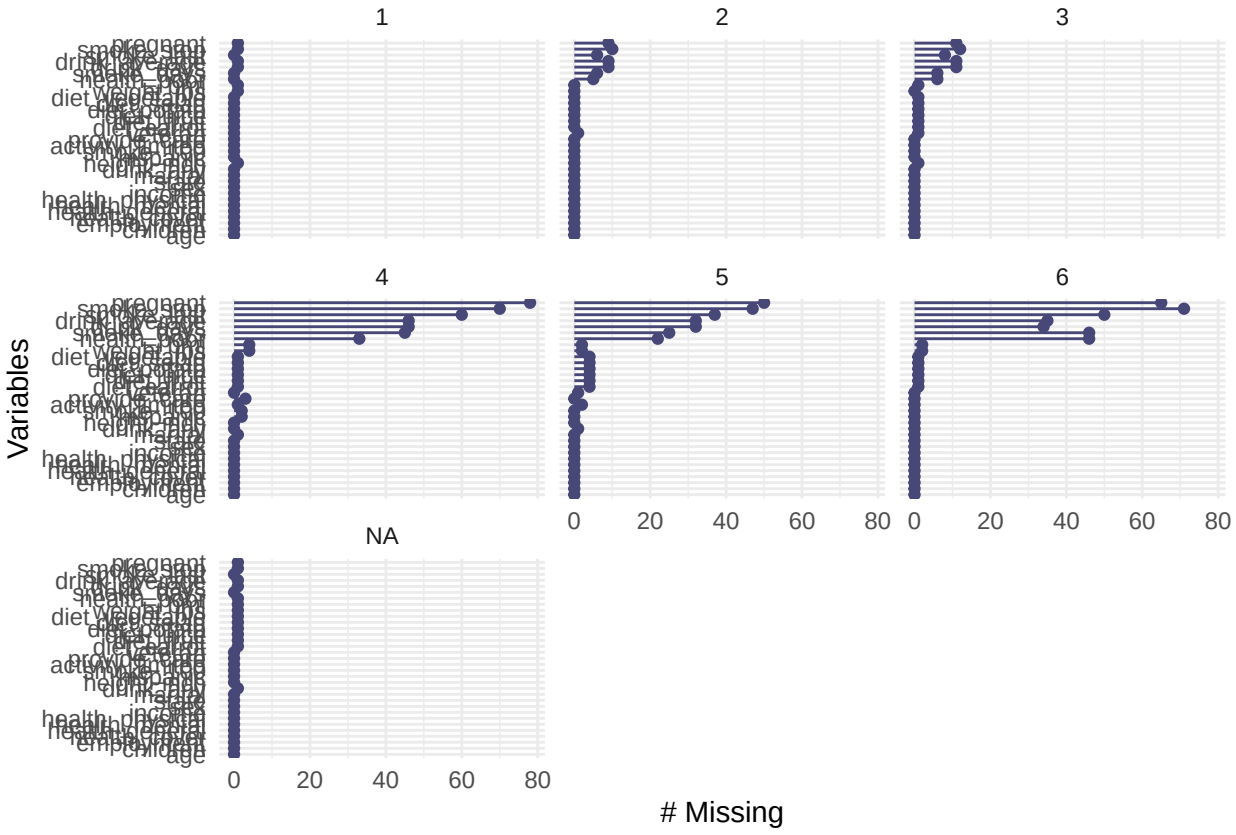

```
# Explorar el número de valores faltantes en casos y facetar por la variable `education`
gg_miss_case(riskfactors, facet = education)
```




```
# Visualizar el número de valores faltantes en las variables utilizando gg_miss_var()
gg_miss_var(riskfactors)
```




```
# Explora el numero de valores perdidos en las variables usando `gg_miss_var()`
# y facet por la variable `education`
gg_miss_var(riskfactors, facet = education)
```

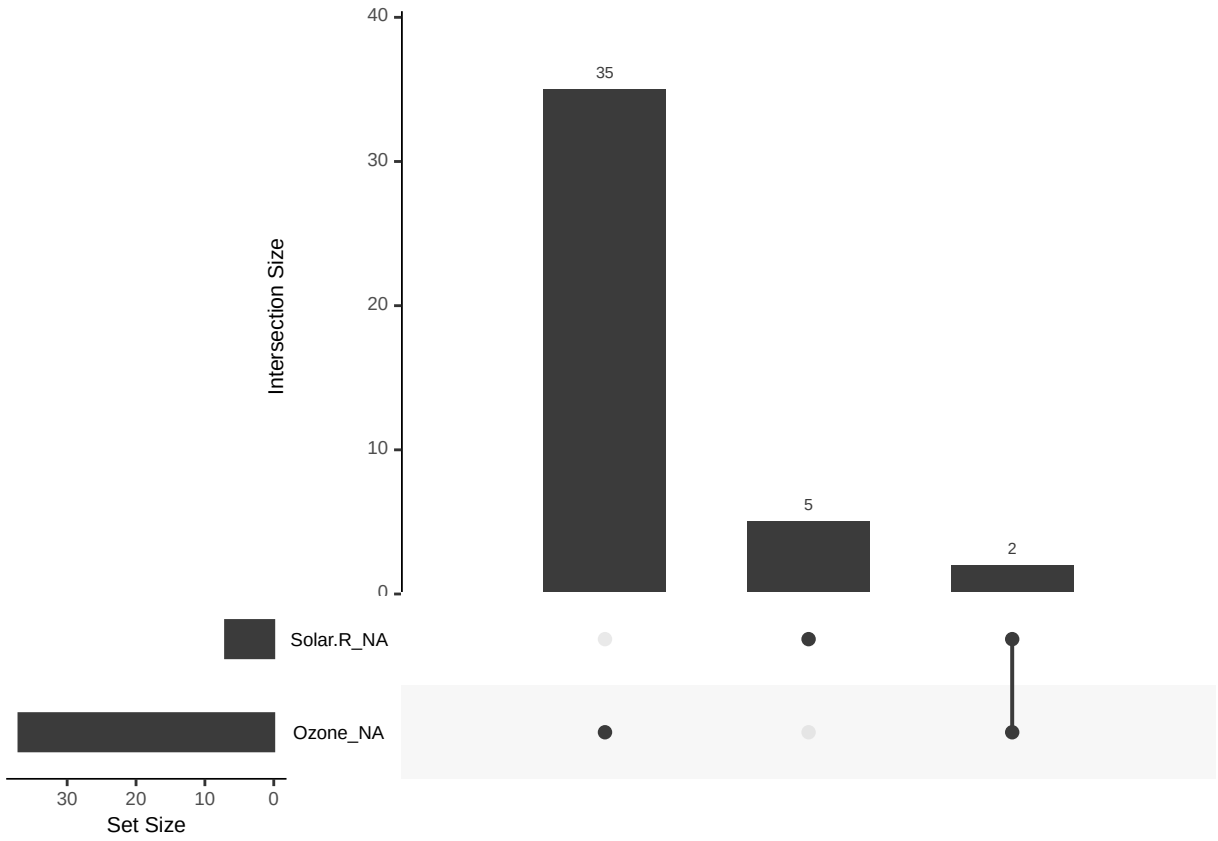


1.7 Visualizando patrones de datos faltantes

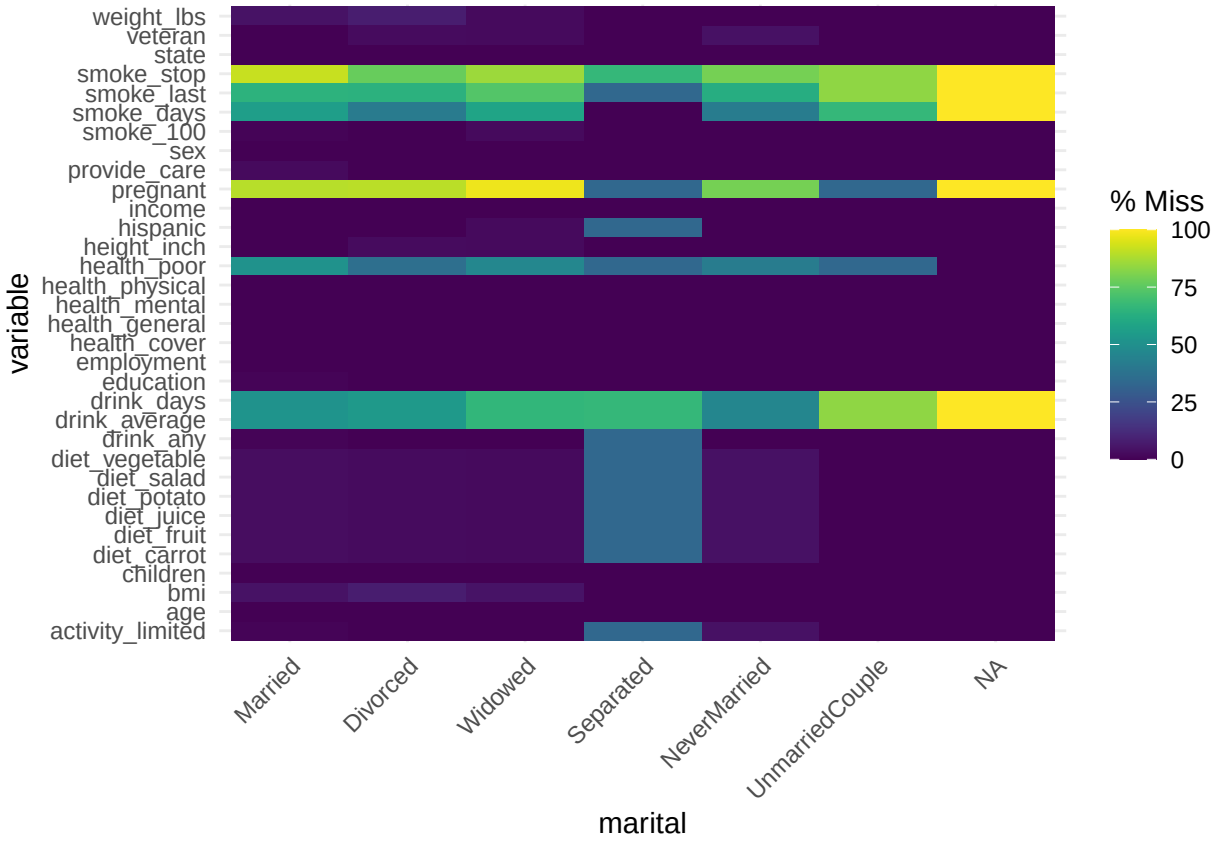
Practiquemos algunas formas diferentes de visualizar patrones de datos faltantes usando:

`gg_miss_upset()` para mostrar un patrón general de datos faltantes. `gg_miss_fct()` para un conjunto de datos que tiene un factor de interés: el estado marital. y `gg_miss_span()` para explorar los datos faltantes en un conjunto de datos de series de tiempo. ¿Qué notas sobre los datos faltantes y la segmentación en los datos?

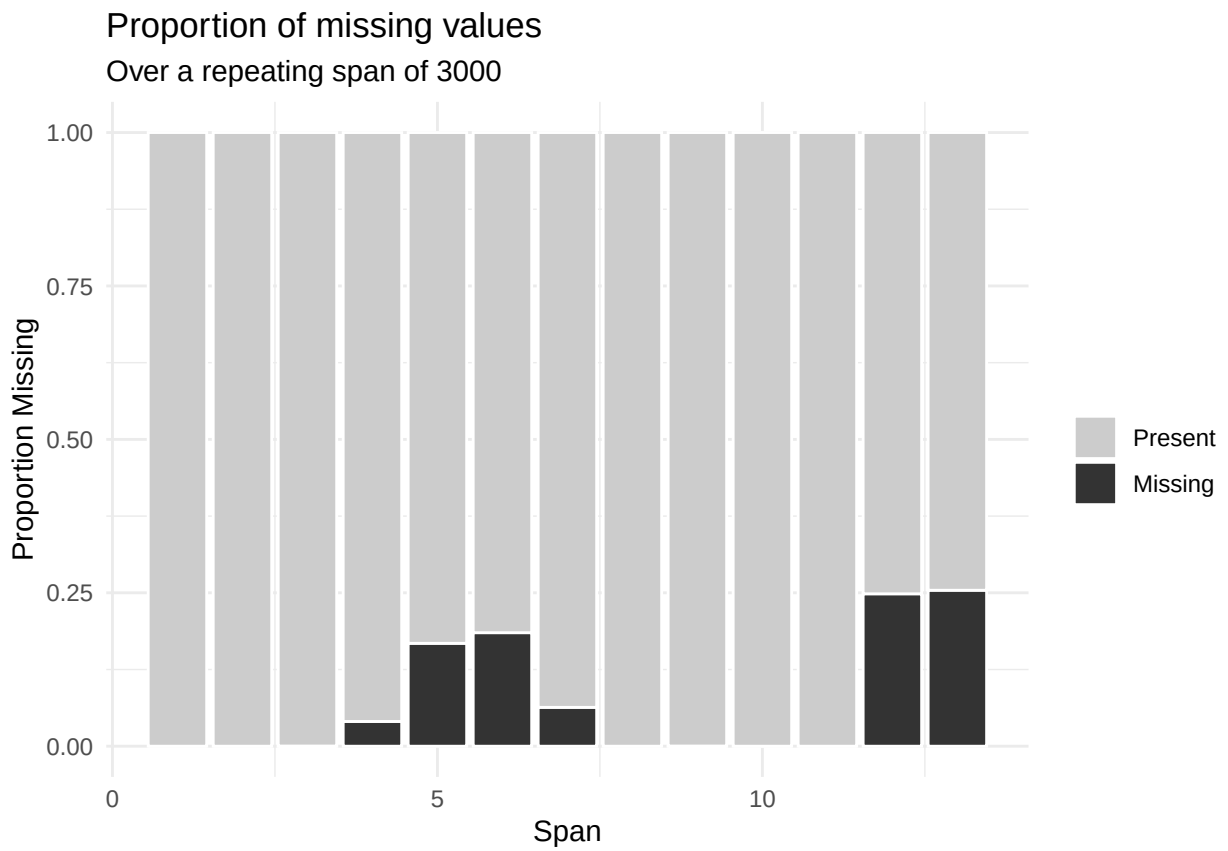
```
# Explore el patrón de ausencia de valores en el conjunto de datos 'airquality'
# con gg_miss_upset()

gg_miss_upset(airquality)
```




```
# Explora cómo cambia la ausencia de valores en relación con la variable
# `marital` con gg_miss_fct()
gg_miss_fct(x = riskfactors, fct = marital)
```




```
# Usando el set de datos pedestrian, explore como la perdida de la variable
# hourly_counts cambia para un span de 3000
gg_miss_span(pedestrian, var = hourly_counts, span_every = 3000)
```




```
# Usando el set de datos `pedestrian`, explore el impacto de la variable
# `month` usando facetting por `month` y explore como la perdida cambia
# para un span de 1000
gg_miss_span(pedestrian, var = hourly_counts , span_every = 1000, facet = month)
```

2 Limpieza y organización de valores faltantes

En la sección dos, aprenderás cómo descubrir valores faltantes ocultos como "missing" o "N/A" y reemplazarlos con `NA`. Aprenderás cómo manejar de manera eficiente los valores faltantes implícitos: aquellos valores que se dan por sentados como faltantes pero que no se enumeran explícitamente. También cubriremos cómo explorar la dependencia de datos faltantes, discutiendo el tipo de patrón de perdida Missing Completely at Random (MCAR), Missing At Random (MAR), Missing Not At Random (MNAR) y lo que significan para el análisis de tus datos.

2.1 Búsqueda y reemplazo de valores faltantes

2.1.1 Usando miss_scan_count

Tienes un conjunto de datos con valores faltantes codificados como `"N/A"`, `"missing"` y `"na"`. Pero antes de seguir adelante y comenzar a reemplazarlos con `NA`, deberíamos tener una idea de qué tan grande es el problema.

Utiliza `miss_scan_count` para contar los valores faltantes posibles en el conjunto de datos `pacman`, que contiene tres columnas:

`year`: el año en que la persona obtuvo esa puntuación. `initial`: las iniciales de la persona. `score`: las puntuaciones de esa persona.

```
# Usando el dataframe de ejemplo
pacman<-readRDS("dealing/pacman.rds")
```

```r
# Explore los raros valores missing "N/A"
miss_scan_count(data = pacman, search = list("N/A"))
```

```
## # A tibble: 6 x 2
##   Variable     n
##   <chr>    <int>
## 1 year         0
## 2 month        0
## 3 day          0
## 4 initial      0
## 5 score      100
## 6 country      0
```

```r
# Explore los raros valores missing "missing"
miss_scan_count(data = pacman, search = list("missing"))
```

```
## # A tibble: 6 x 2
##   Variable     n
##   <chr>    <int>
## 1 year        93
## 2 month       93
## 3 day         93
## 4 initial      0
## 5 score        0
## 6 country      0
```

```r
# Explore los raros valores missing "na"
miss_scan_count(data = pacman, search = list("na"))
```

```
## # A tibble: 6 x 2
##   Variable     n
##   <chr>    <int>
## 1 year       100
## 2 month      100
## 3 day        100
## 4 initial      0
## 5 score        0
## 6 country      0
```

```r
# Explore los raros valores missing " " (un simple espacio)
miss_scan_count(data = pacman, search = list(" "))
```

```
## # A tibble: 6 x 2
##   Variable     n
##   <chr>    <int>
## 1 year         0
## 2 month        0
## 3 day          0
## 4 initial      0
## 5 score        0
## 6 country    100
```

```r
# Explore todos los raros valores, "N/A", "missing", "na", " "
miss_scan_count(data = pacman, search = list("N/A", "missing","na", " "))
```

```
## # A tibble: 6 x 2
##   Variable     n
```

```
##   <chr>    <int>
## 1 year       193
## 2 month      193
## 3 day        193
## 4 initial      0
## 5 score      100
## 6 country    100
```

2.1.2 Usando replace_with_na

Siguiendo con el conjunto de datos anterior, ahora sabemos que tenemos algunos valores faltantes extraños.

Ahora, vamos a hacer algo al respecto y reemplazar estos valores con valores faltantes (por ejemplo, `NA`) usando la función `replace_with_na()`. Que en este caso es como se definen por convención en R como un valor perdido real.

```
# Imprimir el inicio de los datos de pacman utilizando `head()`
head(pacman)
```

```
## # A tibble: 6 x 6
##   year  month day   initial score   country
##   <chr> <chr> <chr> <chr>   <chr>   <chr>
## 1 2007  10    27    LEX     2065812 "CA"
## 2 1995  8     23    PNY     1163465 "JP"
## 3 1980  2     8     MBJ     175380  " "
## 4 1982  5     9     QRC     2025632 "ES"
## 5 na    na    na    YPZ     925357  "NZ"
## 6 2013  11    15    RVJ     319733  "AU"
```

```
# Reemplazar los valores faltantes extraños "N/A", "na" y "missing" con `NA` para las variables
# year y score
pacman_clean <- replace_with_na(pacman, replace = list(year = c("N/A", "na", "missing"),
                                score = c("N/A", "na", "missing")))

# ¿Se conservan estos valores en `pacman_clean`?
miss_scan_count(pacman_clean, search = list("N/A", "na", "missing"))
```

```
## # A tibble: 6 x 2
##   Variable     n
##   <chr>    <int>
## 1 year         0
## 2 month      193
## 3 day        193
## 4 initial      0
## 5 score        0
## 6 country      0
```

2.1.3 Usando las variantes "scoped" de replace_with_na

Para reducir la repetición de código al reemplazar valores con `NA`, utiliza las variantes "scoped" de `replace_with_na()`:

- `replace_with_na_at()`
- `replace_with_na_if()`
- `replace_with_na_all()`

La sintaxis de reemplazo se ve así:

`~.x == "N/A"` Esto reemplaza todos los casos que son iguales a `"N/A"`.

`~.x %in% c("N/A", "missing", "na", " ")` Reemplaza todos los casos que tienen `"N/A"`, `"missing"`, `"na"` o `" "`.

```
# Usa `replace_with_na_at()` para replazar con NA
replace_with_na_at(pacman,
                   .vars = c("year", "month", "day"),
                   ~.x %in% c("N/A", "missing", "na", " "))
```

```
## # A tibble: 2,000 x 6
##    year  month day   initial score   country
##    <chr> <chr> <chr> <chr>   <chr>   <chr>
##  1 2007  10    27    LEX     2065812 "CA"
##  2 1995  8     23    PNY     1163465 "JP"
##  3 1980  2     8     MBJ     175380  " "
##  4 1982  5     9     QRC     2025632 "ES"
##  5 <NA>  <NA>  <NA>  YPZ     925357  "NZ"
##  6 2013  11    15    RVJ     319733  "AU"
##  7 2003  12    4     VKD     3322668 "US"
##  8 2016  9     9     ZIS     2137806 "CN"
##  9 2013  3     20    IYD     3059716 "CN"
## 10 1993  5     19    CHQ     231892  "AU"
## # i 1,990 more rows
```

```
# Usa `replace_with_na_if()` para replazar con NA el valor character usando
# `is.character`
replace_with_na_if(pacman,
                   .predicate = is.character,
                   ~.x %in% c("N/A", "missing", "na", " "))
```

```
## # A tibble: 2,000 x 6
##    year  month day   initial score   country
##    <chr> <chr> <chr> <chr>   <chr>   <chr>
##  1 2007  10    27    LEX     2065812 CA
##  2 1995  8     23    PNY     1163465 JP
##  3 1980  2     8     MBJ     175380  <NA>
##  4 1982  5     9     QRC     2025632 ES
##  5 <NA>  <NA>  <NA>  YPZ     925357  NZ
##  6 2013  11    15    RVJ     319733  AU
##  7 2003  12    4     VKD     3322668 US
##  8 2016  9     9     ZIS     2137806 CN
##  9 2013  3     20    IYD     3059716 CN
## 10 1993  5     19    CHQ     231892  AU
## # i 1,990 more rows
```

```
# Usa `replace_with_na_all()` para replazar con NA
replace_with_na_all(pacman, ~.x %in% c("N/A", "missing", "na", " "))
```

```
## # A tibble: 2,000 x 6
##    year  month day   initial score   country
##    <chr> <chr> <chr> <chr>   <chr>   <chr>
##  1 2007  10    27    LEX     2065812 CA
##  2 1995  8     23    PNY     1163465 JP
##  3 1980  2     8     MBJ     175380  <NA>
##  4 1982  5     9     QRC     2025632 ES
##  5 <NA>  <NA>  <NA>  YPZ     925357  NZ
```

```
##  6 2013  11    15    RVJ     319733  AU
##  7 2003  12    4     VKD     3322668 US
##  8 2016  9     9     ZIS     2137806 CN
##  9 2013  3     20    IYD     3059716 CN
## 10 1993  5     19    CHQ     231892  AU
## # i 1,990 more rows
```

2.2 Rellenando valores faltantes hacia abajo

2.2.1 Arreglando valores faltantes implícitos usando `complete()`

Vamos a explorar un nuevo conjunto de datos, `frogger`.

Este conjunto de datos contiene 4 puntuaciones por jugador registradas en diferentes momentos: `morning`, `afternoon`, `evening` y `late_night`.

Cada jugador debería haber jugado 4 partidas, una en cada uno de estos momentos, pero parece que no todos los jugadores completaron todos estos juegos.

Utiliza la función `complete()` para hacer explícitos estos valores faltantes implícitos.

```
frogger <- read_excel("dealing/frogger.xlsx")
```

```
# muestra los datos dee frogger para observarlos
head(frogger)
```

```
## # A tibble: 6 x 3
##   name  time       value
##   <chr> <chr>      <chr>
## 1 jesse morning    6678
## 2 jesse afternoon  800060
## 3 jesse evening    475528
## 4 jesse late_night 143533
## 5 andy  morning    425115
## 6 andy  afternoon  587468
```

```
# Usa `complete()` en la variable `time` y `name` para
# hacer los valores perdidos implicitos, hacerlos explicitos
frogger_tidy <- frogger %>% complete(time, name)
```

2.2.2 Arreglando valores faltantes explícitos usando `fill()`

Un tipo de valor faltante que puede ser obvio de tratar es donde se da la primera entrada de un grupo, pero las entradas subsiguientes están marcadas como `NA`.

Estos valores faltantes a menudo son el resultado de valores vacíos en hojas de cálculo para evitar ingresar múltiples nombres varias veces; así como para "legibilidad humana".

Este tipo de problema se puede resolver utilizando la función `fill()` del paquete `tidyr`.

```
# Imprimir los datos de frogger para examinarlos
frogger
```

```
## # A tibble: 15 x 3
##    name  time       value
##    <chr> <chr>      <chr>
##  1 jesse morning    6678
##  2 jesse afternoon  800060
##  3 jesse evening    475528
```

```
##  4 jesse late_night 143533
##  5 andy  morning    425115
##  6 andy  afternoon  587468
##  7 andy  late_night 111000
##  8 nic   afternoon  588532
##  9 nic   late_night 915533
## 10 dan   morning    388148
## 11 dan   evening    180912
## 12 alex  morning    552670
## 13 alex  afternoon  98355
## 14 alex  evening    266055
## 15 alex  late_night 121056
```

```
# Utilizar `fill()` para rellenar la variable name en el conjunto de
# datos frogger
frogger %>% fill(name)
```

```
## # A tibble: 15 x 3
##    name  time       value
##    <chr> <chr>      <chr>
##  1 jesse morning    6678
##  2 jesse afternoon  800060
##  3 jesse evening    475528
##  4 jesse late_night 143533
##  5 andy  morning    425115
##  6 andy  afternoon  587468
##  7 andy  late_night 111000
##  8 nic   afternoon  588532
##  9 nic   late_night 915533
## 10 dan   morning    388148
## 11 dan   evening    180912
## 12 alex  morning    552670
## 13 alex  afternoon  98355
## 14 alex  evening    266055
## 15 alex  late_night 121056
```

2.2.3 Usando `complete()` y `fill()` juntos

¡Ahora pongámoslo todo junto!

Utiliza `complete()` y `fill()` juntos para corregir valores faltantes explícitos e implícitos en el conjunto de datos frogger.

```
# Imprime los datos de frogger para examinarlos
frogger
```

```
## # A tibble: 15 x 3
##    name  time       value
##    <chr> <chr>      <chr>
##  1 jesse morning    6678
##  2 jesse afternoon  800060
##  3 jesse evening    475528
##  4 jesse late_night 143533
##  5 andy  morning    425115
##  6 andy  afternoon  587468
##  7 andy  late_night 111000
```

```
##  8 nic   afternoon  588532
##  9 nic   late_night 915533
## 10 dan   morning    388148
## 11 dan   evening    180912
## 12 alex  morning    552670
## 13 alex  afternoon  98355
## 14 alex  evening    266055
## 15 alex  late_night 121056
```

```
# Usa fill() y complete() en los valores faltantes para que nuestro conjunto
# de datos sea coherente
frogger %>%
  fill(name) %>%
  complete(name, time)
```

```
## # A tibble: 20 x 3
##    name  time       value
##    <chr> <chr>      <chr>
##  1 alex  afternoon  98355
##  2 alex  evening    266055
##  3 alex  late_night 121056
##  4 alex  morning    552670
##  5 andy  afternoon  587468
##  6 andy  evening    <NA>
##  7 andy  late_night 111000
##  8 andy  morning    425115
##  9 dan   afternoon  <NA>
## 10 dan   evening    180912
## 11 dan   late_night <NA>
## 12 dan   morning    388148
## 13 jesse afternoon  800060
## 14 jesse evening    475528
## 15 jesse late_night 143533
## 16 jesse morning    6678
## 17 nic   afternoon  588532
## 18 nic   evening    <NA>
## 19 nic   late_night 915533
## 20 nic   morning    <NA>
```

2.3 Dependencia de datos faltantes

2.3.1 Diferencias entre MCAR y MAR

Necesitamos hacer ciertas suposiciones sobre nuestros datos cuando avanzamos en el análisis.

¿Cuál de las siguientes respuestas sobre MCAR y MAR es VERDADERA?

2.4 Posibles respuestas

En general, eliminar observaciones es más seguro para datos MCAR que eliminar observaciones para datos MAR.

MCAR significa que la falta de datos está relacionada con los datos observados, mientras que para MAR, la falta de datos está relacionada con los datos no observados.

MAR y MCAR son efectivamente lo mismo, la distinción no es importante.

Los datos MCAR no están relacionados con los datos observados y no observados. Los datos MAR están relacionados con los datos observados.

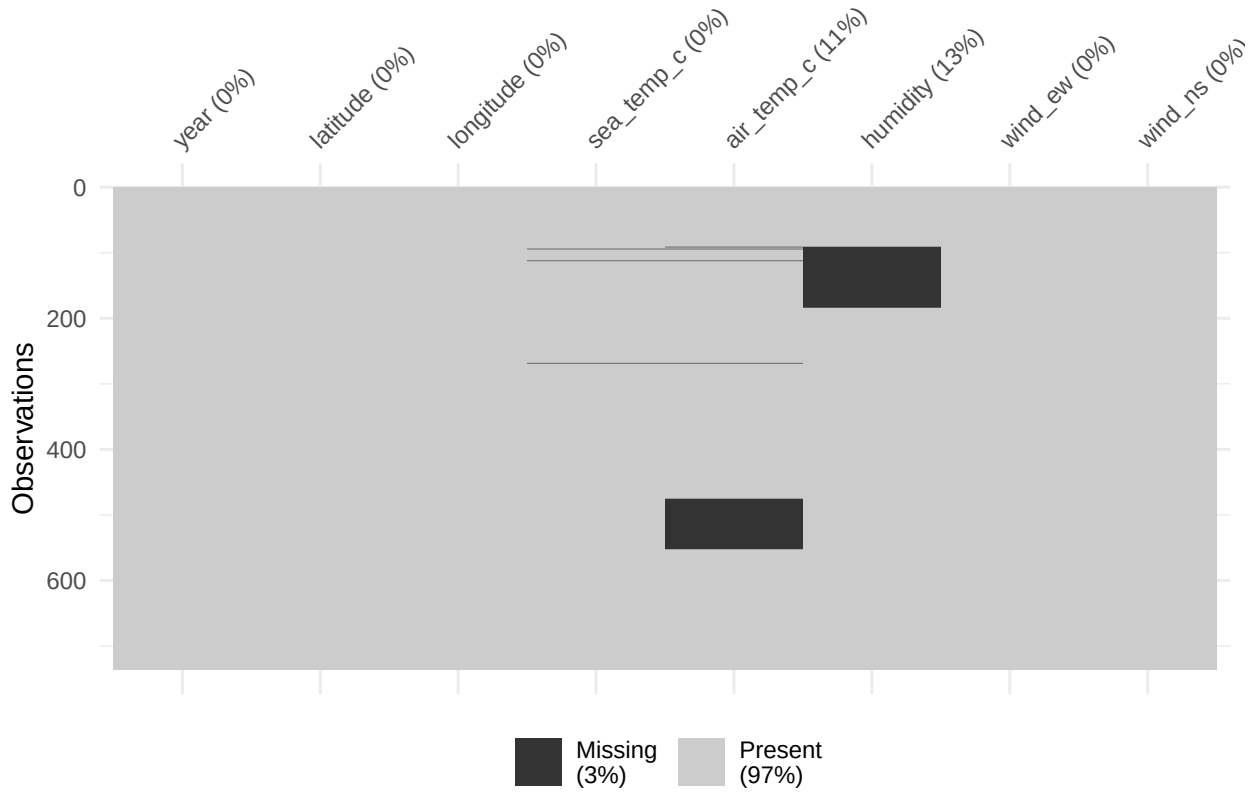
2.4.1 Explorando la dependencia de datos faltantes

Para aprender sobre la estructura de la falta de datos en los datos, podemos explorar cómo cambia la presentación de la falta de datos al ordenar los datos.

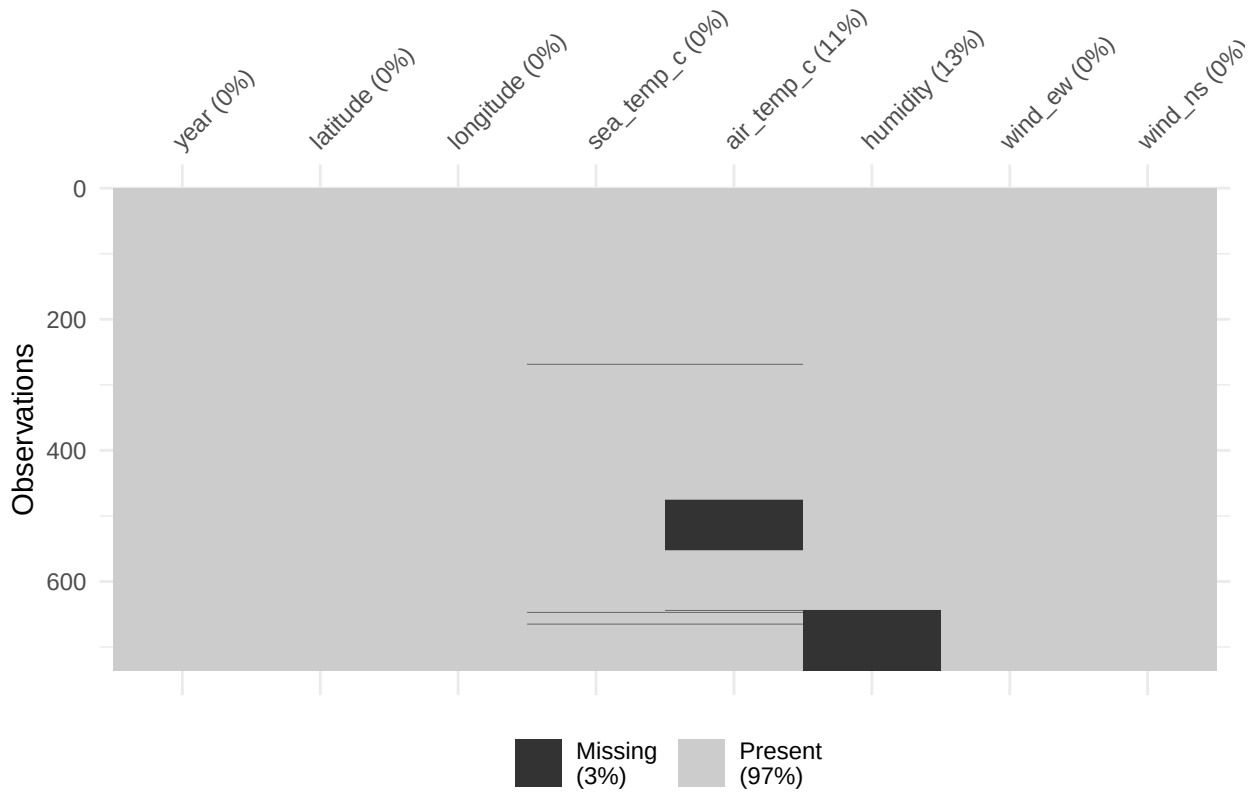
Para el conjunto de datos `oceanbuoys`, explora la falta de datos con `vis_miss()` y luego ordénalo por algunas variables diferentes.

Este no es un proceso definitivo, pero te ayudará a comenzar a hacer las preguntas correctas sobre tus datos. Exploramos técnicas más poderosas en la próxima sección.

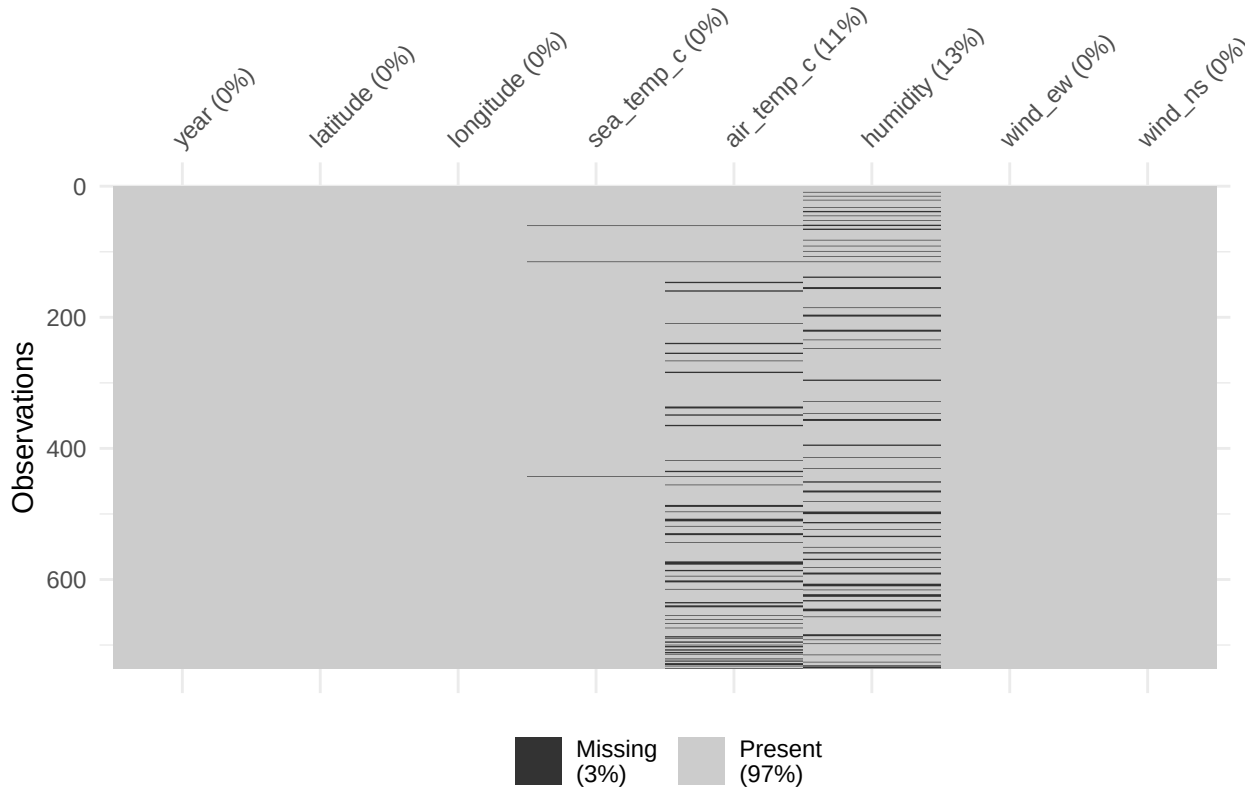
```
# Organizar por año
oceanbuoys %>% arrange(year) %>% vis_miss()
```




```
# Organizar por latitud
oceanbuoys %>% arrange(latitude) %>% vis_miss()
```




```
# Organizar por wind_ew (viento este-oeste)
oceanbuoys %>% arrange(wind_ew) %>% vis_miss()
```



2.4.2 Explorando aún más la dependencia de los datos faltantes

Usando la información previa sobre el conjunto de datos `oceanbuoys`, podemos considerar que *Los datos son MAR: tanto el año como la ubicación son importantes para explicar los datos faltantes.*. Es es posible concluirlo al revisar con `gg_miss_var()` y `gg_miss_case()`, agrupando por año para obtener más información. Por ejemplo:

```
library(naniar); gg_miss_var(oceanbuoys, facet = year)
```

2.5 Herramientas para explorar la dependencia de los datos faltantes

2.5.1 Creando datos de matriz de sombra

Los datos faltantes pueden ser complicados de pensar, ya que normalmente no se proclaman por sí mismos y en su lugar se esconden entre la maleza de los datos.

Una forma de ayudar a visualizar los valores faltantes es cambiar la forma en que pensamos sobre los datos, pensando en cada valor de datos como faltante o no faltante.

La función `as_shadow()` en R transforma un dataframe en una matriz de sombra, un formato de datos especial donde los valores son o bien faltantes (`NA`) o no faltantes (`!NA`).

Los nombres de columna de una matriz de sombra son los mismos que los datos, pero tienen un sufijo agregado `_NA`.

Para realizar un seguimiento y comparar los valores de los datos con su estado de falta, use la función `bind_shadow()`. Tener los datos en este formato, con la columna de matriz de sombra unida a los datos regulares, se llama datos `nabular`.

```r
# Crear una matriz de sombra de datos con `as_shadow()`
as_shadow(oceanbuoys)
```

```
## # A tibble: 736 x 8
##    year_NA latitude_NA longitude_NA sea_temp_c_NA air_temp_c_NA humidity_NA
##    <fct>   <fct>       <fct>        <fct>         <fct>         <fct>
##  1 !NA     !NA         !NA          !NA           !NA           !NA
##  2 !NA     !NA         !NA          !NA           !NA           !NA
##  3 !NA     !NA         !NA          !NA           !NA           !NA
##  4 !NA     !NA         !NA          !NA           !NA           !NA
##  5 !NA     !NA         !NA          !NA           !NA           !NA
##  6 !NA     !NA         !NA          !NA           !NA           !NA
##  7 !NA     !NA         !NA          !NA           !NA           !NA
##  8 !NA     !NA         !NA          !NA           !NA           !NA
##  9 !NA     !NA         !NA          !NA           !NA           !NA
## 10 !NA     !NA         !NA          !NA           !NA           !NA
## # i 726 more rows
## # i 2 more variables: wind_ew_NA <fct>, wind_ns_NA <fct>
```

```r
# Crear datos nablulares al unir la sombra a los datos con `bind_shadow()`
bind_shadow(oceanbuoys)
```

```
## # A tibble: 736 x 16
##     year latitude longitude sea_temp_c air_temp_c humidity wind_ew wind_ns
##    <dbl>    <dbl>     <dbl>      <dbl>      <dbl>    <dbl>   <dbl>   <dbl>
##  1  1997        0      -110       27.6       27.1     79.6   -6.40    5.40
##  2  1997        0      -110       27.5       27.0     75.8   -5.30    5.30
##  3  1997        0      -110       27.6       27       76.5   -5.10    4.5
##  4  1997        0      -110       27.6       26.9     76.2   -4.90    2.5
##  5  1997        0      -110       27.6       26.8     76.4   -3.5     4.10
##  6  1997        0      -110       27.8       26.9     76.7   -4.40    1.60
##  7  1997        0      -110       28.0       27.0     76.5   -2       3.5
##  8  1997        0      -110       28.0       27.1     78.3   -3.70    4.5
##  9  1997        0      -110       28.0       27.2     78.6   -4.20    5
## 10  1997        0      -110       28.0       27.2     76.9   -3.60    3.5
## # i 726 more rows
## # i 8 more variables: year_NA <fct>, latitude_NA <fct>, longitude_NA <fct>,
## #   sea_temp_c_NA <fct>, air_temp_c_NA <fct>, humidity_NA <fct>,
## #   wind_ew_NA <fct>, wind_ns_NA <fct>
```

```r
# Unir solo las variables con valores faltantes utilizando
# bind_shadow(only_miss = TRUE)
bind_shadow(oceanbuoys, only_miss = TRUE)
```

```
## # A tibble: 736 x 11
##     year latitude longitude sea_temp_c air_temp_c humidity wind_ew wind_ns
##    <dbl>    <dbl>     <dbl>      <dbl>      <dbl>    <dbl>   <dbl>   <dbl>
##  1  1997        0      -110       27.6       27.1     79.6   -6.40    5.40
##  2  1997        0      -110       27.5       27.0     75.8   -5.30    5.30
##  3  1997        0      -110       27.6       27       76.5   -5.10    4.5
##  4  1997        0      -110       27.6       26.9     76.2   -4.90    2.5
##  5  1997        0      -110       27.6       26.8     76.4   -3.5     4.10
##  6  1997        0      -110       27.8       26.9     76.7   -4.40    1.60
##  7  1997        0      -110       28.0       27.0     76.5   -2       3.5
##  8  1997        0      -110       28.0       27.1     78.3   -3.70    4.5
```

```
##  9  1997        0      -110       28.0       27.2     78.6   -4.20    5
## 10  1997        0      -110       28.0       27.2     76.9   -3.60    3.5
## # i 726 more rows
## # i 3 more variables: sea_temp_c_NA <fct>, air_temp_c_NA <fct>,
## #   humidity_NA <fct>
```

2.5.2 Realizando resúmenes agrupados de valores faltantes

Ahora que puedes crear datos nabulares, vamos a utilizarlos para explorar los datos. Vamos a calcular estadísticas de resumen basadas en los valores faltantes de otra variable.

Para hacer esto, vamos a seguir los siguientes pasos:

Primero, `bind_shadow()` convierte los datos en datos nabulares.

A continuación, realiza algunos resúmenes en los datos utilizando `group_by()` y `summarize()` para calcular la media y la desviación estándar, utilizando las funciones `mean()` y `sd()`.

```
# `bind_shadow()` y `group_by()` para la ausencia de humedad (`humidity_NA`)
oceanbuoys %>%
  bind_shadow() %>%
  group_by(humidity_NA) %>%
  summarize(wind_ew_mean = mean(wind_ew), #calcula la media de wind_ew
            wind_ew_sd = sd(wind_ew)) #calcula la desviación estándar de wind_ew
```

```
## # A tibble: 2 x 3
##   humidity_NA wind_ew_mean wind_ew_sd
##   <fct>              <dbl>      <dbl>
## 1 !NA                -3.78       1.90
## 2 NA                 -3.30       2.31
```

```
# Repetir esto, pero calculando resúmenes para el viento norte-sur (`wind_ns`)
oceanbuoys %>%
  bind_shadow() %>%
  group_by(humidity_NA) %>%
  summarize(wind_ns_mean = mean(wind_ns),
            wind_ns_sd = sd(wind_ns))
```

```
## # A tibble: 2 x 3
##   humidity_NA wind_ns_mean wind_ns_sd
##   <fct>              <dbl>      <dbl>
## 1 !NA                 2.78       2.06
## 2 NA                  1.66       2.23
```

2.6 Explorando combinaciones adicionales de valores faltantes

Puede ser útil obtener un poco de información adicional sobre el número de casos en cada condición de valores faltantes.

En este ejercicio, vamos a añadir información sobre el número de casos observados utilizando `n()` dentro de la función `summarize()`.

Luego añadiremos un nivel adicional de agrupamiento al examinar la combinación de datos faltantes de humedad (`humidity_NA`) y datos faltantes de temperatura del aire (`air_temp_c_NA`).

```
# Resumir wind_ew según los datos faltantes de`air_temp_c_NA`
oceanbuoys %>%
  bind_shadow() %>%
  group_by(air_temp_c_NA) %>%
```

```
  summarize(wind_ew_mean = mean(wind_ew),
            wind_ew_sd = sd(wind_ew),
            n_obs = n())
```

```
## # A tibble: 2 x 4
##   air_temp_c_NA wind_ew_mean wind_ew_sd n_obs
##   <fct>                <dbl>      <dbl> <int>
## 1 !NA                  -3.91       1.85   655
## 2 NA                   -2.17       2.14    81
```

```
# Resumir wind_ew según los datos faltantes de `air_temp_c_NA` y `humidity_NA`
oceanbuoys %>%
  bind_shadow() %>%
  group_by(air_temp_c_NA, humidity_NA) %>%
  summarize(wind_ew_mean = mean(wind_ew),
            wind_ew_sd = sd(wind_ew),
            n_obs = n())
```

```
## # A tibble: 4 x 5
## # Groups:   air_temp_c_NA [2]
##   air_temp_c_NA humidity_NA wind_ew_mean wind_ew_sd n_obs
##   <fct>         <fct>              <dbl>      <dbl> <int>
## 1 !NA           !NA                -4.01       1.74   565
## 2 !NA           NA                 -3.24       2.31    90
## 3 NA            !NA                -2.06       2.08    78
## 4 NA            NA                 -4.97       1.74     3
```

2.7 Visualizando los valores faltantes de una variable

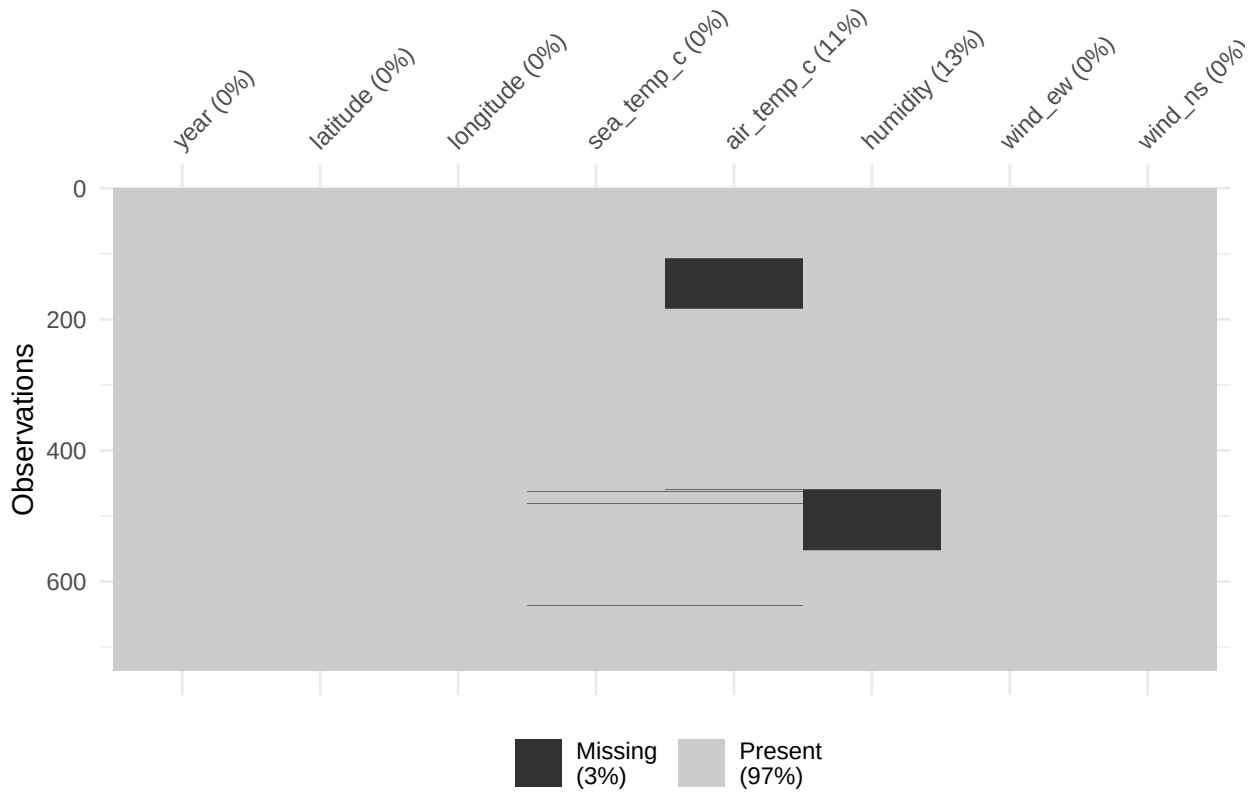
2.7.1 Datos nabulares y llenado por valores faltantes

Los estadísticos de resumen son útiles para calcular, pero como dicen, una imagen vale más que mil palabras.

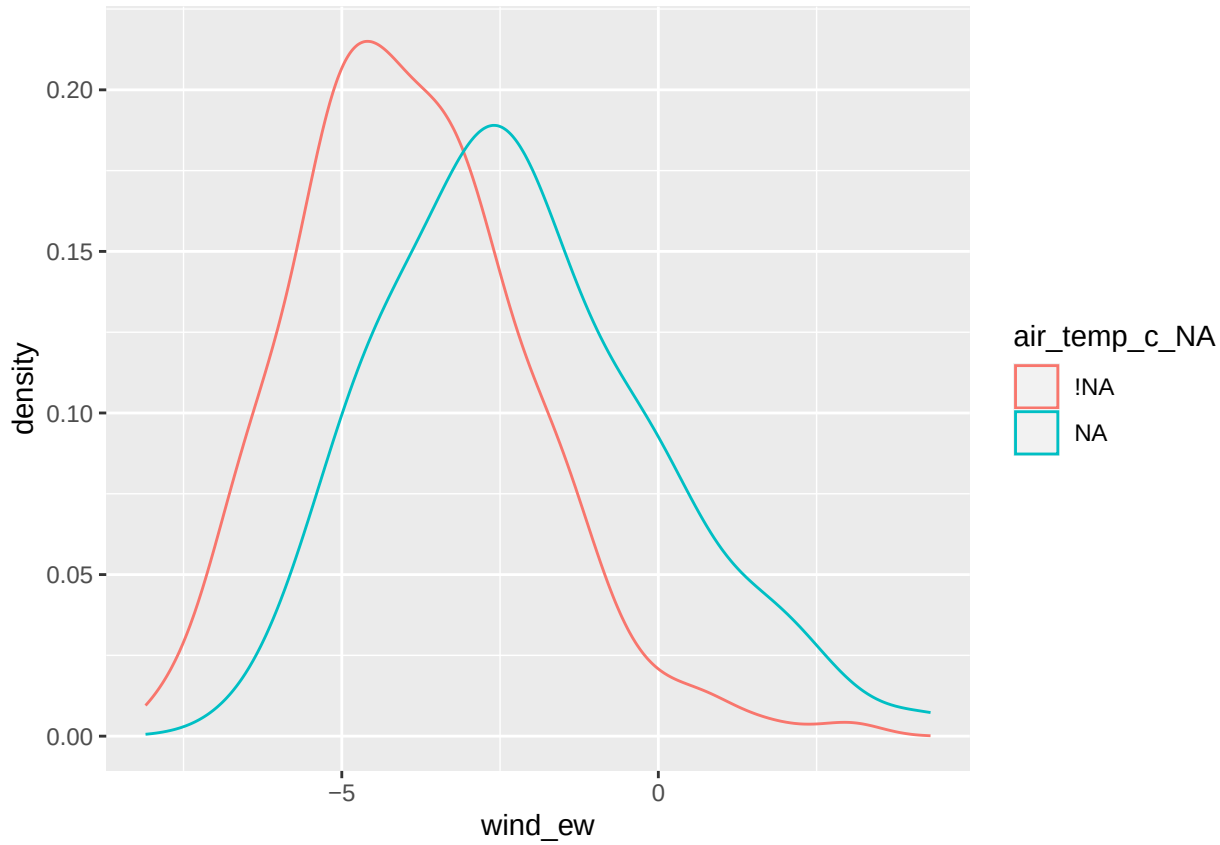
En este ejercicio, vamos a explorar cómo puedes utilizar datos `nabular` para explorar la variación en una variable según los valores faltantes de otra.

Vamos a utilizar el conjunto de datos `oceanbuoys` de `naniar`.

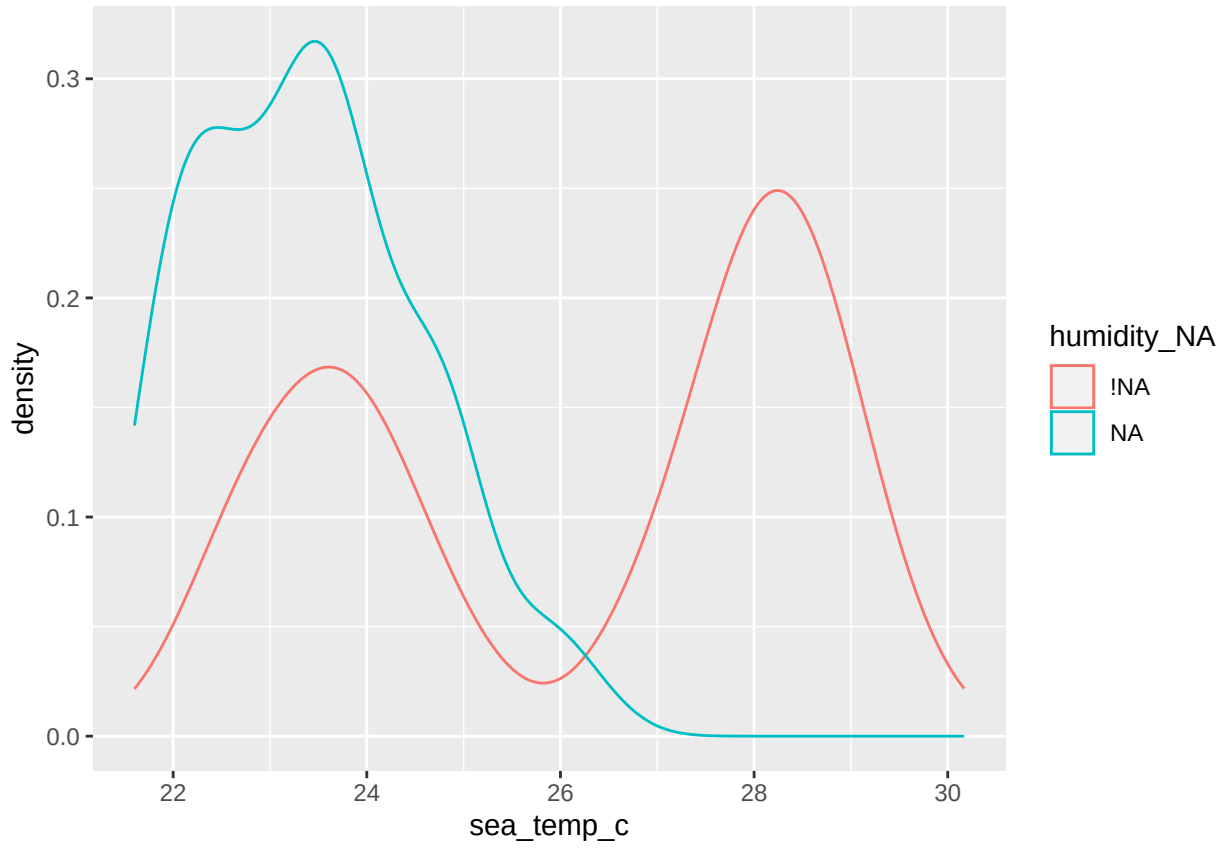
```
# Explorar primero la estructura de valores faltantes en `oceanbuoys`
# utilizando `vis_miss()`
vis_miss(oceanbuoys)
```




```
# Explorar la distribución de `wind_ew` según la ausencia de `air_temp_c_NA`
# utilizando `geom_density()`
bind_shadow(oceanbuoys) %>%
  ggplot(aes(x = wind_ew,
             color = air_temp_c_NA)) +
  geom_density()
```




```
# Explorar la distribución de la temperatura del mar según la ausencia de
# humedad (humidity_NA) utilizando `geom_density()`
bind_shadow(oceanbuoys) %>%
  ggplot(aes(x = sea_temp_c,
             color = humidity_NA)) +
  geom_density()
```



Ahora puedes utilizar datos nabulares para visualizar y explorar datos faltantes utilizando gráficos de densidad.

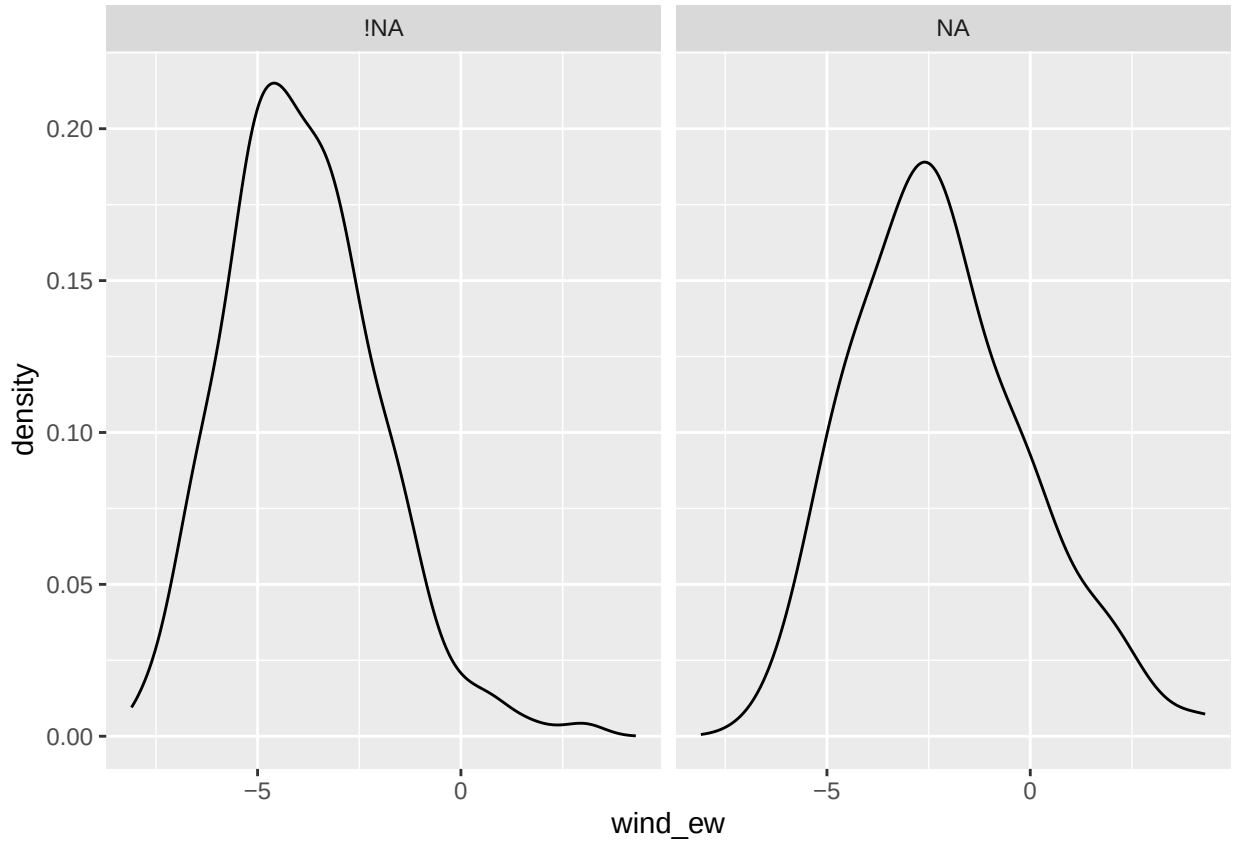
2.7.2 Datos nabulares y resumen por valores faltantes

En este ejercicio, vamos a explorar cómo utilizar datos `nabular` para explorar la variación en una variable según los valores faltantes de otra.

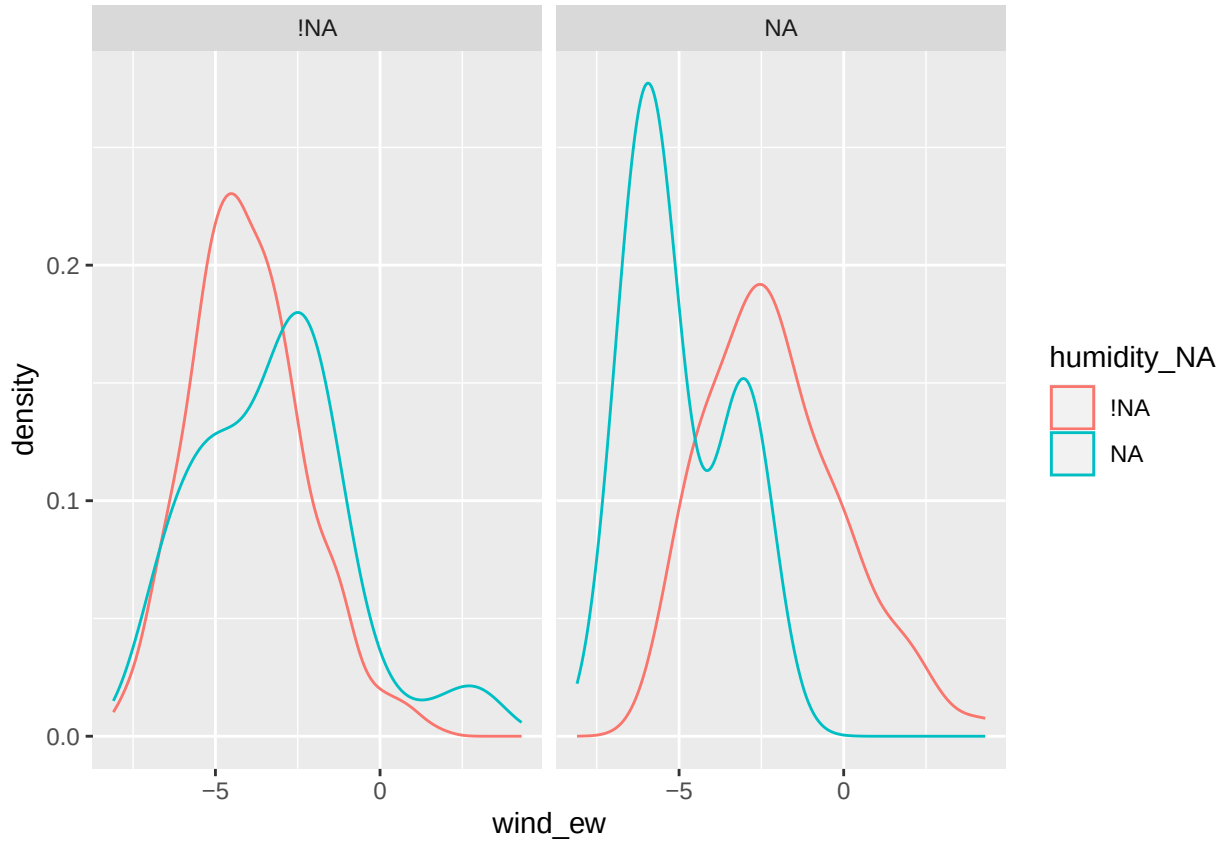
Vamos a utilizar el conjunto de datos `oceanbuoys` de `naniar`, y luego crear múltiples gráficos de los datos utilizando facets.

Esto te permite explorar diferentes capas de valores faltantes.

```
# Exploramos la distribución del viento este-oeste (wind_ew) en función de
# los datos faltantes de temperatura del aire utilizando geom_density() y
# facetando por los datos perdidos de temperatura del aire (air_temp_c_NA).
oceanbuoys %>%
  bind_shadow() %>%
  ggplot(aes(x = wind_ew)) +
  geom_density() +
  facet_wrap(~air_temp_c_NA)
```




```
# Ampliamos esta visualización al colorear por los datos perdidos
# de humedad (humidity_NA).
oceanbuoys %>%
  bind_shadow() %>%
  ggplot(aes(x = wind_ew,
             color = humidity_NA)) +
  geom_density() +
  facet_wrap(~air_temp_c_NA)
```



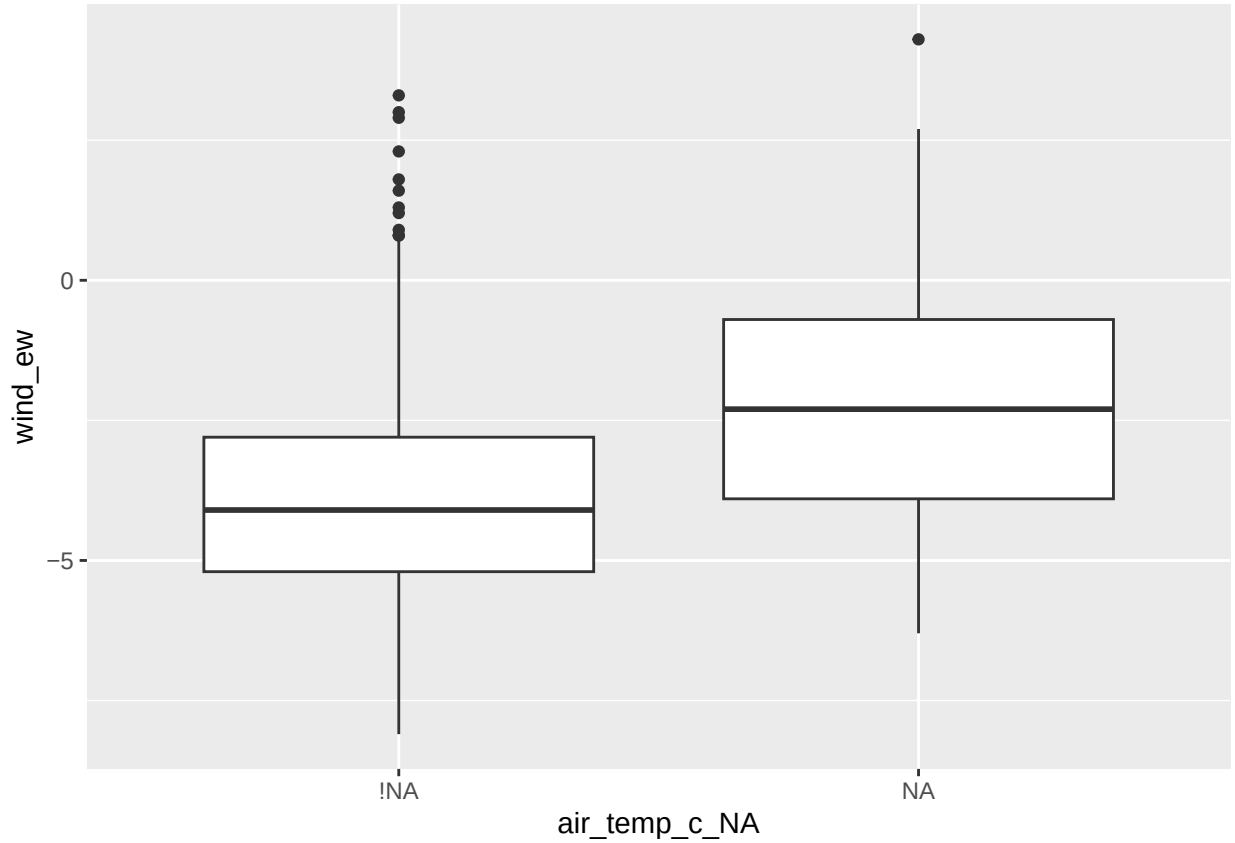
Ahora puedes utilizar datos nabulares para visualizar y explorar datos faltantes.

2.7.3 Explorar variación por valores faltantes: gráficos de caja

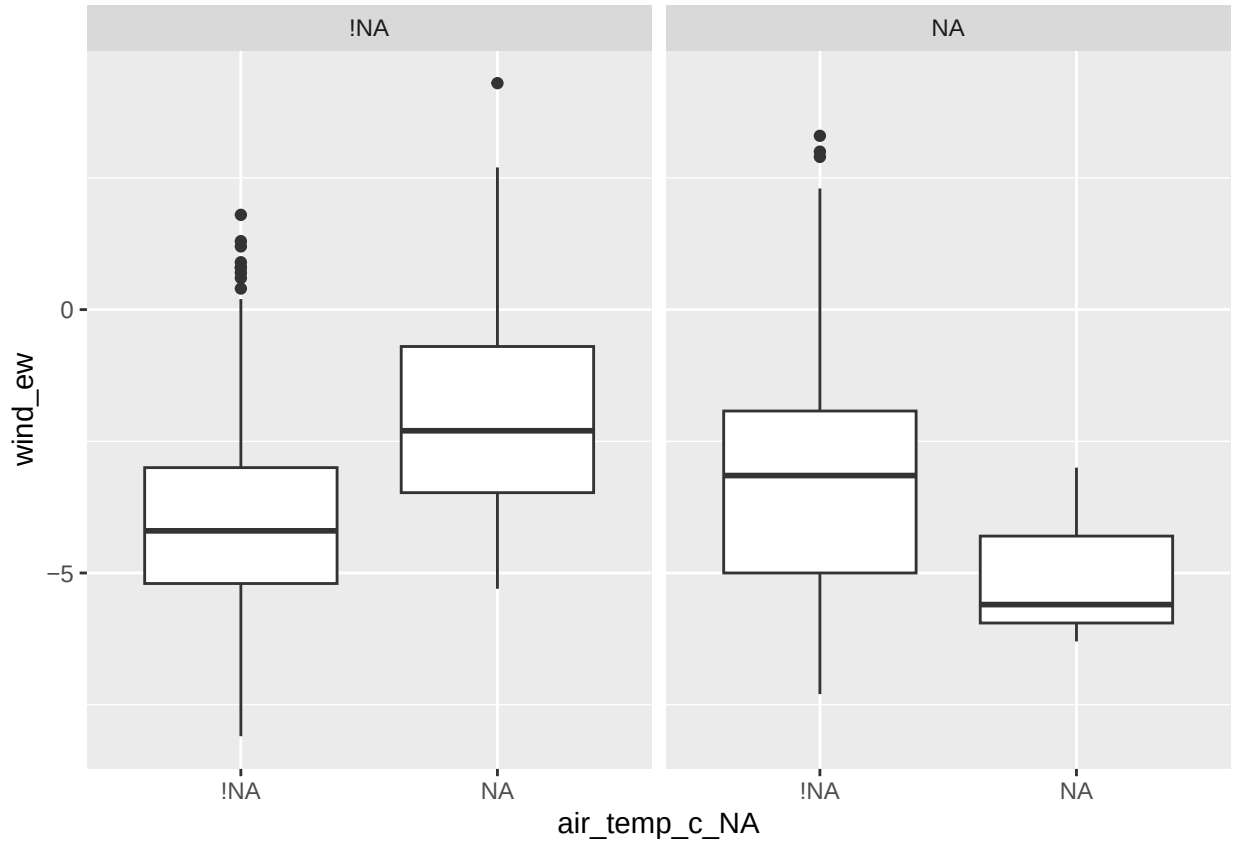
Los ejercicios anteriores utilizan datos `nabular` junto con gráficos de densidad para explorar la variación en una variable según los valores faltantes de otra.

Vamos a utilizar el conjunto de datos `oceanbuoys` de `naniar`, utilizando gráficos de caja en lugar de facets u otros para explorar diferentes capas de valores faltantes.

```
# Explora la distribución del viento este-oeste (`wind_ew`) en función de
# los datos perdidos de temperatura del aire utilizando `geom_boxplot()`.
oceanbuoys %>%
  bind_shadow() %>%
  ggplot(aes(x = air_temp_c_NA,
             y = wind_ew)) +
  geom_boxplot()
```




```
# Ampliamos esta visualización al facetar por la falta de
# humedad (`humidity_NA`).
oceanbuoys %>%
  bind_shadow() %>%
  ggplot(aes(x = air_temp_c_NA,
             y = wind_ew)) +
  geom_boxplot() +
  facet_wrap(~humidity_NA)
```



Ahora podemos utilizar datos nabulares para visualizar y explorar datos faltantes con gráficos de caja y envolturas de facts.

2.8 Visualizar valores faltantes en dos variables

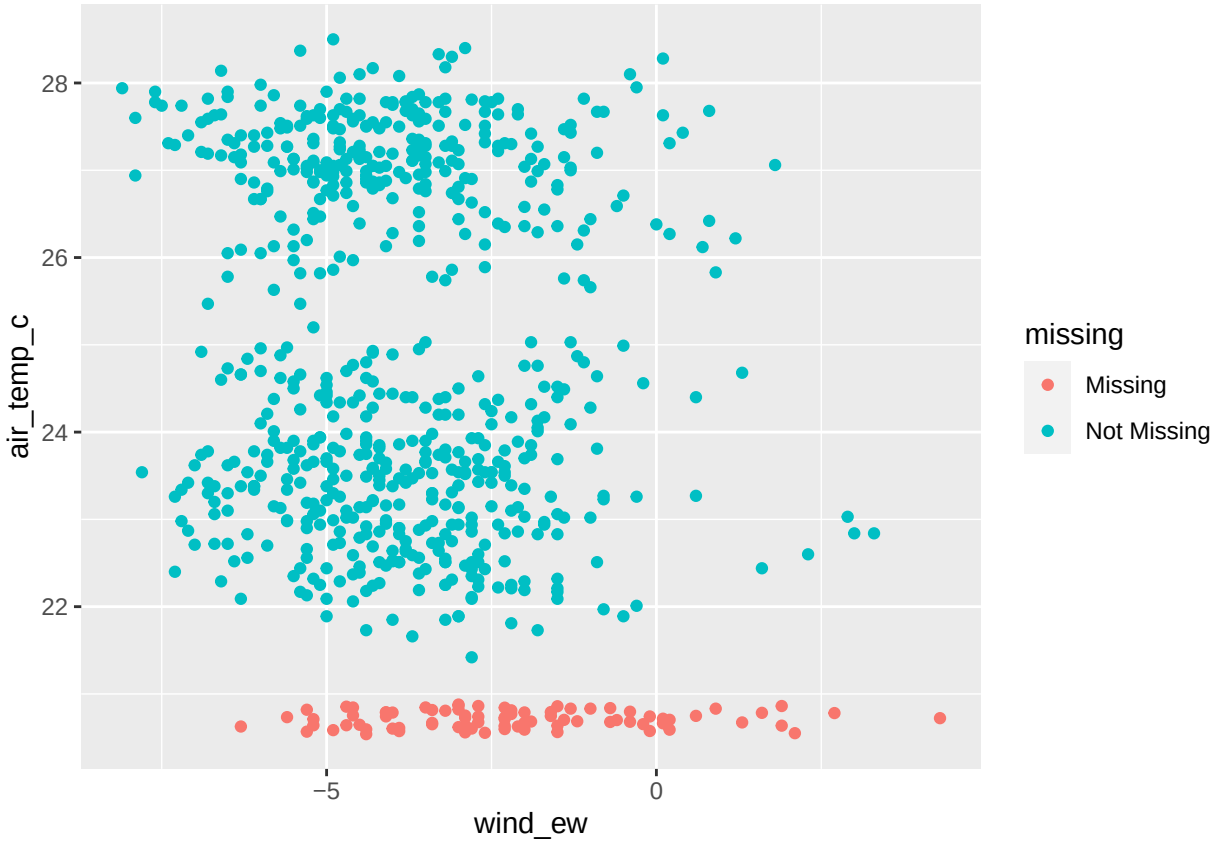
2.8.1 Explorando datos faltantes con gráficos de dispersión

Los valores faltantes en un gráfico de dispersión en `ggplot2` se eliminan por defecto, con una advertencia.

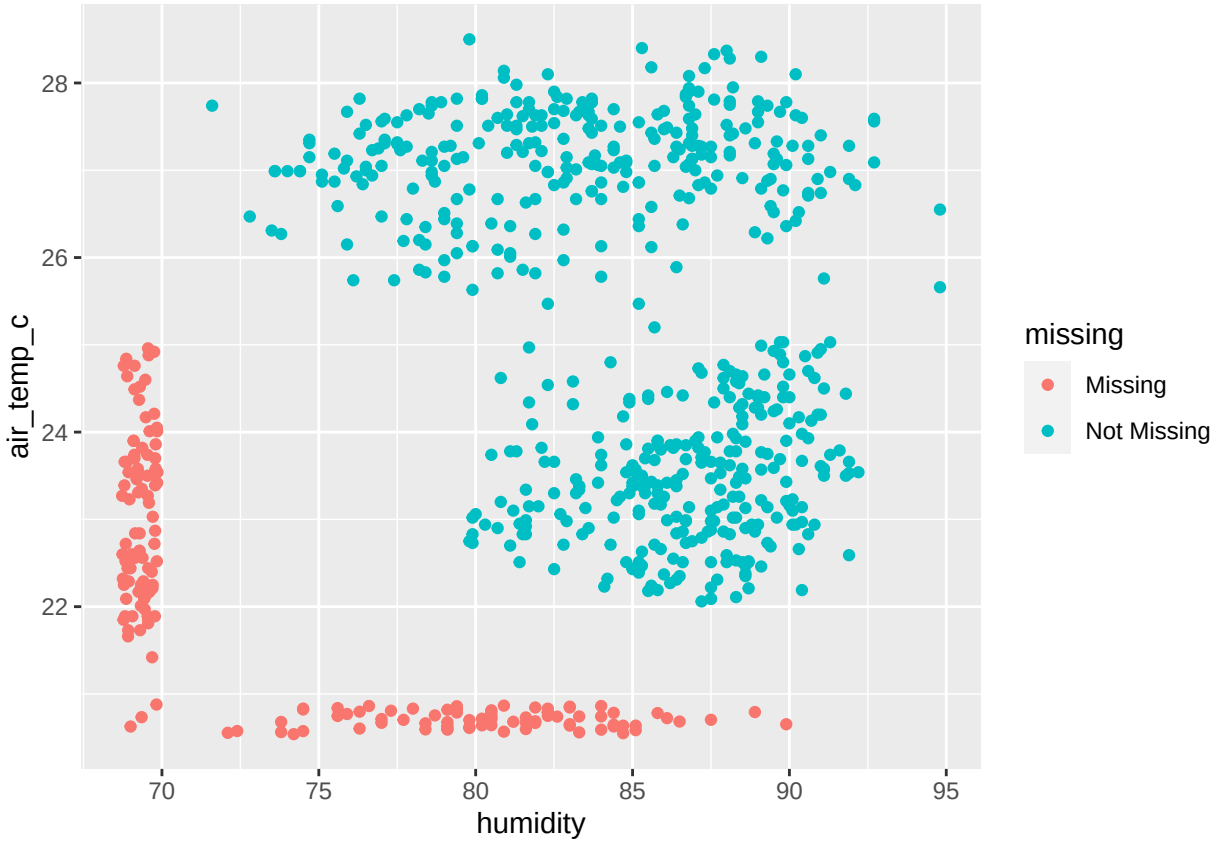
Podemos mostrar los valores faltantes en un gráfico de dispersión utilizando `geom_miss_point()` - una geometría especial de `ggplot2` que desplaza los valores faltantes dentro del gráfico, mostrándolos un 10% por debajo del mínimo de la variable.

Practiquemos esta visualización con el conjunto de datos `oceanbuoys`.

```
# Exploramos la ausencia de datos en el viento y la temperatura del aire, y
# mostramos la ausencia utilizando `geom_miss_point()`
ggplot(oceanbuoys,
       aes(x = wind_ew,
           y = air_temp_c)) +
  geom_miss_point()
```




```
# Exploramos la ausencia de datos en la humedad y la temperatura del aire,
# y mostrar la ausencia utilizando `geom_miss_point()`
ggplot(oceanbuoys,
       aes(x = humidity,
           y = air_temp_c)) +
  geom_miss_point()
```



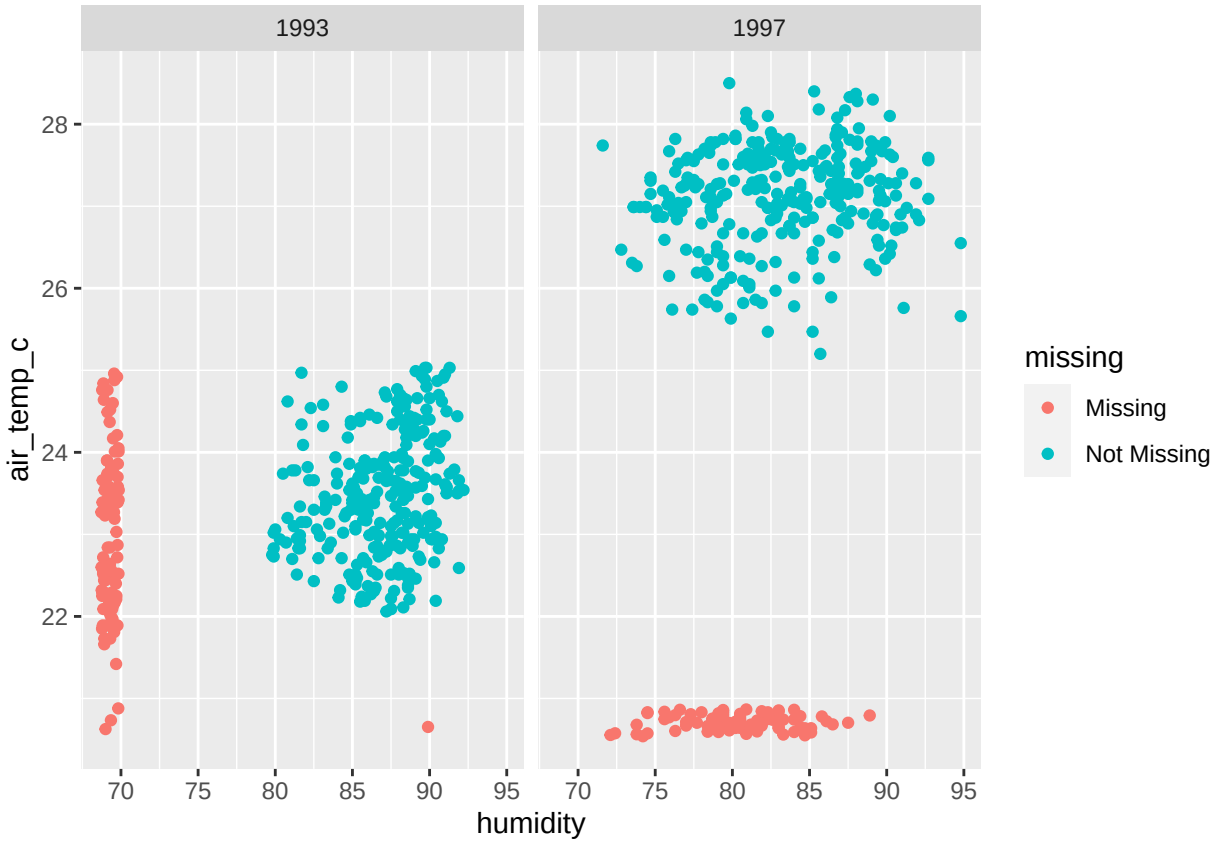
2.8.2 Usando facets para explorar valores faltantes

Debido a que `geom_miss_point()` es una geometría de ggplot, puedes usarla con características de `ggplot2` como el uso de facetas.

Esto significa que podemos explorar rápidamente los valores faltantes y permanecer dentro de los límites familiares de `ggplot2`.

```
# Exploramos la ausencia de datos en el viento y temperatura del aire, y
# mostrar la ausencia utilizando `geom_miss_point()`. Facetar por año para
# explorar más a fondo.
ggplot(oceanbuoys,
       aes(x = wind_ew,
           y = air_temp_c)) +
  geom_miss_point() +
  facet_wrap(~year)
```


```
# Explorar la ausencia de datos en la humedad y la temperatura del aire,
# y mostrar la ausencia utilizando `geom_miss_point()`. Facetar por año
# para explorar más a fondo.
ggplot(oceanbuoys,
       aes(x = humidity,
           y = air_temp_c)) +
  geom_miss_point() +
  facet_wrap(~year)
```



Ahora puedes usar `geom_miss_point()` para explorar valores faltantes en un gráfico de dispersión y puedes usar facetas para expandir y explorar aún más.

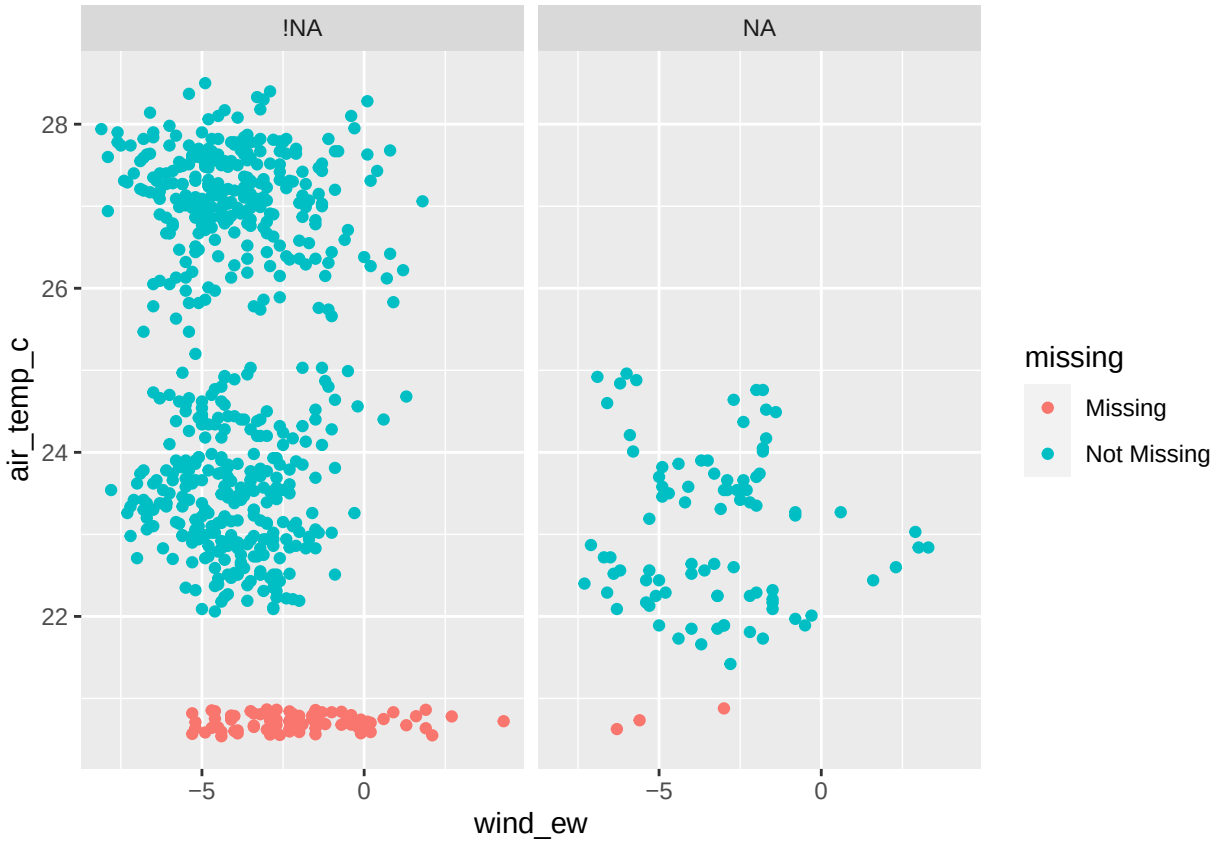
2.8.3 Facets para explorar valores faltantes (múltiples gráficos)

Otra técnica útil con `geom_miss_point()` es explorar los valores faltantes creando múltiples gráficos.

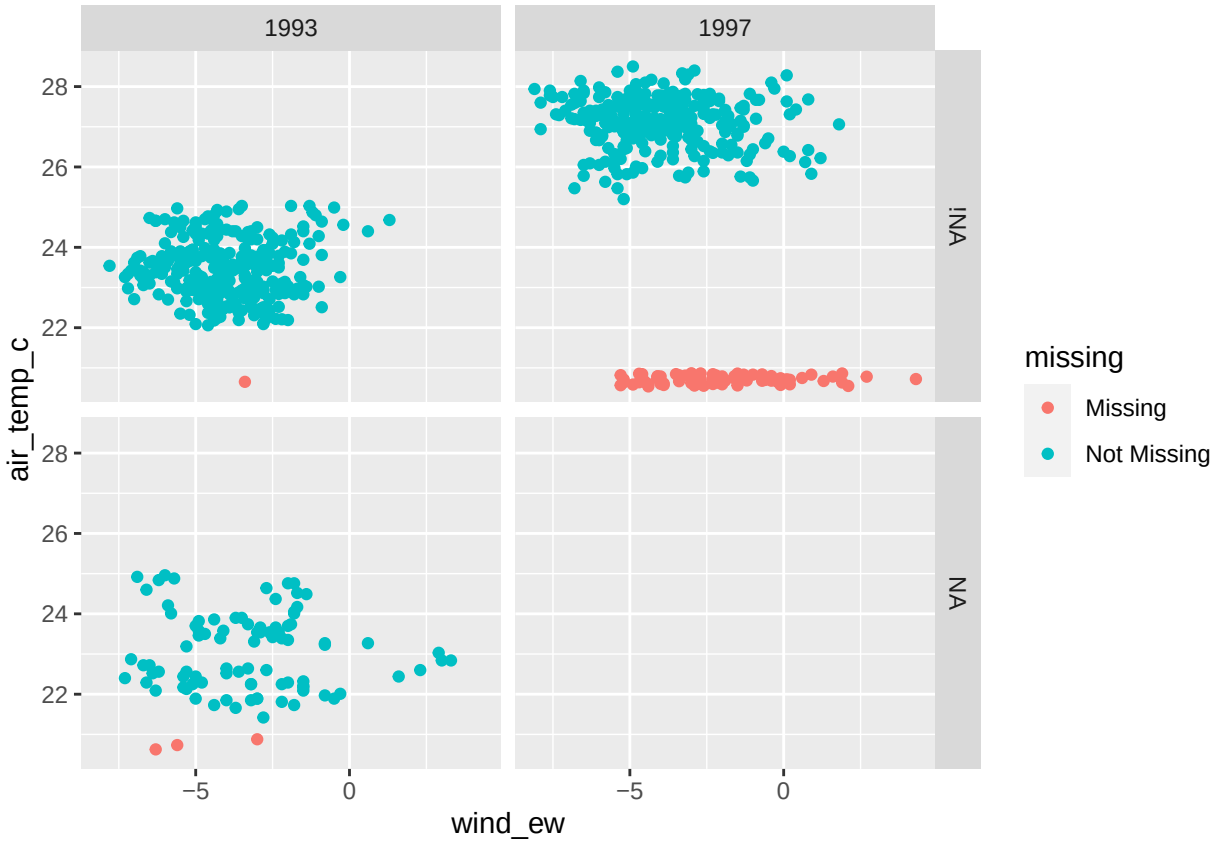
Así como lo hemos hecho en ejercicios anteriores, podemos usar los datos nabulares para ayudarnos a crear gráficos facetados adicionales.

Incluso podemos crear múltiples gráficos facetados según los valores en los datos, como el año, y las características de los datos, como la falta de valores.

```
# Utilizar geom_miss_point() y facet_wrap para explorar cómo la ausencia
# de datos en wind_ew y air_temp_c es diferente para los datos
# perdidos de humedad
bind_shadow(oceanbuoys) %>%
  ggplot(aes(x = wind_ew,
             y = air_temp_c)) +
  geom_miss_point() +
  facet_wrap(~humidity_NA)
```




```
# Utilizar geom_miss_point() y facet_grid para explorar cómo la ausencia
# de datos en wind_ew y air_temp_c es diferente para la ausencia de
# humedad Y por año - utilizando `facet_grid(humidity_NA ~ year)`
bind_shadow(oceanbuoys) %>%
  ggplot(aes(x = wind_ew,
             y = air_temp_c)) +
  geom_miss_point() +
  facet_grid(humidity_NA~year)
```



##Realizando y rastreando imputaciones.

2.8.4 Imputar datos por debajo del rango con datos nulos.

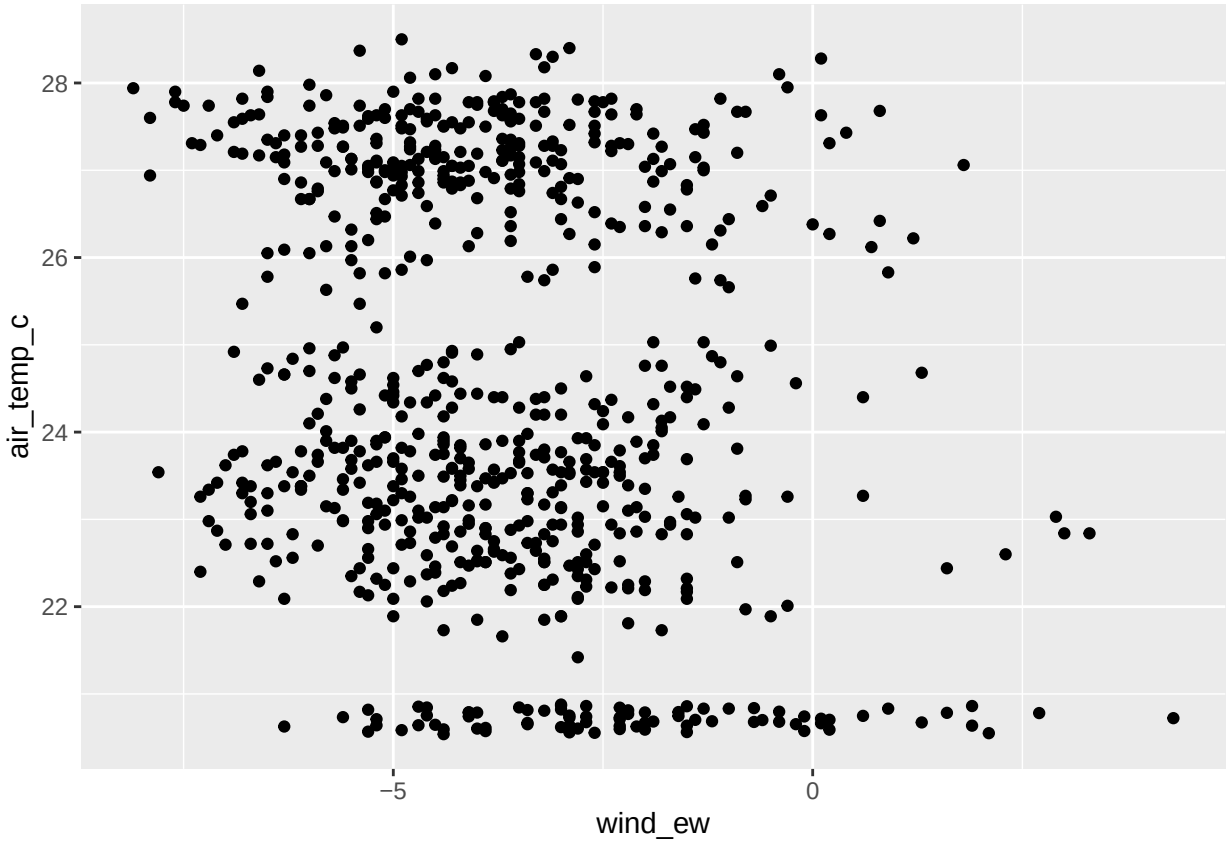
Queremos hacer un seguimiento de los valores que imputamos. Si no lo hacemos, es muy difícil evaluar qué tan buenos son los valores imputados.

Vamos a practicar imputando datos y recreando visualizaciones en el conjunto de ejercicios anterior mediante la imputación de valores por debajo del rango de los datos.

Esta es una forma muy útil de ayudar a explorar aún más la falta de datos y también proporciona el marco para imputar valores faltantes.

Primero, vamos a imputar los datos por debajo del rango usando `impute_below_all()`, y luego visualizar los datos. Notamos que aunque podemos ver dónde están los valores faltantes en este caso, necesitamos alguna forma de hacer un seguimiento de ellos. El patrón de programación de seguimiento de datos faltantes puede ayudar con esto.

```
# Imputar los datos por debajo del rango utilizando `impute_below_all`.
ocean_imp <- impute_below_all(oceanbuoys)

# Visualizar los nuevos valores faltantes
ggplot(ocean_imp,
       aes(x = wind_ew, y = air_temp_c)) +
  geom_point()
```




```
# Imputar y rastrear datos con `bind_shadow`, `impute_below`,
# y `add_label_shadow`
ocean_imp_track <- bind_shadow(oceanbuoys) %>%
  impute_below_all() %>%
  add_label_shadow()

# Observar los valores imputados
ocean_imp_track
```

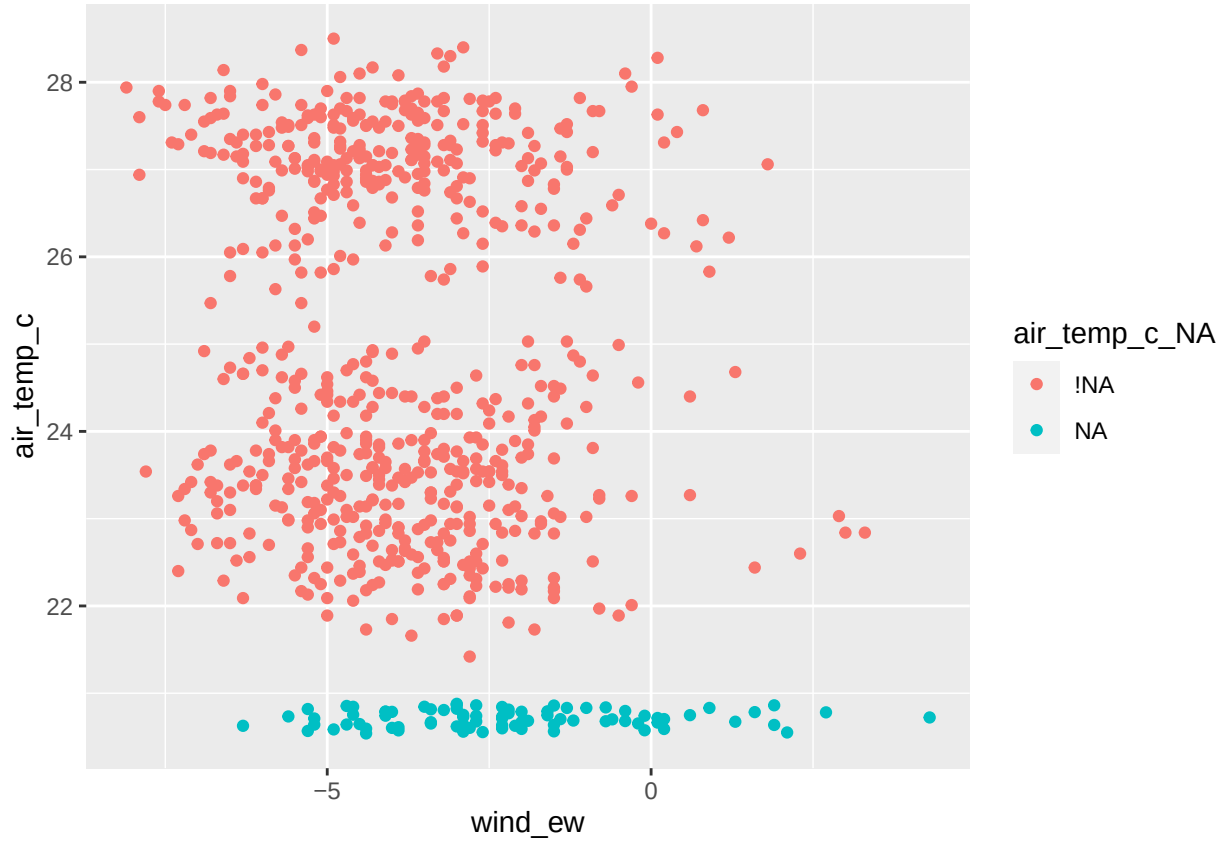
```
## # A tibble: 736 x 17
##     year latitude longitude sea_temp_c air_temp_c humidity wind_ew wind_ns
##    <dbl>    <dbl>     <dbl>      <dbl>      <dbl>    <dbl>   <dbl>   <dbl>
##  1  1997        0      -110       27.6       27.1     79.6   -6.40    5.40
##  2  1997        0      -110       27.5       27.0     75.8   -5.30    5.30
##  3  1997        0      -110       27.6       27       76.5   -5.10    4.5
##  4  1997        0      -110       27.6       26.9     76.2   -4.90    2.5
##  5  1997        0      -110       27.6       26.8     76.4   -3.5     4.10
##  6  1997        0      -110       27.8       26.9     76.7   -4.40    1.60
##  7  1997        0      -110       28.0       27.0     76.5   -2       3.5
##  8  1997        0      -110       28.0       27.1     78.3   -3.70    4.5
##  9  1997        0      -110       28.0       27.2     78.6   -4.20    5
## 10  1997        0      -110       28.0       27.2     76.9   -3.60    3.5
## # i 726 more rows
## # i 9 more variables: year_NA <fct>, latitude_NA <fct>, longitude_NA <fct>,
## #   sea_temp_c_NA <fct>, air_temp_c_NA <fct>, humidity_NA <fct>,
## #   wind_ew_NA <fct>, wind_ns_NA <fct>, any_missing <chr>
```

2.8.5 Visualizar valores imputados en un gráfico de dispersión.

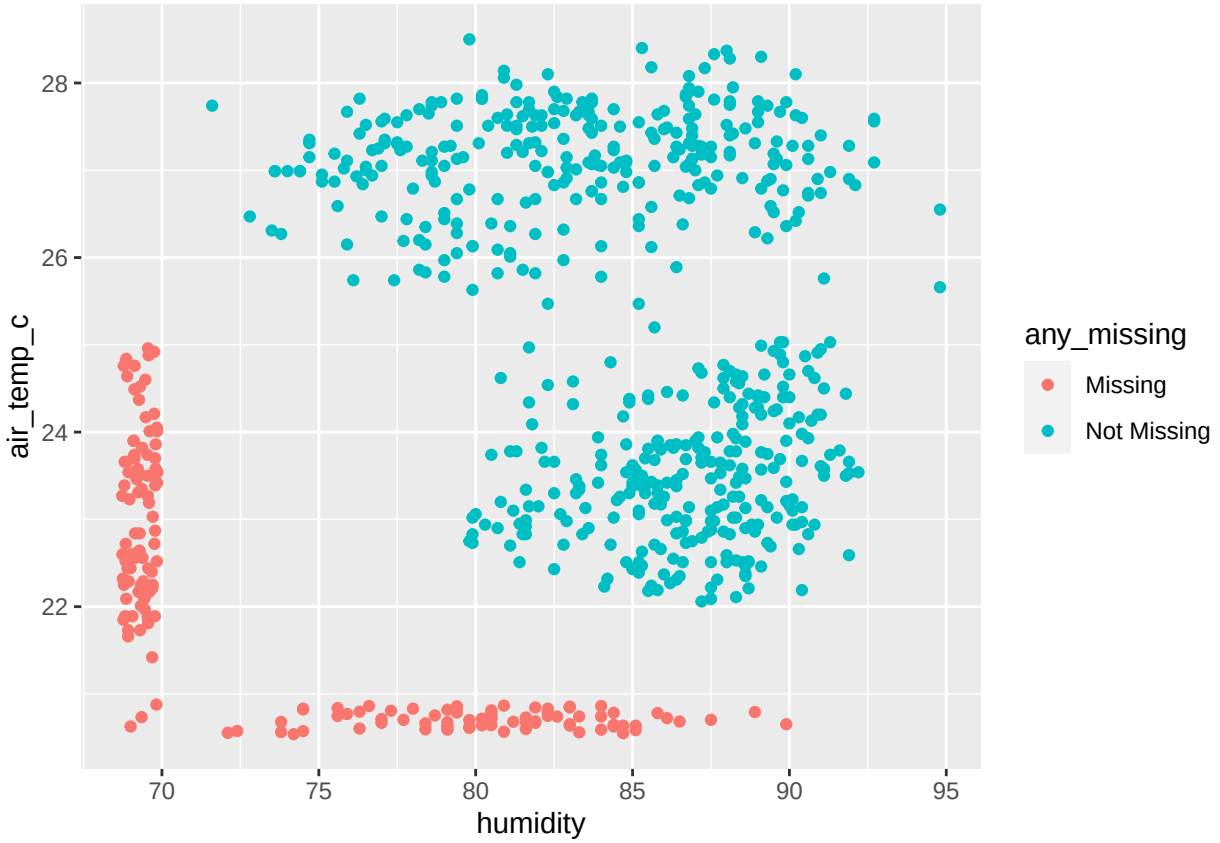
Ahora, vamos a recrear uno de los gráficos anteriores que vimos que utilizaba `geom_miss_point()`.

Para hacer esto, necesitamos imputar los datos por debajo del rango de los datos. Esta es una imputación especial para explorar los datos. Esta imputación ilustrará lo que necesitamos practicar: cómo hacer un seguimiento de los valores faltantes. Para imputar los datos por debajo del rango de los datos, usamos la función `impute_below_all()`.

```
# Imputar y hacer un seguimiento de los valores faltantes
ocean_imp_track <- bind_shadow(oceanbuoys) %>%
  impute_below_all() %>%
  add_label_shadow()

# Visualizar la falta de datos en el viento y la temperatura del aire,
# coloreando los valores faltantes de la temperatura del aire con air_temp_c_NA
ggplot(ocean_imp_track,
       aes(x = wind_ew, y = air_temp_c, color = air_temp_c_NA)) +
  geom_point()
```




```
# Visualizar la humedad y la temperatura del aire, coloreando los casos
# faltantes utilizando la variable any_missing
ggplot(ocean_imp_track,
       aes(x = humidity, y = air_temp_c, color = any_missing)) +
  geom_point()
```

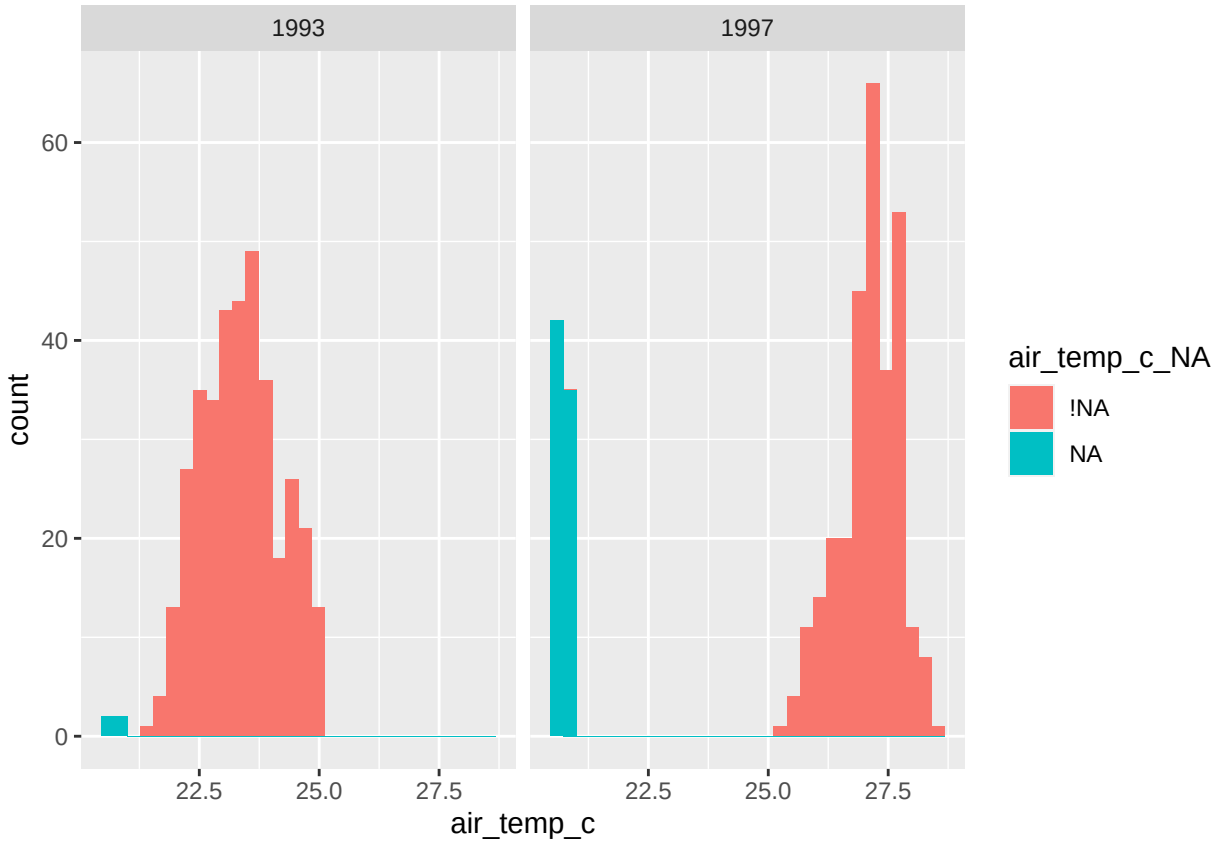


2.8.6 Crear un histograma de los datos imputados.

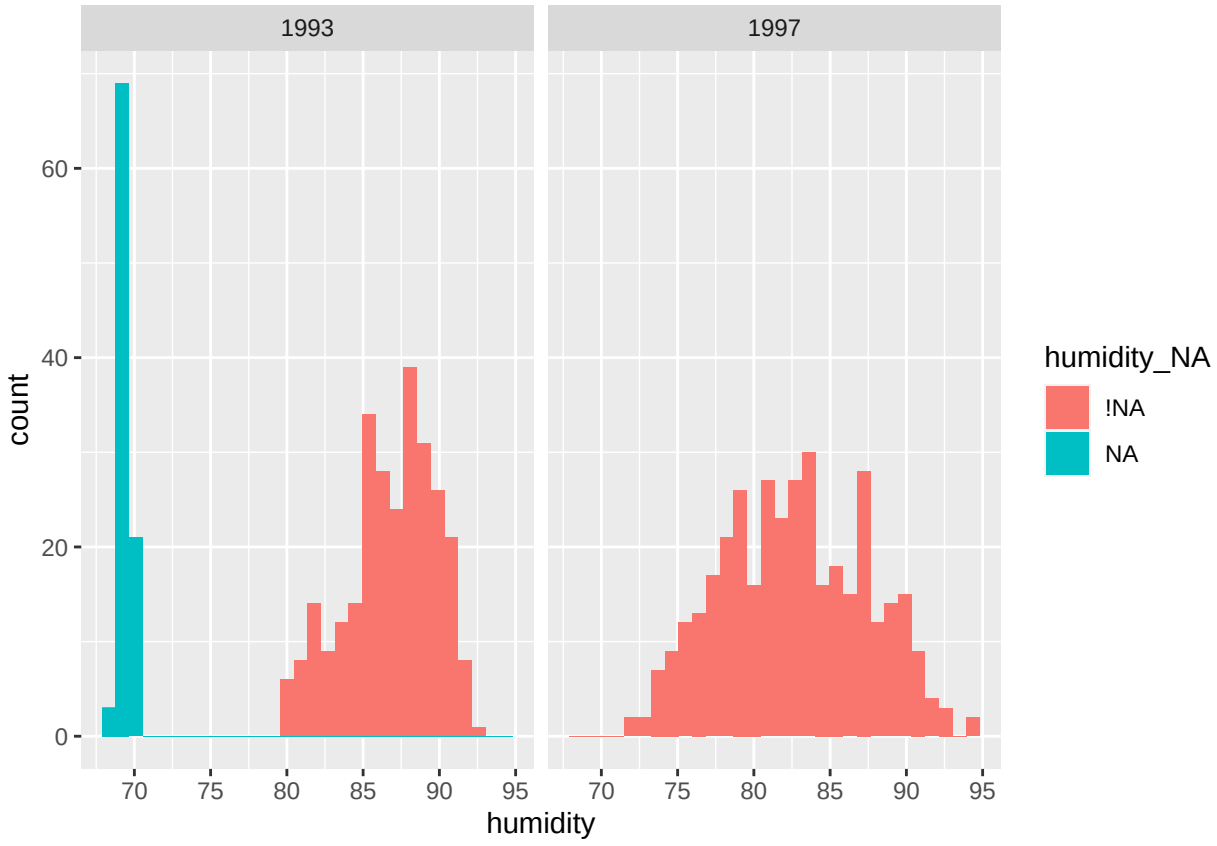
Ahora que podemos recrear la primera visualización de `geom_miss_point()`, vamos a explorar cómo podemos aplicar esto a otras tareas exploratorias.

Una tarea útil es evaluar el número de valores faltantes en una variable dada usando un histograma. Podemos hacer esto usando el conjunto de datos `ocean_imp_track` que creamos en el último ejercicio, el cual está cargado en esta sesión.

```
# Explorar los valores de air_temp_c, visualizando la cantidad de datos
# faltantes con `air_temp_c_NA`.
p <- ggplot(ocean_imp_track, aes(x = air_temp_c, fill = air_temp_c_NA)) + geom_histogram()

# Explorar los datos faltantes en la humedad utilizando humidity_NA
p2 <- ggplot(ocean_imp_track, aes(x = humidity, fill = humidity_NA)) + geom_histogram()

# Explorar los datos faltantes en air_temp_c según el año, utilizando
# `facet_wrap(~year)`.
p + facet_wrap(~year)
```




```
# Explorar los datos faltantes en humedad según el año, utilizando
# `facet_wrap(~year)`.
p2 + facet_wrap(~year)
```



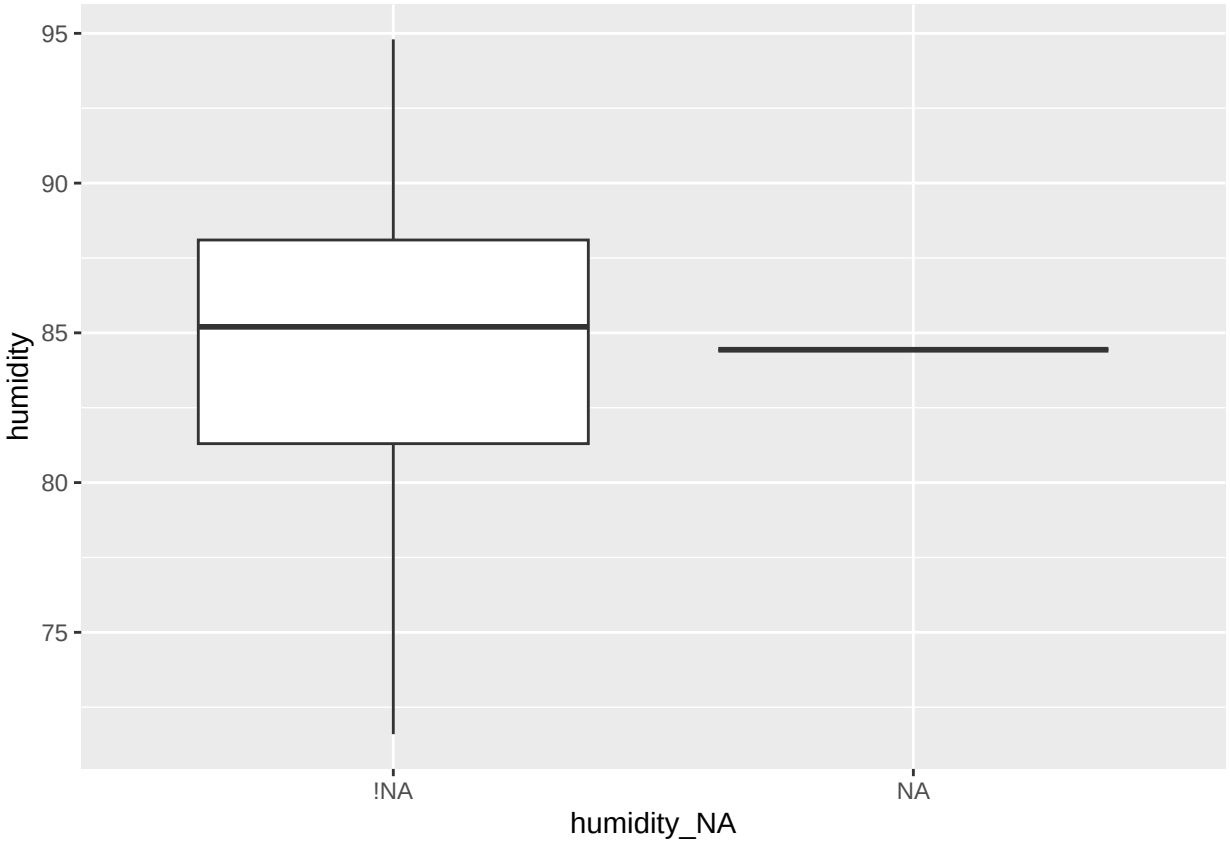
2.9 ¿Qué hace una buena imputación?

2.9.1 Evaluando imputaciones malas.

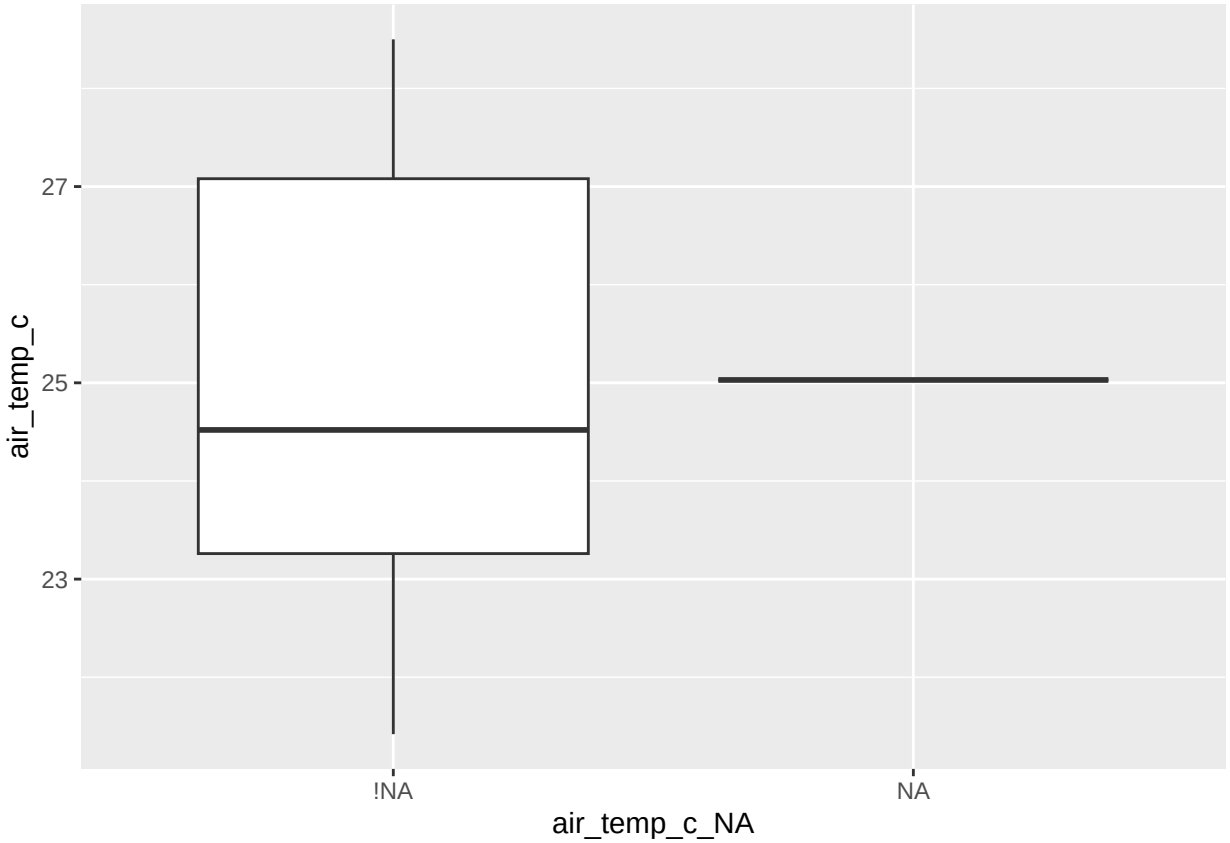
Para evaluar imputaciones, es útil saber cómo se ve algo malo. Para explorar esto, veamos un método de imputación típicamente malo: imputar usando el valor promedio.

En este ejercicio vamos a explorar cómo funciona el método de imputación de promedio usando un diagrama de caja, utilizando el conjunto de datos `oceanbuoys`.

```
# Imputar el valor medio y rastrear las imputaciones
ocean_imp_mean <- bind_shadow(oceanbuoys) %>%
  impute_mean_all() %>%
  add_label_shadow()

# Explorar los valores medios de la humedad en el conjunto de datos imputado
ggplot(ocean_imp_mean,
       aes(x = humidity_NA, y = humidity)) +
  geom_boxplot()
```




```
# Explorar los valores de la temperatura del aire en el conjunto de datos
# imputados
ggplot(ocean_imp_mean,
       aes(x = air_temp_c_NA, y = air_temp_c)) +
  geom_boxplot()
```

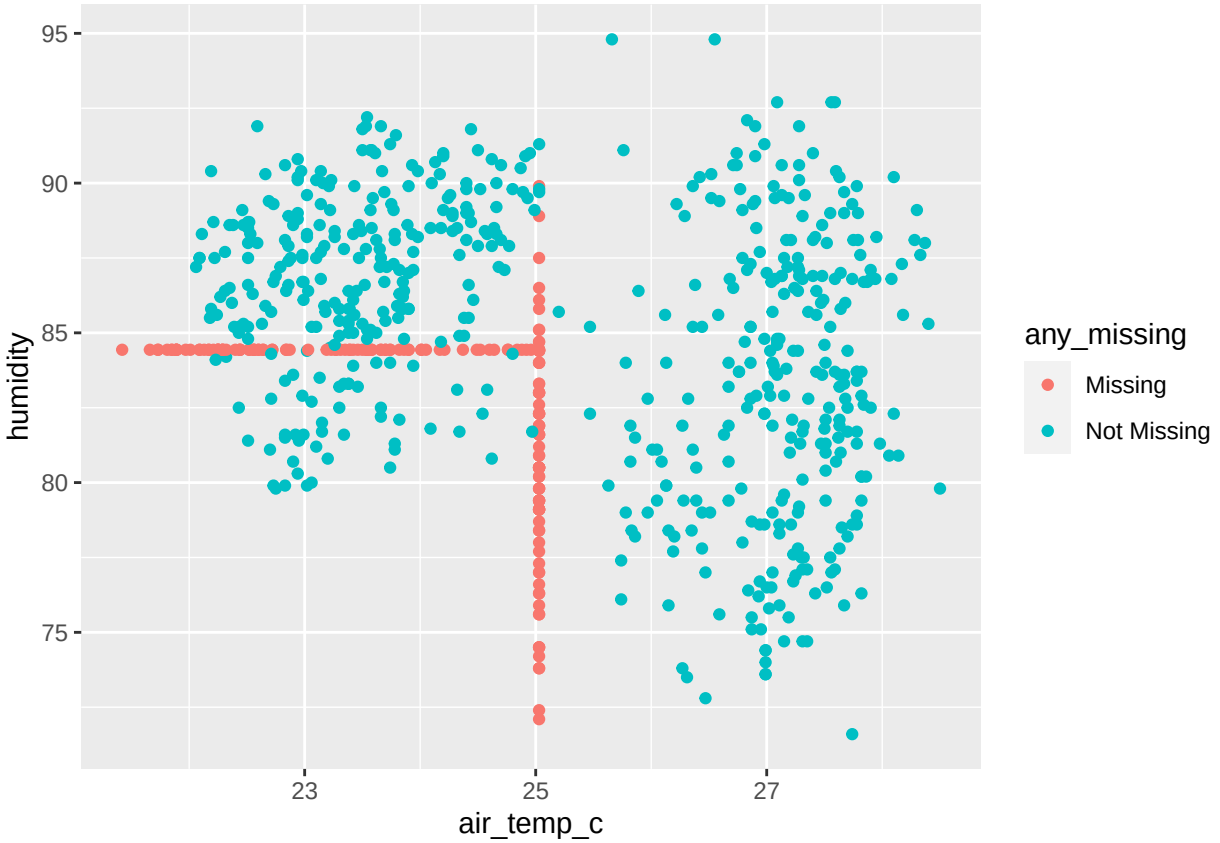


Evaluando imputaciones: La escala

Si bien la imputación de la media puede no parecer tan mala cuando la comparamos mediante un diagrama de caja, es importante tener una idea de la variación en los datos. Es por eso que es importante explorar cómo cambia la escala y la dispersión de los valores imputados en comparación con los datos.

Una forma de evaluar la adecuación de la escala de las imputaciones es usar un gráfico de dispersión para explorar si los valores son apropiados o no.

```
# Explorar las imputaciones en la temperatura del aire y la humedad,
# coloreando por la variable any_missing
ggplot(ocean_imp_mean,
       aes(x = air_temp_c, y = humidity, color = any_missing)) +
  geom_point()
```




```
# Explorar las imputaciones en la temperatura del aire y la humedad,
# coloreando por la variable any_missing y facetando por año
ggplot(ocean_imp_mean,
       aes(x = air_temp_c, y = humidity, color = any_missing)) +
  geom_point() +
  facet_wrap(~year)
```

2.10 Realizando imputaciones.

2.10.1 Utilizando simputation para imputar datos.

Existen muchos paquetes de imputación en R. Nos enfocaremos en usar el paquete `simputation`, que proporciona una interfaz simple y potente para realizar imputaciones.

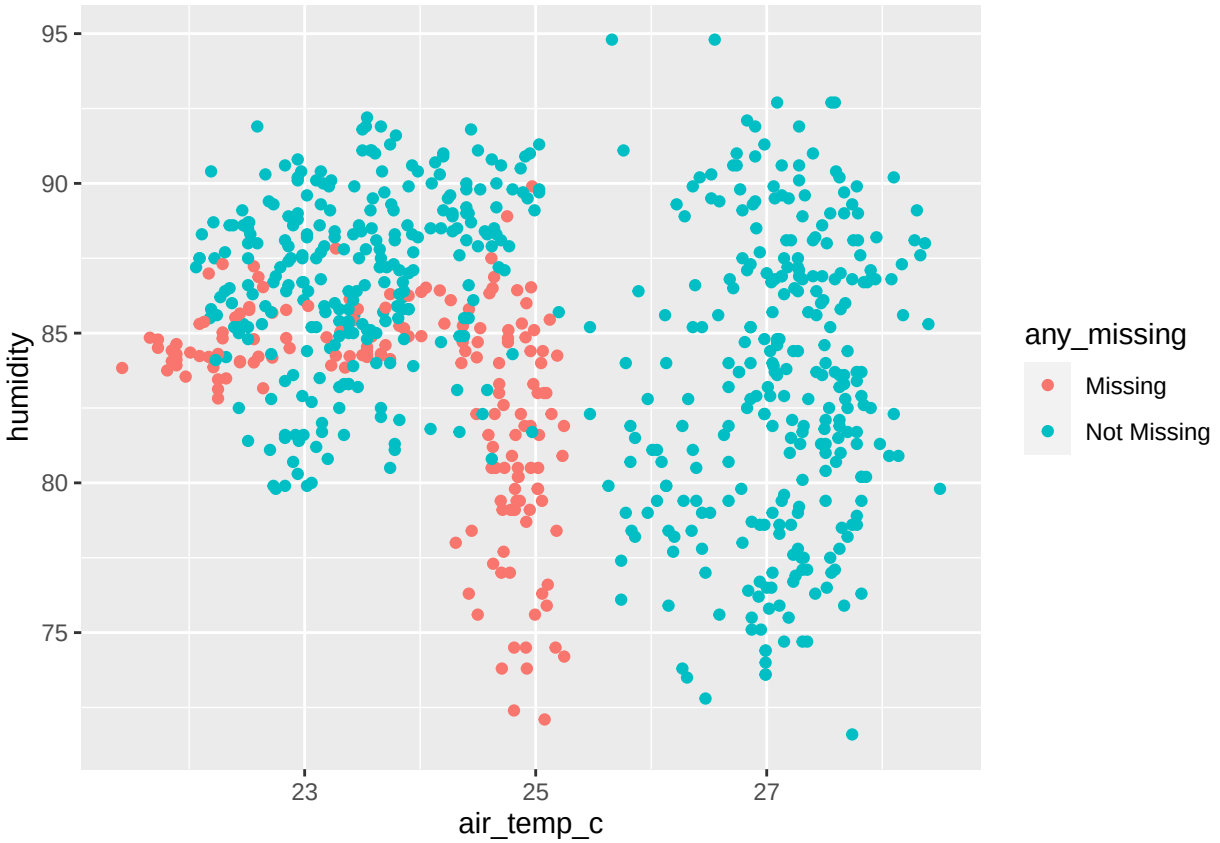
Construir un buen modelo de imputación es muy importante, pero es un tema complejo: hay tanto para construir un buen modelo de imputación como para construir un buen modelo estadístico. En este sección, nos enfocaremos en cómo evaluar las imputaciones.

Primero, vamos a ver cómo utilizar la función `impute_lm()`, que imputa valores de acuerdo a un modelo lineal especificado.

En este ejercicio, vamos a aplicar las técnicas de evaluación anteriores a los datos con `impute_lm()`, y luego construir sobre este método de imputación en lecciones posteriores.

```
library(simputation)
# Imputar la humedad y la temperatura del aire utilizando wind_ew y wind_ns,
# y rastrear los valores faltantes
ocean_imp_lm_wind <- oceanbuoys %>%
    bind_shadow() %>%
    impute_lm(air_temp_c ~ wind_ew + wind_ns) %>%
    impute_lm(humidity ~ wind_ew + wind_ns) %>%
    add_label_shadow()

# Graficar los valores imputados para air_temp_c y humedad,
# coloreados por la ausencia de datos
```

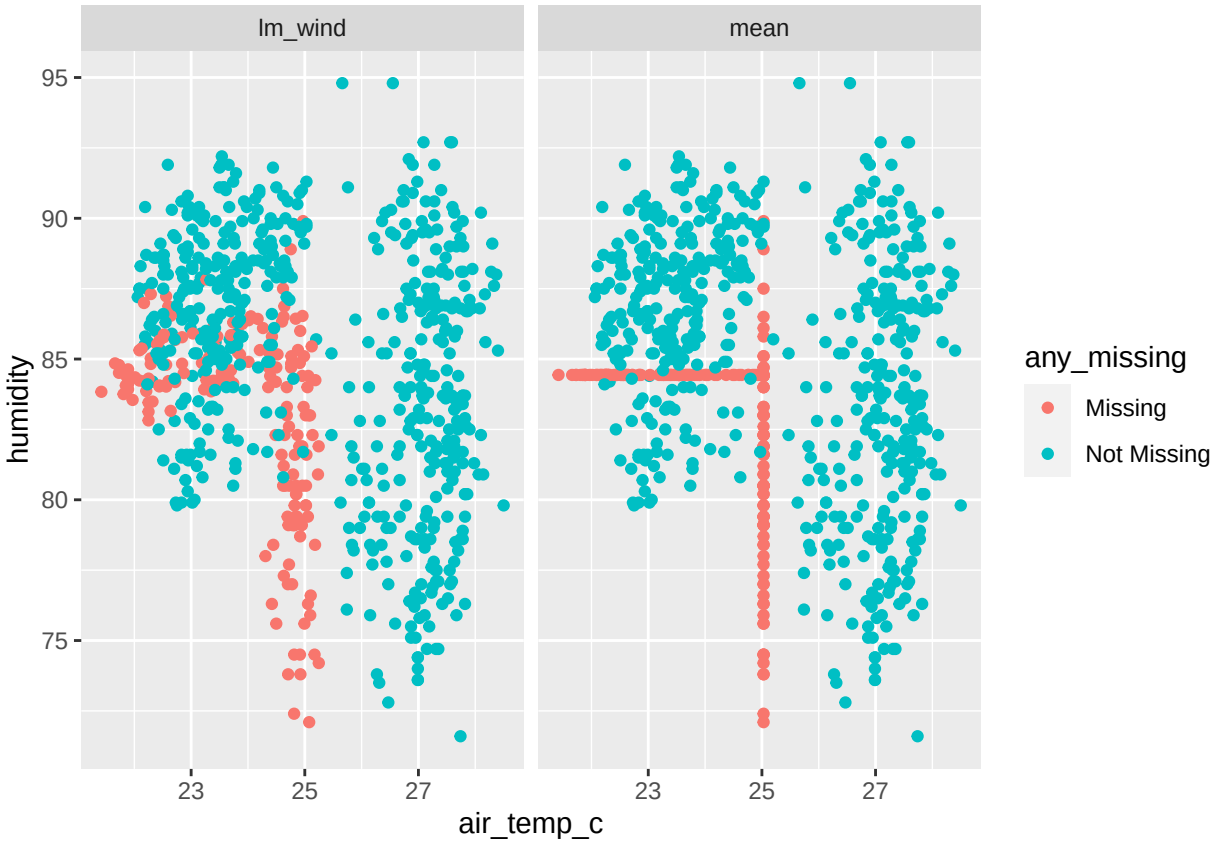
```
ggplot(ocean_imp_lm_wind,
       aes(x = air_temp_c, y = humidity, color = any_missing)) +
  geom_point()
```



Evaluando y comparando imputaciones

Cuando se construye un modelo de imputación, es una buena idea compararlo con otro método. En esta lección, vamos a comparar el conjunto de datos previamente imputados creado utilizando `impute_lm()` con el conjunto de datos imputados con la media. Ambos conjuntos de datos están incluidos en este ejercicio como `ocean_imp_lm_wind` y `ocean_imp_mean`, respectivamente.

```
# Unir los modelos juntos
bound_models <- bind_rows(mean = ocean_imp_mean,
                          lm_wind = ocean_imp_lm_wind,
                          .id = "imp_model")

# Inspeccionar los valores de la temperatura del aire y la humedad en
# un gráfico de dispersión
ggplot(bound_models,
       aes(x = air_temp_c,
           y = humidity,
           color = any_missing)) +
  geom_point() +
  facet_wrap(~imp_model)
```

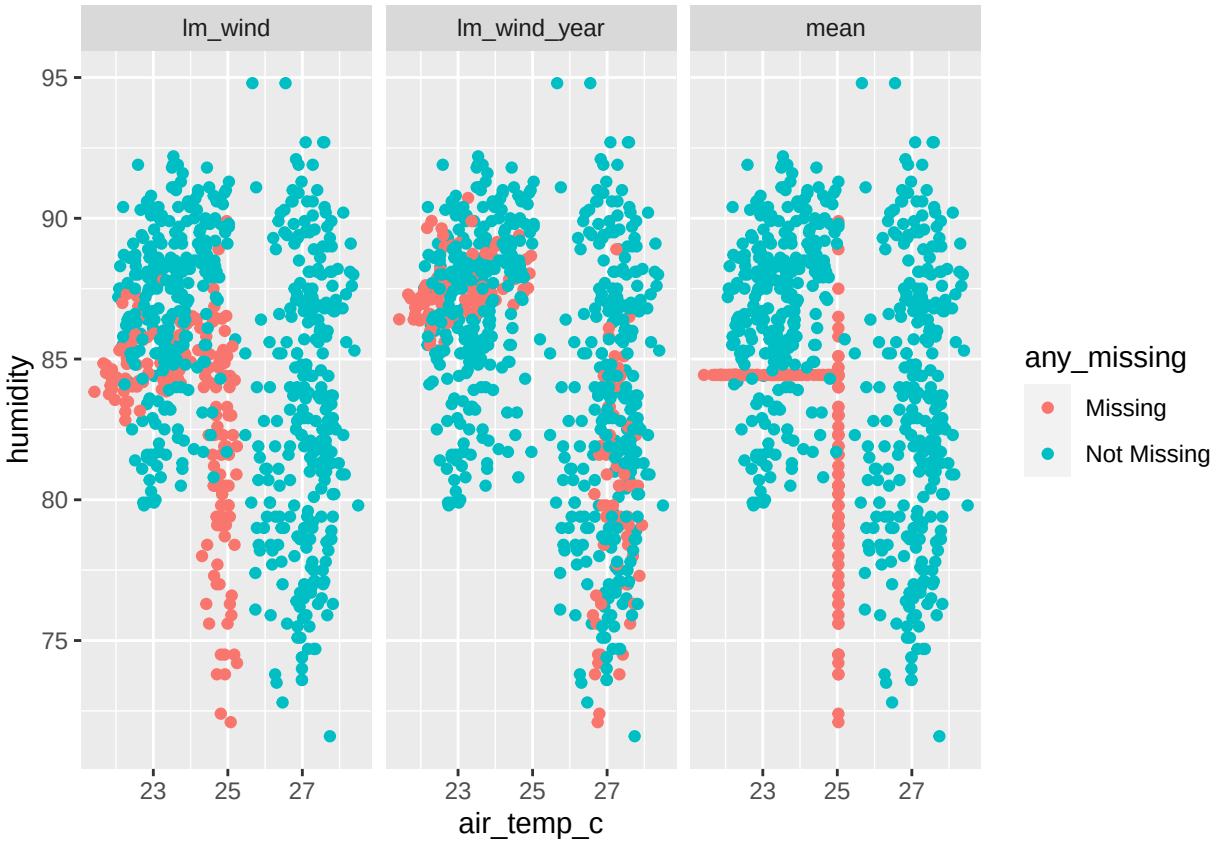


Evaluando imputaciones (muchos modelos y variables)

Cuando se construye un modelo de imputación, es una buena idea compararlo con otro método.

En esta lección, se te pedirá que agregues un modelo de imputación final que contenga una pieza adicional de información útil que ayude a explicar parte de la variación en los datos. Luego, compararás los valores, como se hizo anteriormente en la última lección.

```
# Construir un modelo añadiendo el año a la variable objetivo
ocean_imp_lm_wind_year <- bind_shadow(oceanbuoys) %>%
  impute_lm(air_temp_c ~ wind_ew + wind_ns + year) %>%
  impute_lm(humidity ~ wind_ew + wind_ns + year) %>%
  add_label_shadow()

# Unir los modelos mean, lm_wind y lm_wind_year juntos
bound_models <- bind_rows(mean = ocean_imp_mean,
                          lm_wind = ocean_imp_lm_wind,
                          lm_wind_year = ocean_imp_lm_wind_year,
                          .id = "imp_model")

# Explorar la temperatura del aire y la humedad, coloreando los valores
# faltantes y facetando por modelo de imputación
ggplot(bound_models, aes(x = air_temp_c, y = humidity, color = any_missing)) +
  geom_point() + facet_wrap(~imp_model)
```



2.11 Evaluando imputaciones y modelos

2.11.1 Combinación y comparación de muchos modelos de imputación

Para evaluar los diferentes métodos de imputación, necesitamos ponerlos en un único dataframe. A continuación, compararás tres enfoques diferentes para manejar los datos faltantes utilizando el conjunto de datos `oceanbuoys`.

El primer método es utilizar solo los casos completados y se carga como `ocean_cc`. El segundo método es imputar valores utilizando un modelo lineal con predicciones hechas con wind y se carga como `ocean_imp_lm_wind`. Crearás el tercer conjunto de datos imputados, `ocean_imp_lm_all`, utilizando un modelo lineal e imputando las variables `sea_temp_c`, `air_temp_c` y `humidity` utilizando las variables `wind_ew`, `wind_ns`, `year`, `latitude` y `longitude`.

Luego unirás todos los conjuntos de datos (`ocean_cc`, `ocean_imp_lm_wind` y `ocean_imp_lm_all`), llamándolo `bound_models`.

```
ocean_cc<-oceanbuoys %>%
na.omit() %>%
bind_shadow %>%
add_label_shadow()
```

```
# Crear un conjunto de datos imputado utilizando modelos lineales
ocean_imp_lm_all <- bind_shadow(oceanbuoys) %>%
  add_label_shadow() %>%
  impute_lm(sea_temp_c ~ wind_ew + wind_ns + year + latitude + longitude) %>%
  impute_lm(air_temp_c ~ wind_ew + wind_ns + year + latitude + longitude) %>%
  impute_lm(humidity ~ wind_ew + wind_ns + year + latitude + longitude)
```

```
# Unir los conjuntos de datos
bound_models <- bind_rows(cc = ocean_cc,
                          imp_lm_wind = ocean_imp_lm_wind,
                          imp_lm_all = ocean_imp_lm_all,
                          .id = "imp_model")

# Observar los modelos
bound_models
```

```
## # A tibble: 2,037 x 18
##    imp_model  year latitude longitude sea_temp_c air_temp_c humidity wind_ew
##    <chr>     <dbl>    <dbl>     <dbl>      <dbl>      <dbl>    <dbl>   <dbl>
##  1 cc         1997        0      -110       27.6       27.1     79.6   -6.40
##  2 cc         1997        0      -110       27.5       27.0     75.8   -5.30
##  3 cc         1997        0      -110       27.6       27       76.5   -5.10
##  4 cc         1997        0      -110       27.6       26.9     76.2   -4.90
##  5 cc         1997        0      -110       27.6       26.8     76.4   -3.5
##  6 cc         1997        0      -110       27.8       26.9     76.7   -4.40
##  7 cc         1997        0      -110       28.0       27.0     76.5   -2
##  8 cc         1997        0      -110       28.0       27.1     78.3   -3.70
##  9 cc         1997        0      -110       28.0       27.2     78.6   -4.20
## 10 cc         1997        0      -110       28.0       27.2     76.9   -3.60
## # i 2,027 more rows
## # i 10 more variables: wind_ns <dbl>, year_NA <fct>, latitude_NA <fct>,
## #   longitude_NA <fct>, sea_temp_c_NA <fct>, air_temp_c_NA <fct>,
## #   humidity_NA <fct>, wind_ew_NA <fct>, wind_ns_NA <fct>, any_missing <chr>
```

2.11.2 Evaluando los diferentes parámetros en el modelo

stamos imputando nuestros datos por una razón: ¡queremos analizar los datos!

En este ejemplo, estamos interesados en predecir la temperatura del mar, así que vamos a construir un modelo lineal que prediga la temperatura del mar.

Ajustaremos este modelo a cada uno de los conjuntos de datos que creamos y luego exploraremos los coeficientes en los datos.

Los objetos de la lección anterior (`ocean_cc`, `ocean_imp_lm_wind`, `ocean_imp_lm_all` y `bound_models`) se cargan en el espacio de trabajo.

```
# Crear el resumen del modelo para cada conjunto de datos
model_summary <- bound_models %>%
  group_by(imp_model) %>%
  nest() %>%
  mutate(mod = purrr::map(data, ~lm(sea_temp_c ~ air_temp_c + humidity + year, data = .)),
         res = purrr::map(mod, residuals),
         pred = purrr::map(mod, predict),
         tidy = purrr::map(mod, broom::tidy))

# Explorar los coeficientes en el modelo
model_summary %>%
    select(imp_model,tidy) %>%
    unnest()
```

```
## # A tibble: 12 x 6
## # Groups:   imp_model [3]
```

```
##    imp_model   term          estimate std.error statistic   p.value
##    <chr>       <chr>            <dbl>     <dbl>     <dbl>     <dbl>
##  1 cc          (Intercept)  -735.      45.9        -16.0  8.19e- 48
##  2 cc          air_temp_c      0.864    0.0231      37.4  2.64e-154
##  3 cc          humidity        0.0341   0.00390      8.74 2.69e- 17
##  4 cc          year            0.369    0.0232      15.9  3.46e- 47
##  5 imp_lm_wind (Intercept) -1742.      56.1        -31.0  1.83e-135
##  6 imp_lm_wind air_temp_c      0.365    0.0279      13.1  2.73e- 35
##  7 imp_lm_wind humidity        0.0225   0.00690      3.26 1.17e-  3
##  8 imp_lm_wind year            0.880    0.0283      31.1  6.79e-136
##  9 imp_lm_all  (Intercept)  -697.      51.8        -13.5  5.04e- 37
## 10 imp_lm_all  air_temp_c      0.890    0.0255      35.0  2.90e-158
## 11 imp_lm_all  humidity        0.0127   0.00463      2.75 6.03e-  3
## 12 imp_lm_all  year            0.351    0.0262      13.4  1.12e- 36
```

```
best_model <- "imp_lm_all"
```

3 Evaluación de la no respuesta

3.1 Actividad

Considerando el siguien set de datos.

```
library(dplyr)
library(ggplot2)
biopics <- read.csv("curso_imputacion/biopics.csv")
```

Muestra las primeras 10 observaciones de los datos `biopics` y familiarízate con las variables.

```
# Muestra las primeras 10 observaciones
head(biopics, 10)
```

```
##    country year earnings sub_num sub_type sub_race non_white sub_sex
## 1       UK 1971       NA       1 Criminal     <NA>         0    Male
## 2    US/UK 2013   56.700       1    Other  African         1    Male
## 3    US/UK 2010   18.300       1  Athlete     <NA>         0    Male
## 4   Canada 2014       NA       1    Other    White         0    Male
## 5       US 1998    0.537       1    Other     <NA>         0    Male
## 6       US 2008   81.200       1    Other    other         1    Male
## 7       UK 2002    1.130       1 Musician    White         0    Male
## 8       US 2013   95.000       1  Athlete  African         1    Male
## 9       US 1994   19.600       1  Athlete     <NA>         0    Male
## 10   US/UK 1987    1.080       2   Author     <NA>         0    Male
```

```
# Obtiene el numero de valores perdidos por variable
biopics %>%
    is.na() %>%
    colSums()
```

```
##   country      year  earnings   sub_num  sub_type  sub_race non_white   sub_sex
##         0         0       324         0         0       197         0         0
```

3.2 Reconociendo los mecanismos de datos faltantes

En este ejercicio, se presentarán seis escenarios diferentes en los que faltan algunos datos. Intenta asignar a cada uno de ellos el mecanismo de datos faltantes más probable. Como recordatorio, aquí hay algunas pautas generales:

Si la razón de la falta de datos es puramente aleatoria, es MCAR. Si la razón de la falta de datos puede explicarse por otra variable, es MAR. Si la razón de la falta de datos depende del valor faltante en sí mismo, es MNAR.

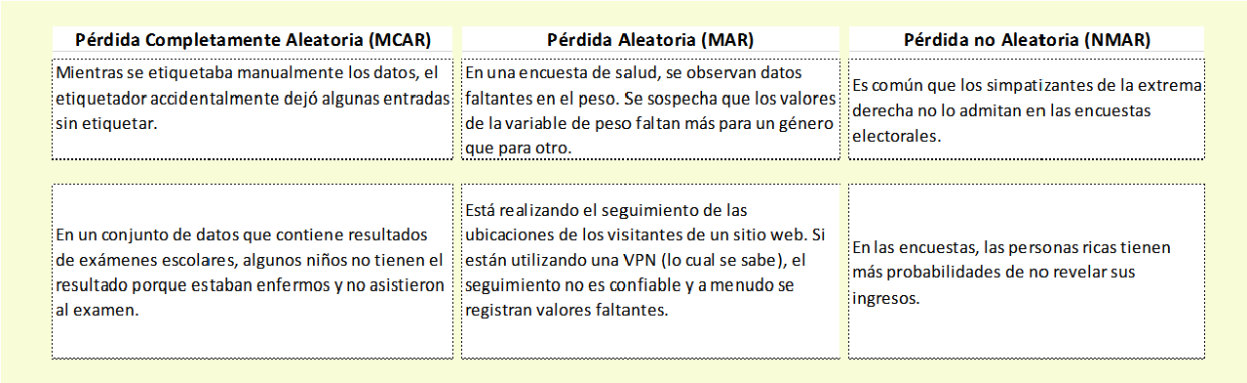


Figura 1: alt text

3.2.1 Prueba t para perdida MAR

3.2.1.1 Preparación de los datos De los tres, MAR es posiblemente el más importante de detectar, ya que muchos métodos de imputación asumen que los datos son MAR. En este ejercicio práctico con `R`, buscaremos identificar si el patrón de pérdida es MAR.

Trabajaremos con los datos de la base `biopics`. El objetivo es probar si el número de valores faltantes en `earnings` difiere por género del sujeto. En este ejercicio, solo se preparan los datos para aplicar una *prueba t*. Primero, se crea una variable ficticia que indica la falta de datos en `earnings`. Luego, se divide por género filtrando los datos para mantener uno de los géneros y luego sacando la variable ficticia. Para filtrar, puede ser útil imprimir el `head()` de `biopics` en la consola y examinar la variable de género.

```
# Crea una variable dummy para la perdida en el gasto
biopics <- biopics %>%
  mutate(missing_earnings = is.na(earnings))

# Obtiene la perdida del gasto para hombres
missing_earnings_males <- biopics %>%
  filter(sub_sex == "Male") %>%
  pull(missing_earnings)

# Obtiene la perdida del gasto para mujeres
missing_earnings_females <- biopics %>%
  filter(sub_sex == "Female") %>%
  pull(missing_earnings)
```

3.2.1.2 Interpretación En el ejercicio anterior, hemos preparado dos vectores con los valores faltantes de ingresos para cada sexo: `perdidos_gastos_hombres` y `perdidos_gastos_mujeres`. Ambos están disponibles en tu espacio de trabajo. Ahora es posible realizar la **prueba t** para verificar si sus medias difieren significativamente entre sí con el siguiente script

```
# Ejecuta el t-test
t.test(missing_earnings_males, missing_earnings_females)
```

```
##
##  Welch Two Sample t-test
```

```
##
## data:  missing_earnings_males and missing_earnings_females
## t = 1.1116, df = 294.39, p-value = 0.2672
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.03606549  0.12969214
## sample estimates:
## mean of x mean of y
## 0.4366438 0.3898305
```

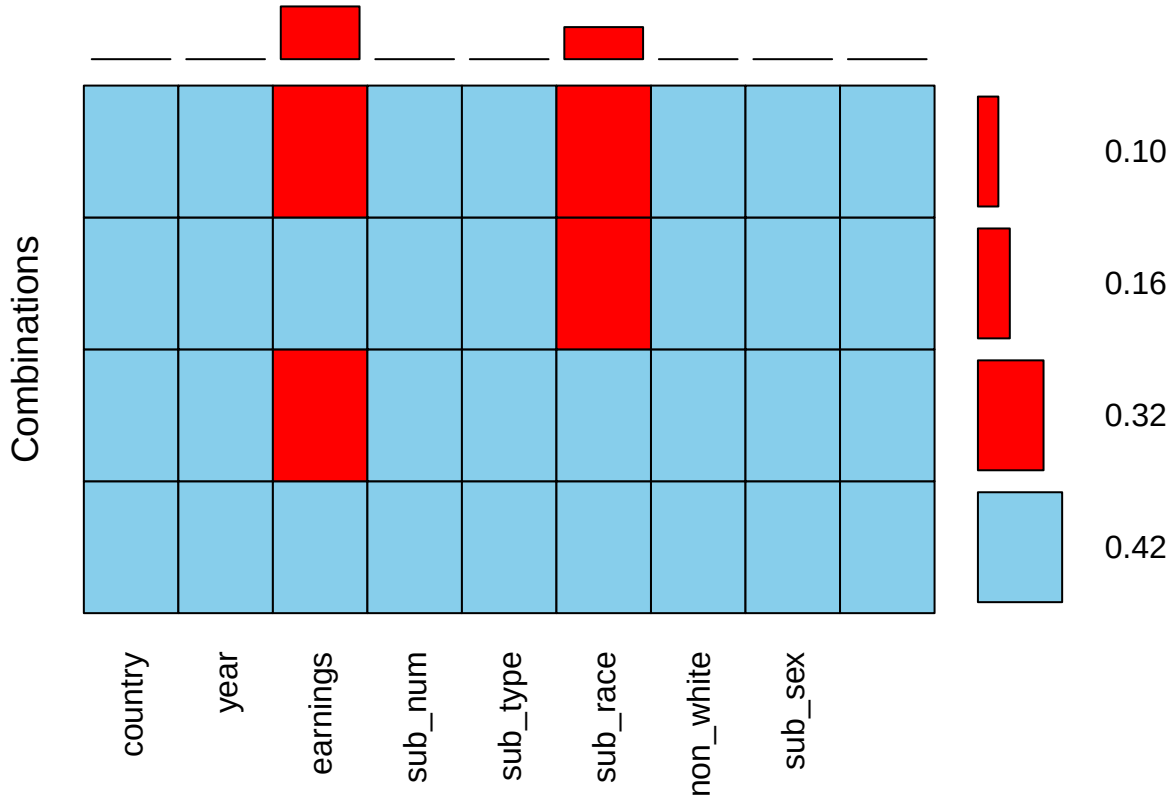
El resultado muestra que no existe diferencia estadísticamente significativa ($\alpha > 0.05$) entre ambos grupos. Por lo tanto, se concluye que la perdida es MAR.

3.2.2 Aggregation plot

El gráfico de agregación proporciona la respuesta a la pregunta básica que uno puede hacer sobre un conjunto de datos incompleto: ¿en qué combinaciones de variables faltan datos y con qué frecuencia? Es muy útil para obtener una visión general de alto nivel de los patrones de ausencia de datos. Por ejemplo, hace visible inmediatamente si hay alguna combinación de variables que faltan juntas con frecuencia, lo que podría sugerir alguna relación entre ellas.

En este ejercicio, primero aplicaremos el gráfico de agregación para los datos de `biopics` y luego practicarás sacando conclusiones basadas en él. ¡Vamos a hacer algunos gráficos!

```
# Carga el paquete VIM
library(VIM)

# Dibuja un aggregation plot para biopics
biopics %>%
    aggr(combined = TRUE, numbers = TRUE)
```



3.2.3 Cuestiones aclaratorias

Basado en el gráfico de agregación que acaba de crear, ¿cuál de las siguientes afirmaciones es falsa?

Posibles respuestas:

a. El 10% de las observaciones tienen valores faltantes tanto en `earnings` como en `sub_race`.

b. Hay más valores faltantes en `sub_race` que en `earnings`.

c. El 42% de las observaciones no tiene entradas faltantes.

d. Exactamente dos variables en los datos `biopics` tienen valores faltantes.

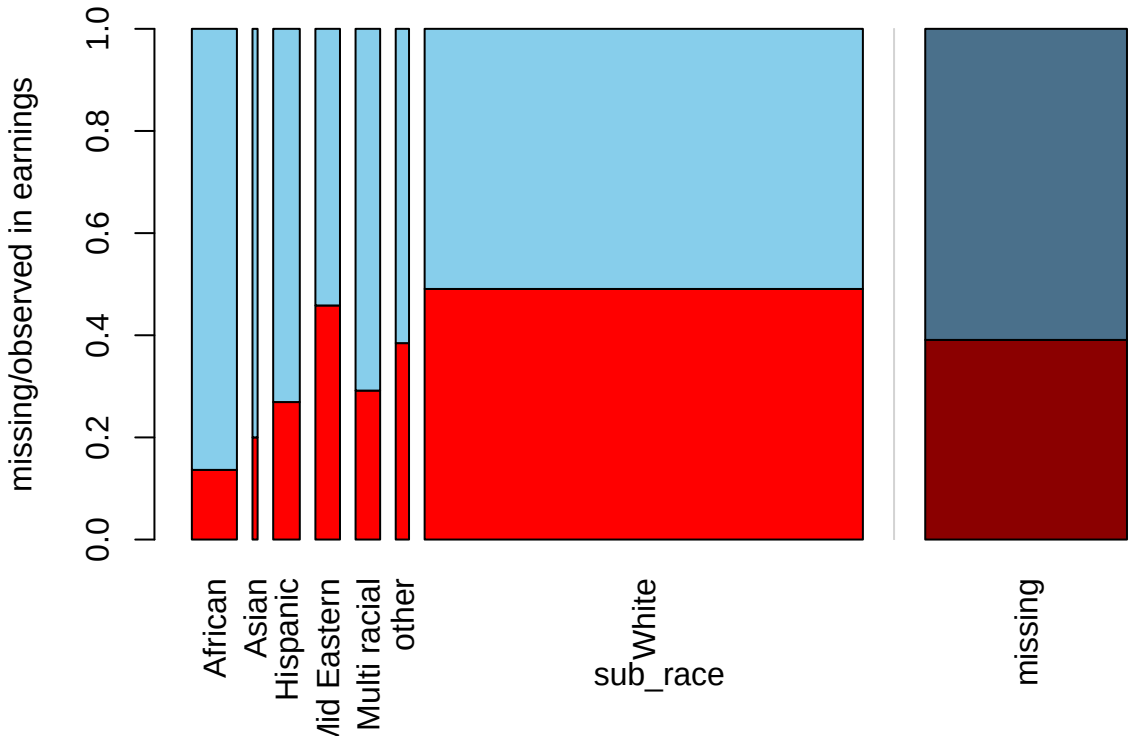
3.2.4 gráfico de Mosaico

El gráfico de agregación que has dibujado en el ejercicio anterior te dio una visión general de alto nivel de los datos faltantes. Si estás interesado en la interacción entre variables específicas, un gráfico de Mosaico es el camino a seguir. Te permite estudiar el porcentaje de valores faltantes en una variable para diferentes valores de la otra, lo cual es conceptualmente muy similar a los test t que has estado realizando en la lección anterior.

En este ejercicio, dibujarás un gráfico de Mosaico para investigar el porcentaje de datos faltantes en `earnings` para diferentes categorías de `sub_race`. ¿Hay más datos faltantes en `earnings` para algunas razas específicas del personaje principal de la película? ¡Vamos a descubrirlo! El paquete `VIM` ya ha sido cargado para ti.

```
# Dibujamos un spine plot para visualizar los valores
# perdidos en earnings por sub_race

biopics %>%
```

```
    dplyr::select(sub_race,earnings) %>%
    spineMiss()
```



3.2.5 Cuestiones aclaratorias

Basándose en la gráfica de Mosaico que acabas de crear, ¿cuál de las siguientes afirmaciones es falsa?

Opciones de respuesta:

a. En la gran mayoría de las películas, el personaje principal es blanco.

b. Cuando el sujeto principal es africano, es más probable que tengamos información completa sobre las ganancias.

c. En lo que respecta a las ganancias y la subraza, los datos parecen ser MAR.

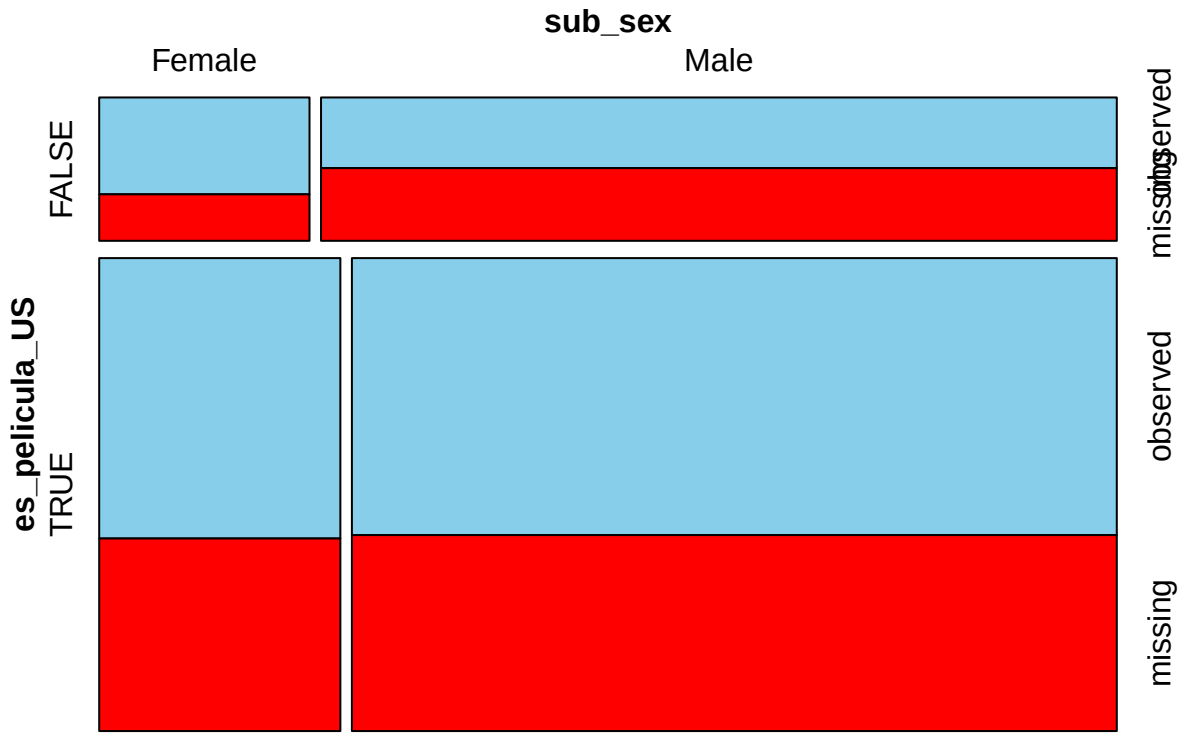
d. La raza que aparece con menos frecuencia en los datos tiene alrededor del 40% de las ganancias faltantes. (incorrecta)

3.2.6 Mosaic plot

La gráfica de Mosaico que hemos creado en el ejercicio anterior permite estudiar los patrones de datos faltantes entre dos variables a la vez. Esta idea se generaliza a más variables en forma de un gráfico de mosaico.

En este ejercicio, comenzarás por crear una variable ficticia que indique si Estados Unidos participó en la producción de cada película. Para hacer esto, utilizarás la función `grepl()`, que verifica si la cadena pasada como su primer argumento está presente en el objeto pasado como su segundo argumento. Luego, crearemos un gráfico de mosaico para ver si el género del sujeto se correlaciona con la cantidad de datos faltantes en `earnings` tanto para películas estadounidenses como no estadounidenses.

```
# Preparar los datos para trazar un gráfico y dibujar un gráfico de mosaico
biopics %>%
    # Crear una variable ficticia para películas producidas en EE. UU.
    mutate(es_pelicula_US = grepl("US", country)) %>%
    # Dibujar un gráfico de mosaico
    mosaicMiss(highlight = "earnings",
               plotvars = c("es_pelicula_US", "sub_sex"))
```



3.2.7 Olfateando el peligro de la imputación por la media

Uno de los métodos de imputación más populares es la imputación por media, en la cual los valores faltantes en una variable se reemplazan con la media de los valores observados en esa variable. Sin embargo, en muchos casos, este enfoque simple es una mala elección. A veces, una mirada rápida a los datos puede alertarnos sobre los peligros de la imputación por la media.

En esta etapa, trabajaremos con una submuestra de los datos del proyecto de *Atmósfera Tropical Oceánica* (`tao`). El conjunto de datos consiste en mediciones atmosféricas tomadas en dos períodos de tiempo diferentes en cinco ubicaciones distintas. Los datos vienen con el paquete `VIM`.

En este ejercicio, nos familiarizaremos con los datos y realizaremos un análisis simple que indicará cuáles podrían ser las consecuencias de la imputación por la media.

```
data(tao, package = "VIM")
names(tao)<-tolower(names(tao))
names(tao)<-sub("[.]", "_", names(tao))
names(tao)<-sub("[.]", "_", names(tao))
```

```
# Imprime las primeras 10 observaciones
head(tao, 10)
```

```
##    year latitude longitude sea_surface_temp air_temp humidity uwind vwind
## 1  1997        0      -110            27.59    27.15     79.6  -6.4   5.4
## 2  1997        0      -110            27.55    27.02     75.8  -5.3   5.3
## 3  1997        0      -110            27.57    27.00     76.5  -5.1   4.5
## 4  1997        0      -110            27.62    26.93     76.2  -4.9   2.5
## 5  1997        0      -110            27.65    26.84     76.4  -3.5   4.1
## 6  1997        0      -110            27.83    26.94     76.7  -4.4   1.6
## 7  1997        0      -110            28.01    27.04     76.5  -2.0   3.5
## 8  1997        0      -110            28.04    27.11     78.3  -3.7   4.5
## 9  1997        0      -110            28.02    27.21     78.6  -4.2   5.0
## 10 1997        0      -110            28.05    27.25     76.9  -3.6   3.5
```

```
# Obtiene el numero de valores perdidos por columna
tao %>%
  is.na() %>%
  colSums()
```

```
##             year         latitude        longitude sea_surface_temp
##                0                0                0                3
##         air_temp         humidity            uwind            vwind
##               81               93                0                0
```

```
# Calcula el numero de valores perdidos o missing values en air_temp por year
tao %>%
  group_by(year) %>%
  summarize(num_miss = sum(is.na(air_temp)))
```

```
## # A tibble: 2 x 2
##    year num_miss
##   <int>    <int>
## 1  1993        4
## 2  1997       77
```

3.2.8 Imputación de la media en temperatura

Imputar la media en la temperatura puede ser arriesgado. Si la variable que se está imputando está correlacionada con otras variables, esta correlación podría ser destruida por los valores imputados. Lo viste en el ejercicio anterior cuando analizaste la variable `air_temp`.

Para averiguar si estas preocupaciones son válidas, en este ejercicio realizarás una imputación de la media en `air_temp`, creando también un indicador binario para mostrar dónde se imputan los valores. Será útil en el siguiente ejercicio, cuando evaluarás el desempeño de tu imputación. ¡Vamos a completar esos valores faltantes!

```
tao_imp <- tao %>%
  # Crear un indicador binario para valores faltantes en air_temp
  mutate(air_temp_imp = ifelse(is.na(air_temp), TRUE, FALSE)) %>%
  # Imputar air_temp con su promedio
  mutate(air_temp = ifelse(is.na(air_temp), mean(air_temp,
                                                 na.rm = TRUE), air_temp))

# Imprimir las primeras 10 filas de tao_imp
head(tao_imp, 10)
```

```
##    year latitude longitude sea_surface_temp air_temp humidity uwind vwind
## 1  1997        0      -110            27.59    27.15     79.6  -6.4   5.4
## 2  1997        0      -110            27.55    27.02     75.8  -5.3   5.3
## 3  1997        0      -110            27.57    27.00     76.5  -5.1   4.5
## 4  1997        0      -110            27.62    26.93     76.2  -4.9   2.5
## 5  1997        0      -110            27.65    26.84     76.4  -3.5   4.1
## 6  1997        0      -110            27.83    26.94     76.7  -4.4   1.6
## 7  1997        0      -110            28.01    27.04     76.5  -2.0   3.5
## 8  1997        0      -110            28.04    27.11     78.3  -3.7   4.5
## 9  1997        0      -110            28.02    27.21     78.6  -4.2   5.0
## 10 1997        0      -110            28.05    27.25     76.9  -3.6   3.5
##    air_temp_imp
## 1         FALSE
## 2         FALSE
## 3         FALSE
## 4         FALSE
## 5         FALSE
## 6         FALSE
## 7         FALSE
## 8         FALSE
## 9         FALSE
## 10        FALSE
```

```
head(filter(tao_imp, air_temp_imp==TRUE))
```

```
##   year latitude longitude sea_surface_temp air_temp humidity uwind vwind
## 1 1997        0       -95            27.69 25.02925     79.8  -0.6   4.2
## 2 1997        0       -95            27.63 25.02925     74.5  -3.9   5.8
## 3 1997        0       -95            27.51 25.02925     76.3  -2.8   5.3
## 4 1997        0       -95            27.54 25.02925     81.6  -2.3   5.6
## 5 1997        0       -95            27.47 25.02925     81.9  -4.6   6.1
## 6 1997        0       -95            27.44 25.02925     74.2  -4.4   6.5
##   air_temp_imp
## 1         TRUE
## 2         TRUE
## 3         TRUE
## 4         TRUE
## 5         TRUE
## 6         TRUE
```

Nos damos cuenta que no tiene mucho sentido imputar por la media, ya que puede agregar inconsistencias entre las variables correlacionadas.

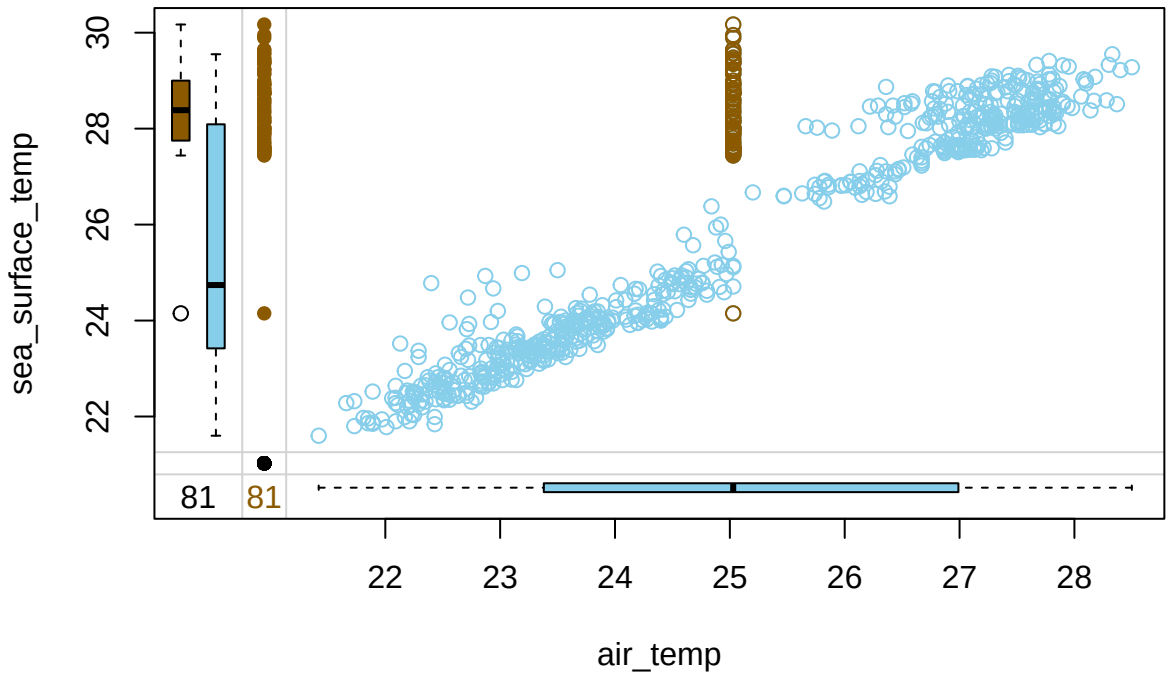
3.2.9 Evaluar la calidad de la imputación con un marginplot

En el último ejercicio, hemos imputado la media de `air_temp` y hemos agregado una variable indicadora para denotar cuáles valores fueron imputados, llamada `air_temp_imp`. Ahora es momento de ver qué tan bien funciona esto.

Al examinar los datos de `tao`, podríamos haber notado que también contiene una variable llamada `sea_surface_temp`, que razonablemente se esperaría que esté positivamente correlacionada con `air_temp`. Si ese es el caso, esperaríamos que estas dos temperaturas sean altas o bajas al mismo tiempo. Imputar la temperatura media del aire cuando la temperatura del mar es alta o baja rompería esta relación.

Para averiguarlo, en este ejercicio seleccionaremos las dos variables de temperatura y la variable indicadora y las usaremos para crear un `marginplot`.

```
# Creamos un marginplot de air_temp vs sea_surface_temp
tao_imp %>%
  select(air_temp, sea_surface_temp, air_temp_imp) %>%
  marginplot(delimiter = "imp")
```



3.2.10 Imputación por hot-deck

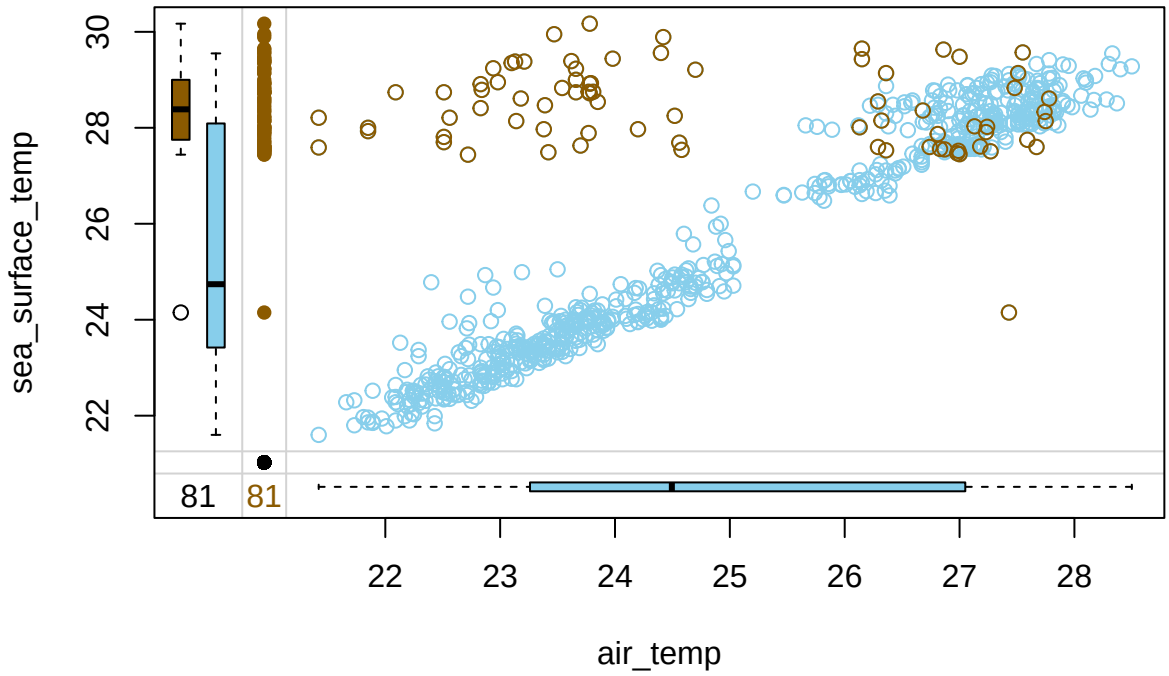
La imputación por hot-deck es un método simple que reemplaza cada valor faltante en una variable por el último valor observado en esa variable. Es muy rápido, ya que solo se necesita una revisión por los datos, pero en su forma más simple, hot-deck a veces puede romper las relaciones entre las variables.

En este ejemplo, lo probaremos en el conjunto de datos `tao`. Imputaremos los valores faltantes en la columna de temperatura del aire `air_temp` por hot-deck y luego visualizaremos un gráfico de margen (`marginplot`) para analizar la relación entre los valores imputados y la columna de temperatura de la superficie del mar `sea_surface_temp`.

```
# Cargar el paquete VIM
library(VIM)

# Imputar la temperatura del aire en tao con imputación por hot-deck
tao_imp <- hotdeck(tao, variable = "air_temp")

# Comprobar la cantidad de valores faltantes en cada variable
tao_imp %>%
    is.na() %>%
    colSums()
```

```
##             year         latitude        longitude sea_surface_temp
##                0                0                0                3
##         air_temp         humidity            uwind            vwind
##                0               93                0                0
##     air_temp_imp
##                0
```

```
# Dibujar un gráfico de márgenes de temperatura del aire vs. temperatura
# de la superficie del mar
tao_imp %>%
    select(air_temp, sea_surface_temp, air_temp_imp) %>%
    marginplot(delimiter = "imp")
```



¿Se ve bien la imputación? Observa las observaciones en la parte superior izquierda del gráfico con los datos de `air_temp` imputados y los valores altos en `sea_surface_temp`. Estas observaciones deben haber sido precedidas por observaciones con bajos valores de `air_temp` en el `data frame`, y por lo tanto, después de la imputación hot-deck, terminaron siendo valores atípicos con `air_temp` bajos y `sea_surface_temp` altos.

3.2.11 Hot-deck trucos y consejos I: imputando dentro de dominios

Un truco que puede ayudar cuando la imputación por hot-deck rompe las relaciones entre las variables es imputar dentro de los dominios. Esto significa que si la variable a imputar está correlacionada con otra variable categórica, se puede ejecutar hot-deck por separado para cada una de sus categorías.

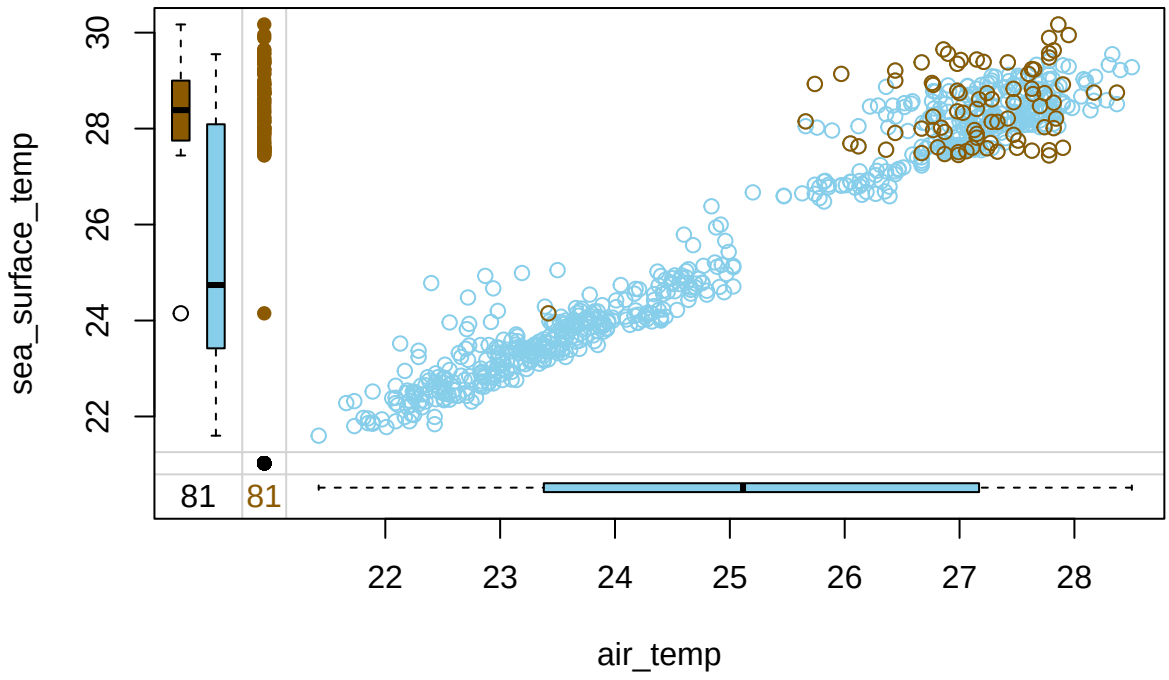
Por ejemplo, se podría esperar que la temperatura del aire dependa del tiempo, ya que estamos viendo que las temperaturas promedio aumentan debido al calentamiento global. El indicador de tiempo que tenemos disponible en los datos de `tao` es una variable categórica, `year`. Primero, comprobaremos si la temperatura media del aire es diferente en cada uno de los dos años estudiados y luego ejecutaremos hot-deck dentro de

los dominios de los años. Finalmente, volveremos a crear el `marginplot` para evaluar el rendimiento de la imputación.

```
# Calcular la temperatura media del aire por año
tao %>%
    group_by(year) %>%
    summarize(temperatura_media_aire = mean(air_temp, na.rm = TRUE))
```

```
## # A tibble: 2 x 2
##    year temperatura_media_aire
##   <int>                  <dbl>
## 1  1993                   23.4
## 2  1997                   27.1
```

```
# Imputar temperatura del aire en tao por dominio de año utilizando hot-deck
tao_imp <- hotdeck(tao, variable = "air_temp", domain_var = "year")

# Dibujar un gráfico de márgenes de temperatura del aire vs. temperatura de
# la superficie del mar
tao_imp %>%
    select(air_temp, sea_surface_temp, air_temp_imp) %>%
    marginplot(delimiter = "imp")
```



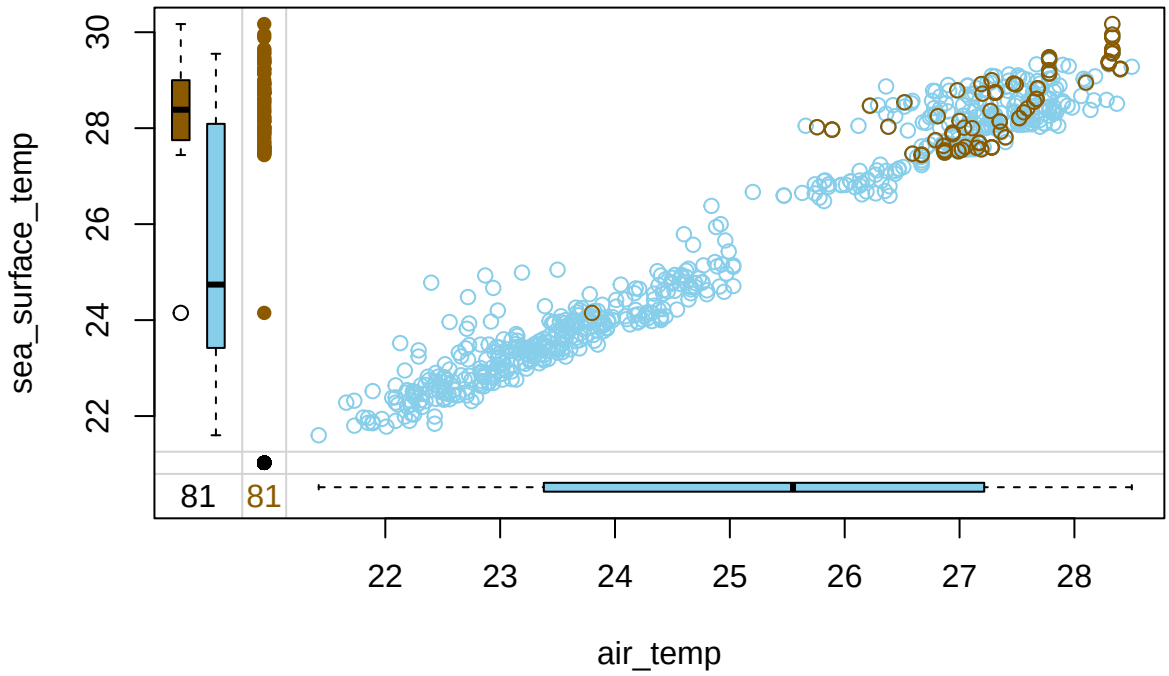
Los resultados se ven mucho mejor esta vez. Sin embargo, si observas la esquina superior derecha del gráfico, verás que la varianza en los valores imputados (naranja) es algo mayor que entre los valores observados (azul). Veamos si podemos mejorar aún más en el próximo ejercicio.

3.2.12 Hot-deck trucos y consejos II: ordenando por variables correlacionadas

Otro truco que puede mejorar el rendimiento de la imputación hot-deck es ordenar los datos por variables correlacionadas con la que queremos imputar.

Por ejemplo, en todos los `marginplot` que hemos estado usando recientemente, se ha visto que la temperatura del aire está fuertemente correlacionada con la temperatura de la superficie del mar, lo cual tiene mucho sentido. Podemos aprovechar este conocimiento para mejorar la imputación hot-deck. Si primero ordenamos los datos por `sea_surface_temp`, entonces cada valor imputado de `air_temp` vendrá de un donante con una `sea_surface_temp` similar.

```
# Imputar temperatura del aire en tao utilizando hot-deck, ordenando por
# temperatura de la superficie del mar
tao_imp <- hotdeck(tao, variable = "air_temp", ord_var = "sea_surface_temp")

# Dibujar un gráfico de márgenes de temperatura del aire vs. temperatura
# de la superficie del mar
tao_imp %>%
    select(air_temp, sea_surface_temp, air_temp_imp) %>%
    marginplot(delimiter = "imp")
```



Esta vez la imputación parece no afectar la relación entre las temperaturas del aire y la superficie del mar: si no fuera por los colores, probablemente no sabriamos cuáles son los valores imputados. La imputación hot-deck, posiblemente mejorada con la imputación por dominios o el ordenamiento, es un método rápido y sencillo que puede funcionar bien en muchas situaciones. Sin embargo, a veces puede ser necesario un enfoque más complejo.

3.2.13 Elegir el número de vecinos

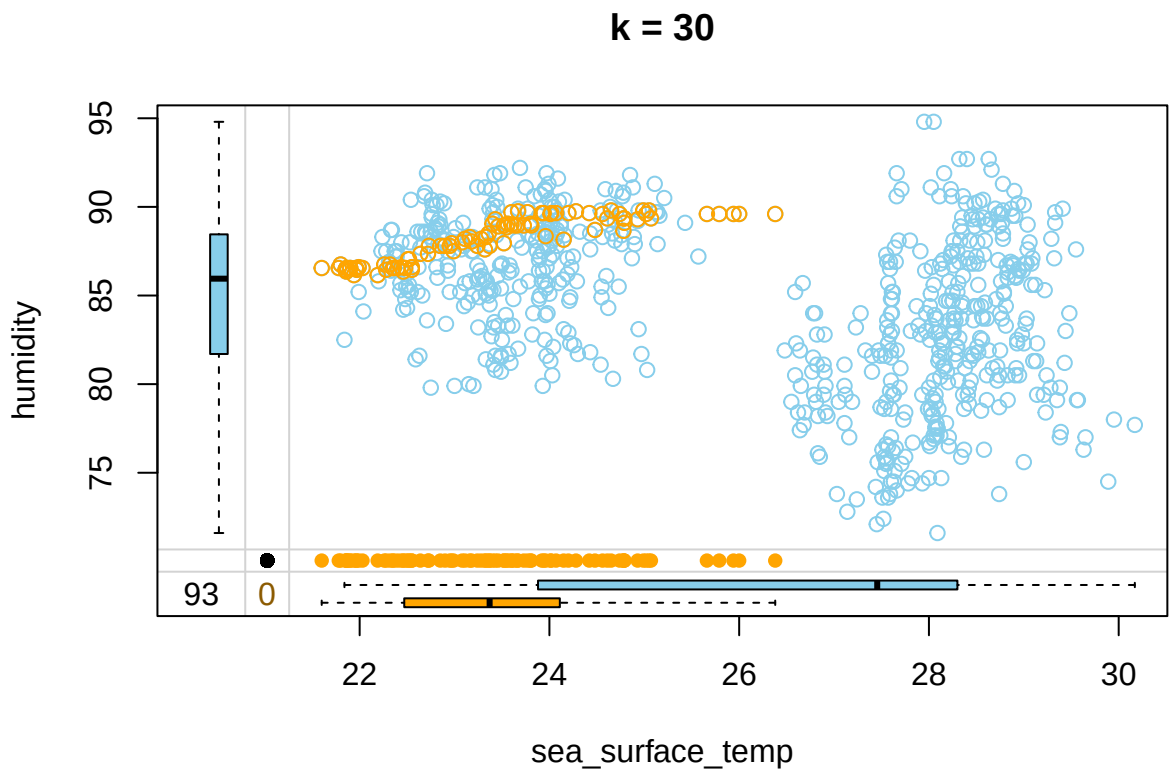
La imputación de k-Nearest-Neighbors (o kNN) imputa los valores faltantes en una observación en función de los valores que provienen de las k otras observaciones más similares a ella. El número de estas observaciones similares, llamadas vecinos, se consideran que es un parámetro que debe elegirse de antemano.

¿Cómo elegir k? Una forma es probar diferentes valores y ver cómo afectan las relaciones entre los datos imputados y observados.

Intentemos imputar `humidity` en los datos de `tao` utilizando tres valores diferentes de k y ver cómo se ajustan los valores imputados a la relación entre `humidity` y `sea_surface_temp`.

Imputamos `humidity` con la imputación de kNN usando 30 vecinos y visualizadolo mediante un `marginplot()` de `sea_surface_temp` vs `humidity`.

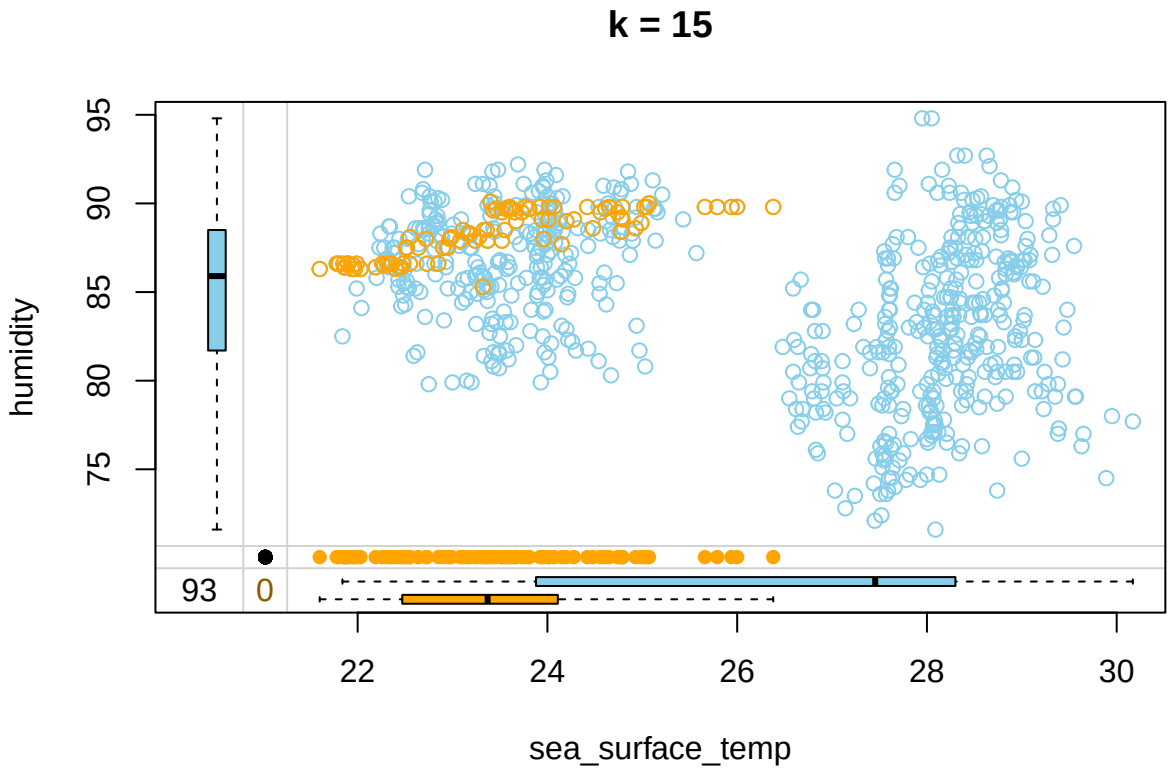
```
# Imputar humedad utilizando 30 vecinos
tao_imp <- kNN(tao, k = 30, variable = "humidity")

# Dibujar un gráfico de márgenes de temperatura de la superficie del
# mar vs. humedad
tao_imp %>%
    select(sea_surface_temp, humidity, humidity_imp) %>%
    marginplot(delimiter = "imp", main = "k = 30")
```



Ahora, imputamos `humidity` con imputación kNN usando 15 vecinos y vemos mediante el `marginplot` de `sea_surface_temp` vs `humidity`.

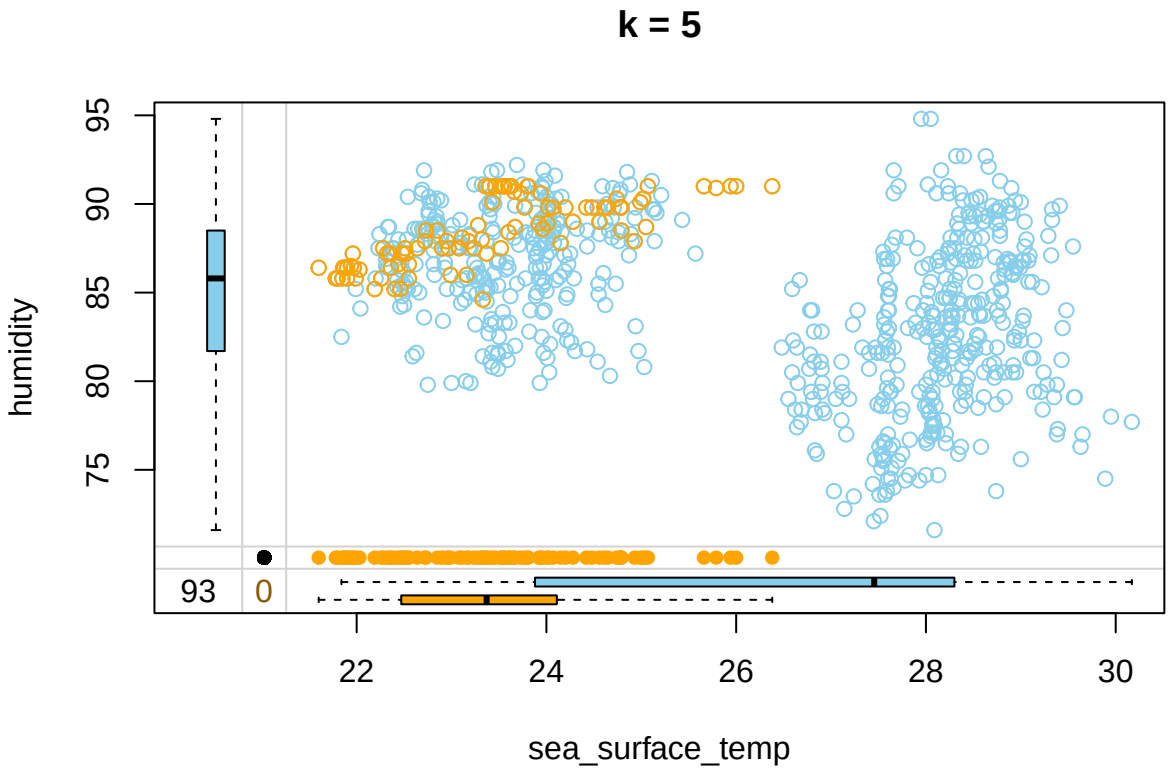
```
# Imputar la humedad utilizando 15 vecinos
tao_imp <- kNN(tao, k = 15, variable = "humidity")
```

```
# Dibujar un gráfico de márgenes de temperatura de la
# superficie del mar vs. humedad
tao_imp %>%
    select(sea_surface_temp, humidity, humidity_imp) %>%
    marginplot(delimiter = "imp", main = "k = 15")
```



Finalmente, imputamos `humidity` con imputación kNN usando 5 vecinos y visualizando los resulados mediante `marginplot` de `sea_surface_temp` vs `humidity`.

```
# Imputar la humedad utilizando 5 vecinos
tao_imp <- kNN(tao, k = 5, variable = "humidity")

# Dibujar un gráfico de márgenes de temperatura de la
# superficie del mar vs. humedad
tao_imp %>%
    select(sea_surface_temp, humidity, humidity_imp) %>%
    marginplot(delimiter = "imp", main = "k = 5")
```

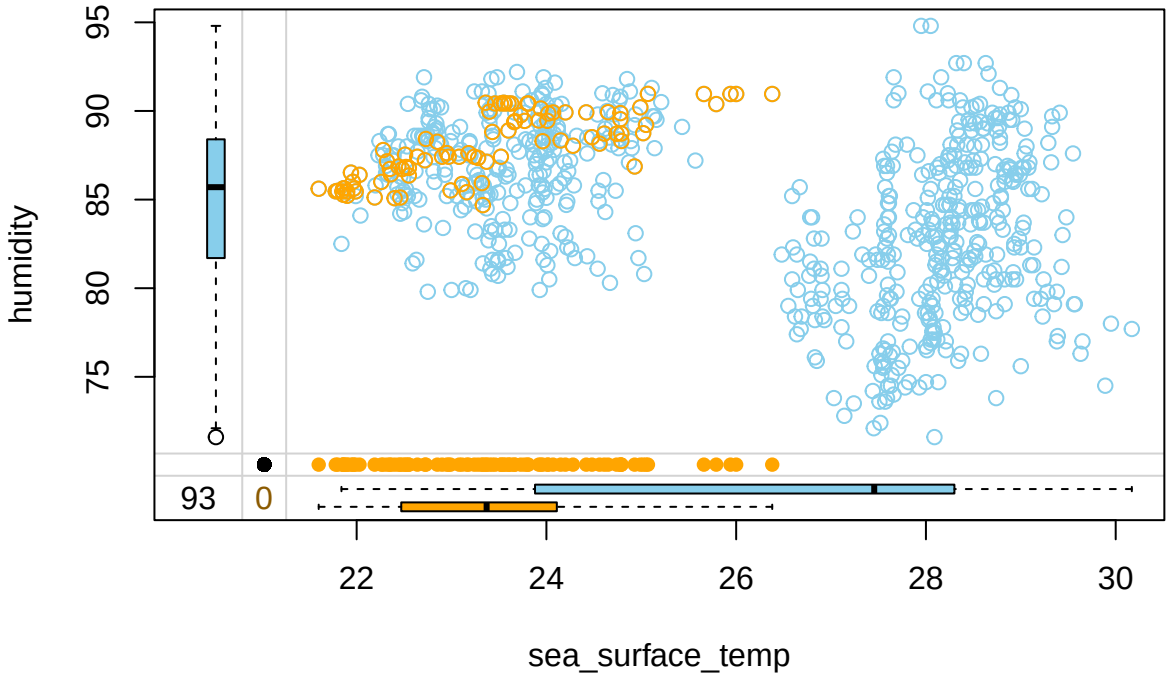


3.2.14 kNN trucos y consejos I: ponderando los donantes

Una variación de la imputación `kNN` que se aplica con frecuencia utiliza la llamada agregación ponderada por distancia. Lo que esto significa es que cuando agregamos los valores de los vecinos para obtener un reemplazo para un valor faltante, lo hacemos usando la media ponderada y las ponderaciones son las distancias invertidas de cada vecino. Como resultado, los vecinos más cercanos tienen más impacto en el valor imputado.

En este ejercicio, aplicamos la agregación ponderada por distancia mientras imputamos los datos de `tao`. Esto solo requerirá dar dos argumentos adicionales a la función `kNN()`.

```
# Cargar el paquete VIM
library(VIM)

# Imputar la humedad con kNN utilizando la media ponderada por distancia
tao_imp <- kNN(tao,
               k = 5,
               variable = "humidity",
               numFun = weighted.mean,
               weightDist = TRUE)

tao_imp %>%
    select(sea_surface_temp, humidity, humidity_imp) %>%
    marginplot(delimiter = "imp")
```

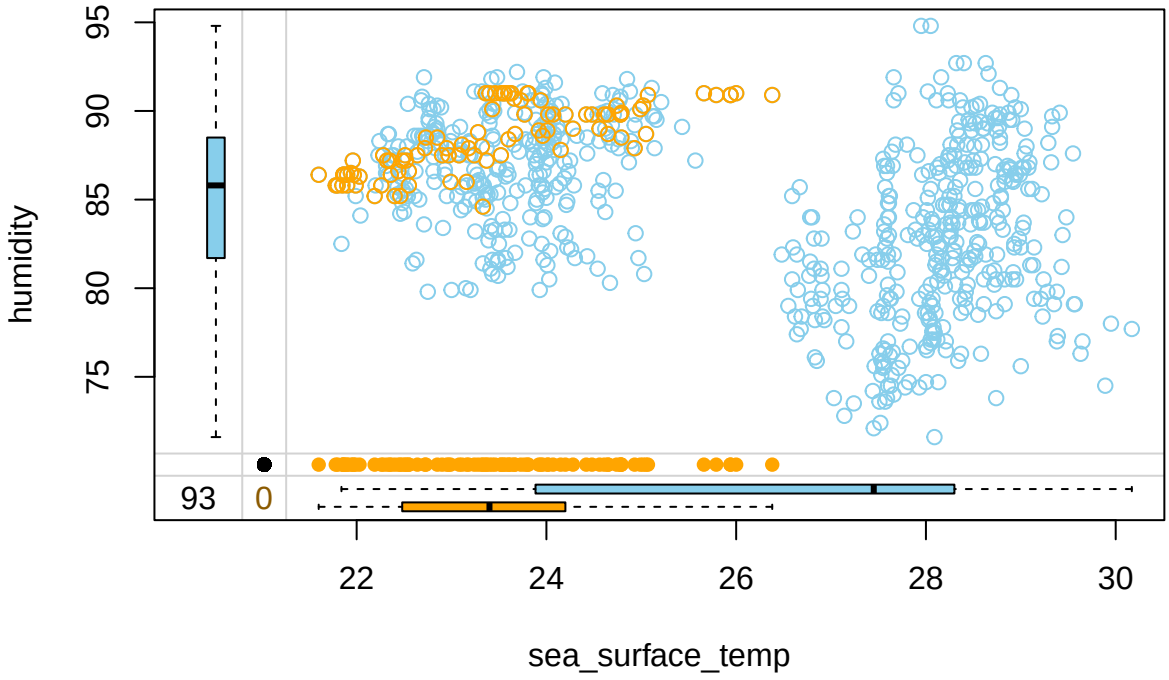


Trucos y consejos de kNN II: ordenar variables

Mientras el algoritmo de k-Nearest Neighbors recorre las variables en los datos para imputarlos, calcula las distancias entre observaciones utilizando otras variables, algunas de las cuales ya han sido imputadas en los pasos anteriores. Esto significa que si las variables ubicadas al principio de los datos tienen muchas valores faltantes, entonces el cálculo de la distancia posterior se basa en muchos valores imputados. Esto introduce ruido en el cálculo de la distancia.

Por esta razón, una buena práctica es ordenar las variables por el número de valores faltantes antes de realizar la imputación kNN. De esta manera, cada cálculo de distancia se basa en tantos datos observados y tan pocos datos imputados como sea posible.

```
# Obtener los nombres de las variables de tao ordenados por la cantidad
# de valores faltantes
vars_by_NAs <- tao %>%
  is.na() %>%
  colSums() %>%
  sort(decreasing = FALSE) %>%
  names()

# Ordenar las variables de tao y alimentarlas en la imputación de kNN
tao_imp <- tao %>%
  select(vars_by_NAs) %>%
  kNN(k = 5)

tao_imp %>%
  select(sea_surface_temp, humidity, humidity_imp) %>%
  marginplot(delimiter = "imp")
```



El kNN que acabamos de programar debería ser más preciso y resistente a imputaciones defectuosas, así que recordemos ordenar las variables primero antes de realizar la imputación con kNN.

3.2.15 Imputación con regresión lineal

A veces, se puede utilizar el conocimiento del dominio, la investigación previa o simplemente el sentido común para describir las relaciones entre las variables en sus datos. En tales casos, la imputación basada en modelos es una gran solución, ya que permite imputar cada variable de acuerdo con un modelo estadístico que puede especificar uno mismo, teniendo en cuenta cualquier suposición que pueda tener sobre cómo las variables impactan entre sí.

Para variables continuas, una elección de modelo popular es la regresión lineal. Siempre puede incluir un cuadrado o un logaritmo de una variable en los predictores. En este caso, trabajaremos con el paquete `simputation` para ejecutar una sola imputación de regresión lineal en los datos `tao` y analizar los resultados.

```
# Lee la libreria simputation
library(simputation)

# Imputa air_temp y humidity con una regresion lineal
formula <- air_temp + humidity ~ year + latitude + sea_surface_temp
tao_imp <- impute_lm(tao, formula)
```

```
# Obtenemos el numero de valores missing por columna
tao_imp %>%
  is.na() %>%
  colSums()
```

```
##             year         latitude        longitude sea_surface_temp
##                0                0                0                3
```

```
##         air_temp         humidity            uwind            vwind
##                3                2                0                0
```

```
# Imprime las celdas de tao_imp en donde air_temp o humidity siguen missing
tao_imp %>%
  filter(is.na(air_temp) | is.na(humidity))
```

```
##   year latitude longitude sea_surface_temp air_temp humidity uwind vwind
## 1 1993        0       -95               NA       NA       NA  -5.6   3.1
## 2 1993        0       -95               NA       NA       NA  -6.3   0.5
## 3 1993       -2       -95               NA       NA     89.9  -3.4   2.4
```

La regresión lineal falla cuando al menos uno de los predictores está ausente. En este caso, fue `sea_surface_temp`. En el próximo ejercicio, lo solucionaremos inicializando los valores faltantes antes de ejecutar `impute_lm()`.

3.2.16 Inicialización de valores perdidos e iteración sobre variables

Como acabamos de ver, la ejecución de `impute_lm()` podría no llenar todos los valores perdidos. Para asegurarte de imputar todos ellos, debemos inicializar los valores perdidos con un método simple, como la imputación de hot-deck que de la sección anterior, que simplemente retroalimenta el último valor observado.

Además, una sola imputación generalmente no es suficiente. Se basa en los valores iniciales básicos y podría estar sesgada. Un enfoque adecuado es iterar sobre las variables, imputándolas una a la vez en las ubicaciones donde originalmente faltan.

En este ejercicio, primero inicializaremos los valores perdidos con la imputación de hot-deck y luego iteraremos cinco veces sobre `air_temp` y `humidity` de los datos `tao` para imputarlos con la regresión lineal.

```
# Inicializa los valores missing con hot-deck
tao_imp <- hotdeck(tao)

# Crea un indicador booleano desde donde air_temp y humidity son missing
missing_air_temp <- tao_imp$air_temp_imp
missing_humidity <- tao_imp$humidity_imp

for (i in 1:5) {
 # Define air_temp como NA en los lugares donde faltaban originalmente y
  # re-imputa
 tao_imp$air_temp[missing_air_temp] <- NA
 tao_imp <- impute_lm(tao_imp, air_temp ~ year + latitude +
                        sea_surface_temp + humidity)
 # Define humidity como NA en los lugares donde faltan originalmente y
 # re-imputa
 tao_imp$humidity[missing_humidity] <- NA
 tao_imp <- impute_lm(tao_imp, humidity ~ year + latitude +
                        sea_surface_temp + air_temp)
}
```

Esa es una aproximación apropiada a la imputación basada en modelos que acabamos de codificar, pero, ¿cómo sabemos que 5 es el número adecuado de iteraciones para ejecutar?.

3.2.17 Detectando convergencia

¿Cuántas iteraciones son necesarias? Cuando los valores imputados no cambian con la nueva iteración, podemos detenernos.

Ahora extenderás nuestro código para calcular las diferencias entre las variables imputadas en las iteraciones

subsiguientes. Para hacer esto, usaremos la función de cambio porcentual promedio absoluto (`mapc`), definida de la siguiente manera:

```
mapc <- function(a, b) {   mean(abs(b - a) / a, na.rm = TRUE) }
```

`mapc()` es una función que devuelve un solo número que te dice cuánto difiere b de a. La usaremos para verificar cuánto cambian las variables imputadas en las iteraciones siguientes. En base a esto, decidiremos cuántas iteraciones son necesarias.

Los indicadores booleanos `missing_air_temp` y `missing_humidity` son usados aquí, al igual que los datos de `tao_imp` inicializados con hot-deck.

```
mapc<- function(a, b) {
  mean(abs(b - a) / a, na.rm = TRUE)
}
```

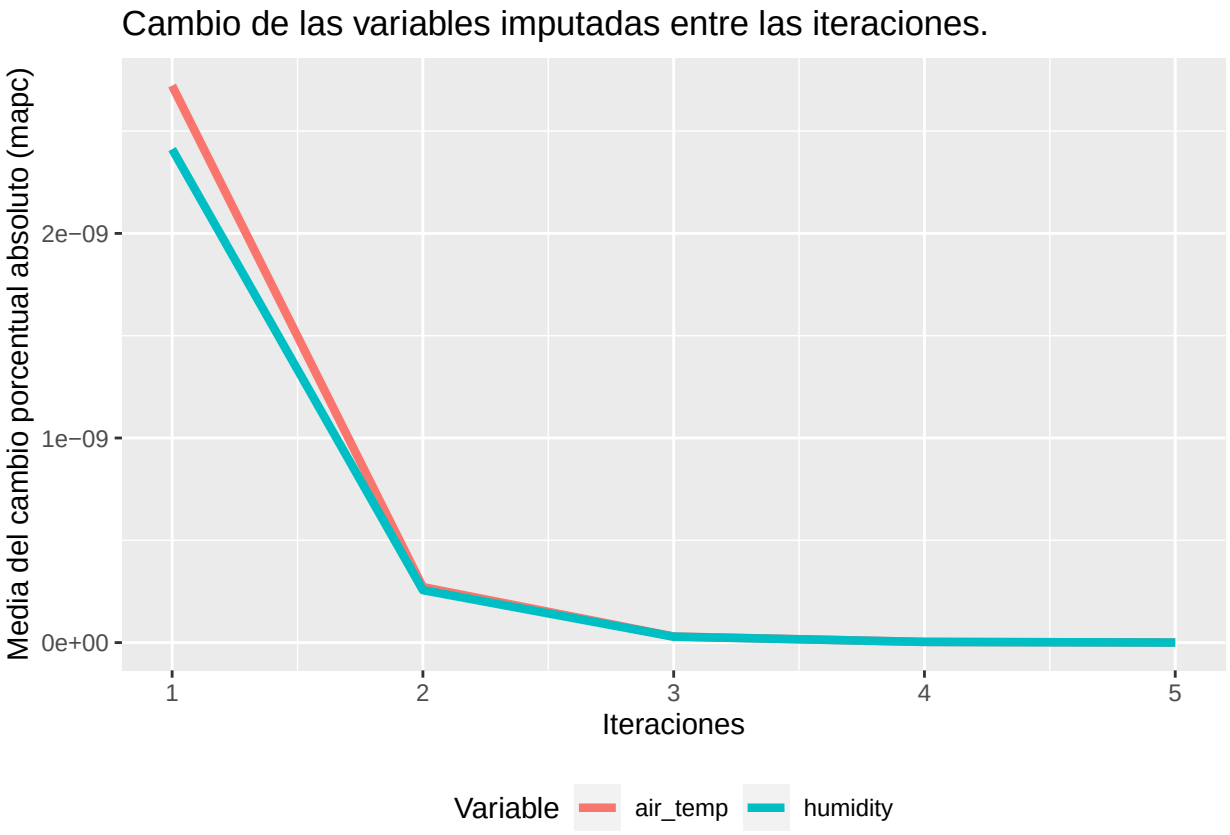
```
diff_air_temp <- c()
diff_humidity <- c()

for (i in 1:5) {
  # Asigna el resultado de la iteración anterior (o inicialización) a prev_iter
  prev_iter <- tao_imp
  # Imputa air_temp y humidity en las ubicaciones que originalmente faltaban
  tao_imp$air_temp[missing_air_temp] <- NA
  tao_imp <- impute_lm(tao_imp, air_temp ~ year + latitude +
                         sea_surface_temp + humidity)
  tao_imp$humidity[missing_humidity] <- NA
  tao_imp <- impute_lm(tao_imp, humidity ~ year + latitude +
                         sea_surface_temp + air_temp)
  # Calcula MAPC para air_temp y humidity y los incluye a la
  # iteración anterior de MAPC
  diff_air_temp <- c(diff_air_temp, mapc(prev_iter$air_temp, tao_imp$air_temp))
  diff_humidity <- c(diff_humidity, mapc(prev_iter$humidity, tao_imp$humidity))
}
```

¿Cuál es un número suficiente de iteraciones para ejecutar, según las diferencias almacenadas en `diff_air_temp` y `diff_humidity`?

Para responder a esta pregunta, puedemos imprimir los dos vectores en la consola y analizar los números, o trazarlos usando la función proporcionada: simplemente ejecutamos `plot_diffs(diff_air_temp, diff_humidity)` en la consola.

```
plot_diffs <- function(a, b) {
  data.frame("mapc" = c(a, b),
             "Variable" = c(rep("air_temp", length(a)),
                            rep("humidity", length(b))),
             "Iteraciones" = c(1:length(a), 1:length(b))) %>%
    ggplot(aes(Iteraciones, mapc, color = Variable)) +
    geom_line(size = 1.5) +
    ylab("Media del cambio porcentual absoluto (mapc)") +
    ggtitle("Cambio de las variables imputadas entre las iteraciones.") +
    theme(legend.position = "bottom")
}
```

```
plot_diffs(diff_air_temp, diff_humidity)
```



3.2.18 Imputación por regresión logística

Una opción popular para imputar variables binarias es la regresión logística. Desafortunadamente, no hay una función similar a `impute_lm()` que lo haga. Por eso crearemos una función para ello.

Llamemos a la función `impute_logreg()`. Su primer argumento será un data frame `df`, cuyos valores faltantes se han inicializado y solo contiene valores faltantes en la columna a imputar. El segundo argumento será una fórmula para el modelo de regresión logística.

La función hará lo siguiente:

1. Mantendrá las ubicaciones de los valores faltantes.
2. Construirá el modelo.
3. Realizará predicciones.
4. Reemplazará los valores faltantes con las predicciones.

No te preocupes por la línea que crea `imp_var` - esto es solo una forma de extraer el nombre de la columna a imputar de la fórmula.

```
impute_logreg <- function(df, formula) {
  # Extrae el nombre de la variable respuesta
  imp_var <- as.character(formula[2])
  # Guarda los lugares donde la respuesta es missing
  missing_imp_var <- is.na(df[imp_var])
  # Ajusta una regresion del modo logistica
  logreg_model <- glm(formula, data = df, family = binomial)
  # Predice la respuesta y la convierte 0s y 1s
  preds <- predict(logreg_model, type = "response")
```

```
  preds <- ifelse(preds >= 0.5, 1, 0)
  # Imputa los valores missing con las predicciones
  df[missing_imp_var, imp_var] <- preds[missing_imp_var]
  return(df)
}
```

La función está completamente operativa y se puede enchufar en el bucle sobre las variables que viste en la sección previa, al igual que `impute_lm()` del paquete `simputation`. Pronto, combinaremos estos dos para imputar tanto variables continuas como binarias. Pero antes, mejoraremos `impute_logreg()` para que reproduzca mejor la variabilidad en los datos imputados.

3.2.19 Crear una distribución condicional

Simplemente llamar a `predict()` en un modelo siempre devolverá el mismo valor para los mismos valores de los predictores. Esto da como resultado una pequeña variabilidad en los datos imputados. Para aumentarla y que la imputación replique la variabilidad de los datos originales, que podemos extraer de la distribución condicional. Esto significa que en lugar de siempre predecir 1 cuando el modelo devuelve una probabilidad mayor que `0.5` , podemos extraer la predicción de una distribución binomial descrita por la probabilidad devuelta por el modelo.

Trabajaremos en el código del ejercicio anterior. La siguiente línea fue eliminada:

```
preds <- ifelse(preds >= 0.5, 1, 0)
```

Nuestra tarea es llenar su lugar con la creación de una distribución binomial.

```
impute_logreg <- function(df, formula) {
  # Extrae el nombre de la variable respuesta
  imp_var <- as.character(formula[2])
  # Guarda las posiciones donde la respuesta es missing
  missing_imp_var <- is.na(df[imp_var])
  # Ajusta una regresion del modo logistico
  logreg_model <- glm(formula, data = df, family = binomial)
  # Predice la respuesta
  preds <- predict(logreg_model, type = "response")
  # Toma una muestra de las predicciones de la distribución binomial
  # preds <- ifelse(preds >= 0.5, 1, 0)
  preds <- rbinom(length(preds), size = 1, prob = preds)
  # Imputa los valores missing con las predicciones
  df[missing_imp_var, imp_var] <- preds[missing_imp_var]
  return(df)
}
```

Crear la distribución condicional hará que la variabilidad de los datos imputados sea mucho parecida a la del conjunto de datos observados originales. Con esta potente función en nuestras manos, ahora podemos diseñar un flujo de imputación basado en modelos que se encargue tanto de variables continuas como binarias.

3.2.20 Imputación basada en modelos con varios tipos de variables

En este ejercicio, combinaremos lo que hemos aplicado hasta ahora sobre imputación basada en modelos para imputar diferentes tipos de variables en los datos de `tao`.

Nuestra tarea es iterar sobre las variables como lo hemos hecho previamente e imputar dos variables:

`is_hot`, una nueva variable binaria que se creó a partir de `air_temp`, que es 1 si `air_temp` está a 26 grados o más y 0 de lo contrario; `humidity`, una variable continua con la que ya estamos familiarizados.

Tendremos que utilizar la función de regresión lineal que aprendimos antes, así como la función para la regresión logística.

```
tao$is_hot<-ifelse(tao$air_temp>= 26, 1,0)
```

```
# Inicializamos los valores missing con hot-deck
tao_imp <- hotdeck(tao)

# Creamos el indicador booleano desde donde is_hot y humidity son missing
missing_is_hot <- tao_imp$is_hot_imp
missing_humidity <- tao_imp$humidity_imp

for (i in 1:3) {
  # Define is_hot como NA en los lugares donde fue originalmente missing
  # y re-imputa
  tao_imp$is_hot[missing_is_hot] <- NA
  tao_imp <- impute_logreg(tao_imp, is_hot ~ sea_surface_temp)
  # Define humidity como NA en los lugares donde fue originalmente
  # missing y re-imputa
  tao_imp$humidity[missing_humidity] <- NA
  tao_imp <- impute_lm(tao_imp, humidity ~ sea_surface_temp + air_temp)
}
```

3.2.21 Imputación con bosques aleatorios

Un enfoque de aprendizaje automático para la imputación puede ser más preciso y más fácil de implementar en comparación con modelos estadísticos tradicionales. Primero, no requiere que especifiques relaciones entre variables. Además, los modelos de aprendizaje automático como los *random forest* son capaces de descubrir relaciones altamente complejas y no lineales y explotarlas para predecir valores faltantes.

En este ejercicio, usaremos el paquete `missForest`, que construye un bosque aleatorio separado para predecir valores faltantes para cada variable, uno por uno. Llamaremos a la función de imputación sobre los datos de películas, `biopics`, con los que hemos trabajado anteriormente y luego extraeremos los datos completos, así como los errores de imputación estimados.

```
# leemos nuevamente los datos
biopics <-read.csv("curso_imputacion/biopics.csv")
# cargamos la libreria
library(missForest)

# trasformación de character a factor
biopics <- type.convert(biopics, as.is=FALSE)


# imputa los datos de biopics usando missForest
imp_res <- missForest(biopics)

# Extrae los datos imputados y revisa por valores missing
imp_data <- imp_res$ximp
print(sum(is.na(imp_data)))
```

```
## [1] 0
```

```
# Extrae e imprime los errores de imputacion
imp_err <- imp_res$OOBerror
print(imp_err)
```

```
##      NRMSE        PFC
## 0.02049845 0.04299645
```

En el ejercicio anterior hemos extraído los errores de imputación estimados a partir de la salida de `missForest`. Esto te dio dos números:

el error cuadrático medio raíz normalizado (NRMSE) para todas las variables continuas; la proporción de entradas falsamente clasificadas (PFC) para todas las variables categóricas.

Sin embargo, podría darse el caso de que el modelo de imputación funcione muy bien para una variable continua y muy mal para otra. Para diagnosticar tales casos, basta con decirle a `missForest` que produzca estimaciones de error por variable. Esto se hace estableciendo el argumento `variablewise` en `TRUE`.

```
# Imputa los datos de biopics con missForest calculando
# los errores por variable
imp_res <- missForest(biopics, variablewise = TRUE)

# Extrae e imprime los errores de imputacion
per_variable_errors <- imp_res$OOBerror
print(per_variable_errors)
```

```
##         PFC         MSE         MSE         MSE         PFC         PFC
##    0.000000    0.000000 1341.080970    0.000000    0.000000    0.179078
##         MSE         PFC
##    0.000000    0.000000
```

```
# Renombra las columnas para incluir el nombre de las variables
names(per_variable_errors) <- paste(names(biopics),
                                    names(per_variable_errors),
                                    sep = "_")

# Imprime los errores renombrados
print(per_variable_errors)
```

```
##   country_PFC      year_MSE  earnings_MSE   sub_num_MSE  sub_type_PFC
##      0.000000      0.000000   1341.080970      0.000000      0.000000
##  sub_race_PFC non_white_MSE   sub_sex_PFC
##      0.179078      0.000000      0.000000
```

Observa cómo produjimos una serie de medidas de error en lugar de las dos por defecto que habiamos visto antes. Ahora podemos evaluar la calidad de imputación para cada variable por separado. Esto es útil cuando necesitamos saber cómo se desempeña el modelo para una variable en particular que deseamos modelar o analizar más a fondo.

3.2.22 Trade-off velocidad-precisión

En este sentido existen dos paramétros que podemos ajustar para influir en el rendimiento de los bosques aleatorios (*random forest*):

. Número de árboles de decisión en cada bosque. . Número de variables utilizadas para la división dentro de los árboles de decisión.

Aumentar cada uno de ellos puede mejorar la precisión del modelo de imputación, pero también requerirá más tiempo para ejecutarse. En este ejercicio, exploraremos estas ideas ajustando `missForest()` a los datos de `biopics` dos veces con diferentes configuraciones. Mientras seguimos estos pasos, pongamos atención a los errores que imprimiremos y al tiempo que tomará la ejecución del código.

```
# Determina el tiempo inicial del primer enfoque
 t <- proc.time()
```

```
# Define el numero de arboles a 5 y el numero de variables
# usadas para dividir en 2
imp_res <- missForest(biopics, mtry = 2, ntree = 5)
tiempo1<-proc.time() - t
# Imprime los resultados de los errores de la imputacion
print(imp_res$OOBerror)
```

```
##      NRMSE        PFC
## 0.02418617 0.07988166
```

```
# Determina el tiempo inicial del segundo enfoque
 t <- proc.time()
# Define el numero de arboles a 50 y el numero de variables usadas para
# dividir en 6
imp_res <- missForest(biopics, mtry = 6, ntree = 50)
tiempo2<-proc.time() - t
# Imprime los errores resultantes de la imputacion
print(imp_res$OOBerror)
```

```
##      NRMSE        PFC
## 0.02158728 0.04388298
```

Compara los errores y los tiempos de ejecución de los dos modelos de imputación. ¿Puedes ver una relación? Como dicen, "no hay nada gratuito". Para obtener una imputación más precisa, tuvimos que invertir más tiempo de computación.

```
tiempo1
```

```
##    user  system elapsed
##    0.17    0.00    0.17
```

```
tiempo2
```

```
##    user  system elapsed
##    2.45    0.00    2.46
```

3.2.23 La imputación y el modelado en una función

Siempre que realice cualquier análisis o modelado en datos imputados, debe tener en cuenta la incertidumbre de la imputación. Ejecutar un modelo en un conjunto de datos imputados, se ignora el hecho de que la imputación estima los valores faltantes con incertidumbre. Los errores estándar de dicho modelo tienden a ser demasiado pequeños. La solución a esto es la imputación múltiple y una forma de implementarla es mediante bootstrap.

Trabajaremos con los datos `biopics`. El objetivo es utilizar la imputación múltiple mediante bootstrap y la regresión lineal para ver si, en función de los datos disponibles, las películas biográficas con mujeres ganan menos que las de hombres.

Comencemos escribiendo una función que construya una muestra de bootstrap, la impute y ajuste un modelo de regresión lineal.

```
calc_gender_coef <- function(data, indices) {
  # Obtener una muestra bootstrap
  data_boot <- data[indices, ]
  # Imputa con imputacion kNN
  data_imp <- kNN(data_boot, k = 5)
  # Ajusta una regresion lineal
  linear_model <- lm(earnings ~ sub_sex + sub_type + year, data = data_imp)
```

```
  # Extrae y calcula coeficiente para gender
  gender_coefficient <- coef(linear_model)[2]
  return(gender_coefficient)
}
```

La función `calc_gender_coef()` toma los datos y los índices de bootstrap como entradas, y produce nuestra estadística de interés: el impacto del género en las ganancias de la regresión lineal. Ahora podemos usar esta función en el algoritmo de bootstrapping.

3.2.24 Ejecutando bootstrap

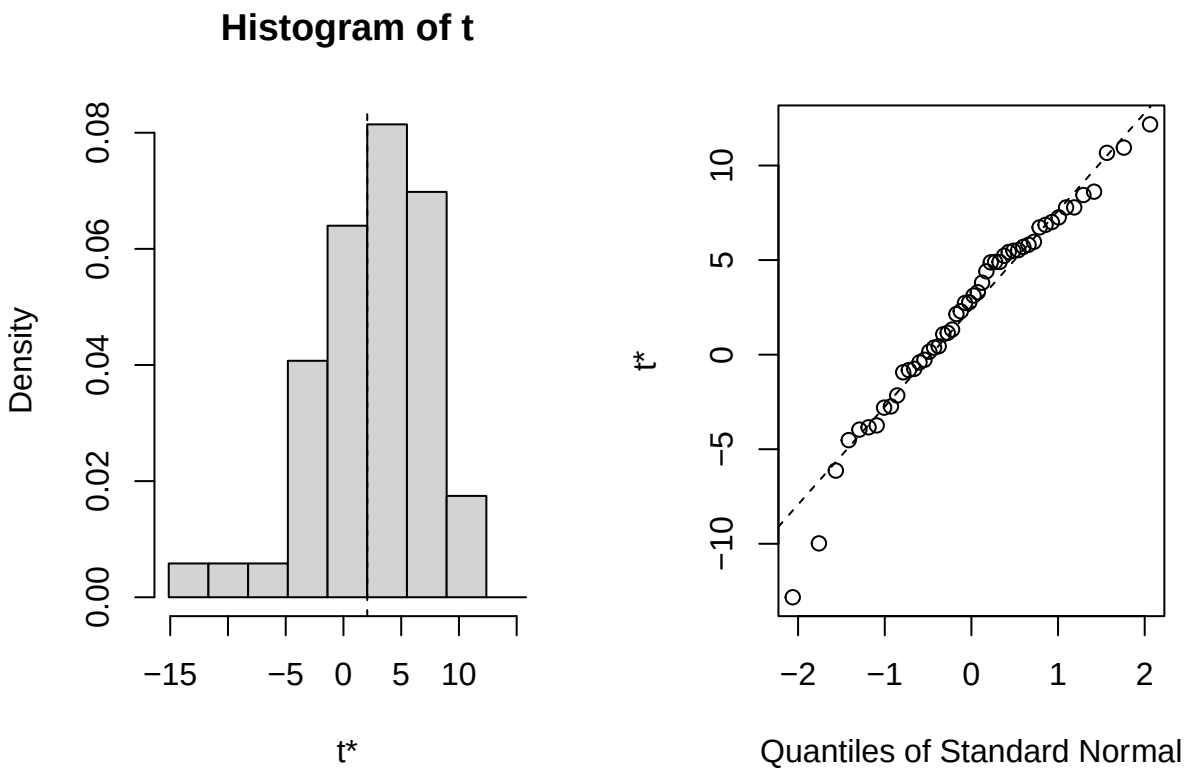
Esta función crea una muestra de bootstrap, la imputa y produce el coeficiente de regresión lineal que describe el impacto de que el tema de la película sea femenino en las ganancias de la película.

En este ejercicio, usarás el paquete `boot` para obtener una distribución de bootstrap de estos coeficientes. La propagación de esta distribución capturará la incertidumbre de la imputación. También verás cómo la distribución de bootstrap difiere de una imputación y regresión.

```
# Carga la libreria boot
library(boot)

# Ejecuta bootstrap sobre los datos biopics
boot_results <- boot(biopics, statistic = calc_gender_coef, R = 50)

# Imprime y grafica los resultados del bootstrapping
print(boot_results)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = biopics, statistic = calc_gender_coef, R = 50)
##
##
## Bootstrap Statistics :
##     original    bias    std. error
## t1* 2.060346 0.3662713    5.175466
```

```
plot(boot_results)
```



Si hubieramos ejecutado la imputación `kNN` y el análisis de regresión en los datos de `biopics` solo una vez, habríamos obtenido un coeficiente de `-1.45` para las películas sobre mujeres (llamado "original" en la salida de la consola), lo que sugiere que las películas sobre mujeres ganan menos. Sin embargo, al corregir la incertidumbre de la imputación, hemos obtenido una distribución que cubre tanto valores negativos como positivos.

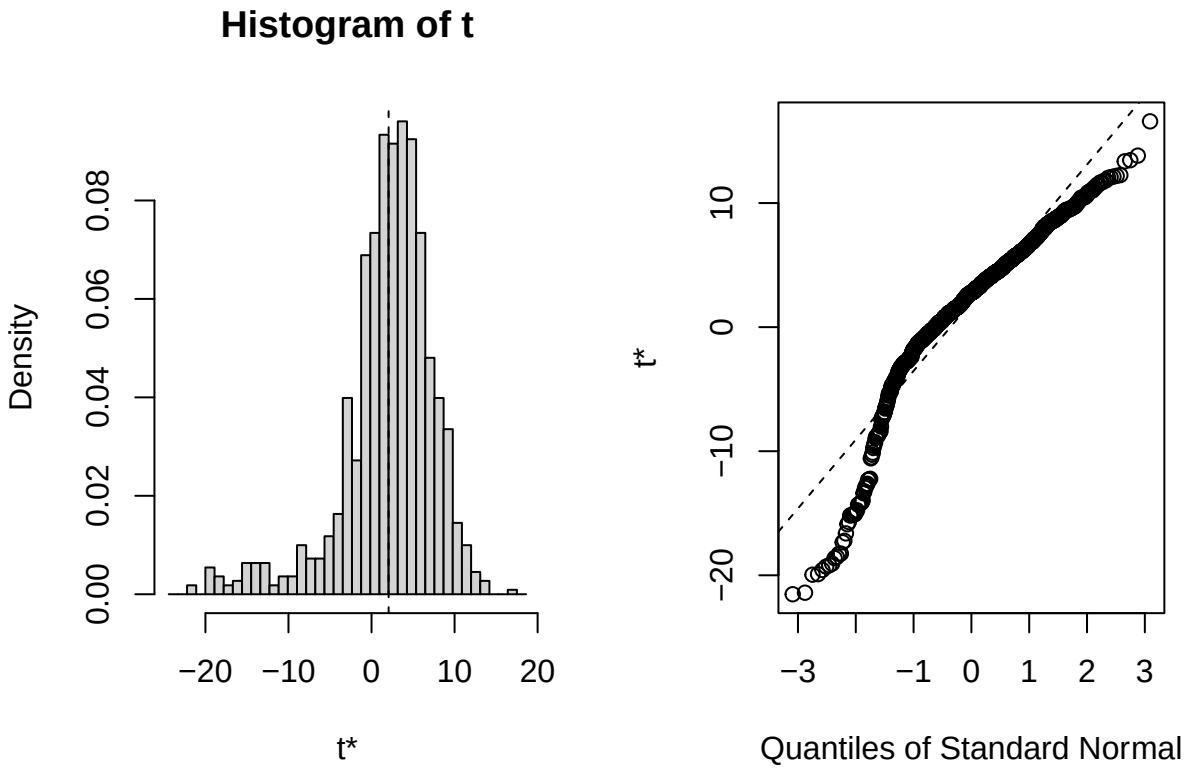
3.2.25 Bootstrapping para intervalos de confianza

Después de haber generado la distribución del coeficiente del efecto femenino en el último ejercicio, ahora podemos usarla para estimar un intervalo de confianza. Esto permitirá hacer la siguiente evaluación sobre los datos: "Dada la incertidumbre de la imputación, estamos 95% seguros de que el efecto femenino en las ganancias se encuentra entre `a` y `b`", donde `a` y `b` son los límites inferior y superior del intervalo.

En el último ejercicio, ejecutamos la técnica de bootstrapping con `R = 50` réplicas. Sin embargo, en la mayoría de las aplicaciones esto no es suficiente. En este ejercicio, puedes utilizar los `boot_results` que se prepararon utilizando 1000 réplicas. Primero, verás si la distribución de bootstrapping parece normal. Si es así, entonces podrás confiar en la distribución normal para calcular el intervalo de confianza.

```
# Ejecuta bootstrap sobre los datos biopics y mide el tiempo de ejecucion
boot_results <- boot(biopics, statistic = calc_gender_coef, R = 1000)
```

```
# Plot and print boot_results
plot(boot_results)
```




```
print(boot_results)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = biopics, statistic = calc_gender_coef, R = 1000)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 2.060346 -0.04651124    5.542859
```

```
# Calculate and print confidence interval
boot_ci <- boot.ci(boot_results, conf = 0.95, type = "norm")
print(boot_ci)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_results, conf = 0.95, type = "norm")
##
## Intervals :
## Level      Normal
## 95%   (-8.757, 12.971 )
```

Calculations and Intervals on Original Scale

A pesar de que la tendencia general parece ser una relación negativa, las réplicas de bootstrap muestran que algunas películas con protagonistas femeninas en realidad ganan más. Al tener en cuenta la incertidumbre de la imputación, no se puede estar al 100% seguro acerca de la dirección de esta relación, aunque un análisis único sugiera lo contrario.

3.2.26 El flujo de MICE: mice - with - pool

El flujo de `MICE` (imputación múltiple por ecuaciones encadenadas) nos permite estimar la incertidumbre de la imputación mediante la imputación de un conjunto de datos varias veces mediante la imputación basada en modelos, mientras se extrae de las distribuciones condicionales. De esta manera, cada conjunto de datos imputados es ligeramente diferente. Luego, se realiza un análisis en cada uno de ellos y se combinan los resultados, obteniendo las cantidades de interés junto con sus intervalos de confianza que reflejan la incertidumbre de la imputación.

En este ejercicio, practicaremos el flujo típico de la imputación con `MICE`: `mice() - with() - pool()`. Realizaremos un análisis de regresión en los datos de `biopics` para ver qué tipo de ocupación de sujeto, `sub_type`, está asociada con mayores ingresos en las películas.

```
# Carga el paquete mice
library(mice)

# Imputa biopics con mice usando 5 imputaciones
biopics_multiimp <- mice(biopics, m = 5, seed = 3108)
```

```
##
##  iter imp variable
##   1   1  earnings  sub_race
##   1   2  earnings  sub_race
##   1   3  earnings  sub_race
##   1   4  earnings  sub_race
##   1   5  earnings  sub_race
##   2   1  earnings  sub_race
##   2   2  earnings  sub_race
##   2   3  earnings  sub_race
##   2   4  earnings  sub_race
##   2   5  earnings  sub_race
##   3   1  earnings  sub_race
##   3   2  earnings  sub_race
##   3   3  earnings  sub_race
##   3   4  earnings  sub_race
##   3   5  earnings  sub_race
##   4   1  earnings  sub_race
##   4   2  earnings  sub_race
##   4   3  earnings  sub_race
##   4   4  earnings  sub_race
##   4   5  earnings  sub_race
##   5   1  earnings  sub_race
##   5   2  earnings  sub_race
##   5   3  earnings  sub_race
##   5   4  earnings  sub_race
##   5   5  earnings  sub_race
```

```
# Ajusta una regresion lineal para cada set de datos imputados
lm_multiimp <- with(biopics_multiimp, lm(earnings ~ year + sub_type))
```

```r
# Combina las estimaciones por las reglas de Rubin (pool)
lm_pooled <- pool(lm_multiimp)
summary(lm_pooled, conf.int = TRUE, conf.level = 0.95)
```

```
##                              term     estimate  std.error   statistic
## 1                     (Intercept) -287.8866750 490.062824 -0.58744851
## 2                            year    0.1612176   0.239277  0.67376974
## 3  sub_typeAcademic (Philosopher)  -32.8423364  39.593926 -0.82947915
## 4                sub_typeActivist  -17.4281787  16.052579 -1.08569339
## 5                   sub_typeActor  -30.7740287  19.378762 -1.58802862
## 6                 sub_typeActress  -27.3033456  21.249448 -1.28489670
## 7      sub_typeActress / activist   16.0962907  39.192849  0.41069458
## 8                  sub_typeArtist  -27.4779956  18.148136 -1.51409465
## 9                 sub_typeAthlete   -4.2336944  12.503822 -0.33859202
## 10     sub_typeAthlete / military   79.1943734  38.055447  2.08102595
## 11                 sub_typeAuthor  -23.6540827  18.471621 -1.28056347
## 12          sub_typeAuthor (poet)  -23.4506193  19.985477 -1.17338304
## 13               sub_typeComedian  -22.9330598  21.576033 -1.06289508
## 14               sub_typeCriminal   -2.6478096  16.754147 -0.15803906
## 15             sub_typeGovernment   -5.0221576  21.879188 -0.22954040
## 16             sub_typeHistorical   -9.3734433  19.183957 -0.48860845
## 17             sub_typeJournalist  -26.5071032  29.005345 -0.91386960
## 18                  sub_typeMedia  -11.5302020  26.461566 -0.43573393
## 19               sub_typeMedicine    4.7788761  15.006237  0.31845932
## 20               sub_typeMilitary   10.9417685  19.325322  0.56618816
## 21    sub_typeMilitary / activist   37.2248141  39.752579  0.93641257
## 22               sub_typeMusician  -19.7931797  17.320867 -1.14273611
## 23                  sub_typeOther  -15.5895605  16.044509 -0.97164460
## 24             sub_typePolitician  -13.6751859  39.752579 -0.34400752
## 25                sub_typeSinger    0.6677979  17.844592  0.03742298
## 26               sub_typeTeacher   51.1575084  39.268925  1.30274786
## 27          sub_typeWorld leader    5.9976436  16.882980  0.35524793
##            df    p.value        2.5 %       97.5 %
## 1    3.983993 0.58858265 -1650.677928 1074.9045785
## 2    4.030421 0.53712441    -0.501149    0.8235843
## 3  139.785256 0.40824758  -111.122704   45.4380311
## 4   10.576004 0.30174143   -52.933253   18.0768961
## 5   10.847500 0.14097812   -73.499669   11.9516115
## 6    7.876739 0.23532207   -76.438456   21.8317642
## 7  169.869958 0.68181408   -61.271473   93.4640548
## 8    8.280382 0.16719720   -69.082290   14.1262992
## 9   15.246283 0.73953513   -30.847513   22.3801244
## 10 336.810947 0.03818668     4.338081  154.0506662
## 11   7.047583 0.24087818   -67.272814   19.9646489
## 12  10.612273 0.26630514   -67.635233   20.7339945
## 13  16.686775 0.30297041   -68.519689   22.6535693
## 14   7.275709 0.87872330   -41.962758   36.6671385
## 15  81.573759 0.81902349   -48.550237   38.5059216
## 16   6.172715 0.64199292   -55.998343   37.2514567
## 17 113.284343 0.36272632   -83.970362   30.9561559
## 18   6.128774 0.67795894   -75.950965   52.8905609
## 19 247.938063 0.75040464   -24.777080   34.3348320
## 20   6.645835 0.58986247   -35.253693   57.1372305
## 21 130.043524 0.35079655   -41.420661  115.8702894
```

```
## 22   6.996299 0.29074124   -60.754913   21.1685537
## 23   7.168854 0.36286255   -53.348394   22.1692725
## 24 130.043524 0.73139625   -92.320661   64.9702894
## 25  10.377953 0.97085786   -38.897026   40.2326216
## 26 163.415623 0.19449369   -26.382404  128.6974204
## 27  14.000056 0.72769899   -30.212733   42.2080205
```

En este caso, hemos seguido el flujo "mice-with-pool" para imputar, modelar y agrupar los resultados. Ahora, echemos un vistazo a la salida en la consola: algunos `sub_types` tienen un impacto positivo en las ganancias. Sin embargo, al tener en cuenta la incertidumbre de la imputación con una confianza del 95%, nunca estamos seguros de estos efectos, ya que los límites inferiores son negativos. Con una excepción: para `sub_typeAthlete / military`, tanto los límites inferiores como los superiores son positivos. Lo que podemos decir con seguridad es que las películas sobre atletas militares son populares.

3.2.26.1 Selección de modelos por defecto MICE crea un modelo de imputación separado para cada variable en los datos. El tipo de modelo depende del tipo de variable en cuestión. Una forma popular de especificar los tipos de modelos que queremos usar es establecer un modelo predeterminado para cada uno de los cuatro tipos de variables.

Podemos hacer esto usando el argumento defaultMethod en la función `mice()`, que debe ser un vector de longitud 4 que contenga los métodos de imputación predeterminados para:

Variables continuas, Variables binarias, Variables categóricas (factores no ordenados), Variables factoriales (factores ordenados).

En este caso, aprovecharemos la documentación de `mice` para ver la lista de métodos disponibles y seleccionar los deseados para que el algoritmo los use.

```
# Imputa biopics usando los metodos especificados en la instruccion
biopics_multiimp <- mice(biopics, m = 20,
                         defaultMethod = c("cart", "lda", "pmm", "polr"))
```

```
##
##  iter imp variable
##   1   1  earnings  sub_race
##   1   2  earnings  sub_race
##   1   3  earnings  sub_race
##   1   4  earnings  sub_race
##   1   5  earnings  sub_race
##   1   6  earnings  sub_race
##   1   7  earnings  sub_race
##   1   8  earnings  sub_race
##   1   9  earnings  sub_race
##   1   10  earnings  sub_race
##   1   11  earnings  sub_race
##   1   12  earnings  sub_race
##   1   13  earnings  sub_race
##   1   14  earnings  sub_race
##   1   15  earnings  sub_race
##   1   16  earnings  sub_race
##   1   17  earnings  sub_race
##   1   18  earnings  sub_race
##   1   19  earnings  sub_race
##   1   20  earnings  sub_race
##   2   1  earnings  sub_race
##   2   2  earnings  sub_race
##   2   3  earnings  sub_race
```

```
##   2   4  earnings  sub_race
##   2   5  earnings  sub_race
##   2   6  earnings  sub_race
##   2   7  earnings  sub_race
##   2   8  earnings  sub_race
##   2   9  earnings  sub_race
##   2   10  earnings  sub_race
##   2   11  earnings  sub_race
##   2   12  earnings  sub_race
##   2   13  earnings  sub_race
##   2   14  earnings  sub_race
##   2   15  earnings  sub_race
##   2   16  earnings  sub_race
##   2   17  earnings  sub_race
##   2   18  earnings  sub_race
##   2   19  earnings  sub_race
##   2   20  earnings  sub_race
##   3   1  earnings  sub_race
##   3   2  earnings  sub_race
##   3   3  earnings  sub_race
##   3   4  earnings  sub_race
##   3   5  earnings  sub_race
##   3   6  earnings  sub_race
##   3   7  earnings  sub_race
##   3   8  earnings  sub_race
##   3   9  earnings  sub_race
##   3   10  earnings  sub_race
##   3   11  earnings  sub_race
##   3   12  earnings  sub_race
##   3   13  earnings  sub_race
##   3   14  earnings  sub_race
##   3   15  earnings  sub_race
##   3   16  earnings  sub_race
##   3   17  earnings  sub_race
##   3   18  earnings  sub_race
##   3   19  earnings  sub_race
##   3   20  earnings  sub_race
##   4   1  earnings  sub_race
##   4   2  earnings  sub_race
##   4   3  earnings  sub_race
##   4   4  earnings  sub_race
##   4   5  earnings  sub_race
##   4   6  earnings  sub_race
##   4   7  earnings  sub_race
##   4   8  earnings  sub_race
##   4   9  earnings  sub_race
##   4   10  earnings  sub_race
##   4   11  earnings  sub_race
##   4   12  earnings  sub_race
##   4   13  earnings  sub_race
##   4   14  earnings  sub_race
##   4   15  earnings  sub_race
##   4   16  earnings  sub_race
##   4   17  earnings  sub_race
```

```
##   4   18  earnings  sub_race
##   4   19  earnings  sub_race
##   4   20  earnings  sub_race
##   5   1  earnings  sub_race
##   5   2  earnings  sub_race
##   5   3  earnings  sub_race
##   5   4  earnings  sub_race
##   5   5  earnings  sub_race
##   5   6  earnings  sub_race
##   5   7  earnings  sub_race
##   5   8  earnings  sub_race
##   5   9  earnings  sub_race
##   5   10  earnings  sub_race
##   5   11  earnings  sub_race
##   5   12  earnings  sub_race
##   5   13  earnings  sub_race
##   5   14  earnings  sub_race
##   5   15  earnings  sub_race
##   5   16  earnings  sub_race
##   5   17  earnings  sub_race
##   5   18  earnings  sub_race
##   5   19  earnings  sub_race
##   5   20  earnings  sub_race
```

```
# Imprime biopics_multiimp
print(biopics_multiimp)
```

```
## Class: mids
## Number of multiple imputations:  20
## Imputation methods:
##   country      year  earnings   sub_num  sub_type  sub_race non_white   sub_sex
##        ""        ""    "cart"        ""        ""     "pmm"        ""        ""
## PredictorMatrix:
##          country year earnings sub_num sub_type sub_race non_white sub_sex
## country        0    1        1       1        1        1         1       1
## year           1    0        1       1        1        1         1       1
## earnings       1    1        0       1        1        1         1       1
## sub_num        1    1        1       0        1        1         1       1
## sub_type       1    1        1       1        0        1         1       1
## sub_race       1    1        1       1        1        0         1       1
## Number of logged events:  200
##   it im      dep meth                          out
## 1  1  1 earnings cart sub_typeAcademic (Philosopher)
## 2  1  1 sub_race  pmm    sub_typeMilitary / activist
## 3  1  2 earnings cart sub_typeAcademic (Philosopher)
## 4  1  2 sub_race  pmm    sub_typeMilitary / activist
## 5  1  3 earnings cart sub_typeAcademic (Philosopher)
## 6  1  3 sub_race  pmm    sub_typeMilitary / activist
```

La capacidad de especificar modelos de imputación puede resultar útil cuando se observa que algunos métodos específicos no funcionan bien. Otro factor que influye en cómo funcionan los métodos de imputación es el conjunto de predictores que utilizan. En el siguiente ejercicio, veremos cómo establecer estos predictores.

3.2.26.2 Usando una matriz predictora Se trata de tomar decisiones importantes cuando se utiliza la imputación basada en modelos, como por ejemplo, qué variables deben incluirse como predictores y en qué

modelos. En `mice()`, esto se rige por la matriz de predictores y, por defecto, todas las variables se utilizan para imputar todas las demás.

En caso de tener muchas variables en los datos o poco tiempo para realizar una selección adecuada del modelo, puede utilizar la funcionalidad de `mice` para crear una matriz de predictores basada en las correlaciones entre las variables. Esta matriz se puede incorporar a `mice()`. En este ejercicio, practicaremos exactamente esto: primero construiremos una matriz de predictores de modo que cada variable se imputa utilizando las variables más correlacionadas con ella; luego, usará una matriz de predictores con la función de imputación.

```
# Crea una matriz predictora con correlacion minima de 0.1
pred_mat <- quickpred(biopics, mincor = 0.1)

# Imputa biopics con mice
biopics_multiimp <- mice(biopics,
                         m = 10,
                         predictorMatrix = pred_mat,
                         seed = 3108)
```

```
##
##  iter imp variable
##   1   1  earnings  sub_race
##   1   2  earnings  sub_race
##   1   3  earnings  sub_race
##   1   4  earnings  sub_race
##   1   5  earnings  sub_race
##   1   6  earnings  sub_race
##   1   7  earnings  sub_race
##   1   8  earnings  sub_race
##   1   9  earnings  sub_race
##   1   10  earnings  sub_race
##   2   1  earnings  sub_race
##   2   2  earnings  sub_race
##   2   3  earnings  sub_race
##   2   4  earnings  sub_race
##   2   5  earnings  sub_race
##   2   6  earnings  sub_race
##   2   7  earnings  sub_race
##   2   8  earnings  sub_race
##   2   9  earnings  sub_race
##   2   10  earnings  sub_race
##   3   1  earnings  sub_race
##   3   2  earnings  sub_race
##   3   3  earnings  sub_race
##   3   4  earnings  sub_race
##   3   5  earnings  sub_race
##   3   6  earnings  sub_race
##   3   7  earnings  sub_race
##   3   8  earnings  sub_race
##   3   9  earnings  sub_race
##   3   10  earnings  sub_race
##   4   1  earnings  sub_race
##   4   2  earnings  sub_race
##   4   3  earnings  sub_race
##   4   4  earnings  sub_race
##   4   5  earnings  sub_race
```

```
##   4   6  earnings  sub_race
##   4   7  earnings  sub_race
##   4   8  earnings  sub_race
##   4   9  earnings  sub_race
##   4   10  earnings  sub_race
##   5   1  earnings  sub_race
##   5   2  earnings  sub_race
##   5   3  earnings  sub_race
##   5   4  earnings  sub_race
##   5   5  earnings  sub_race
##   5   6  earnings  sub_race
##   5   7  earnings  sub_race
##   5   8  earnings  sub_race
##   5   9  earnings  sub_race
##   5   10  earnings  sub_race
```

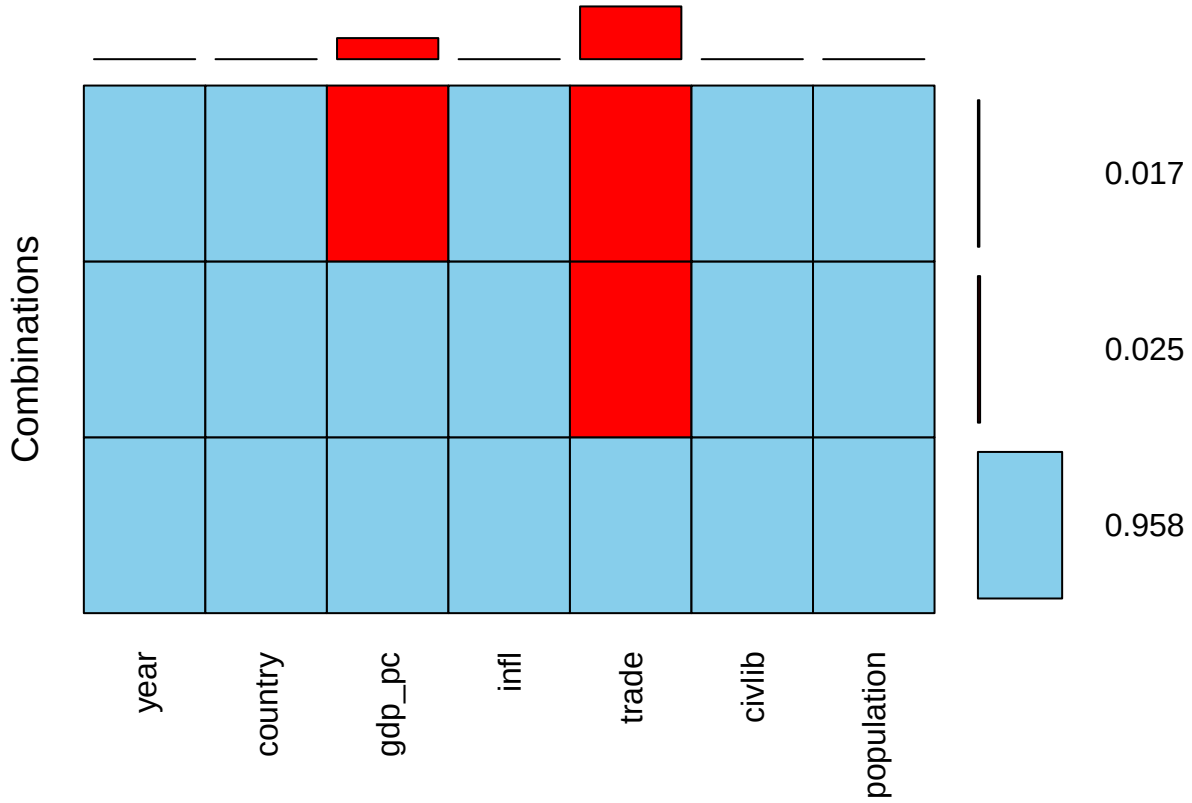
```
# Imprime biopics_multiimp
# print(biopics_multiimp)
```

3.2.27 Analizando los patrones de datos faltantes

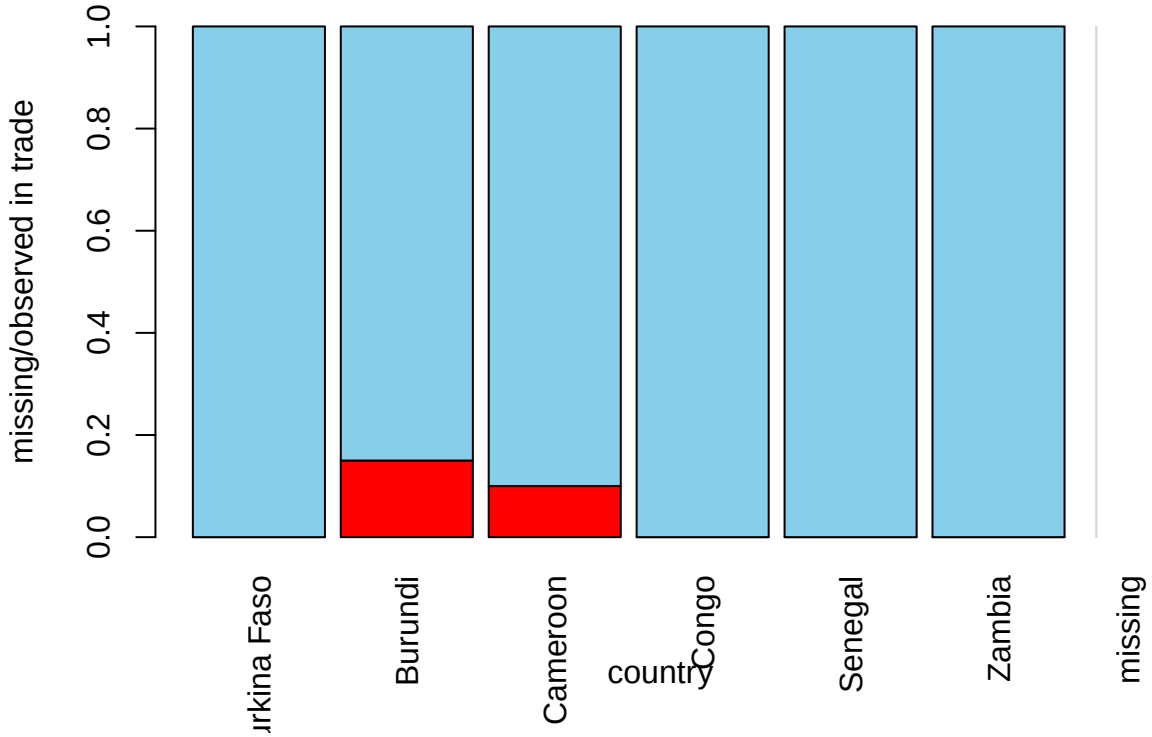
El primer paso para trabajar con datos incompletos es obtener información sobre los patrones de ausencia de datos, y una buena manera de hacerlo es mediante visualizaciones. Comenzarás tu análisis de los datos de `África` empleando el paquete `VIM` para crear dos visualizaciones: el gráfico de agregación y el gráfico de Mosaico. Te dirán cuántos datos faltan, en qué variables y configuraciones, y si podemos decir algo sobre el mecanismo de ausencia de datos. ¡Comencemos con algunas gráficas!

```
africa <- read.csv("handing/africa.csv", sep = ";")
```

```
# carga el paquete VIM
library(VIM)



# Crea un grafico de agregación combinada del set de datos africa
africa %>%
  aggr(combined = TRUE, numbers = TRUE)
```




```
# Crea un grafico spine plot de pais vs trade
africa %>%
  select(country, trade) %>%
  spineMiss()
```



Observamos que no hay tantos valores faltantes. Además, observe en el gráfico de Mosaico para los datos de `africa` parecen ser MAR - al menos con respecto al PIB y al país, lo que significa que se pueden imputar.

3.2.28 Imputando e inspeccionando resultados

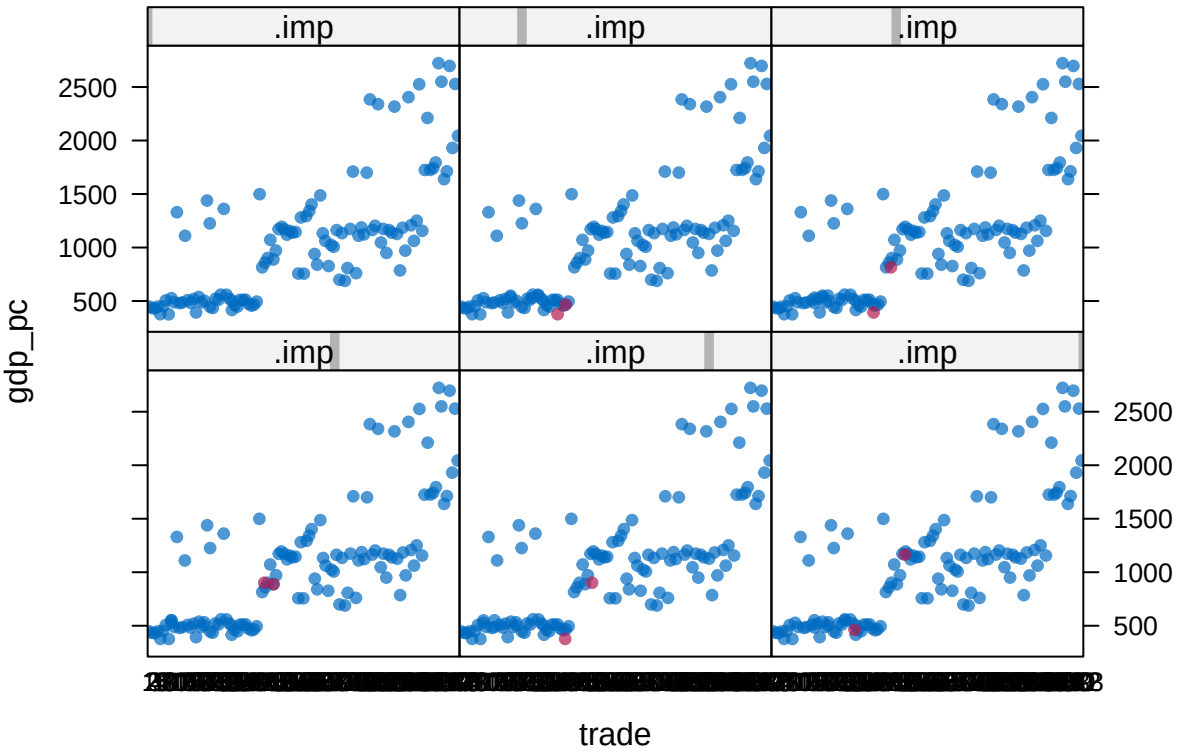
Hemos descubierto que hay algunos datos faltantes en el PIB, `gdp_pc`, y en `trade` como porcentaje del PIB. Además, se sospecha que los datos son MAR, por lo que es posible que sean imputados. En este caso, haremos uso de la imputación múltiple del paquete `mice` para imputar los datos de `africa`. Luego, crearemos un gráfico para `gdp_pc` vs `trade` para ver si los datos imputados no rompen la relación entre estas variables.

```
# Carga mice
library(mice)

# Imputa africa con mice
africa_multiimp <- mice(africa, m = 5, defaultMethod = "cart", seed = 3108)
```

```
##
##  iter imp variable
##   1   1  gdp_pc  trade
##   1   2  gdp_pc  trade
##   1   3  gdp_pc  trade
##   1   4  gdp_pc  trade
##   1   5  gdp_pc  trade
##   2   1  gdp_pc  trade
##   2   2  gdp_pc  trade
##   2   3  gdp_pc  trade
##   2   4  gdp_pc  trade
```

```
##    2   5  gdp_pc  trade
##    3   1  gdp_pc  trade
##    3   2  gdp_pc  trade
##    3   3  gdp_pc  trade
##    3   4  gdp_pc  trade
##    3   5  gdp_pc  trade
##    4   1  gdp_pc  trade
##    4   2  gdp_pc  trade
##    4   3  gdp_pc  trade
##    4   4  gdp_pc  trade
##    4   5  gdp_pc  trade
##    5   1  gdp_pc  trade
##    5   2  gdp_pc  trade
##    5   3  gdp_pc  trade
##    5   4  gdp_pc  trade
##    5   5  gdp_pc  trade
```

```
# Crea un stripplot of gdp_pc versus trade
stripplot(africa_multiimp, gdp_pc ~ trade | .imp, pch = 20, cex = 1)
```



Se observa que la imputación funciona bien: hay pequeños grupos en los gráficos de dispersión, que probablemente corresponden a diferentes países. Cada punto de datos imputado encaja en uno de los grupos, en lugar de ser un valor atípico en algún lugar entre los grupos. Después de haber realizado la imputación, podemos proceder con el modelado.

3.2.28.1 Inferencia con datos imputados En este último caso, hemos utilizado `mice` para imputar los datos de `africa`. En este, implementaremos los otros dos pasos del flujo de "mice - with - pool" que

hemos usado anteriormente. El modelo de interés es una regresión lineal que explica el PIB, `gdp_pc`, con otras variables. Nos interesa particularmente el coeficiente de libertades civiles, `civlib`. ¿Está asociado tener valores más altos en `civlib` en función a un mayor crecimiento económico una vez que incorporamos la incertidumbre de la imputación?

```
# Ajusta im regresion lineal a cada data set  imputado
lm_multiimp <- with(africa_multiimp, lm(gdp_pc ~ country +
                                        year + trade + infl + civlib))

# Combina las estimaciones por las reglas de Rubin (pool)
lm_pooled <- pool(lm_multiimp)

# Summarize pooled results
summary(lm_pooled, conf.int = TRUE, conf.level = 0.9)
```

```
##              term      estimate   std.error  statistic        df      p.value
## 1     (Intercept) -31703.576601 6031.455164 -5.2563728  92.53952 9.387968e-07
## 2   countryBurundi     63.739453   65.610134  0.9714879 103.18378 3.335772e-01
## 3  countryCameroon    622.343351   66.226798  9.3971530  56.28736 3.941785e-13
## 4     countryCongo   1303.940067  119.821885 10.8823197 101.65641 9.508463e-19
## 5   countrySenegal    516.442634   82.535746  6.2571995 106.90957 8.275032e-09
## 6    countryZambia    396.326840   87.729106  4.5176209 106.82235 1.620019e-05
## 7             year     16.156955    3.036095  5.3216230  92.06641 7.199791e-07
## 8            trade      5.468655    1.663523  3.2873939 105.04858 1.375742e-03
## 9             infl     -4.389418    1.031399 -4.2557883 107.91316 4.455825e-05
## 10          civlib    -84.852311  147.925470 -0.5736153  66.50333 5.681630e-01
##               5 %          95 %
## 1  -41724.760108 -21682.393094
## 2     -45.157324    172.636231
## 3     511.587065    733.099637
## 4    1105.037965   1502.842168
## 5     379.496718    653.388550
## 6     250.762899    541.890781
## 7      11.112260     21.201650
## 8       2.708058      8.229252
## 9      -6.100609     -2.678226
## 10   -331.605424    161.900803
```

Basándose en el resumen de los resultados de la regresión agrupada que acabamos de imprimir. Podemos decir que, dado que los límites inferior y superior tienen signos diferentes, no podemos estar seguros de la dirección del efecto.