


Sesión 4

Curso: R Aplicado a los Proyectos de Investigación

Percy Soto-Becerra, M.D., M.Sc(c)

INKASTATS DATA SCIENCE SOLUTIONS | MEDICAL BRANCH

2022-10-10

 <https://github.com/psotob91>



Agenda

1. Manejo de datos 2: Categorización y etiquetado de variables
2. AID / AED en R

Creación de variables con `case_when()`

- Función de apoyo a `mutate()` para crear variables según condiciones más complejas.
- Crea variables de acuerdo a condiciones complejas

```
1 case_when(  
2   condición1 ~ resultado1,  
3   condición2 ~ resultado2,  
4   condición3 ~ resultado3  
5 )
```

- Toda transformación o creación de variables en dplyr ocurre por `mutate`; por tanto, `case_when()` se utiliza dentro de un `mutate`

```
1 datos %>%  
2   mutate(  
3     nueva_var = case_when(  
4       condición1 ~ resultado1,  
5       condición2 ~ resultado2,  
6       condición3 ~ resultado3  
7     )  
8   )
```

- Atajo de teclado para obtener `~`: `Alt + 1 + 2 + 6`

Condición no cumplida en `case_when()`

- Para indicar que todas las condiciones previas no se cumplen, se debe colocar **TRUE**:

```
1 datos %>%  
2   mutate(  
3     nueva_var = case_when(  
4       condición1 ~ resultado1,  
5       condición2 ~ resultado2,  
6       condición3 ~ resultado3,  
7       TRUE ~ resultado_si_condición_no_se_cumple  
8     )  
9   )
```

- Es muy importante siempre colocarla al final de cualquier conjunto de condiciones previas.
- Cuando el resultado de no cumplirse es que se asigne valor perdido, es importante usar la función `as.tipo_var()` sobre el indicador de dato perdido **NA**.
 - Si es character: `as.character(NA)`
 - Si es numeric: `as.numeric(NA)`

case_when() en acción

Categorizar variable numérica

Variable según condición compleja

- Se quieren crear categorías de edad: “20-30”, “31-35” y “36-41”

```
1 datos_fase1 %>%
2   select(id, age) %>%
3   mutate(agecat = case_when(age >= 20 & age <= 30 ~ "20-30",
4                             age >= 31 & age <= 35 ~ "31-35",
5                             age >= 36 & age <= 41 ~ "36-41",
6                             TRUE ~ as.character(NA))
7   )
```

```
# A tibble: 106 × 3
   id   age agecat
  <dbl> <dbl> <chr>
1     1    33 31-35
2     1    32 31-35
3     2    27 20-30
4     2    27 20-30
5     3    25 20-30
6     3    25 20-30
7     4    37 36-41
8     4    38 36-41
9     5    31 31-35
10    5    32 31-35
# ... with 96 more rows
```

- Se desea crear variable indicadora de inclusión en estudio: Solo pacientes sin pareja y que proceden de Callao, Carabayllo, Chorrillos o SJL son elegibles:

```
1 datos_fase1 %>%
2   mutate(elegible = case_when(
3     married2 == "Without couple" & procedence %in% c("Callao", "Carabayllo", "Chorrillos", "SJL") ~ "Elegible",
4     TRUE ~ "No elegible")
5   )
```

```
5 )
6 )
```

```
# A tibble: 106 × 15
  id time      treat    age race married marri...1 proce...2 weight height    e2
  <dbl> <fct>    <fct>  <dbl> <chr> <fct>    <fct>    <chr>    <dbl> <dbl> <dbl>
1     1 1 Baseline Place...   33 Mest... Single Withou... Callao    59    1.4  87.3
2     2 1 3 months Place...   32 Mest... Single Withou... Callao   59.9    1.3  210.
3     3 2 Baseline Dosis...   27 Mest... Single Withou... Santa ...   62    1.5  169.
4     4 2 3 months Dosis...   27 Mest... Single Withou... Santa ...  62.1    1.6   99.9
5     5 3 Baseline Dosis...   25 Mest... Single Withou... Callao    62    1.6   78.8
6     6 3 3 months Dosis...   25 Mest... Single Withou... Callao    60    1.6  155.
7     7 4 Baseline Dosis...   37 Mest... Divorc... Withou... Callao   60.9    1.5   41.0
8     8 4 3 months Dosis...   38 Mest... Divorc... Withou... Callao   61.4    1.5  109.
9     9 5 Baseline Place...   31 Mest... Single Withou... La Mol...   64    1.5   43.0
10    10 5 3 months Place...   32 Mest... Single Withou... La Mol...  58.1    1.6   56.0
# ... with 96 more rows, 4 more variables: lh <dbl>, fsh <dbl>, prog <dbl>,
#   eligible <chr>, and abbreviated variable names 1married2, 2procedence
```

Etiquetar variables con `set_var_labels()`

- La función `set_var_labels()` del paquete `labelled()` es muy útil para etiquetar columnas.
- Los datos deben tener metadatos que permitan ser legibles por el ser humano.
- Primero instalar y cargar paquete:

```
1 library(labelled)
```

- Es preferible usar esta función al final de todo el proceso de limpieza de datos.

Larga

```
1 set_var_labels(.data = DATA, ...)
```

Abreviada

```
1 set_var_labels(DATA, ...)
```

Se estila usar `%>%`

```
1 DATA %>%  
2   set_var_labels(...)
```

Argumento	Descripción
-----------	-------------

<code>.data</code>	Data frame o extensión de data frame (por ejemplo, tibble).
--------------------	---

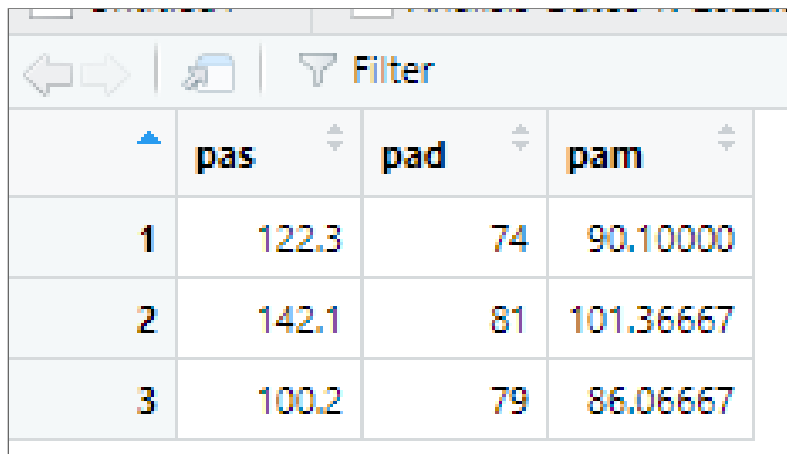
<code>...</code>	<code>variable = 'nueva etiqueta'</code>
------------------	--

set_var_labels() en acción

Sin etiqueta de variable

Con etiqueta de variable




- ¿Qué significan **pas**, **pad** y **pam**?
- Difícil de saber sin una etiqueta



	pas	pad	pam
1	122.3	74	90.10000
2	142.1	81	101.36667
3	100.2	79	86.06667

- Usando `set_variable_labels()`

```
1 datos2 <- datos %>%  
2   set_variable_labels(  
3     pam = "Presión Arterial Media",  
4     pas = "Presión Arterial Sistólica",  
5     pad = "Presión Arterial Diastólica"  
6   )
```


	   Filter		
	pas Presión Arterial Sistólica	pad Presión Arterial Diastólica	pam Presión Arterial Media
1	122.3	74	90.10000
2	142.1	81	101.36667
3	100.2	79	86.06667

Nuestro turno

- Descargue la carpeta denominada [taller04](#) disponible en la carpeta compartida.
- Abra el proyecto denominado [taller04.Rproj](#)
- Complete y ejecute el código faltante en los chunk de código de la PRIMERA PARTE.
- Una vez culmine todo el proceso, renderice el archivo .qmd.

Hagamos una pausa

Tomemos un descanso de 5 minutos...

Estire las piernas ...

Deje de ver las pantallas ...

... cualquier , las del celular también.

05:00

Agenda

1. Manejo de datos 2: Categorización y etiquetado de variables
2. **AID / AED en R**

Paso 1: Resumen global de los datos

¿Qué debo inspeccionar de manera global?

[glimpse\(\)](#) [skim\(\)](#) [describe\(\)](#)

- Dimensiones: columnas y filas
- Variables y tipos
- Datos completos y faltantes
- Variables numéricas: Mínimos, máximos y valores extremos
- Variables categóricas: Valores o categorías muy poco frecuentes y datos perdidos encubiertos
- Heche un vistazo de los datos con [glimpse\(\)](#):

```
1 glimpse(datos)
```

```
Rows: 3  
Columns: 3  
$ pas <dbl> 122.3, 142.1, 100.2  
$ pad <dbl> 74, 81, 79  
$ pam <dbl> 90.10, 101.37, 86.06
```

- La función [skim\(\)](#) del paquete [{skimr}](#) genera un resumen global de los datos:

```
1 skim(datos)
```

Data summary

Name	datos
Number of rows	3
Number of columns	3

Column type frequency:

numeric 3

Group variables None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75
pas	0	1	121.53	20.96	100.20	111.25	122.3	132.20
pad	0	1	78.00	3.61	74.00	76.50	79.0	80.00
pam	0	1	92.51	7.93	86.06	88.08	90.1	95.74

- La función `describe()` del paquete `{Hmisc}` genera un reporte general bien detallado, variable por variable:

```
1 describe(datos)
```

datos

3 Variables 3 Observations

pas

n	missing	distinct	Info	Mean	Gmd
3	0	3	1	121.5	27.93

Value 100.2 122.3 142.1

Frequency 1 1 1

Proportion 0.333 0.333 0.333

pad

n	missing	distinct	Info	Mean	Gmd
3	0	3	1	78	4.667

Paso 2: Detecte y maneje duplicados

Filas duplicadas

ID duplicados

Elimine duplicados

Deduplicación probabilística

- La función `get_dupes()` del paquete `{janitor}` es útil para esto.
- Si solo colocamos `get_dupes()`, entonces nos identifica duplicados de fila completa:

```
1 library(janitor)
2 datos %>%
3   get_dupes()
```

```
[1] id_jaula    id_raton    tratamiento protocolo    peso_inicial
[6] peso_final  peso_uterio chol         glucose      tag
[11] prot        urea        album        dupe_count
<0 rows> (or 0-length row.names)
```

- Si colocamos una o más variables dentro de `get_dupes()`, entonces nos identifica duplicados solo de esa variable.
- A menudo lo hacemos para encontrar individuos duplicados.

```
1 datos %>%
2   get_dupes(id_raton)
```

```
[1] id_raton    dupe_count  id_jaula    tratamiento protocolo
[6] peso_inicial peso_final  peso_uterio chol         glucose
[11] tag         prot        urea        album
<0 rows> (or 0-length row.names)
```

- Si el duplicado es erróneo, lo podemos eliminar con `distinct()` y el argumento `.keep_all = TRUE`.
- Se debe especificar si el duplicado es de fila o de alguna variable (p. ej., id).

```

1 datos <- datos %>%
2   distinct(id_raton, .keep_all = TRUE)
3
4 datos

```

	id_jaula	id_raton	tratamiento	protocolo	peso_inicial
1	1	1	control	ovx	26.00
2	1	2	control	ovx	24.50
3	1	3	control	ovx	20.40
4	2	4	control	hemiovx	26.59
5	2	5	control	ovx	23.50
6	2	6	maca	ovx	25.00
7	2	7	maca	ovx	24.80
8	3	8	maca	ovx	23.20
9	3	9	maca	hemiovx	22.69
10	3	10	maca	ovx	23.90
11	5	11	maca + critro	ovx	21.90
12	5	12	maca + critro	ovx	23.40
13	5	13	maca + critro	ovx	21.90
14	5	14	maca + critro	ovx	22.40
15	8	15	triple dosis maca + citro	ovx	18.90

- ¿Qué pasa si no se sabe si el duplicado es erróneo?
 - Podemos tener dos o más filas con duplicados y no saber cuál es el correcto.
 - En estos casos, el problema es complejo. Una solución puede ser la deduplicación probabilística.

Paso 3: Identifique datos faltantes

- Evalúe número y porcentaje de datos perdidos así como el patrón de estos.
- Hay varios paquetes que permiten manejar datos perdidos:
 - `{VIM}`
 - `{visdat}`
 - `{nanair}`
 - `{otros}`
- Usaremos algunas funciones de `{visdat}`, `{VIM}` y `{nanair}`.
- `{visdat}` y `{nanair}` generan gráficos `{ggplot2}`, mientras que `{VIM}` no lo hace.

Paso 3: Identifique datos faltantes (cont.)

`skim()`

Visualizar tipos de datos

Visualizar % de datos faltantes

- Nuevamente `skim()` nos permite conocer, rápidamente, el número de datos perdidos.

```
1 skim(datos)
```

Data summary

Name	datos
Number of rows	23
Number of columns	13

Column type frequency:	
character	2
numeric	11

Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	w
tratamiento	0	1	4	25	0	5	
protocolo	0	1	3	7	0	3	

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75
---------------	-----------	---------------	------	----	----	-----	-----	-----

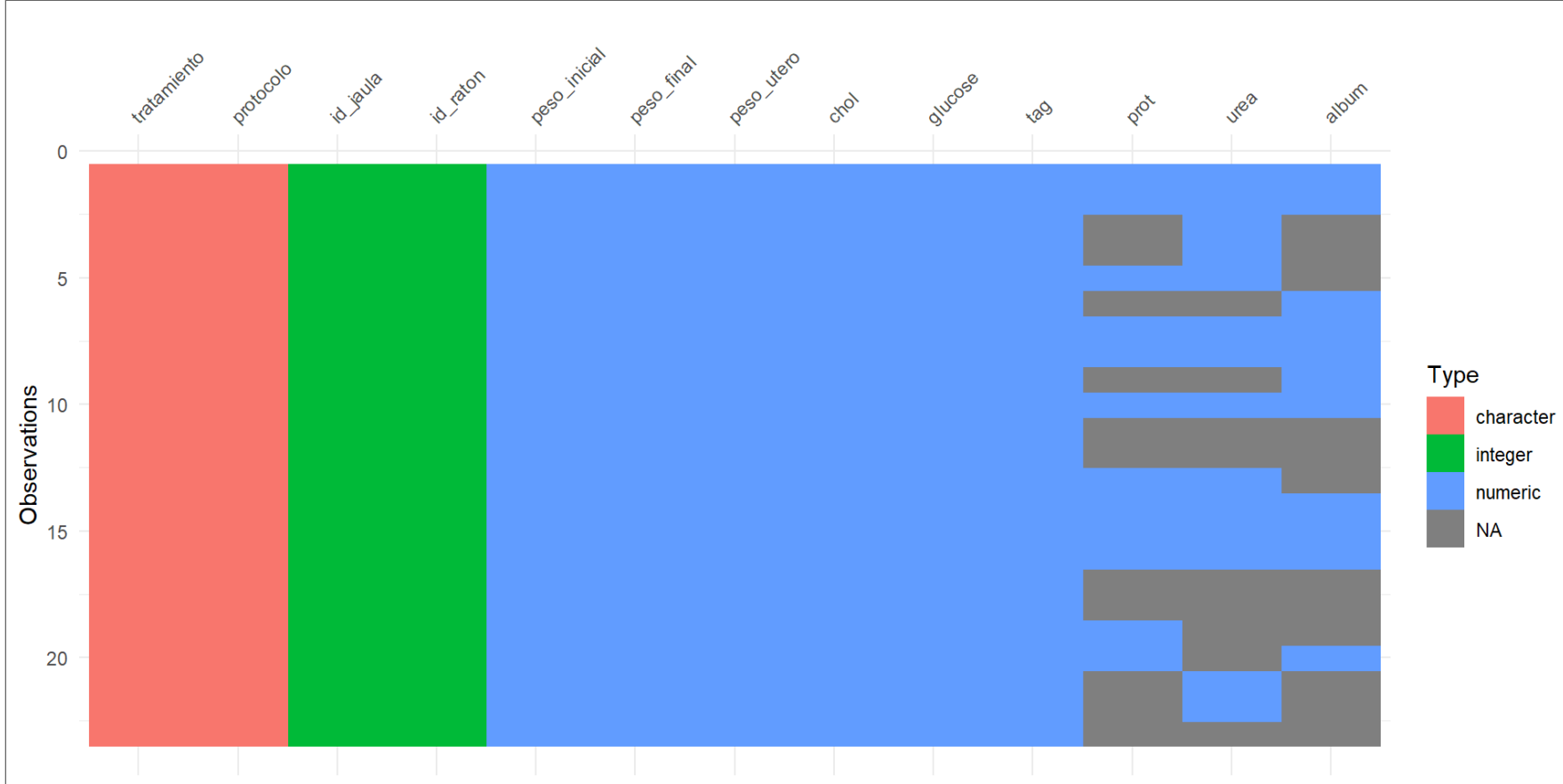
skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75
id_jaula	0	1.00	5.30	3.38	1.00	2.00	5.00	9.00
id_raton	0	1.00	12.00	6.78	1.00	6.50	12.00	17.50
peso_inicial	0	1.00	23.68	1.99	18.90	22.59	23.50	24.90
peso_final	0	1.00	28.59	2.18	23.80	27.08	28.77	30.10
peso_uterio	0	1.00	0.09	0.10	0.01	0.06	0.07	0.09
chol	0	1.00	81.96	12.89	59.28	72.31	82.08	87.29
glucose	0	1.00	124.74	37.27	60.10	99.08	118.37	147.50
tag	0	1.00	153.06	52.36	90.99	108.13	141.10	190.11
prot	11	0.52	5.22	0.49	4.68	4.91	5.08	5.34
urea	9	0.61	56.95	32.34	26.02	38.87	48.16	64.80
album	12	0.48	66.15	8.44	52.77	62.12	66.82	69.62

- El paquete `{visdat}` te permite visualizar el tipo de dato y si hay o no presencia de datos perdidos

```

1 library(visdat)
2 datos %>%
3   vis_dat()

```



- Es importante verificar si el tipo de dato corresponde con la naturaleza de la variable de estudio.
- Algunos datos faltantes pueden no verse por no configurar apropiadamente el tipo de la variable.
- Podemos también generar gráficos para identificar los datos perdidos y sus combinaciones:

```
1 datos %>%
2   vis_miss()
```



- Se aprecia que la variable **prot** tiene 47.83% de sus datos faltantes. La variable **urea** tiene 39.13% de sus datos faltantes.
- La legenda que dice **Missing (10.7%)** indica que el total de datos faltantes en las celdas (no en las filas) es de 10.7%.
- ¿Cuántos datos faltantes en por fila tendremos? ¿Qué combinaciones de datos faltantes tendremos?

Paso 3: Identifique datos faltantes (cont.)

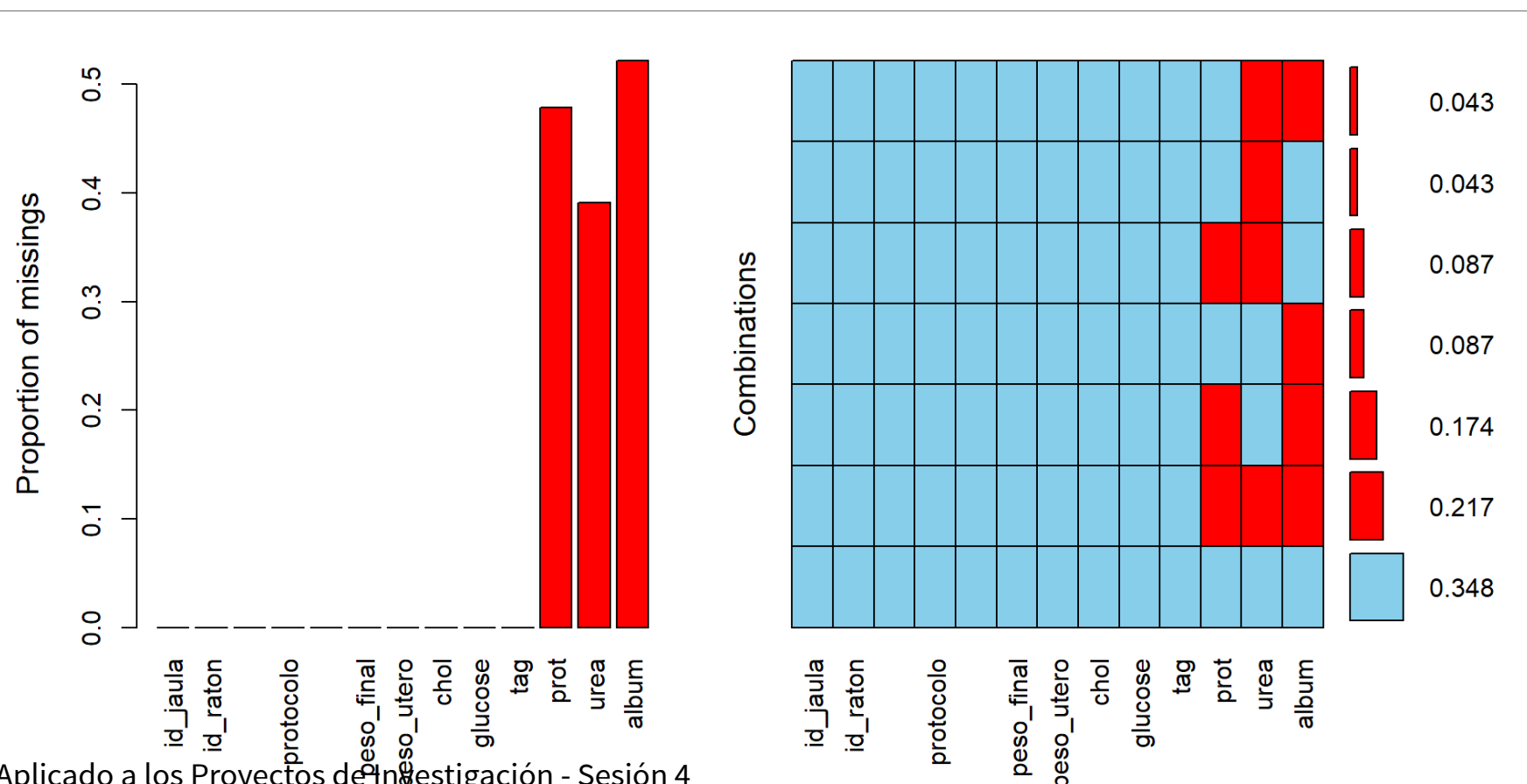
Combinaciones de datos faltantes

Recuperar datos faltantes

Datos perdidos ocultos

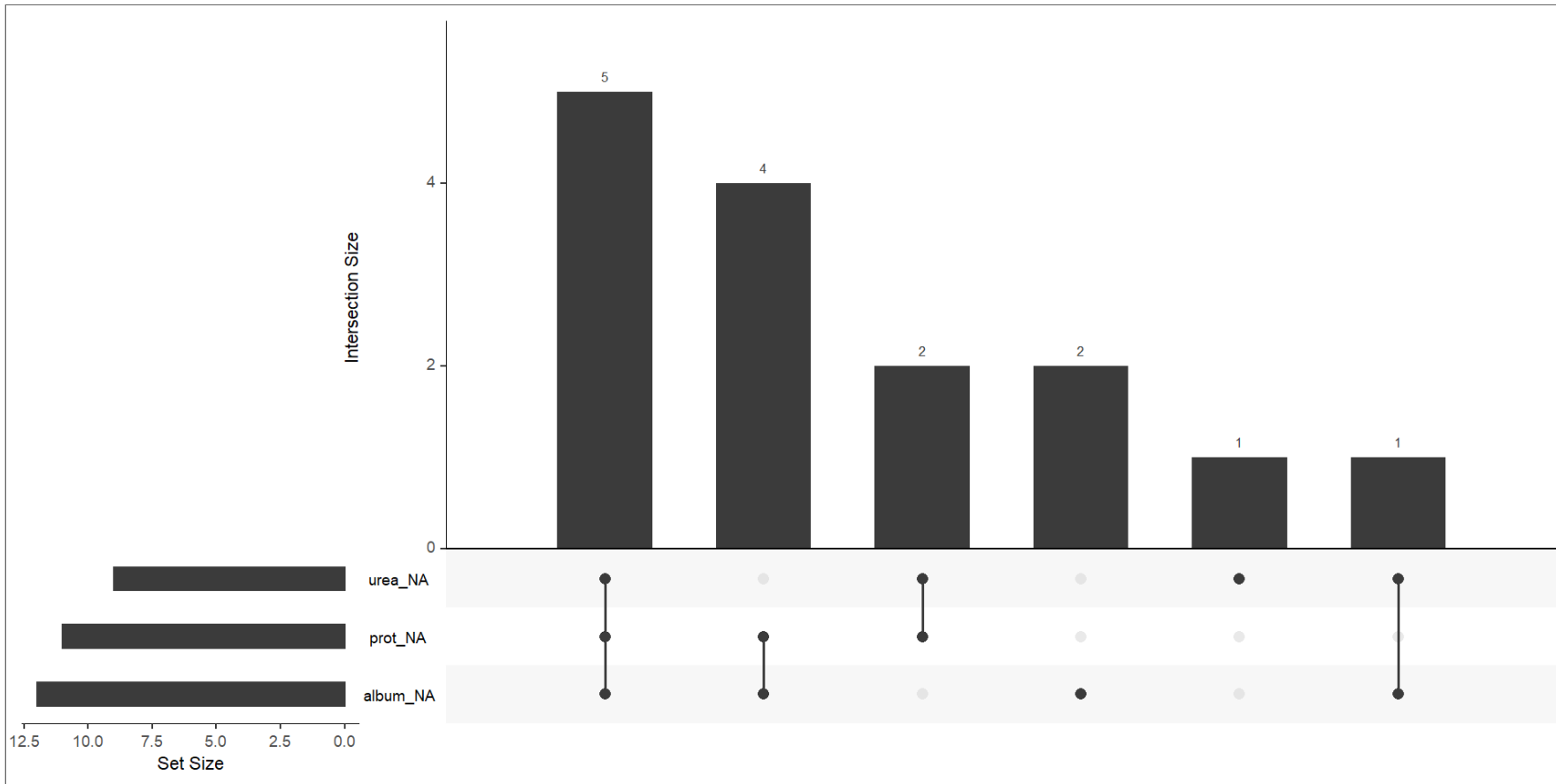
- El paquete `{VIM}` permite identificar datos perdidos por variable y sus combinaciones.
- Podemos visualizar los resultados directamente con la función `aggr()`:

```
1 library(VIM)
2 datos %>%
3   aggr(numbers = TRUE)
```



- También podemos usar la función `gg_miss_upset` del paquete `{naniar}` para evaluar las combinaciones de datos perdidos:

```
1 library(naniar)
2 datos %>%
3   gg_miss_upset()
```



- Lo primero que uno debe tratar de hacer es recuperar los datos faltantes.
 - Volver a revisar documentos fuentes.
 - Recontactar sujetos, etc.
- Podemos identificar a los individuos con datos faltantes en la variable urea usando `filter()`:

```
1 datos %>%
2   select(id_jaula, id_raton, urea) %>%
```

```
3 filter(is.na(urea))
```

	id_jaula	id_raton	urea
1		2	6
2		3	9
3		5	11
4		5	12
5		9	17
6		9	18
7		9	19
8		9	20
9		10	23

- Si se recupera la información, uno puede reemplazar los valores usando código en R.
- La función `replace()` del paquete `{dplyr}` es útil para esto. Supongamos que el dato perdido para el ratón 6 es de 65.2, podemos reemplazar el dato usando `replace()`

```
1 datos %>%
2   select(id_jaula, id_raton, urea) %>%
3   mutate(
4     urea = replace(urea, id_raton == 6, 65.2)
5   )
```

	id_jaula	id_raton	urea
1	1	1	66.27
2	1	2	76.73
3	1	3	52.32
4	2	4	50.71
5	2	5	26.02
6	2	6	65.20
7	2	7	40.78
8	3	8	66.94
9	3	9	NA
10	3	10	34.48
11	5	11	NA
12	5	12	NA
13	5	13	37.96
14	5	14	45.61
15	8	15	58.25

- Los datos perdidos a veces se guardan por defecto con algunos caracteres especiales.
- Pueden ser problemáticas si se guardan con categorías como: -99, 8888, “No aplica”, “No sabe”, etc.
- Una función muy útil para lidiar con estos datos y convertirlos en **NA** es la función `replace_na()` del paquete `{tidyr}`

```
1 library(tidyr)
2 datos_perdidos_comun
```

	edad	diabetes
1	45	Sí
2	23	Sí
3	34	No
4	29	N/A
5	-999	No
6	23	Sí
7	34	No
8	57	N/A
9	88	N/A
10	-999	N/A
11	-999	Sí

- Podemos convertir directamente todos estos valores por default a datos perdidos:

```
1 datos_perdidos_comun %>%
2   na_if(list(edad = -999, diabetes = "N/A")) -> datos_perdidos_limpia
3
4 datos_perdidos_limpia
```

	edad	diabetes
1	45	Sí
2	23	Sí
3	34	No
4	29	<NA>
5	NA	No
6	23	Sí
7	34	No
8	57	<NA>
9	88	<NA>

10	NA	<NA>
11	NA	Sí

Paso 4: Identifique valores extremos no plausibles

[skim\(\)](#)[describe\(\)](#)[Gráficos R base](#)[Datos no plausibles](#)

- Revise, variable por variable **valores extremos no plausibles** o **plausibles, pero sospechosamente extremos**. El valor mínimo es **p0** y el valor máximo es **p100**. Deben ser plausibles.

```
1 skim(datos)
```

Data summary

Name	datos
Number of rows	23
Number of columns	13

Column type frequency:	
character	2
numeric	11

Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	w
tratamiento	0	1	4	25	0	5	
protocolo	0	1	3	7	0	3	

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75
id_jaula	0	1.00	5.30	3.38	1.00	2.00	5.00	9.00
id_raton	0	1.00	12.00	6.78	1.00	6.50	12.00	17.50
peso_inicial	0	1.00	23.68	1.99	18.90	22.59	23.50	24.90
peso_final	0	1.00	28.59	2.18	23.80	27.08	28.77	30.10
peso_uterio	0	1.00	0.09	0.10	0.01	0.06	0.07	0.09
chol	0	1.00	81.96	12.89	59.28	72.31	82.08	87.29
glucose	0	1.00	124.74	37.27	60.10	99.08	118.37	147.50
tag	0	1.00	153.06	52.36	90.99	108.13	141.10	190.11
prot	11	0.52	5.22	0.49	4.68	4.91	5.08	5.34
urea	9	0.61	56.95	32.34	26.02	38.87	48.16	64.80
album	12	0.48	66.15	8.44	52.77	62.12	66.82	69.62

- Permite hacer algo similar

```
1 describe(datos)
```

datos

13 Variables 23 Observations

id_jaula

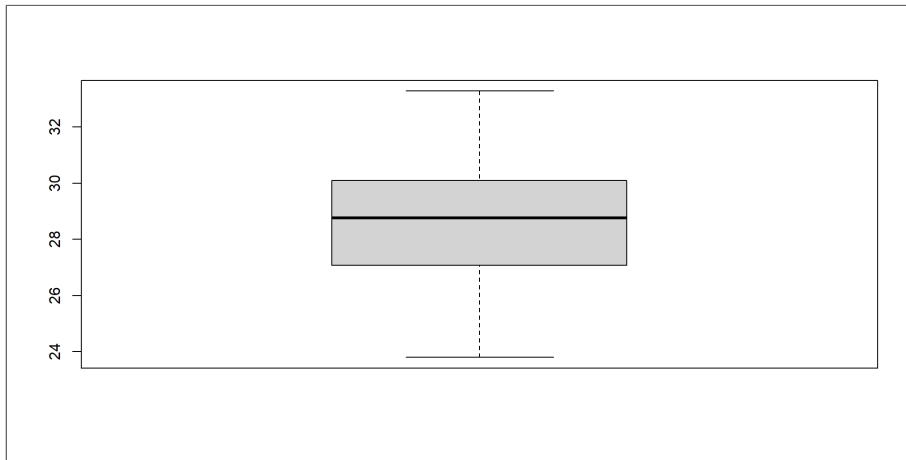
n	missing	distinct	Info	Mean	Gmd
23	0	7	0.979	5.304	3.881

lowest : 1 2 3 5 8, highest: 3 5 8 9 10

Value	1	2	3	5	8	9	10
Frequency	3	4	3	4	2	4	3
Proportion	0.130	0.174	0.130	0.174	0.087	0.174	0.130

- El **gráfico de cajas** nos muestra la distribución de la variable numérica en termino de sus cuantiles.
- Los **puntos aislados**, fuera de las cajas y bigotes, son considerados valores extremos.
- Estos pueden ser **plausibles** o **no plausibles**.
- El gráfico de cajas permite **identificar**, rápidamente, **valores extremos** potencialmente **no plausibles** o **problemáticos**.

```
1 boxplot(datos$peso_final)
```



```
1 boxplot(datos$peso_uterio)
```



- Los datos extremos pueden ser valores anómalos válidos.
- En ocasiones, son valores no plausibles, inválidos, producto del mal recojo de información.
- Cuando se tenga valores extremos no plausibles se puede optar por dos acciones:
 - 1. Corregir el valor extremo no plausible por datos que sí sean plausibles.
 - 2. Si no se puede, convertir los valores extremos no plausibles en datos faltantes (veremos esto).
 - 3. Bonus: A veces puede ser mejor recortar los datos y quedarse con el 1% y 99% percentil más bajo y alto, respectivamente.

Paso 4: Identifique valores extremos no plausibles

Corregir valor extremo no plausible

Convertir valor extremo no plausible a faltante

- Se puede usar la función: `na_if()` del paquete `{dplyr}`.
- Veamos una base de datos juguete con datos de peso (kg) y hemoglobina (mg/dL) de pacientes en un estudio:
 - El peso de 1450 es un valor extremo no plausible. Igualmente, los valores de hemoglobina 213, 3124 y -4 son valores extremos no plausibles.
 - Lo primero que debemos hacer es recuperar es tratar de recuperar estos valores.
 - Supongamos que podemos recuperar los valores: 1450 en realidad es 45 kg; 213, 3124 y -4 son 11.3, 10.44 y 9.2 mg/dL.
- Podemos usar la función `recode` para corregir los valores de peso:

```
1 datos_extremo %>%  
2   mutate(peso = recode(peso, `1450` = 45))
```

	peso	hb
1	56	12
2	34	11
3	23	213
4	78	10
5	46	3124
6	45	-4

- También podemos corregir de varias variables simultáneamente:

```
1 datos_extremo %>%  
2   mutate(  
3     peso = recode(peso, `1450` = 45),  
4     hb = recode(hb, `213` = 11.3, `3124` = 10.44, `-4` = 9.2)  
5   ) -> datos_extremo_recodif
```

```
6
7 datos_extremo_recodif
```

	peso	hb
1	56	12.00
2	34	11.00
3	23	11.30
4	78	10.00
5	46	10.44
6	45	9.20

- Si no podemos recuperar los datos correctos, la otra opción es convertir los valores extremos en datos faltantes:

```
1 datos_extremo %>%
2   mutate(
3     peso = na_if(peso, 1450)
4   )
```

	peso	hb
1	56	12
2	34	11
3	23	213
4	78	10
5	46	3124
6	NA	-4

- Podemos hacerlo de manera simultánea para varias variables

```
1 datos_extremo %>%
2   mutate(
3     peso = na_if(peso, 1450),
4     hb = na_if(hb, 213),
5     hb = na_if(hb, 3124),
6     hb = na_if(hb, -4)
7   )
```


2	34	11
3	23	NA
4	78	10
5	46	NA
6	NA	NA

- O usando `replace()` y una condición lógica:

```
1 datos_extremo %>%
2   mutate(
3     peso = na_if(peso, 1450),
4     hb = replace(hb, hb > 100 | hb < 0, NA)
5   ) -> datos_extremo_recomiss
6
7 datos_extremo_recomiss
```

	peso	hb
1	56	12
2	34	11
3	23	NA
4	78	10
5	46	NA
6	NA	NA

Paso 5: Detecte y corrija inconsistencias mediante consultas (*queries*) de interés

Consulta 1

Consulta 2

Consulta 3

Consulta 4

Muestre el peso inicial mínimo, máximo y promedio del grupo control:

```
1 datos %>%
2   filter(tratamiento == "control") %>%
3   summarise(
4     minimo_peso = min(peso_inicial),
5     maximo_peso = max(peso_inicial),
6     promedio_peso = mean(peso_inicial)
7   )
```

```
  minimo_peso maximo_peso promedio_peso
1         20.4         26.59         24.198
```

Muestre los pesos inicial máximos, mínimo y promedio según grupo de tratamiento. También muestre el número de ratones por grupo:

```
1 datos %>%
2   group_by(tratamiento) %>%
3   summarise(
4     minimo_peso = min(peso_inicial),
5     maximo_peso = max(peso_inicial),
6     promedio_peso = mean(peso_inicial),
7     n_ratones = n()
8   )
```

```
# A tibble: 5 × 5
  tratamiento minimo_peso maximo_peso promedio_peso n_ratones
  <chr>         <dbl>         <dbl>         <dbl>         <int>
1 control         20.4         26.6         24.2           5
2 maca            22.7         25          23.9           5
3 maca            21.8         23.4         22.4           4
```

4 sham operated	22.5	25.5	23.6	4
5 triple dosis maca + citro	18.9	27.9	24.0	5

Muestre los id_jaula con el número de ratones por jaula

```
1 datos %>%
2   group_by(id_jaula) %>%
3   summarise(n_ratones_por_jaula = n())
```

```
# A tibble: 7 × 2
  id_jaula n_ratones_por_jaula
  <int>      <int>
1       1             3
2       2             4
3       3             3
4       5             4
5       8             2
6       9             4
7      10             3
```

Identifique los ID de los ratones del grupo control con una razón glucosa / colesterol > 1

```
1 datos %>%
2   filter(tratamiento == "control" & glucose / chol > 1)
```

```
  id_jaula id_raton tratamiento protocolo peso_inicial peso_final peso_uterio
1       1       1      control      ovx        26.0      33.28      0.089
2       1       3      control      ovx        20.4      29.93      0.078
3       2       5      control      ovx        23.5      30.37      0.052
  chol glucose tag prot urea album
1 85.99 109.97 182.42 5.37 66.27 66.82
2 99.67 118.37 195.16  NA 52.32  NA
3 82.08  95.53 108.13 5.33 26.02  NA
```

Otra forma de hacerlo, es crear primerio la razón glucose / chol y filtrar:

```
1 datos %>%
2   mutate(ratio_gluc_chol = glucose / chol) %>%
3   filter(tratamiento == "control" & ratio_gluc_chol > 1)
```

	id_jaula	id_raton	tratamiento	protocolo	peso_inicial	peso_final	peso_utero
1	1	1	control	ovx	26.0	33.28	0.089
2	1	3	control	ovx	20.4	29.93	0.078
3	2	5	control	ovx	23.5	30.37	0.052

	chol	glucose	tag	prot	urea	album	ratio_gluc_chol
1	85.99	109.97	182.42	5.37	66.27	66.82	1.278870
2	99.67	118.37	195.16	NA	52.32	NA	1.187619
3	82.08	95.53	108.13	5.33	26.02	NA	1.163865

Nuestro turno


- Descargue la carpeta denominada [taller04](#) disponible en la carpeta compartida.
- Abra el proyecto denominado [taller04.Rproj](#)
- Complete y ejecute el código faltante en los chunk de código de la SEGUNDA PARTE.
- Una vez culmine todo el proceso, renderice el archivo .qmd.

10:00

¡Gracias!
¿Preguntas?

@psotob91

<https://github.com/psotob91>

 percys1991@gmail.com