Clase 3a - Visualización de resultados

José Rodríguez de la Fuente

Tabla de contenidos

Construcción de tablas **1**
La función `table()` . . . 2
Algunas librerías para la construcción de tablas . . . 4
Tablas con `dplyr` y `tidyr` . . . 5
El formato *tidy* . . . 8

Exportando los resultados **10**
Exportando a xslx . . . 11
Exportando a docx . . . 14

Construcción de tablas

En esta apartado revisaremos distintas aproximaciones a la construcción de tablas de frecuencias y de tablas de contingencia. Quizás `R` no se destaque por la simpleza del análisis exploratorio a través de tablas, pero existen infinidad de paquetes que nos permitirán llegar a resultados similares. Empezaremos explorando la funciones disponibles en **R base**, luego revisaremos algunos paquetes que harán la mayor parte del trabajo por nosotros y, por último, veremos cómo podemos hacerlo con `dplyr` y `tidyr` de `tidyverse`.

Para este apartado utilizaremos la base de individuos de la EPH de 2025.

```
library(eph)

eph_ind <- get_microdata(year = 2025, period = 2, type = "individual")
eph_hog <- get_microdata(year = 2025, period = 2, type = "hogar")
```

Por otro lado, vamos a trabajar con las siguientes variables que transformaremos en factor:

```r
library(tidyverse)

eph_ind <- eph_ind %>%
  mutate(
    region_f = factor(REGION, labels = c("Gran Buenos Aires", "Noroeste",
    ↪  "Noreste", "Cuyo", "Pampeana", "Patagonia")),
    sexo_f = factor(CH04, labels = c("Varón", "Mujer")),
    estado_f = factor(ESTADO, labels = c("No realizada", "Ocupado",
    ↪  "Desocupado", "Inactivo", "Menor de 10 años")),
    empleo_f = factor(EMPLEO, labels = c("Formal", "Informal", "Ns/Nc")))
```

La función `table()`

La función `table()` es una de las más utilizadas para la construcción de tablas de frecuencias rápidas, tanto para el análisis univariable como bivariable. Por ejemplo, a partir de las variables *estado_f* (Condición de actividad) y *region_f*.

```r
table(eph_ind$estado_f)
```

```
    No realizada          Ocupado       Desocupado         Inactivo
              53            20374             1387            19193
Menor de 10 años
            5079
```

```r
table(eph_ind$estado_f, eph_ind$region_f)
```

```
                 Gran Buenos Aires Noroeste Noreste Cuyo Pampeana Patagonia
  No realizada                  25        4       6    2        11         5
  Ocupado                     3314     4427    1906 2168      6235      2324
  Desocupado                   310      228     134  113       476       126
  Inactivo                    2816     4175    2131 1865      5854      2352
  Menor de 10 años             741     1176     513  563      1549       537
```

Si queremos calcular las proporciones o frecuencias relativas, podemos llamar a la función `prop.table()` sobre la función ya especificada. La opción `margin = 2` indica que queremos calcular los porcentajes por columna.

```
prop.table(table(eph_ind$estado_f, eph_ind$region_f), margin = 2)
```

```
                  Gran Buenos Aires     Noroeste      Noreste         Cuyo
No realizada          0.0034693311 0.0003996004 0.0012793177 0.0004245383
Ocupado               0.4598945323 0.4422577423 0.4063965885 0.4601995330
Desocupado            0.0430197058 0.0227772228 0.0285714286 0.0239864148
Inactivo              0.3907854566 0.4170829171 0.4543710021 0.3958819783
Menor de 10 años      0.1028309742 0.1174825175 0.1093816631 0.1195075356

                     Pampeana    Patagonia
No realizada     0.0007787611 0.0009356287
Ocupado          0.4414159292 0.4348802395
Desocupado       0.0336991150 0.0235778443
Inactivo         0.4144424779 0.4401197605
Menor de 10 años 0.1096637168 0.1004865269
```

Por último, si queremos presentar los resultados en porcentajes, lo multiplicamos por 100:

```
prop.table(table(eph_ind$estado_f, eph_ind$region_f), margin = 2) * 100
```

```
                  Gran Buenos Aires    Noroeste     Noreste        Cuyo
No realizada            0.34693311  0.03996004  0.12793177  0.04245383
Ocupado                45.98945323 44.22577423 40.63965885 46.01995330
Desocupado              4.30197058  2.27772228  2.85714286  2.39864148
Inactivo               39.07854566 41.70829171 45.43710021 39.58819783
Menor de 10 años       10.28309742 11.74825175 10.93816631 11.95075356

                     Pampeana   Patagonia
No realizada      0.07787611  0.09356287
Ocupado          44.14159292 43.48802395
Desocupado        3.36991150  2.35778443
Inactivo         41.44424779 44.01197605
Menor de 10 años 10.96637168 10.04865269
```

Como verán, la función `table()` es muy útil para la construcción rápida de tablas de frecuencias, pero no nos permite realizar análisis complementarios, ni nos otorgan una salida en formato **tidy**.

Sentido de los porcentajes

Por regla general, cuando en una relación entre dos variables podemos establecer que hay una que funciona como independiente y otra como dependiente, solemos ubicar a la primera en la columna y a la segunda en las filas. Al mismo tiempo, los porcentajes suelen calcularse en el sentido de la variable independiente, quedando el 100% en la fila final.
Esto no invalida que, en algunos casos, podamos calcular los porcentajes por fila. Por ejemplo, si queremos conocer la distribución de género por rama de conocimiento.

Algunas librerías para la construcción de tablas

A veces necesitamos hacer procesamientos rápidos, sin necesidad de escribir mucho código. Si nuestra opción es quedarnos usando `R` y no ~~huir~~ migrar a otros programas, podemos utilizar algunas librerías que nos facilitarán la construcción de tablas. Les dejamos una lista con la variedad de librerías que pueden utilizar:

- `janitor`
- `expss`
- `summarytools`
- `gmodels`
- `sjmisc`

Vamos a realizar una primera prueba utilizando la librería `sjmisc`. En primer lugar, la función `frq()` nos devuelve una tabla de frecuencias algo más completa que la función `table()`. Por su parte, tiene la opción *sort.frq* que nos permite ordenar las categorías en forma ascendente o descendente y *weights* que nos permitirá indicar el ponderador. Vamos a probar con la variable *region_f*.

```
install.packages("sjmisc")
```

```
library(sjmisc)

eph_ind %>%
  frq(region_f, sort.frq = "desc", weights = PONDERA)
```

Tabla 1: region_f <categorical>

val	label	frq	raw.prc	valid.prc	cum.prc
Gran Buenos Aires		16130572	53.93	53.93	53.93

Pampeana		6428990	21.50	21.50	75.43
Noroeste		2856595	9.55	9.55	84.98
Cuyo		1874722	6.27	6.27	91.25
Noreste		1479482	4.95	4.95	96.19
Patagonia		1138508	3.81	3.81	100.00
NA	NA	0	0.00	NA	NA
total N=29908869 · valid N=29908869 · x̄=2.43 · =1.79					

El mismo paquete nos permite realizar tablas cruzadas con la función `flat_table`.

```
eph_ind %>%
  flat_table(region_f, empleo_f, margin = "row", weights = PONDERA)
```

```
                 empleo_f Formal Informal Ns/Nc
region_f
Gran Buenos Aires           58.08    41.65  0.27
Noroeste                    46.44    53.56  0.00
Noreste                     48.99    51.01  0.00
Cuyo                        47.34    52.66  0.00
Pampeana                    58.97    40.99  0.03
Patagonia                   72.18    27.74  0.08
```

Tablas con `dplyr` y `tidyr`

Si bien las librerías anteriores nos permiten realizar tablas de frecuencias de forma rápida y sencilla, no siguen la filosofía del `tidyverse` que venimos utilizando y que nos servirá para trabajar en forma ordenada en cada momento de la investigación. Quizás, a principio, la elaboración de tablas con estás funciones sea algo complicado, pero a futuro nos facilitará la posterior presentación de los resultados en documentos o en gráficos.

La secuencia de pasos es similar a como lo hemos hecho en el apartado de **variables agregadas**. Primero agrupamos la información (`group_by()`) y luego contamos los casos. Esto último podemos hacerlo mediante la función `summarise()`, `count()` o `tally()`.

La función `count()` no necesita de la función `group_by()`, ya que cuenta los casos por grupos automáticamente. Por su parte, `tally()` necesita que previamente agrupemos la información con `group_by()`, pero a futuro nos permitirá hacer conteos ponderados.

```
eph_ind %>%
  group_by(region_f) %>%
  summarise(frecuencia = n())
```

```
# A tibble: 6 x 2
  region_f          frecuencia
  <fct>                  <int>
1 Gran Buenos Aires       7206
2 Noroeste               10010
3 Noreste                 4690
4 Cuyo                    4711
5 Pampeana               14125
6 Patagonia               5344
```

```
eph_ind %>%
  count(region_f)
```

```
# A tibble: 6 x 2
  region_f              n
  <fct>             <int>
1 Gran Buenos Aires  7206
2 Noroeste          10010
3 Noreste            4690
4 Cuyo               4711
5 Pampeana          14125
6 Patagonia          5344
```

```
eph_ind %>%
  group_by(region_f) %>%
  tally()
```

```
# A tibble: 6 x 2
  region_f              n
  <fct>             <int>
1 Gran Buenos Aires  7206
2 Noroeste          10010
3 Noreste            4690
4 Cuyo               4711
5 Pampeana          14125
6 Patagonia          5344
```

Para realizar tablas bivariadas, lo primero que adicionaremos es la segunda variable a la función `count()` utilizando la función `wt = PONDERA` para obtener los conteos ponderados.

```r
eph_ind %>%
  count(empleo_f, sexo_f, wt = PONDERA)
```

```
# A tibble: 8 x 3
  empleo_f sexo_f       n
  <fct>    <fct>    <int>
1 Formal   Varón  4292121
2 Formal   Mujer  3238352
3 Informal Varón  3106965
4 Informal Mujer  2646372
5 Ns/Nc    Varón    14689
6 Ns/Nc    Mujer     5882
7 <NA>     Varón  7154370
8 <NA>     Mujer  9450118
```

Como verán, la tabla se nos presenta en un formato extraño, que se denomina *long* o largo, debido a que las dos variables de interés se sitúan en las primeras columnas y las frecuencias en la última. En algunos casos, como por ejemplo para la construcción de gráficos, este formato es apropiado. Pero para la presentación de tablas, necesitamos un formato de tipo *wide* o ancho.

Para presentar la tabla en un formato más amigable, utilizaremos la función `pivot_wider` de `tidyr`. Lo que debemos declarar en dicha función es cual es la variable que pasará a mostrar sus categorías en las columnas y de dónde surgirán los valores que irán en las celdas. En nuestro caso, las columnas se completarán con las categorías de la variable `sexo_f` y los valores serán las frecuencias de los conteos de la variable `n` que construimos en el paso anterior. Aprovecharemos para filtrar los casos que se encuentran ocupados en el mercado de trabajo.

```r
eph_ind %>%
  filter(estado_f == "Ocupado") %>%
  count(empleo_f, sexo_f, wt = PONDERA) %>%
  pivot_wider(names_from = sexo_f, values_from = n)
```

```
# A tibble: 3 x 3
  empleo_f   Varón   Mujer
  <fct>      <int>   <int>
1 Formal   4292121 3238352
2 Informal 3106965 2646372
3 Ns/Nc      14689    5882
```

Como se vera, la tabla se presenta de forma más ordenada y clara, sin embargo, aún no hemos calculado las proporciones. Para ello, la librería `janitor` automáticamente nos calculará distintos tipos de proporciones, nos presentará los datos en formato porcentual y nos dejará agregar los totales por fila o columna. Exploremos la librería.

```
install.packages("janitor")
```

```
library(janitor)

eph_ind %>%
  filter(estado_f == "Ocupado") %>%
  count(empleo_f, sexo_f, wt = PONDERA) %>%
  pivot_wider(names_from = sexo_f, values_from = n) %>%
  adorn_totals("col") %>% # Agregamos los totales por columna
  adorn_percentages("col") %>% # Calculamos las proporciones por columna
  adorn_pct_formatting(digits = 1) # Presentamos los valores en formato
    ↪ porcent
```

```
 empleo_f Varón Mujer Total
   Formal 57.9% 55.0% 56.6%
 Informal 41.9% 44.9% 43.2%
    Ns/Nc  0.2%  0.1%  0.2%
```

El formato *tidy*

Disponer los datos en forma ordenada implica que: 1) cada variable es una columna, 2) cada observación es una fila y 3) cada valor se encuentra en una celda. Sin embargo muchas veces, los datos distan de presentarse en ficho formato. Por ejemplo, es frecuente que:

- los encabezados de las columnas sean valores,
- muchas variables sean guardadas en una columna,
- las variables se encuentren en las filas y columnas,
- múltiples unidades de análisis se encuentren en una misma tabla,
- una unidad de análisis se encuentre en varias tablas.

Para resolver estos problemas, `tidyr` nos ofrece las funciones `pivot_longer()` y `pivot_wider()`. La primera nos permite pasar de un formato ancho a uno largo, mientras que la segunda nos permite pasar de un formato largo a uno ancho.

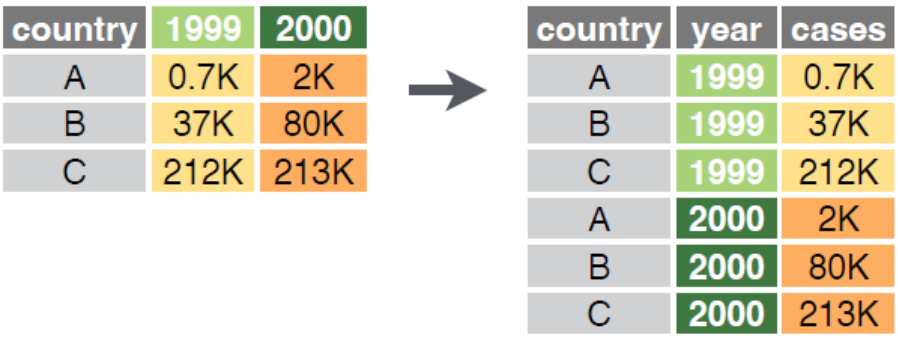


Figura 1: Formato ancho

`pivot_longer()` nos permite pasar de un formato ancho a uno largo. Para ello, debemos especificar las columnas que queremos mantener y las que queremos transformar en filas. Con la función *names_to* especificamos el nombre de la nueva columna que contendrá los nombres de las columnas que estamos transformando en filas, y con *values_to* especificamos el nombre de la nueva columna que contendrá los valores de las columnas que estamos transformando en filas.

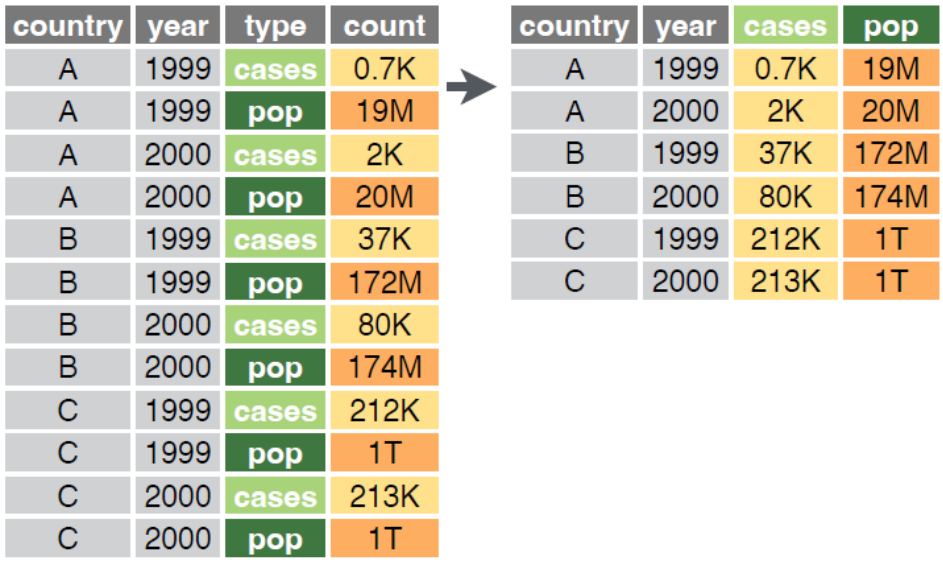


Figura 2: Formato largo

`pivot_wider()` nos permite pasar de un formato largo a uno ancho. Para ello, debemos especificar las columnas que queremos mantener y las que queremos transformar en columnas.

Con la función *names_from* especificamos el nombre de la nueva columna que contendrá los nombres de las columnas que estamos transformando en columnas, y con *values_from* especificamos el nombre de la nueva columna que contendrá los valores de las columnas que estamos transformando en columnas.

Exportando los resultados

Hasta aquí hemos trabajado con tablas de frecuencias y con tablas de contingencia. Si lo que necesitamos es explorar los datos para conocer la información que disponemos, probar hipótesis o realizar análisis descriptivos, con las salidas gráficas que nos ofrece `RStudio` nos bastaría. Sin embargo, a veces, necesitamos presentar los resultados en informes, documentos o presentaciones. Para ello, necesitamos exportar los resultados a formatos comúnmente como xlsx, docx, pdf, etc.

En esta sección vamos a mostrar algunas alternativas para exportar los resultados a una hoja de calculo (xlsx) y a un documento de texto (docx). No obstante, existen muchas otras opciones de exportación que pueden explorarse y que se complementan muy bien con el análisis estadístico en `RStudio`, tales como html o pdf.

> Recomendación
>
> A lxs interesadxs les recomendamos revisar los sistemas de publicación científica tales como `Quarto` o `R Markdown`. Ambos sistemas permiten la integración de código `R` con texto, gráficos y tablas, y la exportación a distintos formatos de salida, permitiendo la creación de **documentos reproducibles**.

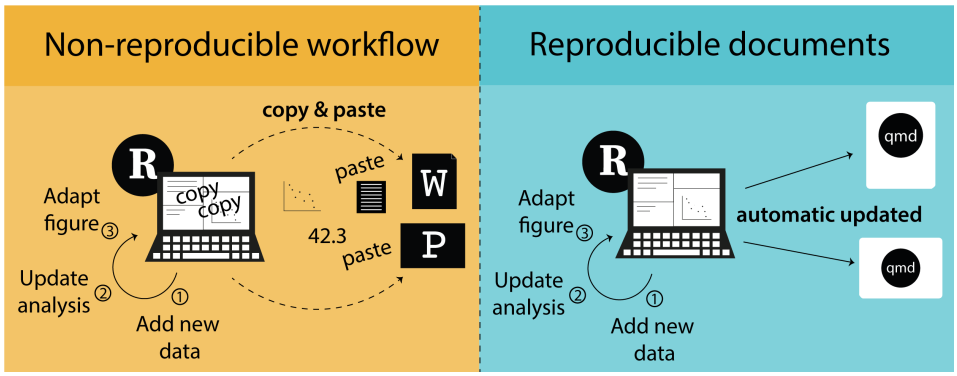


Figura 3: Flujos de trabajo reproducibles y no reproducibles. Fuente: https://biostats-r.github.io/biostats/quarto/

Exportando a xslx

Para exportar los resultados a una hoja de cálculo, utilizaremos la librería `openxlsx`. Esta librería nos permite exportar los resultados a un archivo de `Excel`, añadir hojas de calculo, modificar el formato de las celdas, entre otras cosas. Vamos a revisar las funciones más sencillas del paquete. Para ello, crearemos un objeto con la tabla cruzada que creamos en la sección anterior. Vamos a crear una carpeta dentro del proyecto llamada *resultados* para guardar allí las tablas y gráficos generados.

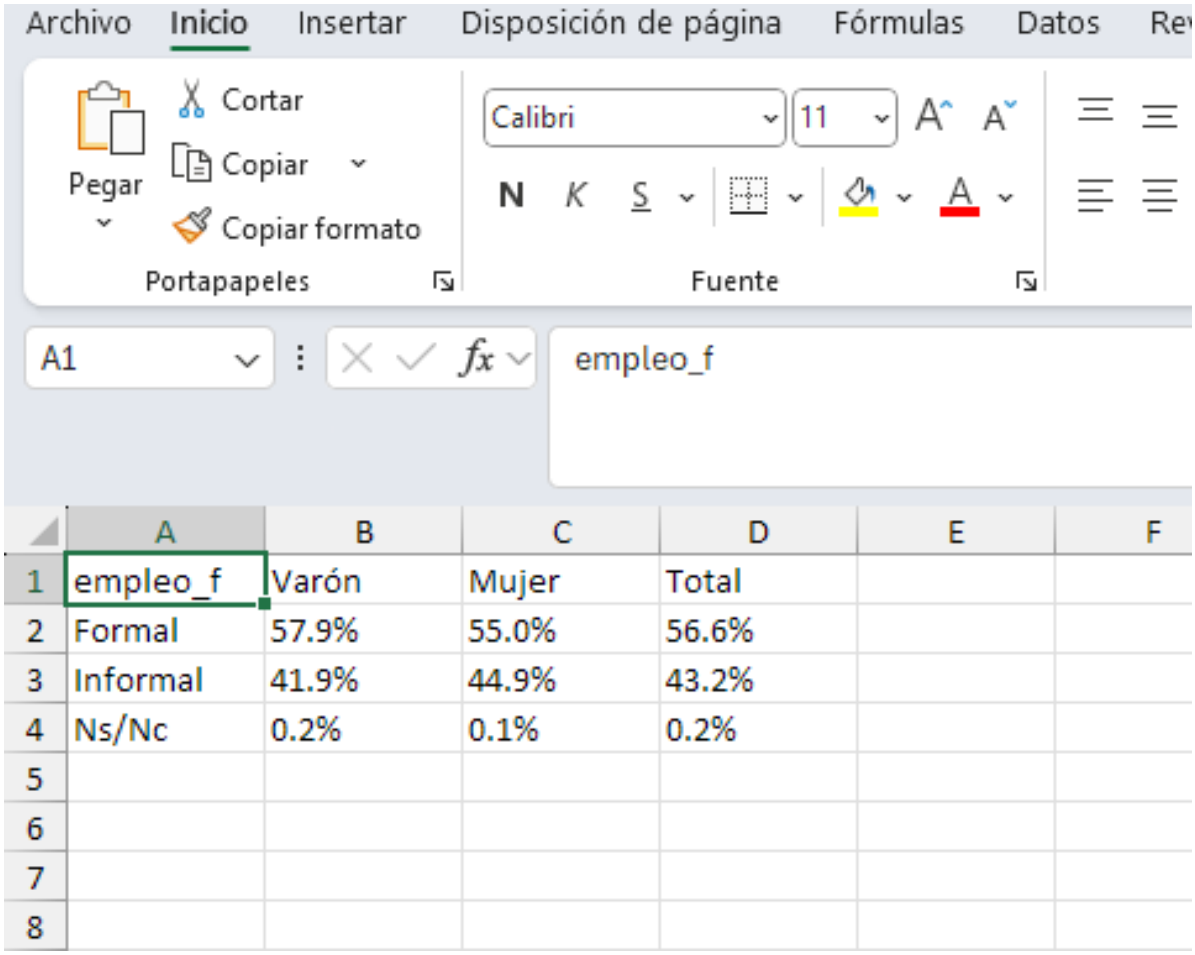
```
install.packages("openxlsx")
```

```
library(openxlsx)

# Creamos un objeto con la tabla de frecuencias
tabla1 <- eph_ind %>%
  filter(estado_f == "Ocupado") %>%
  count(empleo_f, sexo_f, wt = PONDERA) %>%
  pivot_wider(names_from = sexo_f, values_from = n) %>%
  adorn_totals("col") %>% # Agregamos los totales por columna
  adorn_percentages("col") %>% # Calculamos las proporciones por columna
  adorn_pct_formatting(digits = 1) # Presentamos los valores en formato
    ↪ porcent

# Exportamos la tabla a un archivo de Excel
write.xlsx(tabla1, "resultados/tabla1.xlsx")
```

El resultado de la función `write.xlsx` fue la creación de un archivo en la carpeta *resultados* con el nombre tabla1.xlsx. Si abrimos el archivo, veremos que la tabla se ha exportado correctamente.



	A	B	C	D	E	F
1	empleo_f	Varón	Mujer	Total		
2	Formal	57.9%	55.0%	56.6%		
3	Informal	41.9%	44.9%	43.2%		
4	Ns/Nc	0.2%	0.1%	0.2%		
5						
6						
7						
8						

Figura 4: Ejemplo uso write.xlsx

Ahora bien, si dentro de nuestro flujo de trabajo construimos varias tablas y queremos exportarlas a un único archivo, podemos hacerlo creando un *workbook* con la función `createWorkbook()` y luego añadiendo las tablas con la función `addWorksheet()`. Veamos paso a paso cómo hacerlo. En la primera hoja pondremos la tabla que creamos anteriormente y en la segunda hoja incorporaremos la tabla que cruza la variable de informalidad por región.

```
# Creamos un objeto con la tabla de informalidad por región
tabla2 <- eph_ind %>%
  filter(estado_f == "Ocupado") %>%
  count(empleo_f, region_f, wt = PONDERA) %>%
  pivot_wider(names_from = region_f, values_from = n) %>%
  adorn_totals("col") %>%
  adorn_percentages("col") %>%
```

```
  adorn_pct_formatting(digits = 1)

# Creamos un workbook
wb <- createWorkbook()

# Añadimos la primera hoja
addWorksheet(wb, sheetName = "Informalidad x sexo")

# Escribimos la tabla en la primera hoja
writeData(wb, sheet = "Informalidad x sexo", x = "Tipo de inserción laboral
 ↪  según sexo", startRow = 1)
writeData(wb, sheet = "Informalidad x sexo", x = tabla1,
          borders = "rows" , borderStyle = "dashed", startRow = 3)

# Añadimos la segunda hoja
addWorksheet(wb, sheetName = "Informalidad x región")

# Escribimos la tabla en la segunda hoja
writeData(wb, sheet = "Informalidad x región", x = "Tipo de inserción laboral
 ↪  según región", startRow = 1)
writeData(wb, sheet = "Informalidad x región", x = tabla2,
          borders = "rows" , borderStyle = "medium", startRow = 3)

# Guardamos el archivo
saveWorkbook(wb, "resultados/tabla2.xlsx", overwrite = TRUE)
```

Como se ve, el archivo ahora tiene dos hojas, y en cada una de ellas encontramos las tablas que exportamos desde `RStudio`.

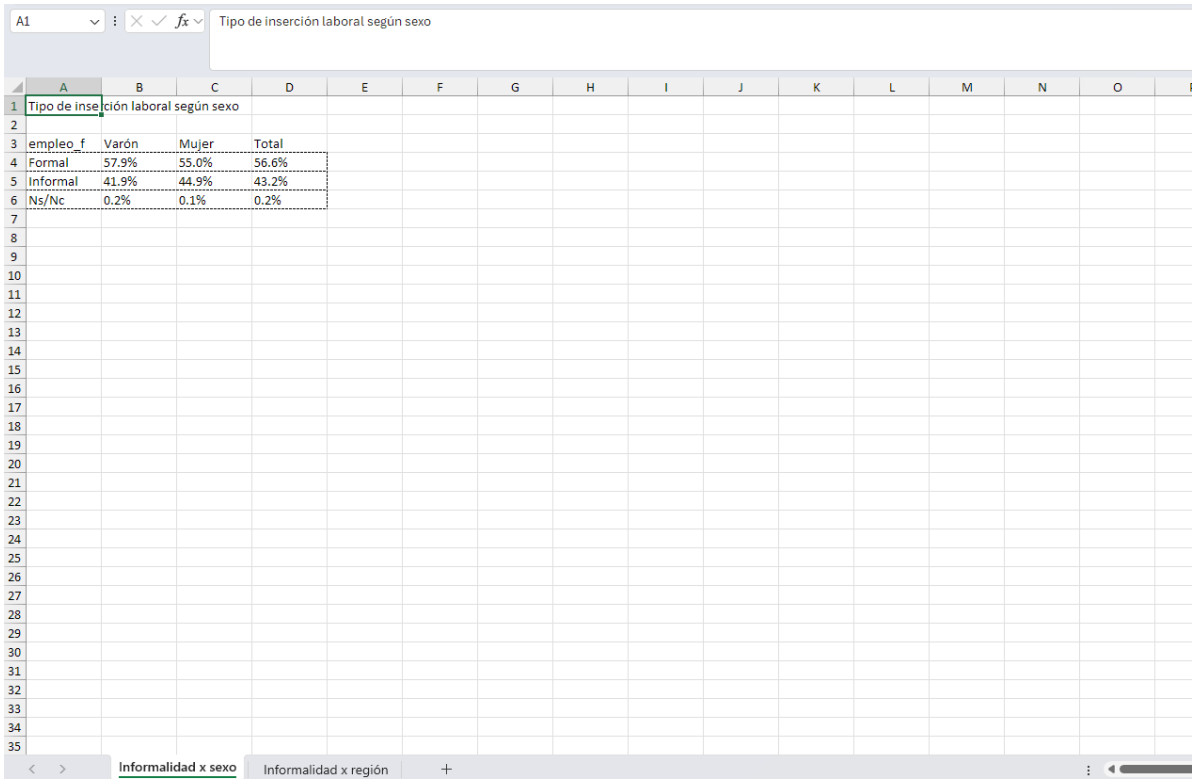


Figura 5: Ejemplo de construcción de hojas en archivo xlsx

Exportando a docx

Otra de las situaciones en la que nos podemos encontrar es ante la necesidad de exportar las tablas realizadas directamente a un archivo .docx en donde luego escribiremos texto. Para ello, utilizaremos la librería `flextable`, que nos permite no solo exportar las tablas, sino editarlas para que queden en un formato amigable para reportes, documentos, *papers*, etc.

Nuevamente, esta librería funciona en formato *tidy* y permite el uso de *pipes*. Lo primero que vamos a hacer es tomar el objeto `tabla2` que creamos anteriormente y exportarlo a un archivo .docx. Para esto, utilizaremos primero la función `flextable()` para convertir la tabla en un objeto de tipo *flextable*.

```
install.packages("flextable")
```

```
library(flextable)

flextable(tabla2)
```

empleo_f	Gran Buenos Aires	Noroeste	Noreste	Cuyo	Pampeana	Patagonia	Total
Formal	58.1%	46.4%	49.0%	47.3%	59.0%	72.2%	56.6%
Informal	41.7%	53.6%	51.0%	52.7%	41.0%	27.7%	43.2%
Ns/Nc	0.3%	-	-	-	0.0%	0.1%	0.2%

La librería incluye muchísimas funciones para editar y automatizar el formato de salida de las tablas. Por ejemplo, podemos cambiar el formato de las celdas, agregar títulos, subtítulos, notas al pie, entre otras cosas.

> Recomendación
>
> Recomendamos revisar la página de la librería `flextable` para conocer las potencialidades que provee.

Vamos a probar ahora agregar en la primera columna la etiqueta de la variable `disciplina_f` y en en una fila superior la etiqueta de la variable `genero_f`. Exploraremos algunas opciones de la función como `set_header_labels()`, `add_header_row()`, `add_footer_lines()`, `set_caption()`, `theme_vanilla()`, `align()` y `autofit()`. El resultado lo guardaremos en un objeto llamado `tabla_doc`.

```
tabla_doc <- flextable(tabla2) %>%
  set_header_labels(empleo_f = "Tipo de inserción") %>% # coloco la etiqueta
    ↪ de la variable empleo_f en la primera columna
  add_header_row(values = c("", "Región"), colwidths = c(1,7)) %>%  # coloco
    ↪ la etiqueta de la variable región en una fila superior
  add_footer_lines(values = c("Fuente: elaboración propia en base a
    ↪ EPH-INDEC")) %>% # agrego una nota al pie
  set_caption("Distribución de la población ocupada según inserción laboral
    ↪ por región. Argentina urbana, 2do trimestre 2025") %>% # agrego un
    ↪ título a la tabla
  theme_vanilla() %>% # cambio el tema de la tabla
  align(i = 1, part = "header", align = "center") %>% # centro el título de
    ↪ la tabla
  autofit() # ajusto el ancho de las columnas automáticamente

tabla_doc
```

	Región					
Tipo de inserción	**Gran Buenos Aires**	**Noroeste**	**Noreste**	**Cuyo**	**Pampeana**	**Pa**
Formal	58.1%	46.4%	49.0%	47.3%	59.0%	72
Informal	41.7%	53.6%	51.0%	52.7%	41.0%	27
Ns/Nc	0.3%	-	-	-	0.0%	0.1
Fuente: elaboración propia en base a EPH-INDEC						

Finalmente, para exportar la tabla a un archivo .docx, utilizaremos la función `save_as_docx()`, asignándole la carpeta donde queremos guardar el archivo y el nombre del mismo.

```
save_as_docx(tabla_doc, path = "resultados/tabla2.docx")
```