

## UNIDAD N° 1

# ESTADÍSTICA DESCRIPTIVA

Manipulación básica de datos en R.  
Introducción al análisis exploratorio de datos.  
Uso de librerías. Trabajo con Data Set. Uso de  
funciones.

## INTRODUCCIÓN

En esta guía correspondiente a la semana 2, trabajaremos con el lenguaje de programación R para construir tablas de frecuencia a partir de datos cualitativos y cuantitativos. ¿Por qué usar R? Porque nos permite automatizar, repetir y escalar los cálculos que en otras circunstancias haríamos a mano, y lo hace de forma precisa, rápida y reproducible.

Pero más allá de eso, queremos que te acerques al pensamiento estadístico desde la lógica de la programación: leer los datos, transformarlos y comprender qué nos están diciendo.

Esta guía está pensada para que no copies y pegues el código, sino que los escribas vos mismo/a en RStudio. Esa práctica activa es la mejor manera de aprender a programar y analizar datos.

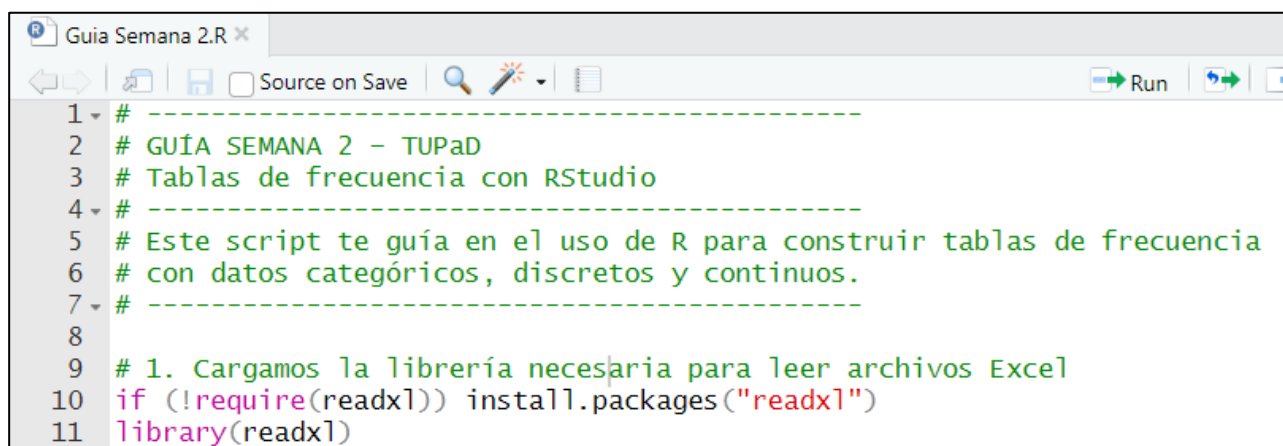
### ¿Qué vamos a trabajar?

- Lectura de datos desde archivos Excel con la librería `readxl`
- Construcción de tablas de frecuencia para variables categóricas y discretas
- Construcción de tablas de frecuencias agrupadas para variables continuas

### Actividad N° 1

En esta actividad vamos a trabajar con un archivo Excel que contiene información sobre 50 estudiantes.

Primero vamos a instalar y cargar la librería `readxl`, que permite importar archivos Excel (.xlsx) a R.



```
1 # -----  
2 # GUÍA SEMANA 2 - TUPaD  
3 # Tablas de frecuencia con RStudio  
4 # -----  
5 # Este script te guía en el uso de R para construir tablas de frecuencia  
6 # con datos categóricos, discretos y continuos.  
7 # -----  
8  
9 # 1. Cargamos la librería necesaria para leer archivos Excel  
10 if (!require(readxl)) install.packages("readxl")  
11 library(readxl)
```

Luego leeremos el archivo “Datos estudiantes de programación.xlsx” para generar una tabla de frecuencia.

```
12  
13 # 2. Carga de archivo Excel desde tu equipo  
14 # Aparecerá una ventana: buscá el archivo "Datos estudiantes de programación.xlsx"  
15 archivo <- file.choose()  
16 datos <- read_excel(archivo)
```

Este archivo contiene información de 50 estudiantes de programación. Incluye las siguientes variables:

- **Estudiante:** nombre de la persona
- **Horas\_Codificacion\_Semana:** horas promedio dedicadas a programar semanalmente
- **Lenguaje\_Favorito:** lenguaje de programación preferido
- **Proyectos\_Completados:** cantidad de proyectos finalizados

Vamos a leer este archivo y construir dos tipos de tablas de frecuencias:

1) Para Lenguaje\_Favorito, una variable cualitativa categórica:

Usamos `table()` para contar cuántos estudiantes eligen cada lenguaje y de esta forma obtener las frecuencias absolutas para cada categoría.

```
21 # Tabla de frecuencia para variable categórica  
22 # Contamos cuántas veces aparece cada lenguaje favorito  
23 tabla_lenguajes<-table(datos$Lenguaje_Favorito)  
24  
25 # Mostramos la tabla de frecuencias  
26 tabla_lenguajes
```

```
> # Mostramos la tabla de frecuencias  
> tabla_lenguajes  
  
      C++      Java JavaScript      Otros      Python  
      9       10         5         1       26  
  
>
```

2) Para `Proyectos_Completados`, una variable cuantitativa discreta:

Creamos la tabla de frecuencias:

```
30 # Tabla de frecuencia para variable discreta: Proyectos_Completados
31 tabla <- table(datos$Proyectos_Completados)
32 f_acum <- cumsum(tabla)
33 f_rel <- prop.table(tabla)
34 f_rel_acum <- cumsum(f_rel)
```

Que hacemos en cada línea:

- Usamos la función `table()` para contar cuántas veces aparece cada valor distinto de la variable `Proyectos_Completados`. El resultado es una tabla de frecuencias absolutas como la creada para la variable `Lenguaje_Favorito`.
- La función `cumsum()` calcula la suma acumulada de las frecuencias absolutas. Así obtenemos cuántos casos hay desde el valor más bajo hasta cada valor sucesivo.
- La función `prop.table()` convierte las frecuencias absolutas en proporciones (frecuencia relativa).
- Volvemos a usar `cumsum()`, esta vez sobre las proporciones, para obtener la frecuencia relativa acumulada.

Con los cálculos anteriores construimos la tabla final:

```
36 tabla_frecuencia <- data.frame(
37   Proyectos = names(tabla),
38   Frec = as.vector(tabla),
39   Frec_Acum = as.vector(f_acum),
40   Frec_Rel = round(as.vector(f_rel), 3),
41   Frec_Rel_Acum = round(as.vector(f_rel_acum), 3)
42 )
```

- Creamos un `data.frame()` para organizar toda la información en una única tabla.
- Utilizamos `names()` para obtener los nombres de las categorías y `as.vector()` para eliminar las etiquetas internas del vector generado por `table()` y quedarnos solo con los valores de frecuencia, lo mismo para el resto de las frecuencias.
- Además, usamos `round(..., 3)` para redondear las proporciones a tres cifras

decimales y facilitar la lectura.

Para terminar, mostramos nuestra tabla de frecuencias:

```
44 tabla_frecuencia
45
46 # Mostrar la tabla sin los índices de fila
47 print(tabla_frecuencia, row.names = FALSE)
48
```

```
> tabla_frecuencia
  Proyectos Frec Frec_Acum Frec_Rel Frec_Rel_Acum
0          0   3         3   0.059         0.059
1          1   6         9   0.118         0.176
2          2   7        16   0.137         0.314
3          3  10        26   0.196         0.510
4          4   9        35   0.176         0.686
5          5   2        37   0.039         0.725
6          6   8        45   0.157         0.882
7          7   6        51   0.118         1.000
> # Mostrar la tabla sin los índices de fila
> print(tabla_frecuencia, row.names = FALSE)
  Proyectos Frec Frec_Acum Frec_Rel Frec_Rel_Acum
         0   3         3   0.059         0.059
         1   6         9   0.118         0.176
         2   7        16   0.137         0.314
         3  10        26   0.196         0.510
         4   9        35   0.176         0.686
         5   2        37   0.039         0.725
         6   8        45   0.157         0.882
         7   6        51   0.118         1.000
> |
```

## **Actividad N° 2**

En esta actividad veremos cómo construir una tabla de frecuencias para datos agrupados para estudiar una variable cuantitativa continua, el tiempo (en minutos) que tardan distintos estudiantes de programación en resolver un trabajo práctico.

Vamos a simular estos tiempos y construir una tabla de frecuencias agrupadas aplicando la regla de Sturges.

Usamos la función `rnorm()` para simular los tiempos (en minutos) de 47 estudiantes, con una media de 55 minutos y una desviación estándar de 15 (ya veremos estos conceptos la semana próxima), lo que interesa ahora es que veas como generar datos aleatorios en R. Redondeamos los valores a 1 decimal y reemplazamos los negativos por 0 (no se puede tener tiempo negativo).

```
2 # Fijamos la semilla para obtener siempre los mismos resultados
3 set.seed(123)
4
5 # Generamos los datos: 47 tiempos simulados en minutos
6 tiempos <- round(rnorm(47, mean = 55, sd = 15), 1)
7
8 # Reemplazamos valores negativos por cero
9 tiempos <- ifelse(tiempos < 0, 0, tiempos)
10
11 # Mostramos los datos simulados
12 tiempos
```

Como estudiante de programación deberás desarrollar habilidades de búsqueda de información sobre el lenguaje que estás utilizando, por eso en este código hay dos cuestiones que tendrás que averiguar, ¿por que utilizamos `set.seed(123)` y cual es la estructura para utilizar `ifelse` en R?

Para agrupar los datos en intervalos, primero necesitamos saber cuántas clases usar. Usamos la regla de Sturges:  $k = 1 + 3.322 * \log n$

Usamos la función `length()` para contar cuántos datos hay en el vector `tiempos`. Esta función devuelve el número total de elementos en un objeto, y es útil para saber cuántas observaciones tenemos.

La función `ceiling()` se usa para redondear hacia arriba y obtener un número entero de clases.

```
14 # Cantidad total de observaciones
15 n <- length(tiempos)
16
17 # Número de clases según la regla de Sturges
18 k <- ceiling(1 + 3.322 * log10(n))
19 k
20
```

Para calcular el rango y la amplitud de clase utilizamos la función `range()` y luego calculamos la amplitud de cada clase.

```
21 rango <- range(tiempos)
22 amplitud <- ceiling((rango[2] - rango[1]) / k)
23 rango
24 amplitud
25
```

Usamos `seq()` para generar los límites de clase (de menor a mayor,

cada amplitud unidades) y luego clasificamos los datos con `cut()`.

¿Qué hace la función `seq()`?

La función `seq()` se usa para generar secuencias numéricas. En este contexto, nos permite definir los límites de los intervalos de clase para agrupar los datos.

- `floor(rango[1])`: redondea hacia abajo el mínimo del vector.
- `ceiling(rango[2]) + amplitud`: asegura que el último valor quede dentro del último intervalo.
- `by = amplitud`: define la distancia entre cortes (el ancho de cada clase).

Este vector `breaks` define los puntos de corte para los intervalos.

```
26 breaks <- seq(floor(rango[1]), ceiling(rango[2]) + amplitud, by = amplitud)
27 clases <- cut(tiempos, breaks = breaks, right = FALSE)
28 head(clases)
```

La función `cut()` toma un vector numérico y lo divide en intervalos definidos por el usuario mediante el argumento `breaks`. Luego asigna a cada valor un factor que indica a qué intervalo pertenece.

- `tiempos`: es el vector de datos que queremos agrupar.
- `breaks`: es un vector con los límites de los intervalos calculado en la línea anterior.
- `right = FALSE`: indica que los intervalos incluyen el extremo izquierdo, pero excluyen el derecho.

Como ya vimos en la sección pasada, esto es importante para que no haya superposición de clases y cada valor caiga en un único intervalo.

Además, `cut()` genera una variable categórica con etiquetas de los intervalos que se puede usar directamente en tablas o gráficos.

Ahora sí, construyamos con toda esta información la tabla de frecuencias:

```
30 tabla_tiempos <- table(clases)
31 f_acum <- cumsum(tabla_tiempos)
32 f_rel <- prop.table(tabla_tiempos)
33 f_rel_acum <- cumsum(f_rel)
```

Nuevamente usamos `data.frame()` para combinar todo en una tabla clara, redondeando las frecuencias relativas con `round()` y mostramos finalmente nuestra

tabla:

```
35 tabla_final <- data.frame(  
36   Intervalo = levels(clases),           # los nombres de los intervalos  
37   Frecuencia = as.vector(tabla_tiempos), # los datos sin nombres internos  
38   Frec_Acumulada = as.vector(f_acum),  
39   Frec_Relativa = round(as.vector(f_rel), 3),  
40   Frec_Rel_Acum = round(as.vector(f_rel_acum), 3)  
41 )  
42  
43 tabla_final
```

```
> tabla_final  
Intervalo Frecuencia Frec_Acumulada Frec_Relativa Frec_Rel_Acum  
1 [25,34)          2             2         0.043         0.043  
2 [34,43)          6             8         0.128         0.170  
3 [43,52)         15            23         0.319         0.489  
4 [52,61)          6            29         0.128         0.617  
5 [61,70)         11            40         0.234         0.851  
6 [70,79)          4            44         0.085         0.936  
7 [79,88)          3            47         0.064         1.000  
8 [88,97)          0            47         0.000         1.000  
> |
```

¿Qué significa esa columna de números a la izquierda de la tabla?

Cuando imprimís una tabla en R (como `tabla_final`), lo primero que aparece a la izquierda es una columna sin título que contiene los números de fila.

Esa columna:

- No es una variable ni forma parte del `data.frame`.
- Es solo una ayuda visual que R te muestra para que sepas en qué posición está cada fila.
- Es útil si querés pedirle a R “Mostrame la fila 4”, lo que harías con `tabla_final[4, ]`

**¿Por qué construimos la tabla con ese código?**

Cuando creamos una tabla de frecuencias agrupadas con `data.frame()`, a veces R agrega columnas de más o pone nombres confusos como `Frec_Relativa.clases`. Esto pasa porque algunos objetos que usamos como columnas ya tienen nombres internos (como etiquetas) y R los trata como si fueran columnas nuevas.

Por ejemplo:

- `table(clases)` no es solo un vector de números, es un objeto con nombres (los



intervalos).

- Lo mismo ocurre con `prop.table()`: sus resultados tienen nombres incorporados que coinciden con los niveles de clase.

Entonces, si usás directamente eso en `data.frame()`, R lo combina como si fueras a mostrar los datos y también los nombres, generando columnas duplicadas o confusas.

Por eso usamos estas funciones:

- `levels(clases)`: extrae solo los nombres de los intervalos, en orden, sin duplicarlos.
- `as.vector(...)`: convierte cada tabla o resultado en un simple vector de números, sin etiquetas.

De esta forma, armamos la tabla nosotros, con los nombres y columnas que queremos, y nada más.

### ¿Qué pasa si no tenemos cuidado? (Un ejemplo de mala práctica)

Si no usamos `levels()` ni `as.vector()` como explicamos antes, la tabla se va a construir con columnas confusas o duplicadas. Mirá este ejemplo que, aunque se ejecuta sin error, genera una tabla mal formada:

```
46 | tabla_final <- data.frame(  
47 |   Intervalo = names(tabla),  
48 |   Frecuencia = as.vector(tabla),  
49 |   Frec_Acumulada = f_acum,  
50 |   Frec_Relativa = round(f_rel, 3),  
51 |   Frec_Rel_Acum = round(f_rel_acum, 3)  
52 | )  
53 |
```

¿Qué está mal acá?

- `names(tabla)` extrae las etiquetas del objeto `tabla`, pero esas ya están dentro del objeto como nombres de fila. Esto genera columnas duplicadas o con conflictos.
- Además, algunos objetos pueden tener estructuras internas que complican la visualización si no los “limpiamos” con `as.vector()`.

Ejecuta este último código y observa estas diferencias.

### **Conclusión**

Puede parecer que el código “funciona”, porque no lanza errores y devuelve algo, pero el resultado no es correcto ni prolijo.

Esto es un claro ejemplo de que, en programación (y especialmente en estadística), no alcanza con que el código corra sin errores, también debe dar un resultado claro, preciso y correctamente estructurado.

En estadística, las tablas y gráficos que generamos no son solo para nosotros, son la base de los informes que compartimos con otros. Por eso, es fundamental que la información que mostramos sea:

- Prolija en su presentación
- Clara en su estructura
- Correcta en su contenido

Una tabla mal construida puede generar confusión o interpretaciones equivocadas. En cambio, una tabla bien diseñada facilita la lectura, el análisis y la toma de decisiones.

Programar bien también es comunicar bien.