

Muestreo y Encuestas Complejas en R

Sesión 5: Análisis con `srvyr` y `survey`

Gabriel Sotomayor

2025-04-07



Objetivos de la Sesión

En esta segunda sesión, nos enfocaremos en **aplicar** lo aprendido:

1. **Repasar** brevemente por qué necesitamos análisis especiales para encuestas complejas.
2. **Entender el concepto** de “objeto de diseño muestral” en R.
3. Conocer las **funciones clave** del paquete **survey** para análisis descriptivo y modelos, entendiendo su lógica.
4. Aprender a usar el paquete **srvyr** para realizar análisis de forma **más intuitiva** (estilo **tidyverse**).
5. **Calcular e interpretar** correctamente estimaciones (medias, proporciones, totales) y sus **errores estándar / ICs** en R.
6. Realizar **análisis por subgrupos** de manera eficiente.



Recordando: ¿Por Qué Análisis Especiales?

- Las encuestas reales (como CAsEN) **no usan Muestreo Aleatorio Simple (MAS)**.
- Usan diseños **complejos** (estratos, conglomerados, ponderadores) por razones prácticas (costos, logística, representatividad).
- Esto **viola los supuestos** de las funciones estadísticas estándar de R (**mean**, **lm**, **t.test**...).

Consecuencias de Ignorar el Diseño:

1. Estimaciones puntuales **sesgadas** (por no usar ponderadores).
2. Errores estándar, ICs y p-valores **incorrectos** (generalmente subestimados, por ignorar conglomerados/estratos).

Necesitamos herramientas que “entiendan” el diseño.



La Solución: Análisis Consciente del Diseño

Para hacer análisis válidos, debemos usar paquetes de R diseñados específicamente para encuestas complejas.

El principal es el paquete **survey**.

La idea central de **survey** es:

1. **Primero, describirle a R CÓMO fue hecha la muestra**, creando un objeto especial que contenga toda la información del diseño.
2. **Luego, usar funciones específicas** que lean la información de ese objeto para realizar los cálculos correctamente.



El Corazón de **survey**: El Objeto de Diseño Muestral

- Piensa en este objeto como una “**receta**” o un “**manual de instrucciones**” para R sobre tu muestra.
- **NO** es solo la tabla de datos. Es un objeto más complejo que **contiene**:
 - Los **datos** mismos.
 - La información sobre los **conglomerados** (UPMs, variable **ids**).
 - La información sobre los **estratos** (variable **strata**).
 - La información sobre los **ponderadores** (variable **weights**).
 - (Opcional) Información sobre población finita (FPC), etapas de muestreo, etc.
- Este objeto es creado **UNA VEZ** (generalmente al inicio del análisis) usando **`survey::svydesign()`** (o **`srvyr::as_survey_design()`**).
- **TODAS** las funciones de análisis posteriores (**`svymean`**, **`svytotal`**, **`svyglm`**...) **necesitan** este objeto como argumento para funcionar correctamente.



Declarando el Diseño:

`survey::svydesign()` (Repaso)

Recordemos cómo creamos este objeto “receta” para CASEN en el práctico anterior:

```
1 casen_design <- svydesign(  
2   ids = ~varunit,      # Le dice a R cuál es la variable de UPMs/Conglomerados  
3   strata = ~varstrat,  # Le dice a R cuál es la variable de Estratos  
4   weights = ~expr,     # Le dice a R cuál es la variable de Ponderadores  
5   data = casen,        # Le dice a R dónde están los datos crudos  
6   nest = TRUE          # Le dice a R que los IDs de UPMs pueden repetirse entre estratos  
7 )
```

- Cada argumento (`ids`, `strata`, `weights`) le da una pieza clave de la “receta” a R.
- El resultado (`casen_design`) encapsula toda esta información.



Entendiendo la Tilde (~) en R

- Han visto la tilde (~) antes, probablemente en modelos como `lm(y ~ x, data=...)`.
- **Función Clave:** La tilde en R se usa para crear **fórmulas**. Una fórmula **describe una relación o especifica variables** *sin* evaluarlas inmediatamente.
- En `svydesign(ids = ~varunit)`:
 - No queremos que R calcule algo con `varunit` en ese momento.
 - Queremos decirle a `svydesign`: “La variable que identifica los conglomerados se llama `varunit` y está *dentro* de los datos que te pasé en `data=casen`”.
 - El `~` logra esto: **nombra la variable dentro del contexto del data frame.**
- En `svymean(svymean(~edad, ...))`:
 - Similarmente, le dice a `svymean`: “Calcula la media de la variable que se llama `edad` que se encuentra *dentro* del objeto `design` que te estoy pasando”.

Es la forma estándar en R para referirse a variables dentro de funciones que operan sobre data frames o entornos específicos.



Usando el Objeto de Diseño: Funciones de `survey`

Una vez creado el `casen_design`, podemos usar las funciones de `survey`. Todas ellas toman el objeto `design` como argumento principal.

- `svymean(~variable, design, na.rm=T)`:
 - Calcula la media (si `variable` es numérica) o proporciones (si `variable` es factor/categórica/dummy).
 - **Importante:** Usa los `weights` del `design` para la ponderación y los `strata` e `ids` para calcular el error estándar (SE) correcto.



Usando el Objeto de Diseño: Funciones de **survey**

- **svytotal(~variable, design, na.rm=T):**
 - Estima el total poblacional de **variable** (ej. suma ponderada de ingresos para estimar el ingreso total del país).
 - También calcula el SE correcto.
- **svyquantile(~variable, design, quantiles=..., na.rm=T):**
 - Calcula cuantiles ponderados (mediana, percentiles).
 - El cálculo del SE para cuantiles es más complejo, pero **survey** lo maneja.



Análisis por Subgrupos con `survey`: `svyby()`

Para calcular estadísticas por grupo (ej. media de ingreso por región), `survey` ofrece `svyby()`.

```
1 # Calcular la media de 'edad' para cada nivel de la variable 'sexo'
2 svyby(
3   formula = ~edad,      # La variable a analizar
4   by = ~sexo,           # La variable que define los grupos
5   design = casen_design, # El objeto de diseño
6   FUN = svymean,        # ¡La función que queremos aplicar a cada grupo!
7   na.rm = TRUE          # Argumento para pasar a svymean
8 )
```

- **Lógica Funcional:** `svyby` es interesante porque toma *otra función* (`svymean` en este caso) como argumento (`FUN`).
- **¿Qué hace?** Divide internamente el `design` según los niveles de `by` y aplica `FUN` (`svymean`) a cada subconjunto, luego junta los resultados.
- Es potente pero la sintaxis (`FUN = ...`) puede ser menos familiar que el `group_by %>% summarise` de `dplyr`.



Modelos Estadísticos con **survey**: **svyglm()**

Para regresión lineal, logística, etc., que consideren el diseño:

```
1 # Modelo lineal: predecir 'esc' (escolaridad) usando 'edad' y 'sexo'
2 modelo_complejo <- svyglm(esc ~ edad + factor(sexo),
3                           design = casen_design,
4                           family = gaussian()) # Para regresión lineal
5
6 summary(modelo_complejo)
7 # OJO: Los errores estándar y p-valores serán diferentes a un lm()
```

- **Sintaxis:** Similar a `glm()` o `lm()`, pero requiere el argumento `design`.
- **Resultado Clave:** Los errores estándar de los coeficientes (β) se calculan **correctamente**, usando la información de estratos, conglomerados y pesos. Esto afecta directamente los **p-valores** y las conclusiones sobre la significancia estadística.



Entra **srvyr**: El Puente hacia el Tidyverse

Si bien **survey** es completo y fundamental, su sintaxis (especialmente para subgrupos) puede sentirse diferente si vienes del mundo **tidyverse**.

El paquete **srvyr** actúa como un “traductor” o “adaptador”:

- Permite usar la sintaxis familiar de **dplyr** (verbos como **group_by**, **summarise**, **mutate**) para realizar análisis de encuestas complejas.
- Internamente, **srvyr** llama a las funciones de **survey**, asegurando que los cálculos sean correctos.
- Hace el código a menudo más legible y fácil de escribir para operaciones comunes.



srvyr: Creando el Objeto `tbl_svy`

El primer paso con `srvyr` es crear un objeto especial `tbl_svy`, análogo al `survey.design`. Se usa `as_survey_design()` (o simplemente `as_survey`).

```
1 # Usando los mismos argumentos que svydesign, pero estilo tidyverse
2 casen_design_srvyr <- as_survey_design(casen,
3                                     ids = varunit,      # Sin tilde, nombre directo
4                                     strata = varstrat,   # Sin tilde
5                                     weights = expr,     # Sin tilde
6                                     nest = TRUE)
7
8 print(casen_design_srvyr)
```

- Notar que ahora pasamos los nombres de las variables **directamente**, sin la tilde (~), como es usual en `dplyr`.
- El objeto `casen_design_srvyr` ahora puede ser usado en un pipeline `%>%`.



srvyr: El Flujo `group_by %>% summarise`

La gran mayoría de los análisis descriptivos con `srvyr` siguen este patrón:

```
1 objeto_srvyr %>%
2   # Opcional: Filtrar casos si es necesario ANTES de agrupar/resumir
3   # filter(condicion) %>%
4
5   # Agrupar por las variables deseadas (si aplica)
6   group_by(variable_grupo1, variable_grupo2) %>%
7
8   # Calcular las estadísticas deseadas DENTRO de summarise
9   summarise(
10     nombre_stat1 = survey_mean(variable1, na.rm = TRUE, vartype = "ci"),
11     nombre_stat2 = survey_total(variable2, na.rm = TRUE),
12     nombre_stat3 = survey_quantile(variable1, quantiles = 0.5, na.rm = TRUE)
13     # etc.
14   )
```

- ¡Igual que `dplyr`! Pero usando las funciones `survey_*()` dentro de `summarise`.



srvyr: Funciones Espejo survey_*()

Dentro de `summarise`, usamos las funciones de `srvyr` que “imitan” a las de `survey`:

- `survey_mean()`: Medias (continuas) y proporciones (categóricas/dummies).
- `survey_total()`: Totales poblacionales.
- `survey_quantile()`: Cuantiles (incluye mediana si `quantiles=0.5`).
- `survey_median()`: Atajo para la mediana.
- `survey_ratio(num, den)`: Ratios.
- `survey_var()` / `survey_sd()`: Varianza / Desv. Estándar poblacional (menos común).



srvyr: Funciones Espejo survey_*()

Fuera de `summarise`:

- `survey_count(...)` / `survey_tally()`: Conteos poblacionales estimados (similares a `dplyr::count`).

Argumentos Importantes:

- * `na.rm = TRUE`: ¡Fundamental para manejar valores perdidos!
- * `vartype = c("se", "ci", "var", "cv")`: Para pedir el tipo de medida de variabilidad (error estándar, intervalo confianza, varianza, coef. variación). Por defecto es "se".
- * `level = 0.95`: Para cambiar el nivel de confianza del IC.



Comparación Final: **survey** vs **srvyr** (Media edad por sexo)

Usando **survey** (**svyby**):

```
1 svyby(~edad, by = ~sexo, design = casen_design, FUN = svymean, na.rm = TRUE)
```

Usando **srvyr**:

```
1 casen_design_srvyr %>%  
2   group_by(sexo) %>%  
3   summarise(edad_media = survey_mean(edad, na.rm = TRUE))
```

Ambos dan el mismo resultado numérico, pero **srvyr** se integra más fluidamente en un flujo de trabajo **tidyverse**. ¡En el práctico usaremos principalmente **srvyr**!



Interpretando Resultados: SE, DEFF, IC (Recordatorio)

No importa si usas **survey** o **srvyr**, el resultado siempre incluirá:

1. **Estimación Puntual Ponderada:** Tu mejor estimación del valor poblacional.
2. **Error Estándar (SE) Correcto:** La medida de incertidumbre, calculada considerando el diseño complejo.

Recuerda:

- El SE suele ser **mayor** que en MAS debido al **Efecto de Diseño (DEFF)**, especialmente por los **conglomerados**.
- Usa el SE para construir **Intervalos de Confianza (ICs)** (**vartype="ci"** lo hace por ti) que te dan un rango plausible para el valor poblacional.
- **Reporta siempre la incertidumbre (SE o IC) junto a tus estimaciones.**



Próximos Pasos: ¡A Practicar con **srvyr**!

En el bloque práctico de hoy:

1. Crearemos el objeto `tbl_svy` para CAsEN usando `as_survey_design`.
2. Calcularemos **medias, proporciones y totales** ponderados con `srvyr`.
3. Practicaremos la obtención e interpretación de **errores estándar e intervalos de confianza**.
4. Realizaremos **análisis por subgrupos** usando `group_by`.
5. Construcción de gráficos integrando ggplot.

¡Volvamos a RStudio!



