

Dplyr

Rodrigo Negrete Pérez

August 6, 2022

- 1 Intro al Tidyverse
- 2 Lectura de bases de datos
- 3 Data Pliers
- 4 Verbos
- 5 Agrupar datos
- 6 Otras funciones
- 7 Merges y cambio de estructura

Section 1

Intro al Tidyverse

Un mundo de datos

- Podemos crear nuestras bases de datos en R
- Sin embargo, su atractivo es manipular datos ya existentes
- Nuestro siguiente paso es descargar bases de datos y prepararlas para el análisis

Paquetes

- Una de las ventajas de R: al ser gratis, hay muchos paquetes con funciones especializadas.
- A menudo, autores publican sus métodos de estimación como paquetes.
- Antes de hacer algo sofisticado, revisa antes si no existe ya un paquete que lo hace.
 - Por ejemplo, muchos loops se pueden hacer con la familia de funciones “apply”

Tidyverse

- Tidyverse es una familia de paquetes estadísticos.
- Vamos a ver “dplyr” -> Data pliers.
- Dedicaremos otras sesión a ““ggplot””.

Instalación de paquetes

- Para todos los paquetes: primero debemos INSTALARLOS UNA SOLA VEZ.
- Encerramos el nombre del paquete entre comillas

```
install.packages('tidyverse')
```

- Antes de usarlos, CARGARLOS CADA VEZ que abramos sesión.

```
library(tidyverse)
```

- Cuando lo cargamos con library() lo escribimos sin comillas.

Section 2

Lectura de bases de datos

Bases de R

- R ya tiene incorporadas algunas bases de datos en el paquete `data`
- Si ejecutamos **`data()`** podemos ver la lista de datos

- Están precargados y podemos acceder a ellos solo con su nombre

```
sleep<- sleep  
cars<- mtcars
```

Descargar bases de datos

- Lo más común es que trabajemos con bases de datos:
 - Agencias del Estado
 - Investigadores
 - Sector Privado

Directorio

- Antes de leer bases de datos, necesitamos hablar un poco de los directorios.
- Hay muchas maneras para leer bases de datos:
 - Una para cada formato: excel, stata, csv, etc.
- Pero en todas tenemos que especificar dónde está el archivo: el path

Checar directorio

- podemos verificar el directorio con `wd()`

```
getwd()
```

```
## [1] "C:/Users/rodri/OneDrive - INSTITUTO TECNOLOGICO AUTON"
```

- Por default:
 - R va a buscar archivos en esta carpeta
 - También va a subir archivos a esta carpeta

Ejemplo

- Usaremos la base del INE de la elección presidencial de 2018
- La pueden descargar del Github
 - RodrigoNP/Curso_R
- **Primero debemos descargar el archivo en nuestra carpeta de elección**

Modificando directorio

- Si elegimos modificar el directorio usamos `setwd()` y ponemos dentro el path de nuestra carpeta.

```
setwd('C:/Users/rodri/OneDrive - INSTITUTO TECNOLOGICO AUTONOM
```

- La última diagonal es la última carpeta en la que está.
- **Observa que es un forward slash**

- Para abrir un archivo excel necesitamos hacer uso de otro paquete: readxl, dentro del tidyverse.
- Una vez modificado el directorio, solo ponemos el nombre y formato del archivo entre como texto (entre comillas)

```
presidencial<- readxl::read_excel('Data/presidencia.xlsx',  
                                   skip = 5)
```


- Fíjate en cómo nombro la función y que guardo el df como una variable
 - Podemos citar un paquete con ::
 - Nombramos el paquete :: la función que queremos llamar directamente
- El archivo no puede estar abierto

- Podemos especificar donde empezar a leer los datos con el argumento **skip**
- Podemos especificar cuales son los NA con `na=""`

Sin modificar directorio

- Si no modificamos el directorio, tenemos que poner todo el path y el nombre del archivo

```
min<- readxl:: read_excel('C:/Users/rodri/OneDrive - INSTITUTO
```

NA's

- Todas las bases de datos tiene NA.
- Denotan celdas que están vacías, no tienen dato. NO SON 0.
- Podemos especificar qué valores fungen como NA al descargar la base de datos, añadiendo `na=c('0', '99', 'etc')`

Section 3

Data Pliers

Data Pliers

- El paquete está pensado para preparar los datos
- El objetivo es que una fila sea una observación y una columna sea una variable
- El paquete dplyr tiene una sintaxis sencilla a lo largo de varias funciones

Pipe (pipa)

- El operador más común es el “pipe”, %>%
- Lo podemos escribir fácilmente con Ctrl+ shift+ m
- Quiere decir: *a los datos de la izquierda (pipa) le voy a aplicar el verbo de la derecha.*
- no obstante, podemos omitir la pipa si ponemos la variable del df como primer argumento de la función

Section 4

Verbos

- Los verbos de dpliers también funcionan sin la pipa
- Tienen una sintaxis similar:
 - El primer argumento es un df
 - El segundo argumento describe qué hacer con el df
 - El resultado es un nuevo df
- Los verbos nunca modifican los inputs, así que es necesario crear un nuevo objeto.

Rename()

- Podemos renombrar variables para manipularlas más fácilmente

```
presidencial<-presidencial %>%  
  rename(votos=TOTAL_VOTOS_CALCULADOS,  
         lista_nominal=LISTA_NOMINAL_CASILLA)
```

- El primer argumento es el nuevo nombre que queremos; el segundo, el nombre original de la variable
- Podemos separar todas las transformaciones con comas

mutate()

- `mutate()` añade una variable (columna).
- Su atractivo es que podemos utilizar operadores lógicos que relacionen variables dentro de una observación.

- Añadamos el porcentaje de participación

Sin la pipa

- O podríamos poner

```
presidencial<-mutate(presidencial,  
                      participacion=votos/  
                      lista_nominal)
```

- Esto es válido para todos los verbos del paquete dplyr
 - Se puede o no usar la pipa
 - La pipa es útil para concatenar

| | CLAVE_CASILLA | participacion | CLAVE_ACTA | ID_ESTADO | N |
|---|---------------|---------------|---------------|-----------|---|
| 1 | '010000M0100' | 0.50 | 010000M010001 | 1.00 | A |
| 2 | '010338B0100' | 0.57 | 010338B010002 | 1.00 | A |
| 3 | '010338C0100' | 0.57 | 010338C010002 | 1.00 | A |
| 4 | '010338C0200' | 0.52 | 010338C020002 | 1.00 | A |
| 5 | '010339B0100' | 0.59 | 010339B010002 | 1.00 | A |
| 6 | '010339C0100' | 0.58 | 010339C010002 | 1.00 | A |

ifelse()

- Si bien ifelse() no es parte del paquete dplyr, es muy útil para la econometría
- En consonancia con mutate(), podemos hacer dummies muy fácilmente

```
df<-mutate(df,  
            dummy=ifelse(condicion a cumplir,  
                           que pasa si se cumple,  
                           que poner si no))
```

Hagamos una dummy que indique si la casilla es rural

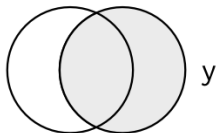
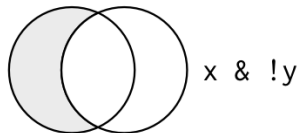
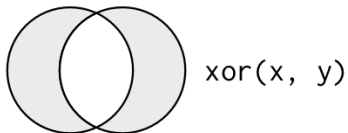
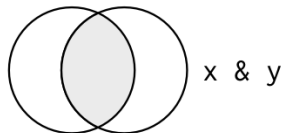
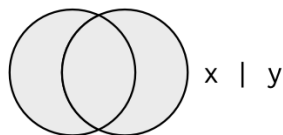
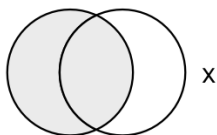
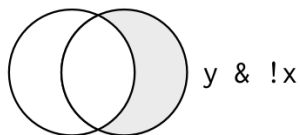
```
presidencial<-presidencial %>%  
  mutate(  
    rural=ifelse(CASILLA=='Rural',1,0)  
  )
```


| | CLAVE_CASILLA | rural | CLAVE_ACTA | ID_ESTADO | NOMBRE |
|---|---------------|-------|---------------|-----------|--------|
| 1 | '010000M0100' | | 010000M010001 | 1.00 | AGUASC |
| 2 | '010338B0100' | 1.00 | 010338B010002 | 1.00 | AGUASC |
| 3 | '010338C0100' | 1.00 | 010338C010002 | 1.00 | AGUASC |
| 4 | '010338C0200' | 1.00 | 010338C020002 | 1.00 | AGUASC |
| 5 | '010339B0100' | 1.00 | 010339B010002 | 1.00 | AGUASC |
| 6 | '010339C0100' | 1.00 | 010339C010002 | 1.00 | AGUASC |

filter()

- filter() ayuda a quedarnos con observaciones que cumplan con ciertos criterios
- Se quedan las observaciones que produzcan un TRUE dentro de los paréntesis.
- Sirve para crear nuevas bases de datos a partir de las observaciones que nos interesan del df anterior

Operadores lógicos



Iguales

- Para las comparaciones, se usa `==`
- usar `=` daría un error
- R utiliza un sistema de punto flotante, así que cuidado con sus comparaciones
- Pueden usar **`near()`** para hacer comparaciones especificando la tolerancia

```
sqrt(2)^2==2
```

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

```
near(sqrt(2)^2, 2)
```

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

Ley de de Morgan

No a o no b es lo mismo que no a y no b

En lenguaje de R

$[!(x \& y) = !x \mid !y]$

- Otro operador muy util es `%in%`

Ejemplo filter()

- Creemos una nueva base de datos en la que solo haya casillas en las que se haya votado por algún independiente

```
independientes<-presidencial %>%  
  filter(CAND_IND_01>0 | CAND_IND_02>0)
```

| | CAND_IND_01 | CAND_IND_02 | CLAVE_CASILLA | CLAVE_ACTA |
|---|-------------|-------------|---------------|-------------|
| 1 | 0.00 | 26.00 | '010000M0100' | 010000M0100 |
| 2 | 0.00 | 14.00 | '010338B0100' | 010338B0100 |
| 3 | 0.00 | 18.00 | '010338C0100' | 010338C0100 |
| 4 | 0.00 | 13.00 | '010338C0200' | 010338C0200 |
| 5 | 0.00 | 14.00 | '010339B0100' | 010339B0100 |
| 6 | 0.00 | 14.00 | '010339C0100' | 010339C0100 |

select()

- `select()` hace lo mismo que `filter()`, pero con variables (columnas).
- Algunos auxiliares son:
 - `starts_with(' ')` para variables con un prefijo en común
 - `ends_with(' ')`
 - `contains`
 - `num_range('x', 1:3)` para `x1`, `x2`, `x3`
 - `everything()` toma el resto del `df`, muy útil para reacomodar

Ejemplo select()

Creemos un df que contenga la clave de la casilla, el estado y solo Morena y sus coaliciones

```
morena<- presidencial %>%  
  select(CLAVE_CASILLA, NOMBRE_ESTADO,  
         contains('MORENA'))
```

| | CLAVE_CASILLA | NOMBRE_ESTADO | MORENA | PT_MORENA |
|---|---------------|----------------|--------|-----------|
| 1 | '010000M0100' | AGUASCALIENTES | 378.00 | |
| 2 | '010338B0100' | AGUASCALIENTES | 153.00 | |
| 3 | '010338C0100' | AGUASCALIENTES | 159.00 | |
| 4 | '010338C0200' | AGUASCALIENTES | 136.00 | |
| 5 | '010339B0100' | AGUASCALIENTES | 151.00 | |
| 6 | '010339C0100' | AGUASCALIENTES | 132.00 | |

Concatenar verbos

- Podemos concatenar funciones de dplyr añadiendo pipas.
- Por ejemplo, podemos hacer lo mismo que anteriormente, pero en un solo paso.

```
presidencial<-presidencial %>%  
  rename(votos=TOTAL_VOTOS_CALCULADOS,  
         lista_nominal=LISTA_NOMINAL_CASILLA) %>%  
  mutate(participacion=votos/  
         lista_nominal) %>%  
  select(CLAVE_CASILLA, NOMBRE_ESTADO,  
         contains('MORENA'))
```

Otros verbos

Algunas otras funciones notables son

- **sample_n()** para seleccionar n observaciones aleatoriamente
- **sample_frac()** para seleccionar un porcentaje de observaciones aleatoriamente.
- **replace_na()** para reemplazar NA's

Section 5

Agrupar datos

group_by()

A veces las observaciones pertenecen a una o más categorías.

- Por ejemplo, los datos panel están agrupados por entidad y por tiempo.

R puede agrupar fácilmente con `group_by()`

- Después de agrupar podemos añadir variables o comprimir bases de datos.
- Podemos agrupar por más de una categoría separando por una coma.

Por ejemplo, agrupemos por estado y añadamos el promedio de votos por morena por estado

```
presidencial<- presidencial %>%  
  group_by(NOMBRE_ESTADO) %>%  
  mutate(  
    voto_morena= mean(MORENA, na.rm=T)  
  )
```

| | voto_morena | CLAVE_CASILLA | CLAVE_ACTA | ID_ESTADO | |
|---|-------------|---------------|---------------|-----------|---|
| 1 | 111.28 | '010000M0100' | 010000M010001 | 1.00 | A |
| 2 | 111.28 | '010338B0100' | 010338B010002 | 1.00 | A |
| 3 | 111.28 | '010338C0100' | 010338C010002 | 1.00 | A |
| 4 | 111.28 | '010338C0200' | 010338C020002 | 1.00 | A |
| 5 | 111.28 | '010339B0100' | 010339B010002 | 1.00 | A |
| 6 | 111.28 | '010339C0100' | 010339C010002 | 1.00 | A |

na.rm=T

- Los NA son contagiosos: casi todo lo que hagamos dará NA
- Filter no incluye los NA en la condición, así que hay que unir con `| is.na()`
- ¿Cómo manejamos los NA?
- Hay funciones que ignoran los NA
- Generalmente el argumento que incertamos es **na.rm=T**

summarise()

- Usamos `summarise()` si queremos comprimir la base de datos.
- La base de datos se modifica totalmente
- Las nuevas observaciones pasan a ser el/los argumento(s) por los que agrupamos.
- Solo tendremos las variables que especifiquemos dentro del `summarise()`

Auxiliares summarise()

- **across()** sirve para hacer la misma operación con varias variables con `mutate()` o `summarise()`
- Al meter las columnas podemos usar la sintaxis y auxiliares que usaríamos con `select()`: `begins_with()`, `contains()`, `everything()`, etc.
- **c_across()** indica que queremos hacer una operación utilizando varias variables
- `n()` sirve para añadir cuantas observaciones hay dentro de cada grupo

Ejemplo summarise() y across()

- Agrupemos por Estado y saquemos los votos totales por partido por estado

```
prom_estados<- presidencial %>%  
  group_by(NOMBRE_ESTADO) %>%  
  summarise(  
    across(PAN:CAND_IND_02, sum)  
  )
```

- Sin el across tendríamos que hacer summarise(pan=pan, prd=prd, etc)

| | NOMBRE_ESTADO | PAN | PRI | PRD | PVEN |
|---|---------------------|-----------|-----------|----------|---------|
| 1 | AGUASCALIENTES | 162349.00 | 88820.00 | 7442.00 | 6216.00 |
| 2 | BAJA CALIFORNIA | 239038.00 | 102272.00 | 13090.00 | 8873.00 |
| 3 | BAJA CALIFORNIA SUR | 51812.00 | 21631.00 | 1409.00 | 2797.00 |
| 4 | CAMPECHE | 43742.00 | 82300.00 | 4945.00 | 2950.00 |
| 5 | CHIAPAS | | | | |
| 6 | CHIHUAHUA | | | | |

row_wise()

- Hasta ahora, utilizamos mutate o summarise calculamos la operación a lo largo de la columna (la variable)
- El summarise() anterior es un claro ejemplo. Vimos al interior del Estado y sumamos los votos a lo largo de la columna

- **rowwise()** es un verbo que nos ayuda a hacer operaciones a lo largo de las filas.
- Si añadimos `rowwise()` le indicamos a dplyr que las operaciones las queremos a lo largo de la fila
- En nuestro ejemplo, sería muy útil para sumar los votos de las coaliciones

Ejemmplo rowwise()

- Calculemos el total de los votos obtenidos por MORENA

```
presidencial<- presidencial %>%  
  rowwise() %>%  
  mutate(  
    total_morena= sum(c_across(contains('MORENA')))- voto_morena  
  )
```

- Restamos voto_morena porque también lo incluiría y es un estadístico que sacamos

| | CLAVE_CASILLA | total_morena | CLAVE_ACTA | ID_ESTADO | |
|---|---------------|--------------|---------------|-----------|---|
| 1 | '010000M0100' | 433.00 | 010000M010001 | 1.00 | A |
| 2 | '010338B0100' | 160.00 | 010338B010002 | 1.00 | A |
| 3 | '010338C0100' | 162.00 | 010338C010002 | 1.00 | A |
| 4 | '010338C0200' | 143.00 | 010338C020002 | 1.00 | A |
| 5 | '010339B0100' | 154.00 | 010339B010002 | 1.00 | A |
| 6 | '010339C0100' | 136.00 | 010339C010002 | 1.00 | A |

across() vs c_across()

- Si queremos añadir múltiples columnas con, por ejemplo, el promedio de cada una de las variables, lo podemos hacer con `across()`
- Si queremos hacer operaciones cuyo output sea una sola variables, pero que necesite de input múltiples variables utilizamos `c_across`

Section 6

Otras funciones

Menciono otras funciones interesantes que pueden llegar a ser útiles al manejar datos y en conjunto con dplyr

case_when()

- Hay veces que queremos hacer un ifelse con varias condiciones
- concatenar un ifelse() se vuelve progresivamente difícil

```
case_when( condicion_1 ~ qué pone R si se cumple,  
condicion_2 ~ " ",  
...  
TRUE ~ qué pone R e.o.c. )
```

Ejemplo case_when()

- Creemos una variable que dependa de los valores que pueda tomar la participación que se llame participativo
 - Si es menor a 30%, que valga “baja”
 - Entre 30 y 60 que sea “promedio”
 - Mayor a 60 que diga “muy”


```
presidencial<- presidencial %>%  
  mutate(  
    participativo=case_when(  
      participacion<0.30 ~ 'poco',  
      participacion>0.30 & participacion<=0.60 ~ 'promedio',  
      participacion>0.60 ~ 'muy'  
    )  
  )
```

| | CLAVE_CASILLA | participativo | CLAVE_ACTA | ID_ESTADO | N |
|---|---------------|---------------|---------------|-----------|---|
| 1 | '010000M0100' | promedio | 010000M010001 | 1.00 | A |
| 2 | '010338B0100' | promedio | 010338B010002 | 1.00 | A |
| 3 | '010338C0100' | promedio | 010338C010002 | 1.00 | A |
| 4 | '010338C0200' | promedio | 010338C020002 | 1.00 | A |
| 5 | '010339B0100' | promedio | 010339B010002 | 1.00 | A |
| 6 | '010339C0100' | promedio | 010339C010002 | 1.00 | A |

recode()

- Hay veces que queremos cambiar el nombre de los factores en un df
- **recode()** preserva las categorizaciones, pero alterando los nombres de los factores.

factor()

- Podemos fácilmente convertir una variable caracter o numérica en factor con **factor()**
- Hay veces que queremos que sea un factor y no solo texto, especialmente cuando grafiquemos.

Section 7

Merges y cambio de estructura

Merges

Hay veces que queremos juntas bases de datos con las mismas observaciones.

- Por ejemplo, nuestra base de ministros con datos del PIB.
- Hay muchos tipos de merges y una familia de funciones para todos los casos
- Evidentemente, las categorías deseadas tienen que estar **escritas idénticamente**.

```
tipo_join(x,y,  
by=c('variable1.x'='variable1.y',  
'variable2.x'='variable2.y'))
```

Tipos de merge

- **left_join()** A la base de la izquierda le añade lo de la derecha. Observaciones solo en la base derecha se pierden.
- **right_join()**
- **inner_join()** la intersección de las bases de datos
- **full_join()** la unión

Cambio de estructura

Habr  veces que queremos cambiar la estructura de los datos. Pensemos en nuestro ejemplo. Cada partido es una variable, sin embargo, todos pertenecen a una misma categor a: partidos.

- Si pudi ramos cambiar la estructura de datos de casilla a casilla-partido podr amos, por ejemplo, agrupar por la variable partido

pivot_longer()

- `pivot_longer()` transforma la estructura de los datos
- Hace exactamente lo antes mencionado: pasa de un formato *wide* a uno *long*
- Veamos cómo pasar todos los partidos a una sola variable, cambiando la unidad de observación de casilla a casilla-partido

```
presidencial_long<- presidencial %>%  
  pivot_longer(PAN:CAND_IND_02,  
               names_to = 'partido',  
               values_to = 'votos_partido') %>%  
  select(CLAVE_CASILLA, partido, votos_partido, everything())
```

| | CLAVE_CASILLA | partido | votos_partido | CL |
|---|---------------|----------------------|---------------|----|
| 1 | '010000M0100' | PAN | 247.00 | 01 |
| 2 | '010000M0100' | PRI | 37.00 | 01 |
| 3 | '010000M0100' | PRD | 8.00 | 01 |
| 4 | '010000M0100' | PVEM | 1.00 | 01 |
| 5 | '010000M0100' | PT | 24.00 | 01 |
| 6 | '010000M0100' | MOVIMIENTO CIUDADANO | 4.00 | 01 |

- `pivot_longer()` es un verbo de dplyr
- el primer argumento son las variables que queremos convertir
 - acepta la sintaxis de `select()`
- el segundo argumento es el nombre de la columna en la que va a colapsar las variables anteriores
 - i.e. que categoría comparten esas variables
- El segundo argumento es cómo se va a llamar la columna en la que va a poner los valores

Así, de tener una `df` con una columna para cada partido y los votos como valores a lo largo de todas esas variables, terminamos con un `df` con una variable de partido y otra de los votos por ese partido, pero con muchas más observaciones. Lo que sucede es que se alteró la unidad de observación: de casilla a casilla-partido.

pivot_wider()

- Evidentemente, hay un verbo que hace lo contrario, que pasa de un formato long a uno wide
- Sirve para dividir categorías en múltiples columnas

```
presidencial_wide<- presidencial_long %>%  
  pivot_wider(names_from = partido,  
              values_from = votos_partido)
```

- Evidentemente, queda como nuestro df original
- Podemos descomponer varias categorías en múltiples variables si añadimos las variables como vector a `values_from=`
- Si quisiéramos hacer una variable por partido-estado

```
presidencial_wide<- presidencial_long %>%  
  pivot_wider(names_from = partido,  
              values_from = c(votos_partido, NOMBRE_ESTADO))
```

arrange()

- `arrange()` es un verbo que sirve para acomodar las observaciones en orden ascendente o descendente de acuerdo con un criterio (variable)

`arrange(variable que contienen letras o números)`

- Por default lo hace en orden ascendente
 - Si queremos descendente, tenemos que encerrar la columna entre **`desc()`**
- Podemos especificar si queremos ordenar al interior de cada grupo con el argumento `.by_group=T`

Auxiliares arrange()

- Podemos aprovechar que los datos ya estén en orden
- `first()` toma la primera observación
- `last()`
- `nth(x, número)`
 - Si queremos el segundo: `nth(vector, 2)`
- `lag()` recrea el vector pero con un rezago
- `lead()` recrea el vector pero un periodo adelantado

Ejemplo arrange()

- Creemos una variable que identifique el ganador para cada casilla

```
presidencial_long <- presidencial_long %>%  
  group_by(CLAVE_CASILLA) %>%  
  arrange(desc(votos_partido), .by_group = T) %>%  
  mutate(ranking=row_number(),  
         ganador=first(partido))
```

- Además añadimos el ranking de cada partido con row_number

| | CLAVE_CASILLA | partido | ranking | ganador |
|---|---------------|---------------|---------|---------|
| 1 | '010000M0100' | MORENA | 1 | MORENA |
| 2 | '010000M0100' | PAN | 2 | MORENA |
| 3 | '010000M0100' | PT_MORENA_PES | 3 | MORENA |
| 4 | '010000M0100' | PRI | 4 | MORENA |
| 5 | '010000M0100' | PAN_PRD_MC | 5 | MORENA |
| 6 | '010000M0100' | CAND_IND_02 | 6 | MORENA |