Notas

Carlos C.

2022-12-25

1. Procesamiento de datos

Existe una serie de funciones del paquete `dplyr` que permite transformar un dataframe a las necesidades que tengamos.

El operador tuberia `%>%` permite combinar múltiples operaciones en `R` en una sola cadena secuencial de acciones. Se escribe haciendo la combinación de teclas `Ctrl + Shift + M`, este toma la salida de una función y luego la "canaliza" para que sea la entrada de la siguiente función. Un truco útil es leer `%>%` como "entonces" o "y luego".

Es importante tener en cuenta que existen muchas funciones avanzadas de manipulación de datos, pero con las siguientes se pueden realizar una gran variedad de tareas.

1.1 Operador filter()

- `filter( condition )`: Permite hacer un filtrado de datos, devuelve aquellos valores que cumplan la condición impuesta.

```
# Filtramos aquellos sismso que se ocurrieron en enero
sismos_enero <- sismos %>% filter(mes==1)
glimpse(sismos_enero)
```

```
Rows: 22,587
Columns: 12
$ Fecha                      <chr> "20/01/1900", "16/01/1902", "13/01/1903", "~
$ Hora                       <chr> "00:33:30", "17:19:00", "19:47:36", "22:21:~
$ Magnitud                   <dbl> 7.4, 7.0, 7.6, 6.4, 6.7, 7.8, 6.8, 6.5, 6.9~
$ Latitud                    <dbl> 20.000, 17.620, 15.000, 19.270, 19.000, 16.~
$ Longitud                   <dbl> -105.000, -99.720, -93.000, -96.970, -107.0~
$ Profundidad                <dbl> 33, 33, 33, 10, 33, 40, 33, 33, 80, 80, 24,~
$ Referencia.de.localizacion <chr> "71 km al NOROESTE de AUTLAN DE NAVARRO, JA~
$ municipio                  <chr> "AUTLAN DE NAVARRO, JAL", "ZUMPANGO DEL RIO~
$ entidad                    <chr> "JAL", "GRO", "CHIS", "VER", "JAL", "OAX", ~
$ zona                       <chr> "Oeste", "Suroeste", "Suroeste", "Oriente",~
$ mes                        <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ placas                     <chr> "Rivera", "Cocos", "Caribe", "Norteam\xe9ri~
```

Veamos más ejemplos

```
# El siguiente comando muestra todos los registros que contengan un NA en "temp"
tabla_prueba <- weather %>%
  filter(temp %in% c(NA))
```

1.2 Operador summarise()

- `summarise( estadistico=function(x) )`: Muestra los estadístico de resumen que se especifiquen de un conjuto de datos .

```
# El siguiente codigo muestra la mean y sd de "temp"
resumen_temp <- weather %>%
  summarize(media=mean(temp,na.rm=T),
            desviacion=sd(temp,na.rm=T))
resumen_temp
```

```
# A tibble: 1 x 2
  media desviacion
  <dbl>      <dbl>
1  55.3       17.8
```

Siempre que el resultado de alguna función sea UN dato se podrá introducir **cualquier función** en `summarise()`. Su input debe ser una columna, veamos más ejemplos de lo que se puede hacer:

```
# Para no prender el na.rm=T pudimos haber realizado esto
weather %>%
  filter(!( temp=c(NA) )) %>%
  summarize(mean(temp))

# Conteo de valores con NA en la variable "temp"
weather %>%
  filter(temp %in% c(NA)) %>%
  summarize(num_rows=n())
```

1.3 Operador group_by()

- `group_by( ~ )`: Los valores de un dataframe se agrupan de acuerdo a la variable `~`. Lo que hace es modificar los metadatos, genera una tabla que está particionada de acuerdo a la variable `~`.

```
tabla_prueba <- weather %>%
  group_by(month)
        # Con esto acreamos una tabla donde
        #   primero aparecen los del mes 1
        #   despues aparecen los del mes 2 y asi sucesivamente
        #   R lo piensa como que existen 12 tablas
```

Con datos agrupados es posible visualizar estadísticos de resumen de una mejor forma:

```
# Aqui calcula la mean por cada una de las 12 tablas
weather %>%
  group_by(month) %>%
  summarise(media=mean(temp,na.rm=T))
```

```
# A tibble: 12 x 2
   month media
   <int> <dbl>
 1     1  35.6
 2     2  34.3
 3     3  39.9
 4     4  51.7
 5     5  61.8
 6     6  72.2
 7     7  80.1
 8     8  74.5
 9     9  67.4
10    10  60.1
11    11  45.0
12    12  38.4
```

Una buena práctica consiste en realizar un `ungroup()` después de hacer un agrupamiento, esto permitirá que el resultado de las operaciones no aparezca "agrupado". Observe el siguiente ejemplo

```
ejercicio <- weather %>%
  filter(hour>=8,hour<=20) %>%  # Hasta aqui incluimos desde 8am y 8pm
  group_by(day,month) %>%
  summarise(temp_media=mean(temp,na.rm=T)) %>%
  arrange(month) %>%
  ungroup()
```

Observemos que se ha usado la función `arrange()`, esto permite ordenar los resultados de forma **descendiente**. Además, es posible hacer varios ordenamientos a la vez, la segunda variable es la que fungirá como criterio de desempate.

1.4 Operador mutate()

- `mutate( nombre=operacion )`: permite crear una nueva columna con información de variables existentes. Esta nueva columna corresponde a una cierta `operacion` y que se genera con el nombre `nombre`.

```
# Creamos varias columnas
tabla <- flights %>%
  mutate(
    gain = dep_delay - arr_delay,
    hours = air_time / 60,
    gain_per_hour = gain / hours)

names(tabla)
```

```
 [1] "year"           "month"          "day"            "dep_time"
 [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
 [9] "arr_delay"      "carrier"        "flight"         "tailnum"
[13] "origin"         "dest"           "air_time"       "distance"
[17] "hour"           "minute"         "time_hour"      "gain"
[21] "hours"          "gain_per_hour"
```

1.5 Operador arrange()

- `arrange( ~ )`: Permite realizar un ordenamiento usando la variable `~`. El orden que hace por defecto es decreciente.

```
# Se puede especificar el ordenamiento deseado de dos formas:

flights %>% group_by(dest) %>%
  summarise(total_de_vuelos=n()) %>%
  arrange(-total_de_vuelos) # Solo sirve para variables numéricas
```

```
# A tibble: 105 x 2
   dest  total_de_vuelos
   <chr>           <int>
 1 ORD             17283
 2 ATL             17215
 3 LAX             16174
 4 BOS             15508
 5 MCO             14082
 6 CLT             14064
 7 SFO             13331
 8 FLL             12055
 9 MIA             11728
10 DCA              9705
# ... with 95 more rows
```

```
# Otra forma de cambiar el orden
flights %>% group_by(dest) %>%
  summarise(total_de_vuelos=n()) %>%
  arrange(desc(total_de_vuelos)) # usando la funcion desc()
```

```
# A tibble: 105 x 2
   dest  total_de_vuelos
   <chr>           <int>
 1 ORD             17283
 2 ATL             17215
 3 LAX             16174
 4 BOS             15508
 5 MCO             14082
 6 CLT             14064
 7 SFO             13331
 8 FLL             12055
 9 MIA             11728
10 DCA              9705
# ... with 95 more rows
```

En la funcion `arrange()` se puede establecer una segunda variable, donde la segunda fungirá como criterio de desempate.

1.6 Operador inner_join()

- `inner_join()`: Permite realizar la intersección de dos tablas mediante una "variable clave". Existen dos casos en los que se puede presentar esta variable clave, los siguientes ejemplos muestran estas situaciones:

1. El primero es donde la segunda tabla *airlines* contiene una variable que tiene exactamente el mismo nombre que la tabla *flights*

```
# El inner_join() se queda unicamente con las filas donde encontro alguna coincidencia
flights_joined <- flights %>% inner_join(airlines)
```

```
Joining, by = "carrier"
```

2. El segundo caso es donde se tiene que especificar cuáles son los nombres de las "variable clave" para pode realizar el inner join. Son variables que tienen los mismos valores pero aparecen con diferente nombre en cada tabla.

```
flights %>% inner_join(airports[,c("faa","name")],
                       by = c("dest"="faa"))
```

```
# A tibble: 329,174 x 20
    year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
   <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
 1  2013     1     1      517        515       2     830     819      11 UA
 2  2013     1     1      533        529       4     850     830      20 UA
 3  2013     1     1      542        540       2     923     850      33 AA
 4  2013     1     1      554        600      -6     812     837     -25 DL
 5  2013     1     1      554        558      -4     740     728      12 UA
 6  2013     1     1      555        600      -5     913     854      19 B6
 7  2013     1     1      557        600      -3     709     723     -14 EV
 8  2013     1     1      557        600      -3     838     846      -8 B6
 9  2013     1     1      558        600      -2     753     745       8 AA
10  2013     1     1      558        600      -2     849     851      -2 B6
# ... with 329,164 more rows, 10 more variables: flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>, name <chr>, and abbreviated variable names
#   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
#   5: arr_delay
```

Además, notemos que hemos restringido qué variables son las que queremos "llevar". El inner join usa el mismo razonamiento que SQL.

1.7 Operador rename()

- `rename()`: Realiza un cambio de nombre a una columna ya creada de un dataframe. La sintaxis que sigue es `nombre_nuevo=nombre_viejo`

```
named_dests <- flights %>%
  group_by(dest) %>%
  summarize(num_flights = n()) %>%
  inner_join(airports, by = c("dest" = "faa")) %>%
  rename(airport_name = name) #a la columna "name" llámala "airport_name"

names(named_dests)
```

```
[1] "dest"         "num_flights"  "airport_name" "lat"          "lon"
[6] "alt"          "tz"           "dst"          "tzone"
```

1.8 Operador select()

- `select()`: Permite seleccionar columnas cuando solo queremos trabajar con algunas columnas.

```
# Muestra el dataframe solo con esas dos columnas
tabla_select <- flights %>%
  select(carrier,flight)
```

Veamos otros ejemplos para mostrar las funciones de `select()`

```
### Se puede establecer el orden en que aparecen las variables
flights %>%
  select(year:day,hour,minute,time_hour,everything()) %>% View()

### Se puede usar select para quitar informacion
flights %>%
  select(-year) %>% View()

### Podemos buscar variables (columnas) que cumplan ciertas condiciones:

# Palabras que empiecen con ~
flights %>% select(starts_with(("a"))) %>% names()

# Palabras que terminen con ~
flights %>% select(ends_with("e")) %>% names()

# Palabras que contienen ~ en cualquier parte de la variable
flights %>% select(contains("time")) %>% names()
```

También es posible modificar el nombre de las columnas de un dataframe con esta función.

```
# Se puede omitir rename(), con select() se puede hacer lo mismo
flights %>% select("departure_time"=dep_time,
                   "arrival_time"=arr_time)
```

00. Acordeón

- `filter( condition )`: Permite hacer un filtrado de datos, devuelve aquellos registros que cumplan la condición impuesta
- `summarise( estadistico=function(x) )`: Muestra los estadísticos de resumen que se especifiquen para un conjunto de datos `x`
- `n()`: es una función que puede usarse dentro de `summarise()` y sirve para hacer un conteo de registros.
- `group_by( ~ )`: Podemos realizar agrupación de datos de un dataframe de acuerdo a las categorias de la variable `~`. Una buena práctica es usar `ungroup()` para que los resultados no aparezcan "agrupados"
- `mutate( nombre=operacion )`: Permite crear una nueva columna con un `nombre` dado y generada a partir de cierta `operacion`
- `arrange( ~ )`: Realiza un ordenamiento de los datos de acuerdo a la variable `~` (por ejemplo, mes). Se puede emplear una segunda variable que sirva como criterio de desempate.
- `inner_join( ~ )`: Permite realizar un inner join con la información de la tabla `~` , mismo razonamiento que en SQL. Si las "variables clave" no coinciden en nombre, se debe especificar con el argumento `by=c(a,b)` cuáles son el nombre de las variables.
- `rename()`: Realiza un cambio de nombre a una columna ya creada de un dataframe. La sintaxis que sigue es `nombre_nuevo=nombre_viejo`
- `select()`: Permite seleccionar algunas variables (columnas) de un dataframe para trabajar exclusivamente con ellas. Es posible cambiar el nombre de las columnas con esta función. Además, permite argumentos como `starts_with("a")`, `ends_with("e")` y `contains("palabra")`