

Manipulación y agrupación de datos

Teresa Ortiz

En estas notas continuamos con la introducción a R para análisis de datos, en particular mostraremos herramientas de manipulación de datos. Trataremos los siguientes puntos:

- Reestructura de datos y el principio de los datos limpios.
- Estrategia divide-aplica-combina.

Es sabido que limpieza y preparación de datos ocupan gran parte del tiempo del análisis de datos: (Dasu y Johnson, 2003 (<http://onlinelibrary.wiley.com/book/10.1002/0471448354>) y NYT's 'Janitor Work' Is Key Hurdle to Insights (<https://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html?mcubz=0>), es por ello que vale la pena dedicar un tiempo a aprender técnicas que faciliten estas tareas, y entender que estructura en los datos es más conveniente para trabajar.

Datos limpios

Una vez que importamos datos a R es conveniente limpiarlos, esto implica almacenarlos de una manera consistente que nos permita enfocarnos en responder preguntas de los datos en lugar de estar luchando con los datos. Entonces, **datos limpios** son datos que facilitan las tareas del análisis de datos:

- **Manipulación:** Manipulación de variables como agregar, filtrar, reordenar, transformar.
- **Visualización:** Resúmenes de datos usando gráficas, análisis exploratorio, o presentación de resultados.
- **Modelación:** Ajustar modelos es sencillo si los datos están en la forma correcta.

Los principios de **datos limpios** (Tidy Data de Hadley Wickham (<http://vita.had.co.nz/papers/tidy-data.pdf>)) proveen una manera estándar de organizar la información:

1. Cada variable forma una columna.
2. Cada observación forma un renglón.
3. Cada tipo de unidad observacional forma una tabla.

Vale la pena notar que los principios de los datos limpios se pueden ver como teoría de algebra relacional para estadísticos, éstos principios equivalen a la tercera forma normal de Codd con enfoque en una sola tabla de datos en lugar de muchas conectadas en bases de datos relacionales.

Veamos un ejemplo:

La mayor parte de las bases de datos en estadística tienen forma rectangular, ¿cuántas variables tiene la siguiente tabla?

	tratamientoA	tratamientoB
Juan Aguirre	-	2
Ana Bernal	16	11
José López	3	1

La tabla anterior también se puede estructurar de la siguiente manera:

	Juan Aguirre	Ana Bernal	José López
tratamientoA	-	16	3
tratamientoB	2	11	1

Si vemos los principios (cada variable forma una columna, cada observación forma un renglón, cada tipo de unidad observacional forma una tabla), ¿las tablas anteriores cumplen los principios?

Para responder la pregunta identifiquemos primero cuáles son las variables y cuáles las observaciones de esta pequeña base. Las variables son: persona/nombre, tratamiento y resultado. Entonces, siguiendo los principios de *datos limpios* obtenemos la siguiente estructura:

nombre	tratamiento	resultado
Juan Aguirre	a	-
Ana Bernal	a	16
José López	a	3
Juan Aguirre	b	2
Ana Bernal	b	11
José López	b	1

Limpieza bases de datos

Los principios de los datos limpios parecen obvios pero la mayor parte de los datos no los cumplen debido a:

1. La mayor parte de la gente no está familiarizada con los principios y es difícil derivarlos por uno mismo.
2. Los datos suelen estar organizados para facilitar otros aspectos que no son análisis, por ejemplo, la captura.

Algunos de los problemas más comunes en las bases de datos que no están *limpias* son:

- Los encabezados de las columnas son valores y no nombres de variables.
- Más de una variable por columna.
- Las variables están organizadas tanto en filas como en columnas.
- Más de un tipo de observación en una tabla.
- Una misma unidad observacional está almacenada en múltiples tablas.

La mayor parte de estos problemas se pueden arreglar con pocas herramientas, a continuación veremos como *limpiar* datos usando 2 funciones del paquete `tidyr`:

- **gather**: recibe múltiples columnas y las junta en pares de valores y nombres, convierte los datos anchos en largos.
- **spread**: recibe 2 columnas y las separa, haciendo los datos más anchos.

Repasaremos los problemas más comunes que se encuentran en conjuntos de datos sucios y mostraremos como se puede manipular la tabla de datos (usando las funciones *gather* y *spread*) con el fin de estructurarla para que cumpla los principios de datos limpios.

1. Los encabezados de las columnas son valores

Usaremos ejemplos para entender los conceptos más fácilmente. La primer base de datos está basada en una encuesta de Pew Research (<http://www.pewforum.org/2009/01/30/income-distribution-within-us-religious-groups/>) que investiga la relación entre ingreso y afiliación religiosa.

¿Cuáles son las variables en estos datos?

```

library(tidyverse)
pew <- read_delim("http://stat405.had.co.nz/data/pew.txt", "\t",
  escape_double = FALSE, trim_ws = TRUE)
#> Parsed with column specification:
#> cols(
#>   religion = col_character(),
#>   `<$10k` = col_integer(),
#>   `$10-20k` = col_integer(),
#>   `$20-30k` = col_integer(),
#>   `$30-40k` = col_integer(),
#>   `$40-50k` = col_integer(),
#>   `$50-75k` = col_integer(),
#>   `$75-100k` = col_integer(),
#>   `$100-150k` = col_integer(),
#>   `>150k` = col_integer(),
#>   `Don't know/refused` = col_integer()
#> )
pew
#> # A tibble: 18 x 11
#>   religion `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k`
#>   <chr>    <int>    <int>    <int>    <int>    <int>
#> 1 Agnostic      27      34      60      81      76
#> 2 Atheist       12      27      37      52      35
#> 3 Buddhist      27      21      30      34      33
#> 4 Catholic     418     617     732     670     638
#> 5 Don't know/refused 15      14      15      11      10
#> 6 Evangelical Prot 575     869    1064     982     881
#> 7 Hindu         1       9       7       9      11
#> 8 Historically Black Prot 228     244     236     238     197
#> 9 Jehovah's Witness  20      27      24      24      21
#> 10 Jewish        19      19      25      25      30
#> 11 Mainline Prot  289     495     619     655     651
#> 12 Mormon        29      40      48      51      56
#> 13 Muslim         6       7       9      10       9
#> 14 Orthodox      13      17      23      32      32
#> 15 Other Christian   9       7      11      13      13
#> 16 Other Faiths     20      33      40      46      49
#> 17 Other World Religions  5       2       3       4       2
#> 18 Unaffiliated   217     299     374     365     341
#> # ... with 5 more variables: `$50-75k` <int>, `$75-100k` <int>,
#> #   `$100-150k` <int>, `>150k` <int>, `Don't know/refused` <int>

```

Esta base de datos tiene 3 variables: religión, ingreso y frecuencia. Para *limpiarla* es necesario apilar las columnas (alargar los datos). Notemos que al alargar los datos desaparecerán las columnas que se agrupan y dan lugar a dos nuevas columnas: la correspondiente a clave y la correspondiente a valor. Entonces, para alargar una base de datos usamos la función `gather` que recibe los argumentos:

- `data`: base de datos que vamos a reestructurar.
- `key`: nombre de la nueva variable que contiene lo que fueron los nombres de columnas que apilamos.
- `value`: nombre de la variable que almacenará los valores que corresponden a cada `key`.
- ...: lo último que especificamos son las columnas que vamos a apilar, la notación para seleccionarlás es la misma que usamos con `select()`.

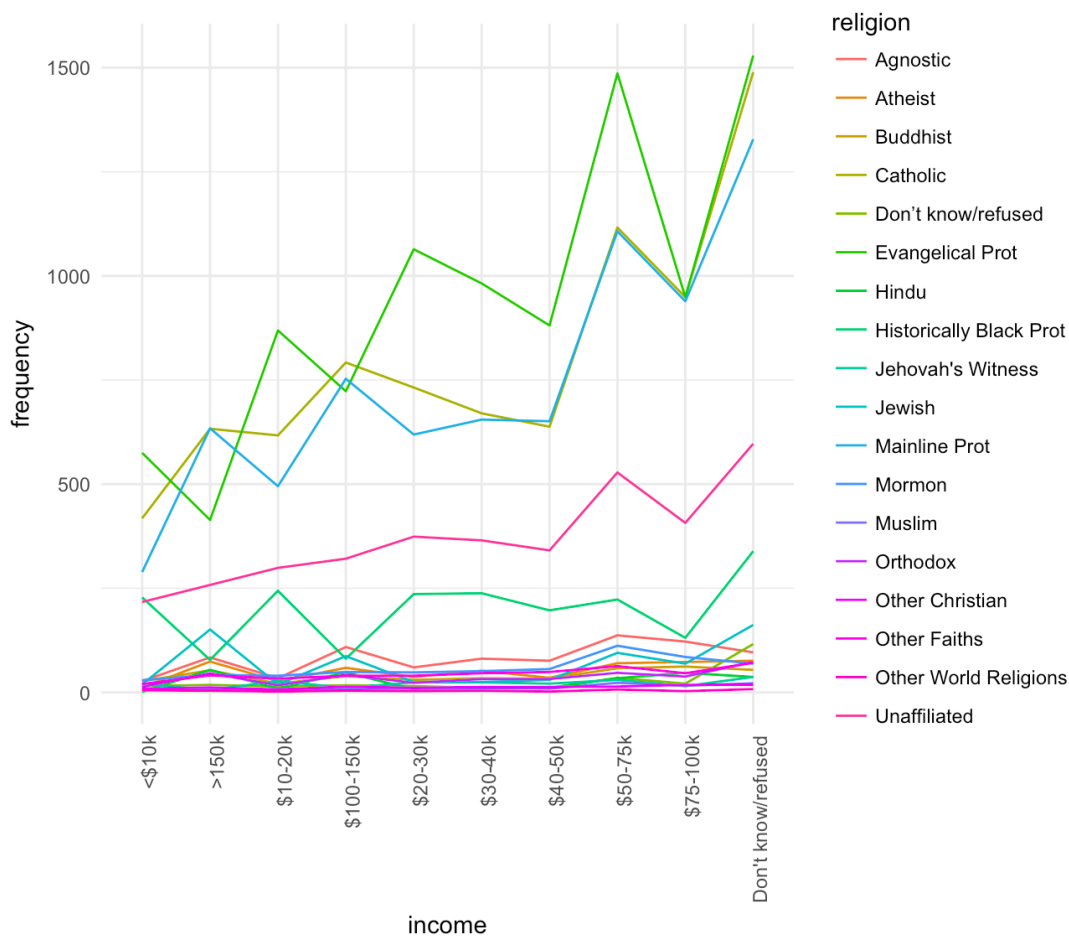
```
pew_tidy <- gather(data = pew, income, frequency, -religion)
pew_tidy
#> # A tibble: 180 x 3
#>
#>       religion income frequency
#>   <chr>    <chr>    <int>
#> 1   Agnostic <$10k         27
#> 2   Atheist <$10k         12
#> 3   Buddhist <$10k         27
#> 4   Catholic <$10k        418
#> 5 Don't know/refused <$10k         15
#> 6 Evangelical Prot <$10k        575
#> 7   Hindu    <$10k          1
#> 8 Historically Black Prot <$10k        228
#> 9   Jehovah's Witness <$10k         20
#> 10 Jewish   <$10k         19
#> # ... with 170 more rows
```

Observemos que en la tabla ancha teníamos bajo la columna <\$10k, en el renglón correspondiente a *Agnostic* un valor de 27, y podemos ver que este valor en la tabla larga se almacena bajo la columna frecuencia y corresponde a religión *Agnostic*, income <\$10k. También es importante ver que en este ejemplo especificamos las columnas a apilar identificando la que **no** vamos a alargar con un signo negativo: es decir apila todas las columnas menos religión.

La nueva estructura de la base de datos nos permite, por ejemplo, hacer fácilmente una gráfica donde podemos comparar las diferencias en las frecuencias.

Nota: En esta sección no explicaremos las funciones de graficación pues estas se cubren en las notas introductorias a R. En esta parte nos queremos concentrar en como limpiar datos y ejemplificar lo sencillo que es trabajar con datos limpios, esto es, una vez que los datos fueron reestructurados es fácil construir gráficas y resúmenes.

```
ggplot(pew_tidy, aes(x = income, y = frequency, color = religion, group = religion)) +
  geom_line() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

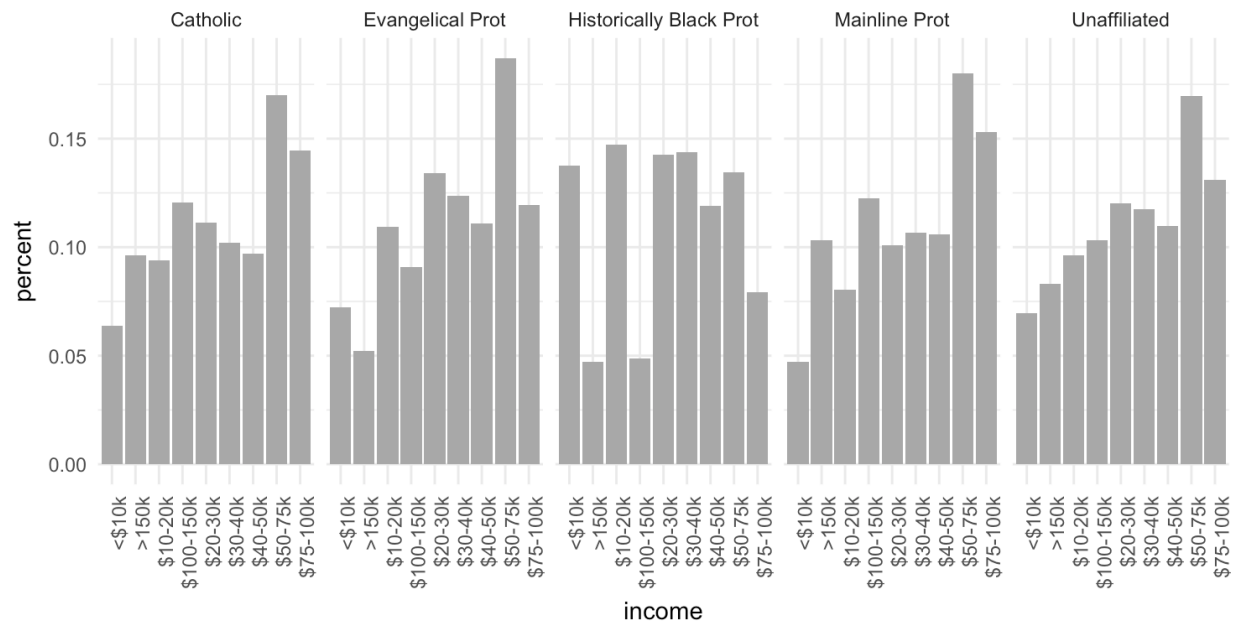


Podemos hacer gráficas más interesantes si creamos nuevas variables:

```
by_religion <- group_by(pew_tidy, religion)
pew_tidy_2 <- pew_tidy %>%
  filter(income != "Don't know/refused") %>%
  group_by(religion) %>%
  mutate(percent = frequency / sum(frequency)) %>%
  filter(sum(frequency) > 1000)

head(pew_tidy_2)
#> # A tibble: 6 x 4
#> # Groups:   religion [5]
#>   religion income frequency percent
#>   <chr>   <chr>     <int>     <dbl>
#> 1 Catholic <$10k         418 0.06367098
#> 2 Evangelical Prot <$10k         575 0.07239078
#> 3 Historically Black Prot <$10k         228 0.13768116
#> 4 Mainline Prot <$10k         289 0.04705308
#> 5 Unaffiliated <$10k         217 0.06977492
#> 6 Catholic $10-20k         617 0.09398324

ggplot(pew_tidy_2, aes(x = income, y = percent, group = religion)) +
  facet_wrap(~ religion, nrow = 1) +
  geom_bar(stat = "identity", fill = "darkgray") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



En el código de arriba utilizamos las funciones `group_by`, `filter` y `mutate` que estudiaremos más adelante. Por ahora concentremonos en `gather` y `spread`.

Otro ejemplo, veamos los datos de *Billboard*, aquí se registra la fecha en la que una canción entra por primera vez al top 100 de Billboard.

```

billboard <- read_csv("data/billboard.csv")
#> Parsed with column specification:
#> cols(
#>   .default = col_integer(),
#>   artist = col_character(),
#>   track = col_character(),
#>   time = col_time(format = ""),
#>   date.entered = col_date(format = ""),
#>   wk66 = col_character(),
#>   wk67 = col_character(),
#>   wk68 = col_character(),
#>   wk69 = col_character(),
#>   wk70 = col_character(),
#>   wk71 = col_character(),
#>   wk72 = col_character(),
#>   wk73 = col_character(),
#>   wk74 = col_character(),
#>   wk75 = col_character(),
#>   wk76 = col_character()
#> )
#> See spec(...) for full column specifications.
billboard
#> # A tibble: 317 x 81
#>   year      artist      track      time date.entered
#>   <int>    <chr>      <chr>    <time>    <date>
#> 1  2000      2 Pac Baby Don't Cry (Keep... 04:22:00 2000-02-26
#> 2  2000      2Ge+her The Hardest Part Of ... 03:15:00 2000-09-02
#> 3  2000    3 Doors Down      Kryptonite 03:53:00 2000-04-08
#> 4  2000    3 Doors Down      Loser 04:24:00 2000-10-21
#> 5  2000      504 Boyz      Wobble Wobble 03:35:00 2000-04-15
#> 6  2000      98^0 Give Me Just One Nig... 03:24:00 2000-08-19
#> 7  2000      A*Teens      Dancing Queen 03:44:00 2000-07-08
#> 8  2000      Aaliyah      I Don't Wanna 04:15:00 2000-01-29
#> 9  2000      Aaliyah      Try Again 04:03:00 2000-03-18
#> 10 2000 Adams, Yolanda      Open My Heart 05:30:00 2000-08-26
#> # ... with 307 more rows, and 76 more variables: wk1 <int>, wk2 <int>,
#> #   wk3 <int>, wk4 <int>, wk5 <int>, wk6 <int>, wk7 <int>, wk8 <int>,
#> #   wk9 <int>, wk10 <int>, wk11 <int>, wk12 <int>, wk13 <int>, wk14 <int>,
#> #   wk15 <int>, wk16 <int>, wk17 <int>, wk18 <int>, wk19 <int>,
#> #   wk20 <int>, wk21 <int>, wk22 <int>, wk23 <int>, wk24 <int>,
#> #   wk25 <int>, wk26 <int>, wk27 <int>, wk28 <int>, wk29 <int>,
#> #   wk30 <int>, wk31 <int>, wk32 <int>, wk33 <int>, wk34 <int>,
#> #   wk35 <int>, wk36 <int>, wk37 <int>, wk38 <int>, wk39 <int>,
#> #   wk40 <int>, wk41 <int>, wk42 <int>, wk43 <int>, wk44 <int>,
#> #   wk45 <int>, wk46 <int>, wk47 <int>, wk48 <int>, wk49 <int>,
#> #   wk50 <int>, wk51 <int>, wk52 <int>, wk53 <int>, wk54 <int>,
#> #   wk55 <int>, wk56 <int>, wk57 <int>, wk58 <int>, wk59 <int>,
#> #   wk60 <int>, wk61 <int>, wk62 <int>, wk63 <int>, wk64 <int>,
#> #   wk65 <int>, wk66 <chr>, wk67 <chr>, wk68 <chr>, wk69 <chr>,
#> #   wk70 <chr>, wk71 <chr>, wk72 <chr>, wk73 <chr>, wk74 <chr>,
#> #   wk75 <chr>, wk76 <chr>

```

Notemos que el rank en cada semana (una vez que entró a la lista) está guardado en 75 columnas wk1 a wk75, este tipo de almacenamiento no es *limpio* pero puede ser útil al momento de ingresar la información.

Para tener datos *limpios* apilamos las semanas de manera que sea una sola columna (nuevamente alargamos los datos):

```
billboard_long <- gather(billboard, week, rank, wk1:wk76, na.rm = TRUE)
billboard_long
#> # A tibble: 5,307 x 7
#>   year      artist      track      time date.entered
#>   <int>    <chr>      <chr>    <time>    <date>
#> 1  2000      2 Pac Baby Don't Cry (Keep... 04:22:00 2000-02-26
#> 2  2000      2Ge+her The Hardest Part Of ... 03:15:00 2000-09-02
#> 3  2000    3 Doors Down      Kryptonite 03:53:00 2000-04-08
#> 4  2000    3 Doors Down      Loser 04:24:00 2000-10-21
#> 5  2000      504 Boyz      Wobble Wobble 03:35:00 2000-04-15
#> 6  2000      98^0 Give Me Just One Nig... 03:24:00 2000-08-19
#> 7  2000      A*Teens      Dancing Queen 03:44:00 2000-07-08
#> 8  2000      Aaliyah      I Don't Wanna 04:15:00 2000-01-29
#> 9  2000      Aaliyah      Try Again 04:03:00 2000-03-18
#> 10 2000 Adams, Yolanda      Open My Heart 05:30:00 2000-08-26
#> # ... with 5,297 more rows, and 2 more variables: week <chr>, rank <chr>
```

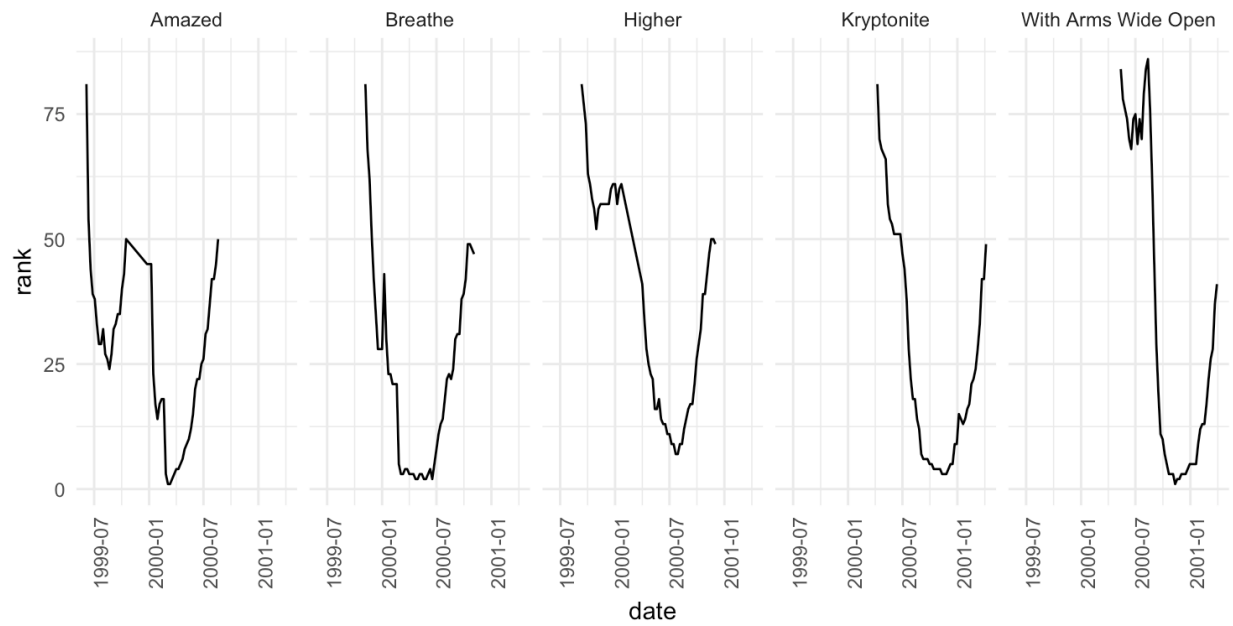
Notemos que en esta ocasión especificamos las columnas que vamos a apilar indicando el nombre de la primera de ellas seguido de : y por último el nombre de la última variable a apilar. Por otra parte, la instrucción `na.rm = TRUE` se utiliza para eliminar los renglones con valores faltantes en la columna de `value` (`rank`), esto es, eliminamos aquellas observaciones que tenían NA en la columna `wknum` de la tabla ancha. Ahora realizamos una limpieza adicional creando mejores variables de fecha.

```
billboard_tidy <- billboard_long %>%
  mutate(
    week = parse_number(week),
    date = date.entered + 7 * (week - 1),
    rank = as.numeric(rank)
  ) %>%
  select(-date.entered)
billboard_tidy
#> # A tibble: 5,307 x 7
#>   year      artist      track      time week rank
#>   <int>    <chr>      <chr>    <time> <dbl> <dbl>
#> 1  2000      2 Pac Baby Don't Cry (Keep... 04:22:00     1    87
#> 2  2000      2Ge+her The Hardest Part Of ... 03:15:00     1    91
#> 3  2000    3 Doors Down      Kryptonite 03:53:00     1    81
#> 4  2000    3 Doors Down      Loser 04:24:00     1    76
#> 5  2000      504 Boyz      Wobble Wobble 03:35:00     1    57
#> 6  2000      98^0 Give Me Just One Nig... 03:24:00     1    51
#> 7  2000      A*Teens      Dancing Queen 03:44:00     1    97
#> 8  2000      Aaliyah      I Don't Wanna 04:15:00     1    84
#> 9  2000      Aaliyah      Try Again 04:03:00     1    59
#> 10 2000 Adams, Yolanda      Open My Heart 05:30:00     1    76
#> # ... with 5,297 more rows, and 1 more variables: date <date>
```

Nuevamente, podemos hacer gráficas fácilmente.

```
tracks <- filter(billboard_tidy, track %in%
  c("Higher", "Amazed", "Kryptonite", "Breathe", "With Arms Wide Open"))

ggplot(tracks, aes(x = date, y = rank)) +
  geom_line() +
  facet_wrap(~track, nrow = 1) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

2. Una columna asociada a más de una variable

La siguiente base de datos proviene de la Organización Mundial de la Salud y contiene el número de casos confirmados de tuberculosis por país y año, la información esta por grupo demográfico de acuerdo a sexo (m, f), y edad (0-4, 5-14, etc). Los datos están disponibles en <http://www.who.int/tb/country/data/download/en/> (<http://www.who.int/tb/country/data/download/en/>).

```

tb <- read_csv("data/tb.csv")
#> Parsed with column specification:
#> cols(
#>   .default = col_integer(),
#>   year = col_character()
#> )
#> See spec(...) for full column specifications.
#> Warning in rbind(names(probs), probs_f): number of columns of result is not
#> a multiple of vector length (arg 1)
#> Warning: 5769 parsing failures.
#> row # A tibble: 5 x 5 col      row    col    expected    actual      file expected  <int> <chr>
  <chr>      <chr>      <chr> actual 1      1 <NA> 22 columns 23 columns 'data/tb.csv' file 2      2 <NA>
> 22 columns 23 columns 'data/tb.csv' row 3      3 <NA> 22 columns 23 columns 'data/tb.csv' col 4      4 <
NA> 22 columns 23 columns 'data/tb.csv' expected 5      5 <NA> 22 columns 23 columns 'data/tb.csv'
#> ... ..
#> See problems(...) for more details.
tb
#> # A tibble: 5,769 x 22
#>   iso2 year new_sp_m04 new_sp_m514 new_sp_m014 new_sp_m1524
#>   <int> <chr>      <int>      <int>      <int>      <int>
#> 1     1     AD      1989         NA         NA         NA
#> 2     2     AD      1990         NA         NA         NA
#> 3     3     AD      1991         NA         NA         NA
#> 4     4     AD      1992         NA         NA         NA
#> 5     5     AD      1993         NA         NA         NA
#> 6     6     AD      1994         NA         NA         NA
#> 7     7     AD      1996         NA         NA         0
#> 8     8     AD      1997         NA         NA         0
#> 9     9     AD      1998         NA         NA         0
#> 10    10     AD      1999         NA         NA         0
#> # ... with 5,759 more rows, and 16 more variables: new_sp_m2534 <int>,
#> #   new_sp_m3544 <int>, new_sp_m4554 <int>, new_sp_m5564 <int>,
#> #   new_sp_m65 <int>, new_sp_mu <int>, new_sp_f04 <int>,
#> #   new_sp_f514 <int>, new_sp_f014 <int>, new_sp_f1524 <int>,
#> #   new_sp_f2534 <int>, new_sp_f3544 <int>, new_sp_f4554 <int>,
#> #   new_sp_f5564 <int>, new_sp_f65 <int>, new_sp_fu <int>

```

 De manera similar a los ejemplos anteriores, utiliza la función `gather` para apilar las columnas correspondientes a sexo-edad.

Piensa en como podemos separar la “variable” sexo-edad en dos columnas.

Ahora separaremos las variables sexo y edad de la columna `demo`, para ello debemos pasar a la función `separate()`, esta recibe como parámetros:

- el nombre de la base de datos,
- el nombre de la variable que deseamos separar en más de una,
- la posición de donde deseamos “cortar” (hay más opciones para especificar como separar, ver `?separate`). El default es separar valores en todos los lugares que encuentre un caracter que no es alfanumérico (espacio, guión,...).

```

tb_tidy <- separate(tb_long, demo, c("sex", "age"), 8)
tb_tidy
#> # A tibble: 41,247 x 5
#>   iso2 year      sex age      n
#>   * <int> <chr>    <chr> <chr> <int>
#> 1     1    AD new_sp_m 04 1989
#> 2     2    AD new_sp_m 04 1990
#> 3     3    AD new_sp_m 04 1991
#> 4     4    AD new_sp_m 04 1992
#> 5     5    AD new_sp_m 04 1993
#> 6     6    AD new_sp_m 04 1994
#> 7     7    AD new_sp_m 04 1996
#> 8     8    AD new_sp_m 04 1997
#> 9     9    AD new_sp_m 04 1998
#> 10    10    AD new_sp_m 04 1999
#> # ... with 41,237 more rows
table(tb_tidy$sex)
#>
#> new_sp_f new_sp_m
#>   17831   23416

# creamos un mejor código de genero
tb_tidy <- mutate(tb_tidy, sex = substr(sex, 8, 8))
table(tb_tidy$sex)
#>
#>      f      m
#> 17831 23416

```

3. Variables almacenadas en filas y columnas

El problema más difícil es cuando las variables están tanto en filas como en columnas, veamos una base de datos de clima en Cuernavaca. ¿Cuáles son las variables en estos datos?

```

clima <- read_delim("data/clima.txt", "\t", escape_double = FALSE, trim_ws = TRUE)
#> Parsed with column specification:
#> cols(
#>   .default = col_integer(),
#>   id = col_character(),
#>   element = col_character(),
#>   d9 = col_character(),
#>   d12 = col_character(),
#>   d18 = col_character(),
#>   d19 = col_character(),
#>   d20 = col_character(),
#>   d21 = col_character(),
#>   d22 = col_character(),
#>   d24 = col_character()
#> )
#> See spec(...) for full column specifications.

```

Estos datos tienen variables en columnas individuales (id, año, mes), en múltiples columnas (día, d1-d31) y en filas (tmin, tmax). Comencemos por apilar las columnas.

```

clima_long <- gather(clima, day, value, d1:d31, na.rm = TRUE)
clima_long
#> # A tibble: 66 x 6
#>       id year month element   day value
#>   *   <chr> <int> <int>   <chr> <chr> <chr>
#> 1 MX000017004 2010    12    TMAX    d1    299
#> 2 MX000017004 2010    12    TMIN    d1    138
#> 3 MX000017004 2010     2    TMAX    d2    273
#> 4 MX000017004 2010     2    TMIN    d2    144
#> 5 MX000017004 2010    11    TMAX    d2    313
#> 6 MX000017004 2010    11    TMIN    d2    163
#> 7 MX000017004 2010     2    TMAX    d3    241
#> 8 MX000017004 2010     2    TMIN    d3    144
#> 9 MX000017004 2010     7    TMAX    d3    286
#> 10 MX000017004 2010     7    TMIN    d3    175
#> # ... with 56 more rows

```

Podemos crear algunas variables adicionales.

```

clima_vars <- clima_long %>%
  mutate(day = parse_number(day),
         value = as.numeric(value) / 10) %>%
  select(id, year, month, day, element, value) %>%
  arrange(id, year, month, day)
clima_vars
#> # A tibble: 66 x 6
#>       id year month   day element value
#>   <chr> <int> <int> <dbl>   <chr> <dbl>
#> 1 MX000017004 2010     1    30    TMAX  27.8
#> 2 MX000017004 2010     1    30    TMIN  14.5
#> 3 MX000017004 2010     2     2    TMAX  27.3
#> 4 MX000017004 2010     2     2    TMIN  14.4
#> 5 MX000017004 2010     2     3    TMAX  24.1
#> 6 MX000017004 2010     2     3    TMIN  14.4
#> 7 MX000017004 2010     2    11    TMAX  29.7
#> 8 MX000017004 2010     2    11    TMIN  13.4
#> 9 MX000017004 2010     2    23    TMAX  29.9
#> 10 MX000017004 2010     2    23    TMIN  10.7
#> # ... with 56 more rows

```

Finalmente, la columna *element* no es una variable, sino que almacena el nombre de dos variables, la operación que debemos aplicar (spread) es el inverso de apilar (gather):

```

clima_tidy <- spread(clima_vars, element, value)
clima_tidy
#> # A tibble: 33 x 6
#>       id year month   day TMAX TMIN
#>   *   <chr> <int> <int> <dbl> <dbl> <dbl>
#> 1 MX000017004 2010     1    30  27.8  14.5
#> 2 MX000017004 2010     2     2  27.3  14.4
#> 3 MX000017004 2010     2     3  24.1  14.4
#> 4 MX000017004 2010     2    11  29.7  13.4
#> 5 MX000017004 2010     2    23  29.9  10.7
#> 6 MX000017004 2010     3     5  32.1  14.2
#> 7 MX000017004 2010     3    10  34.5  16.8
#> 8 MX000017004 2010     3    16  31.1  17.6
#> 9 MX000017004 2010     4    27  36.3  16.7
#> 10 MX000017004 2010     5    27  33.2  18.2
#> # ... with 23 more rows

```

Ahora es inmediato no solo hacer gráficas sino también ajustar un modelo.

```
# ajustamos un modelo lineal donde la variable respuesta es temperatura
# máxima, y la variable explicativa es el mes
clima_lm <- lm(TMAX ~ factor(month), data = clima_tidy)
summary(clima_lm)
#>
#> Call:
#> lm(formula = TMAX ~ factor(month), data = clima_tidy)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#>  -3.65  -0.92  -0.02   1.05   3.18
#>
#> Coefficients:
#>              Estimate Std. Error t value Pr(>|t|)
#> (Intercept)    27.8000     1.8610  14.938 5.34e-13 ***
#> factor(month)2   -0.0500     2.0807   -0.024  0.98104
#> factor(month)3    4.7667     2.1489    2.218  0.03717 *
#> factor(month)4    8.5000     2.6319    3.230  0.00385 **
#> factor(month)5    5.4000     2.6319    2.052  0.05228 .
#> factor(month)6    1.2500     2.2793    0.548  0.58892
#> factor(month)7    1.4500     2.2793    0.636  0.53123
#> factor(month)8    0.4714     1.9895    0.237  0.81488
#> factor(month)10   1.1000     2.0386    0.540  0.59491
#> factor(month)11   0.3200     2.0386    0.157  0.87670
#> factor(month)12   1.0500     2.2793    0.461  0.64955
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 1.861 on 22 degrees of freedom
#> Multiple R-squared:  0.6182, Adjusted R-squared:  0.4447
#> F-statistic: 3.563 on 10 and 22 DF,  p-value: 0.006196
```

4. Mas de un tipo de observación en una misma tabla

En ocasiones las bases de datos involucran valores en diferentes niveles, en diferentes tipos de unidad observacional. En la limpieza de datos, cada unidad observacional debe estar almacenada en su propia tabla (esto esta ligado a normalización de una base de datos), es importante para evitar inconsistencias en los datos.

¿Cuáles son las unidades observacionales de los datos de billboard?

```
billboard_tidy
#> # A tibble: 5,307 x 7
#>   year      artist      track      time week rank
#>   <int>      <chr>      <chr>      <time> <dbl> <dbl>
#> 1  2000      2 Pac Baby Don't Cry (Keep... 04:22:00     1    87
#> 2  2000      2Ge+her The Hardest Part Of ... 03:15:00     1    91
#> 3  2000      3 Doors Down      Kryptonite 03:53:00     1    81
#> 4  2000      3 Doors Down      Loser 04:24:00     1    76
#> 5  2000      504 Boyz      Wobble Wobble 03:35:00     1    57
#> 6  2000      98^0 Give Me Just One Nig... 03:24:00     1    51
#> 7  2000      A*Teens      Dancing Queen 03:44:00     1    97
#> 8  2000      Aaliyah      I Don't Wanna 04:15:00     1    84
#> 9  2000      Aaliyah      Try Again 04:03:00     1    59
#> 10 2000 Adams, Yolanda      Open My Heart 05:30:00     1    76
#> # ... with 5,297 more rows, and 1 more variables: date <date>
```

Separaremos esta base de datos en dos: la tabla canción que almacena artista, nombre de la canción y duración; la tabla rank que almacena el ranking de la canción en cada semana.

```

song <- billboard_tidy %>%
  select(artist, track, year, time) %>%
  unique() %>%
  arrange(artist) %>%
  mutate(song_id = row_number(artist))
song
#> # A tibble: 317 x 5
#>   artist          track year    time song_id
#>   <chr>          <chr> <int> <time>   <int>
#> 1      2 Pac Baby Don't Cry (Keep... 2000 04:22:00     1
#> 2    2Ge+her The Hardest Part Of ... 2000 03:15:00     2
#> 3 3 Doors Down      Kryptonite 2000 03:53:00     3
#> 4 3 Doors Down      Loser 2000 04:24:00     4
#> 5    504 Boyz      Wobble Wobble 2000 03:35:00     5
#> 6    98^0 Give Me Just One Nig... 2000 03:24:00     6
#> 7    A*Teens      Dancing Queen 2000 03:44:00     7
#> 8    Aaliyah      I Don't Wanna 2000 04:15:00     8
#> 9    Aaliyah      Try Again 2000 04:03:00     9
#> 10 Adams, Yolanda    Open My Heart 2000 05:30:00    10
#> # ... with 307 more rows

rank <- billboard_tidy %>%
  left_join(song, c("artist", "track", "year", "time")) %>%
  select(song_id, date, week, rank) %>%
  arrange(song_id, date) %>%
  tbl_df
rank
#> # A tibble: 5,307 x 4
#>   song_id    date  week  rank
#>   <int>   <date> <dbl> <dbl>
#> 1      1 2000-02-26     1    87
#> 2      1 2000-03-04     2    82
#> 3      1 2000-03-11     3    72
#> 4      1 2000-03-18     4    77
#> 5      1 2000-03-25     5    87
#> 6      1 2000-04-01     6    94
#> 7      1 2000-04-08     7    99
#> 8      2 2000-09-02     1    91
#> 9      2 2000-09-09     2    87
#> 10     2 2000-09-16     3    92
#> # ... with 5,297 more rows

```

5. Una misma unidad observacional está almacenada en múltiples tablas

También es común que los valores sobre una misma unidad observacional estén separados en muchas tablas o archivos, es común que estas tablas esten divididas de acuerdo a una variable, de tal manera que cada archivo representa a una persona, año o ubicación. Para juntar los archivos hacemos lo siguiente:

1. Leemos los archivos en una lista de tablas.
2. Para cada tabla agregamos una columna que registra el nombre del archivo original.
3. Combinamos las tablas en un solo data frame.

Veamos un ejemplo, descarga la carpeta specdata (<https://www.dropbox.com/sh/c0mgho95gwjc1mv/AACVLPr33O6ENW68xmL7hyUna?dl=0>), ésta contiene 332 archivos csv que almacenan información de monitoreo de contaminación en 332 ubicaciones de EUA. Cada archivo contiene información de una unidad de monitoreo y el número de identificación del monitor es el nombre del archivo.

Los pasos en R (usando el paquete `purrr`), primero creamos un vector con los nombres de los archivos en un directorio, eligiendo aquellos que contengan las letras ".csv".

```
paths <- dir("data/specdata", pattern = "\\..csv$", full.names = TRUE)
```

Después le asignamos el nombre del csv al nombre de cada elemento del vector. Este paso se realiza para preservar los nombres de los archivos ya que estos los asignaremos a una variable mas adelante.

```
paths <- set_names(paths, basename(paths))
```

La función `map_df` itera sobre cada dirección, lee el csv en dicha dirección y los combina en un data frame.

```
specdata_us <- map_df(paths, ~read_csv(., col_types = "Tddi"), .id = "filename")

# eliminamos la basura del id
specdata <- specdata_us %>%
  mutate(monitor = parse_number(filename)) %>%
  select(id = ID, monitor, date = Date, sulfate, nitrate)
specdata
#> # A tibble: 772,087 x 5
#>   id monitor      date sulfate nitrate
#>   <int>   <dbl>   <dtm>   <dbl>   <dbl>
#> 1     1     1 2003-01-01     NA     NA
#> 2     2     1 2003-01-02     NA     NA
#> 3     3     1 2003-01-03     NA     NA
#> 4     4     1 2003-01-04     NA     NA
#> 5     5     1 2003-01-05     NA     NA
#> 6     6     1 2003-01-06     NA     NA
#> 7     7     1 2003-01-07     NA     NA
#> 8     8     1 2003-01-08     NA     NA
#> 9     9     1 2003-01-09     NA     NA
#> 10    10     1 2003-01-10     NA     NA
#> # ... with 772,077 more rows
```

6. Otras consideraciones

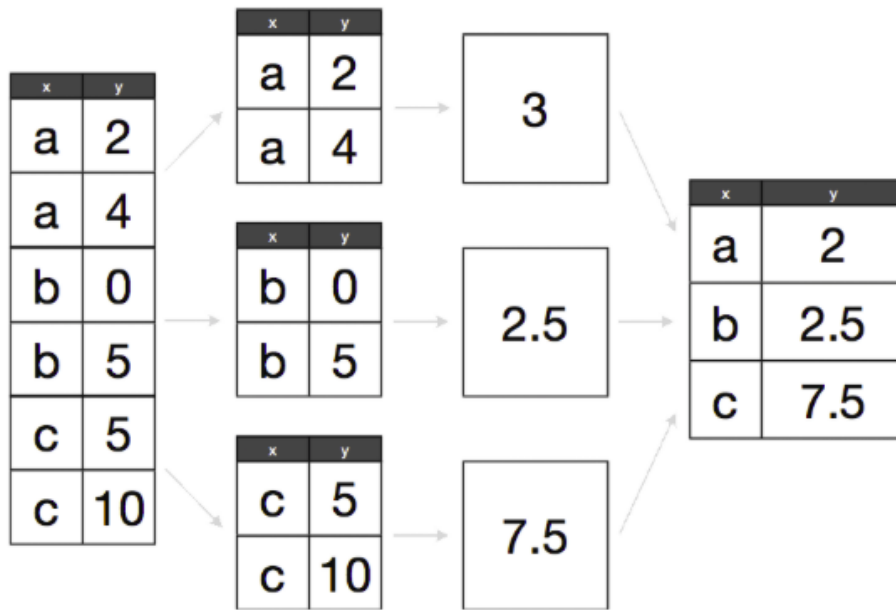
En las buenas prácticas es importante tomar en cuenta los siguientes puntos:

- Incluir un encabezado con el nombre de las variables.
- Los nombres de las variables deben ser entendibles (e.g. `AgeAtDiagnosis` es mejor que `AgeDx`).
- En general los datos se deben guardar en un archivo por tabla.
- Escribir un script con las modificaciones que se hicieron a los *datos crudos* (reproducibilidad).
- Otros aspectos importantes en la *limpieza* de datos son: selección del tipo de variables (por ejemplo fechas), datos faltantes, *typos* y detección de valores atípicos.

Divide-aplica-combina (*split-apply-combine*)

Muchos problemas de análisis de datos involucran la aplicación de la estrategia divide-aplica-combina, (Hadley Wickham, 2011 (<http://www.jstatsoft.org/v40/i01/paper>)) esta consiste en romper un problema en pedazos (de acuerdo a una variable de interés), operar sobre cada subconjunto de manera independiente (ej. calcular la media de cada grupo, ordenar observaciones por grupo, estandarizar por grupo) y después unir los pedazos nuevamente. El siguiente diagrama ejemplifica el paradigma de divide-aplica-combina:

- **Separa** la base de datos original.
- **Aplica** funciones a cada subconjunto.
- **Combina** los resultados en una nueva base de datos.



En esta sección trabajaremos con las siguientes bases de datos para ejemplificar las funciones de divide-aplica-combina:


```

flights <- read_csv("data/flights.csv")
#> Parsed with column specification:
#> cols(
#>   date = col_datetime(format = ""),
#>   hour = col_integer(),
#>   minute = col_integer(),
#>   dep = col_integer(),
#>   arr = col_integer(),
#>   dep_delay = col_integer(),
#>   arr_delay = col_integer(),
#>   carrier = col_character(),
#>   flight = col_integer(),
#>   dest = col_character(),
#>   plane = col_character(),
#>   cancelled = col_integer(),
#>   time = col_integer(),
#>   dist = col_integer()
#> )
flights
#> # A tibble: 227,496 x 14
#>       date      hour minute   dep   arr dep_delay arr_delay
#>   <dtm>    <int>  <int> <int> <int>      <int>    <int>
#> 1 2011-01-01 12:00:00    14     0  1400  1500         0       -10
#> 2 2011-01-02 12:00:00    14     1  1401  1501         1        -9
#> 3 2011-01-03 12:00:00    13    52  1352  1502        -8       -8
#> 4 2011-01-04 12:00:00    14     3  1403  1513         3         3
#> 5 2011-01-05 12:00:00    14     5  1405  1507         5        -3
#> 6 2011-01-06 12:00:00    13    59  1359  1503        -1        -7
#> 7 2011-01-07 12:00:00    13    59  1359  1509        -1        -1
#> 8 2011-01-08 12:00:00    13    55  1355  1454        -5       -16
#> 9 2011-01-09 12:00:00    14    43  1443  1554         43        44
#> 10 2011-01-10 12:00:00   14    43  1443  1553         43        43
#> # ... with 227,486 more rows, and 7 more variables: carrier <chr>,
#> #   flight <int>, dest <chr>, plane <chr>, cancelled <int>, time <int>,
#> #   dist <int>

```

```

weather <- read_csv("data/weather.csv")
#> Parsed with column specification:
#> cols(
#>   date = col_date(format = ""),
#>   hour = col_integer(),
#>   temp = col_double(),
#>   dew_point = col_double(),
#>   humidity = col_integer(),
#>   pressure = col_double(),
#>   visibility = col_double(),
#>   wind_dir = col_character(),
#>   wind_dir2 = col_integer(),
#>   wind_speed = col_double(),
#>   gust_speed = col_double(),
#>   precip = col_double(),
#>   conditions = col_character(),
#>   events = col_character()
#> )
weather
#> # A tibble: 8,723 x 14
#>       date      hour temp dew_point humidity pressure visibility wind_dir
#>   <date>    <int> <dbl>    <dbl>    <int>    <dbl>      <dbl>    <chr>
#> 1 2011-01-01     0  59.0    28.9      32    29.86        10    NNE
#> 2 2011-01-01     1  57.2    28.4      33    29.88        10    NNE
#> 3 2011-01-01     2  55.4    28.4      36    29.93        10   NNW
#> 4 2011-01-01     3  53.6    28.4      38    29.94        10  North
#> 5 2011-01-01     4   NA      NA      NA    29.99        10   NNW
#> 6 2011-01-01     5   NA      NA      NA    30.02        10  North

```

```
#> 7 2011-01-01      6 53.1      17.1      24 30.05      10 North
#> 8 2011-01-01      7 53.1      16.0      23 30.07      10 North
#> 9 2011-01-01      8 54.0      18.0      24 30.09      10 North
#> 10 2011-01-01     9 55.4      17.6      23 30.09      10 NNE
#> # ... with 8,713 more rows, and 6 more variables: wind_dir2 <int>,
#> #   wind_speed <dbl>, gust_speed <dbl>, precip <dbl>, conditions <chr>,
#> #   events <chr>
```

```
planes <- read_csv("data/planes.csv")
```

```
#> Parsed with column specification:
```

```
#> cols(
```

```
#>   plane = col_character(),
```

```
#>   year = col_integer(),
```

```
#>   mfr = col_character(),
```

```
#>   model = col_character(),
```

```
#>   no.eng = col_integer(),
```

```
#>   no.seats = col_integer(),
```

```
#>   speed = col_integer(),
```

```
#>   engine = col_character(),
```

```
#>   type = col_character()
```

```
#> )
```

```
planes
```

```
#> # A tibble: 2,853 x 9
```

```
#>   plane year      mfr      model no.eng no.seats speed
```

```
#>   <chr> <int>      <chr>      <chr> <int>    <int> <int>
```

```
#> 1 N576AA 1991 MCDONNELL DOUGLAS DC-9-82(MD-82)      2      172    NA
```

```
#> 2 N557AA 1993      MARZ BARRY      KITFOX IV      1        2    NA
```

```
#> 3 N403AA 1974      RAVEN      S55A      NA        1    60
```

```
#> 4 N492AA 1989 MCDONNELL DOUGLAS DC-9-82(MD-82)      2      172    NA
```

```
#> 5 N262AA 1985 MCDONNELL DOUGLAS DC-9-82(MD-82)      2      172    NA
```

```
#> 6 N493AA 1989 MCDONNELL DOUGLAS DC-9-82(MD-82)      2      172    NA
```

```
#> 7 N477AA 1988 MCDONNELL DOUGLAS DC-9-82(MD-82)      2      172    NA
```

```
#> 8 N476AA 1988 MCDONNELL DOUGLAS DC-9-82(MD-82)      2      172    NA
```

```
#> 9 N504AA   NA AUTHIER ANTHONY P      TIERRA II      1        2    NA
```

```
#> 10 N565AA 1987 MCDONNELL DOUGLAS DC-9-83(MD-83)      2      172    NA
```

```
#> # ... with 2,843 more rows, and 2 more variables: engine <chr>, type <chr>
```

```
airports <- read_csv("data/airports.csv")
```

```
#> Parsed with column specification:
```

```
#> cols(
```

```
#>   iata = col_character(),
```

```
#>   airport = col_character(),
```

```
#>   city = col_character(),
```

```
#>   state = col_character(),
```

```
#>   country = col_character(),
```

```
#>   lat = col_double(),
```

```
#>   long = col_double()
```

```
#> )
```

```
airports
```

```
#> # A tibble: 3,376 x 7
```

```
#>   iata      airport      city state country      lat
```

```
#>   <chr>      <chr>      <chr> <chr>    <chr>    <dbl>
```

```
#> 1 00M      Thigpen      Bay Springs MS      USA 31.95376
```

```
#> 2 00R Livingston Municipal Livingston TX      USA 30.68586
```

```
#> 3 00V      Meadow Lake Colorado Springs CO      USA 38.94575
```

```
#> 4 01G      Perry-Warsaw      Perry NY      USA 42.74135
```

```
#> 5 01J      Hilliard Airpark      Hilliard FL      USA 30.68801
```

```
#> 6 01M      Tishomingo County      Belmont MS      USA 34.49167
```

```
#> 7 02A      Gragg-Wade      Clanton AL      USA 32.85049
```

```
#> 8 02C      Capitol      Brookfield WI      USA 43.08751
```

```
#> 9 02G      Columbiana County East Liverpool OH      USA 40.67331
```

```
#> 10 03D      Memphis Memorial      Memphis MO      USA 40.44726
```

```
#> # ... with 3,366 more rows, and 1 more variables: long <dbl>
```

Cuando pensamos como implementar la estrategia divide-aplica-combina es natural pensar en iteraciones, por ejemplo utilizar un ciclo *for* para recorrer cada grupo de interés y aplicar las funciones, sin embargo la aplicación de ciclos *for* desemboca en código difícil de entender. Adicionalmente, *dplyr* es mucho más veloz.

Estudiaremos las siguientes funciones:

- **filter**: obten un subconjunto de las filas de acuerdo a un criterio.
- **select**: selecciona columnas de acuerdo al nombre
- **arrange**: reordena las filas
- **mutate**: agrega nuevas variables
- **summarise**: reduce variables a valores (crear nuevas bases de datos con resúmenes de variables de la base original)

Estas funciones trabajan de manera similar, el primer argumento que reciben es un *data frame* (usualmente en formato *limpio*), los argumentos que siguen indican que operación se va a efectuar y el resultado es un nuevo *data frame*.

Adicionalmente, se pueden usar con **group_by** que cambia el dominio de cada función, pasando de operar en el conjunto de datos completos a operar en grupos.

Veamos con ejemplos.

Filtrar

Creemos una base de datos de juguete para mostrar el funcionamiento de cada instrucción:

```
df_ej <- tibble(genero = c("mujer", "hombre", "mujer", "mujer", "hombre"),
  estatura = c(1.65, 1.80, 1.70, 1.60, 1.67))
df_ej
#> # A tibble: 5 x 2
#>   genero estatura
#>   <chr>   <dbl>
#> 1  mujer    1.65
#> 2  hombre    1.80
#> 3  mujer    1.70
#> 4  mujer    1.60
#> 5  hombre    1.67
```

El primer argumento de `filter()` es el nombre del *data frame*, los subsecuentes son las expresiones que indican que filas filtrar.

```
filter(df_ej, genero == "mujer")
#> # A tibble: 3 x 2
#>   genero estatura
#>   <chr>   <dbl>
#> 1  mujer    1.65
#> 2  mujer    1.70
#> 3  mujer    1.60
filter(df_ej, estatura > 1.65 & estatura < 1.75)
#> # A tibble: 2 x 2
#>   genero estatura
#>   <chr>   <dbl>
#> 1  mujer    1.70
#> 2  hombre    1.67
```

Algunos operadores importantes para filtrar son:

```
x > 1
x >= 1
x < 1
x <= 1
x != 1
x == 1
x %in% ("a", "b")
```

Debemos tener cuidado al usar `==`

```
sqrt(2) ^ 2 == 2
#> [1] FALSE
1/49 * 49 == 1
#> [1] FALSE
```

Los resultados de arriba se deben a que las computadoras usan aritmética de precisión finita:

```
print(1/49 * 49, digits = 20)
#> [1] 0.999999999999999988898
```

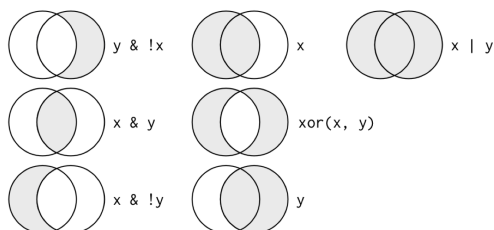
Para estos casos es útil usar la función `near()`

```
near(sqrt(2) ^ 2, 2)
#> [1] TRUE
near(1 / 49 * 49, 1)
#> [1] TRUE
```

Los operadores booleanos también son convenientes para filtrar:

```
# Conjuntos
a | b
a & b
a & !b
xor(a, b)
```

```
knitr::include_graphics("imagenes/transform-logical.png")
```



 Encuentra todos los vuelos hacia SFO ó OAK.

Los vuelos con un retraso mayor a una hora.

En los que el retraso de llegada es más del doble que el retraso de salida.

Seleccionar

Elegir columnas de un conjunto de datos.

```
df_ej
#> # A tibble: 5 x 2
#>   genero estatura
#>   <chr>    <dbl>
#> 1  mujer    1.65
#> 2  hombre    1.80
#> 3  mujer    1.70
#> 4  mujer    1.60
#> 5  hombre    1.67
select(df_ej, genero)
#> # A tibble: 5 x 1
#>   genero
#>   <chr>
#> 1  mujer
#> 2  hombre
#> 3  mujer
#> 4  mujer
#> 5  hombre
select(df_ej, -genero)
#> # A tibble: 5 x 1
#>   estatura
#>   <dbl>
#> 1    1.65
#> 2    1.80
#> 3    1.70
#> 4    1.60
#> 5    1.67
```

```
select(df_ej, starts_with("g"))
select(df_ej, contains("g"))
```



Ve la ayuda de select (?select) y escribe tres maneras de seleccionar las variables de retraso (delay).

Arreglar

Arreglar u ordenar de acuerdo al valor de una o más variables:

```
arrange(df_ej, genero)
#> # A tibble: 5 x 2
#>   genero estatura
#>   <chr>    <dbl>
#> 1  hombre    1.80
#> 2  hombre    1.67
#> 3  mujer    1.65
#> 4  mujer    1.70
#> 5  mujer    1.60
arrange(df_ej, desc(estatura))
#> # A tibble: 5 x 2
#>   genero estatura
#>   <chr>    <dbl>
#> 1  hombre    1.80
#> 2  mujer    1.70
#> 3  hombre    1.67
#> 4  mujer    1.65
#> 5  mujer    1.60
```



Ordena los vuelos por fecha de salida y hora.


¿Cuáles son los vuelos con mayor retraso?

¿Qué vuelos *ganaron* más tiempo en el aire?

Mutar

Mutar consiste en crear nuevas variables aplicando una función a columnas existentes:

```
mutate(df_ej, estatura_cm = estatura * 100)
#> # A tibble: 5 x 3
#>   genero estatura estatura_cm
#>   <chr>   <dbl>     <dbl>
#> 1 mujer     1.65       165
#> 2 hombre    1.80       180
#> 3 mujer     1.70       170
#> 4 mujer     1.60       160
#> 5 hombre    1.67       167
mutate(df_ej, estatura_cm = estatura * 100, estatura_in = estatura_cm * 0.3937)
#> # A tibble: 5 x 4
#>   genero estatura estatura_cm estatura_in
#>   <chr>   <dbl>     <dbl>     <dbl>
#> 1 mujer     1.65       165     64.9605
#> 2 hombre    1.80       180     70.8660
#> 3 mujer     1.70       170     66.9290
#> 4 mujer     1.60       160     62.9920
#> 5 hombre    1.67       167     65.7479
```

 Calcula la velocidad en millas por hora a partir de la variable tiempo y la distancia (en millas). ¿Qué vuelo fue el más rápido?

Crea una nueva variable que muestre cuánto tiempo se ganó o perdió durante el vuelo.

Hay muchas funciones que podemos usar para crear nuevas variables con `mutate()`, éstas deben cumplir ser funciones vectorizadas, es decir, reciben un vector de valores y devuelven un vector de la misma dimensión.

Summarise y resúmenes por grupo

Summarise sirve para crear nuevas bases de datos con resúmenes o agregaciones de los datos originales.

```
summarise(df_ej, promedio = mean(estatura))
#> # A tibble: 1 x 1
#>   promedio
#>   <dbl>
#> 1 1.684
```

Podemos hacer resúmenes por grupo, primero creamos una base de datos agrupada:

```
by_genero <- group_by(df_ej, genero)
by_genero
#> # A tibble: 5 x 2
#> # Groups:   genero [2]
#>   genero estatura
#>   <chr>   <dbl>
#> 1 mujer     1.65
#> 2 hombre    1.80
#> 3 mujer     1.70
#> 4 mujer     1.60
#> 5 hombre    1.67
```

y después operamos sobre cada grupo, creando un resumen a nivel grupo y uniendo los subconjuntos en una base nueva:

```
summarise(by_genero, promedio = mean(estatura))
#> # A tibble: 2 x 2
#>   genero promedio
#>   <chr>   <dbl>
#> 1 hombre 1.735
#> 2 mujer 1.650
```

 Calcula el retraso promedio por fecha.

¿Qué otros resúmenes puedes hacer para explorar el retraso por fecha?

- Algunas funciones útiles con *summarise* son `min(x)`, `median(x)`, `max(x)`, `quantile(x, p)`, `n()`, `sum(x)`, `sum(x > 1)`, `mean(x > 1)`, `sd(x)`.

```
flights$date_only <- as.Date(flights$date)
by_date <- group_by(flights, date_only)
no_miss <- filter(by_date, !is.na(dep))
delays <- summarise(no_miss, mean_delay = mean(dep_delay), n = n())
```

Operador pipeline

En R cuando uno hace varias operaciones es difícil leer y entender el código:

```
hourly_delay <- filter(summarise(group_by(filter(flights, !is.na(dep_delay)),
  date_only, hour), delay = mean(dep_delay), n = n()), n > 10)
```

La dificultad radica en que usualmente los parámetros se asignan después del nombre de la función usando (). El operador “Forward Pipe”

(%>%) cambia este orden, de manera que un parámetro que precede a la función es enviado (“piped”) a la función: $x \%>\% f(y)$ se vuelve $f(x,y)$, $x \%>\% f(y) \%>\% g(z)$ se vuelve $g(f(x, y), z)$. Es así que podemos reescribir el código para poder leer las operaciones que vamos aplicando de izquierda a derecha y de arriba hacia abajo.

Veamos como cambia el código anterior:

```
hourly_delay <- flights %>%
  filter(!is.na(dep_delay)) %>%
  group_by(date_only, hour) %>%
  summarise(delay = mean(dep_delay), n = n()) %>%
  filter(n > 10)
```

podemos leer %>% como “*después*”.

👉 ¿Qué destinos tienen el promedio de retrasos más alto?

¿Qué vuelos (compañía + vuelo) ocurren diario?

En promedio, ¿Cómo varían a lo largo del día los retrasos de vuelos no cancelados? (pista: hour + minute / 60)

Variables por grupo

En ocasiones es conveniente crear variables por grupo, por ejemplo estandarizar dentro de cada grupo $z = (x - \text{mean}(x)) / \text{sd}(x)$.

Veamos un ejemplo:

```
planes <- flights %>%
  filter(!is.na(arr_delay)) %>%
  group_by(plane) %>%
  filter(n() > 30)

planes %>%
  mutate(z_delay =
    (arr_delay - mean(arr_delay)) / sd(arr_delay)) %>%
  filter(z_delay > 5)
#> # A tibble: 1,403 x 16
#> # Groups:   plane [856]
#>
#>   date       hour minute  dep   arr dep_delay arr_delay
#>   <dtm>    <int>  <int> <int> <int>      <int>    <int>
#> 1 2011-01-28 12:00:00    15    16 1516 1916        351      326
#> 2 2011-01-27 12:00:00    18    22 1822 1945        234      210
#> 3 2011-01-27 12:00:00    21    37 2137 2254        242      219
#> 4 2011-01-27 12:00:00     0    11   11  216        168      137
#> 5 2011-01-27 12:00:00    22    37 2237  153        227      208
#> 6 2011-01-27 12:00:00    21    28 2128  136        231      216
#> 7 2011-01-26 12:00:00    11    46 1146 1633        171      193
#> 8 2011-01-26 12:00:00     9    49  949 1436        144      180
#> 9 2011-01-21 12:00:00    19    11 1911 2352         94      112
#> 10 2011-01-20 12:00:00     6    35  635  807        780      775
#> # ... with 1,393 more rows, and 9 more variables: carrier <chr>,
#> #   flight <int>, dest <chr>, plane <chr>, cancelled <int>, time <int>,
#> #   dist <int>, date_only <date>, z_delay <dbl>
```

Verbos de dos tablas

¿Cómo mostramos los retrasos de los vuelos en un mapa?

Para responder esta pregunta necesitamos unir la base de datos de vuelos con la de aeropuertos.

```
location <- airports %>%
  select(dest = iata, name = airport, lat, long)

flights %>%
  group_by(dest) %>%
  filter(!is.na(arr_delay)) %>%
  summarise(
    arr_delay = mean(arr_delay),
    n = n() ) %>%
  arrange(desc(arr_delay)) %>%
  left_join(location)
#> Joining, by = "dest"
#> # A tibble: 116 x 6
#>   dest arr_delay    n      name      lat
#>   <chr>    <dbl> <int>    <chr>    <dbl>
#> 1  ANC  26.08065  124 Ted Stevens Anchorage International 61.17432
#> 2  CID  17.80049   406 Eastern Iowa 41.88459
#> 3  DSM  15.95110   634 Des Moines International 41.53493
#> 4  SFO  14.89036 2800 San Francisco International 37.61900
#> 5  BPT  14.33333     3 Southeast Texas Regional 29.95083
#> 6  GRR  13.71729   665 Kent County International 42.88082
#> 7  DAY  13.67117   444 James M Cox Dayton Intl 39.90238
#> 8  VPS  12.45718   864 Eglin Air Force Base 30.48325
#> 9  ECP  12.42222   720      <NA>      NA
#> 10 SAV  12.33137   851 Savannah International 32.12758
#> # ... with 106 more rows, and 1 more variables: long <dbl>
```

Hay varias maneras de unir dos bases de datos y debemos pensar en el objetivo:


```

x <- tibble(name = c("John", "Paul", "George", "Ringo", "Stuart", "Pete"),
  instrument = c("guitar", "bass", "guitar", "drums", "bass",
    "drums"))

y <- tibble(name = c("John", "Paul", "George", "Ringo", "Brian"),
  band = c("TRUE", "TRUE", "TRUE", "TRUE", "FALSE"))

x
#> # A tibble: 6 x 2
#>   name instrument
#>   <chr>      <chr>
#> 1 John      guitar
#> 2 Paul      bass
#> 3 George    guitar
#> 4 Ringo     drums
#> 5 Stuart    bass
#> 6 Pete      drums
y
#> # A tibble: 5 x 2
#>   name band
#>   <chr> <chr>
#> 1 John  TRUE
#> 2 Paul  TRUE
#> 3 George TRUE
#> 4 Ringo TRUE
#> 5 Brian FALSE

inner_join(x, y)
#> Joining, by = "name"
#> # A tibble: 4 x 3
#>   name instrument band
#>   <chr>      <chr> <chr>
#> 1 John      guitar TRUE
#> 2 Paul      bass  TRUE
#> 3 George    guitar TRUE
#> 4 Ringo     drums TRUE
left_join(x, y)
#> Joining, by = "name"
#> # A tibble: 6 x 3
#>   name instrument band
#>   <chr>      <chr> <chr>
#> 1 John      guitar TRUE
#> 2 Paul      bass  TRUE
#> 3 George    guitar TRUE
#> 4 Ringo     drums TRUE
#> 5 Stuart    bass  <NA>
#> 6 Pete      drums  <NA>
semi_join(x, y)
#> Joining, by = "name"
#> # A tibble: 4 x 2
#>   name instrument
#>   <chr>      <chr>
#> 1 John      guitar
#> 2 Paul      bass
#> 3 George    guitar
#> 4 Ringo     drums
anti_join(x, y)
#> Joining, by = "name"
#> # A tibble: 2 x 2
#>   name instrument
#>   <chr>      <chr>
#> 1 Pete      drums
#> 2 Stuart    bass

```

Resumamos lo que observamos arriba:

Tipo	Acción
inner	Incluye únicamente las filas que aparecen tanto en x como en y
left	Incluye todas las filas en x y las filas de y que coincidan
semi	Incluye las filas de x que coincidan con y
anti	Incluye las filas de x que no coinciden con y

Ahora combinamos datos a nivel hora con condiciones climáticas, ¿cuál es el tipo de unión adecuado?

```
hourly_delay <- flights %>%
  group_by(date_only, hour) %>%
  filter(!is.na(dep_delay)) %>%
  summarise(
    delay = mean(dep_delay),
    n = n() ) %>%
  filter(n > 10)

delay_weather <- hourly_delay %>% left_join(weather)
#> Joining, by = "hour"

arrange(delay_weather, -delay)
#> # A tibble: 2,091,842 x 17
#> # Groups:   date_only [365]
#>   date_only hour    delay      n    date temp dew_point humidity
#>   <date> <int>    <dbl> <int>   <date> <dbl>    <dbl>    <int>
#> 1 2011-05-12    23 184.2121    33 2011-01-02 43.0      28.9      58
#> 2 2011-05-12    23 184.2121    33 2011-01-03 39.0      27.0      62
#> 3 2011-05-12    23 184.2121    33 2011-01-04 50.0      45.0      83
#> 4 2011-05-12    23 184.2121    33 2011-01-05 62.6      60.8      94
#> 5 2011-05-12    23 184.2121    33 2011-01-06 53.1      36.0      52
#> 6 2011-05-12    23 184.2121    33 2011-01-07 46.9      36.0      66
#> 7 2011-05-12    23 184.2121    33 2011-01-08 50.0      43.0      77
#> 8 2011-05-12    23 184.2121    33 2011-01-09 53.1      30.0      41
#> 9 2011-05-12    23 184.2121    33 2011-01-10 41.0      37.0      86
#> 10 2011-05-12    23 184.2121    33 2011-01-11 39.9      32.0      73
#> # ... with 2,091,832 more rows, and 9 more variables: pressure <dbl>,
#> #   visibility <dbl>, wind_dir <chr>, wind_dir2 <int>, wind_speed <dbl>,
#> #   gust_speed <dbl>, precip <dbl>, conditions <chr>, events <chr>
```



¿Qué condiciones climáticas están asociadas con retrasos en las salidas de Houston?

Explora si los aviones más viejos están asociados a mayores retrasos, responde con una gráfica.

Recursos adicionales

- Tidy Data (<http://vita.had.co.nz/papers/tidy-data.pdf>), Hadley Wickham.
- The Split-Apply-Combine Strategy for Data Analysis (<http://www.jstatsoft.org/v40/i01/paper>), Hadley Wickham 2011.
- Data Wrangling Cheat Sheet (<https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>), RStudio.