Practico 12

Se usaron datos sacados de este paper

Introducción

En este práctico (. . .)

En particular, utilizaremos datos de la publicación ***Comparative transcriptomics analyses across species, organs, and developmental stages reveal functionally constrained lncRNAs*** (Darbellay & Necsulea, 2019).

A su vez se ejemplificarán las funciones básicas a emplear con un dataset estándar empleado en muchas demostraciones: el dataset 'iris'.

Precalentamiento

Precalentamiento

En la mayoría de los casos los datos con los que se trabaja suelen ser importados de archivos alojados en disco duro.

Para este propósito se utiliza la librería **readr**. Las funciones de la misma son capaces de leer archivos de varios formatos: CSV (*comma-separated values*), TSV (*tab-separated > alues*) y otros.

Las funciones alojadas en esta librería comienzan todas con el prefijo **read_***, acompañadas del formato de texto que son capaces de leer. Así, por ejemplo, read_csv() es la función de esta librería diseñada para cargar archivos de texto con formato csv.

En general estas funciones también poseen una sintaxis similar, por lo que al aprenderse a utilizar una ya se posee el conocimiento para lograr cargar otros formatos.

cargando datos

```
# cargamos la librería tidyverse.
# Esta contiene a todas las librerias asociadas (e.g. readr, tidyr, dplyr)
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
# se lee el conjunto de datos a utilizar y se los aloja en una variable
vuelos = readr::read_csv('airports_1.csv')
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   County = col_logical(),
##   `Kobuk Airport` = col_logical(),
##   `Key Largo` = col_logical(),
##   `International Airport` = col_logical(),
##   `Branch County Memorial Airport` = col_logical(),
##   `Offutt Afb` = col_logical(),
##   Kahului = col_logical(),
##   `Ogdensburg Intl` = col_logical(),
##   `Johnson County Airport` = col_logical(),
##   `Will Rogers World` = col_logical(),
##   `LM Clayton Airport` = col_logical(),
##   `Old Harbor Airport` = col_logical(),
##   `Olympia Regional Airpor` = col_logical(),
##   `Nogales Intl` = col_logical(),
##   `San Diego Old Town Transit Center` = col_logical(),
##   `Olive Branch Muni` = col_logical(),
##   `Eppley Afld` = col_logical(),
##   Nome = col_logical(),
##   `Ormond Beach municipal Airport` = col_logical(),
##   `Oneonta Municipal Airport` = col_logical()
##   # ... with 429 more columns
## )
```

```
## See spec(...) for full column specifications.
```

```
## Warning: 451 parsing failures.
##  row                           col           expected                 actual             file
## 1001 County                        1/0/T/F/TRUE/FALSE 27.265833/-80.851111   'airports_1.csv'
## 1002 Kobuk Airport                 1/0/T/F/TRUE/FALSE 66.912222/-156.897222  'airports_1.csv'
## 1003 Key Largo                     1/0/T/F/TRUE/FALSE 25.325393/-80.274775   'airports_1.csv'
## 1004 International Airport         1/0/T/F/TRUE/FALSE 29.1725/-82.224167     'airports_1.csv'
## 1005 Branch County Memorial Airport 1/0/T/F/TRUE/FALSE 41.9335691/-85.0522935 'airports_1.csv'
## .... .......................... .................. ...................... .................
## See problems(...) for more details.
```

```
vuelos
```

```
## # A tibble: 1,458 x 1,441
##    `Flight code (a~ `Lansdowne Airp~ `Moton Field Mu~ `Schaumburg Reg~
##    <chr>            <chr>            <chr>            <chr>
##  1 04G (1044)       41.1304722/-80.~ <NA>             <NA>
##  2 06A (264)        <NA>             32.4605722/-85.~ <NA>
```

```
##  3 06C (801)        <NA>            <NA>            41.9893408/-88.~
##  4 06N (523)        <NA>            <NA>            <NA>
##  5 09J (11)         <NA>            <NA>            <NA>
##  6 0A9 (1593)       <NA>            <NA>            <NA>
##  7 0G6 (730)        <NA>            <NA>            <NA>
##  8 0G7 (492)        <NA>            <NA>            <NA>
##  9 0P2 (1000)       <NA>            <NA>            <NA>
## 10 0S9 (108)        <NA>            <NA>            <NA>
## # ... with 1,448 more rows, and 1,437 more variables: `Randall Airport` <chr>,
## #   `Jekyll Island Airport` <chr>, `Elizabethton Municipal Airport` <chr>,
## #   `Williams County Airport` <chr>, `Finger Lakes Regional Airport` <chr>,
## #   `Shoestring Aviation Airfield` <chr>, `Jefferson County Intl` <chr>,
## #   `Harford County Airport` <chr>, `Galt Field Airport` <chr>, `Port
## #   Bucyrus-Crawford County Airport` <chr>, `Jackson County Airport` <chr>,
## #   `Martin Campbell Field Airport` <chr>, `Mansfield Municipal` <chr>,
## #   `Frazier Lake Airpark` <chr>, `Clow International Airport` <chr>, `Kent
## #   State Airport` <chr>, `Grand Canyon West Airport` <chr>, `Effingham
## #   Memorial Airport` <chr>, `Fortman Airport` <chr>, `Point Roberts
## #   Airpark` <chr>, `Clarke CO` <chr>, `Lowell City Airport` <chr>, `Suwannee
## #   County Airport` <chr>, `Forest Lake Airport` <chr>, `Grove City
## #   Airport` <chr>, `Mark Anton Airport` <chr>, `Plum Island Airport` <chr>,
## #   `Jefferson County Airpark` <chr>, `Somerset County Airport` <chr>, `Shelby
## #   County Airport` <chr>, `Quincy Municipal Airport` <chr>, `Atmautluak
## #   Airport` <chr>, `Heber City Municipal Airport` <chr>, `Lynden
## #   Airport` <chr>, `Ephraim-Gibraltar Airport` <chr>, `Wadsworth
## #   Municipal` <chr>, `Ashland County Airport` <chr>, `Ridgeland
## #   Airport` <chr>, `Put-in-Bay Airport` <chr>, `Perry-Foley Airport` <chr>,
## #   `Braceville Airport` <chr>, `Cherokee County Airport` <chr>, `Gilmer County
## #   Airport` <chr>, `Chemehuevi Valley` <chr>, `Polk County Airport - Cornelius
## #   Moore Field` <chr>, `Clayton County Tara Field` <chr>, `Isbell Field
## #   Airport` <chr>, `Robertson Field` <chr>, `Pittsburgh-Monroeville
## #   Airport` <chr>, `Hamburg Inc Airport` <chr>, `Youngstown Elser Metro
## #   Airport` <chr>, `Putnam County Airport` <chr>, `Dell Flight Strip` <chr>,
## #   `Madison GA Municipal Airport` <chr>, `DeFuniak Springs Airport` <chr>,
## #   `Fernandina Beach Municipal Airport` <chr>, Packwood <chr>, `East Troy
## #   Municipal Airport` <chr>, `Saratoga County Airport` <chr>, `Ocean Isle
## #   Beach Airport` <chr>, `Griffin-Spalding County Airport` <chr>, `Saluda
## #   County` <chr>, `Tok Junction Airport` <chr>, `Big Timber Airport` <chr>,
## #   Florence <chr>, `Welke Airport` <chr>, `Cairo-Grady County Airport` <chr>,
## #   `Spring Hill Airport` <chr>, `Foster Field` <chr>, `Germack Airport` <chr>,
## #   `Spitfire Aerodrome` <chr>, `Garland Airport` <chr>, `Richland
## #   Airport` <chr>, `Bamberg County Airport` <chr>, `Covington Municipal
## #   Airport` <chr>, `Barwick Lafayette Airport` <chr>, `Rock Airport` <chr>,
## #   `Phoenix Regional Airport` <chr>, `Colorado Springs East` <chr>,
## #   `Apalachicola Regional Airport` <chr>, `Andrau Airport` <chr>, `Lehigh
## #   Valley Intl` <chr>, `Abilene Rgnl` <chr>, `Ambler Airport` <chr>,
## #   `Albuquerque International Sunport` <chr>, `Aberdeen Regional
## #   Airport` <chr>, `Southwest Georgia Regional Airport` <chr>, `Jimmy Carter
## #   Regional` <chr>, `Nantucket Mem` <chr>, `Waco Rgnl` <chr>, Arcata <chr>,
## #   `Atlantic City Intl` <chr>, `Adak Airport` <chr>, `Ardmore Muni` <chr>,
## #   Kodiak <chr>, Addison <chr>, `Andrews Afb` <chr>, `Allakaket
## #   Airport` <chr>, `Alexandria Intl` <chr>, `Kake Airport` <chr>, ...
```

Ejercicio 1

Cargue en una variable los datos alojados en el archivo 'gapminder_1.csv'. Esta tabla posee datos acerca de la evolución de diversos parámetros a lo largo de años para varios países. Los mismos son una versión modificada de aquellos disponibles en la página de la Gapminder Foundation.

```
# cargamos la librería tidyverse.
# Esta contiene a todas las librerias asociadas (e.g. readr, tidyr, dplyr)
library(tidyverse)

# se lee el conjunto de datos a utilizar y se los aloja en una variable
gapminder_1 = readr::read_csv('gapminder_1.csv')
```

```
## Parsed with column specification:
## cols(
##   .default = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

edicion

La librería tidyverse fue construída para trabajar sobre conjuntos de datos que poseen un formato en particular: estos datos son llamados *tidy data*.

Para que un set sea caracterizado como *tidy data* debe poseer tres características que están interrelacionadas:

- Cada variable debe estar representada en una columna
- Cada observación debe estar representada en una fila
- Cada valor debe estar alojado en una celda

Esto no siempre se cumple en la vida cotidiana. Los *datos crudos* con los que solemos presentarnos suelen tener uno (o los dos) siguientes defectos que nos > impiden definirlos como *tidy datasets*:

- Una variable puede estar siendo representada en más de una columna
- Una observación puede estar representada en más de una fila

Estos problemas pueden ser resueltos con las funciones de la librería **tidyr**. Como su nombre indica, esta librería tiene como objetivo generar *tidy datasets* > en R.

Sus funciones más importantes son

• **pivot_longer()**: hace a un dataset más *largo*, alojando valores que antes se encontraban en columnas en filas.

• **pivot_wider()**: lo opuesto de pibot_longer(). Un dataset se hace más *ancho* al aumentar el número de columnas y disminuir el número de filas.

• **separate()**: separa observaciones de una columna en varias columnas, al separar los valores de sus celdas según un separador.

- **unite()**: lo opuesto de separate(). Se unen observaciones de diferentes columnas, utilizando para ello un caracter especificado.

Cinco de las funciones de la librería **dplyr** permiten hacer un conjunto de operaciones que, en su conjunto, logran resolver la mayoría de los problemas relacionados con la manipulación de datos en un *tidy dataset*.

Estas funciones son

- **filter()**:
- **arrange()**
- **select()**
- **mutate()**
- **summarise()**

En esta sección nos focalizamos en **filter()**.

El uso efectivo de esta función se basa en los siguientes operadores: > (*mayor*), >= (*mayor o igual*), < (*menor*), <= (*menor o igual*), != (*dstinto de*) y == (*igual*).

A su vez se pueden combinar argumentos en el filtrado usando operadores booleanos clásicos como los encontrados en otros lenguajes (*i.e.* Bash o awk). Estos operadores son: **&** (operador *AND*), | (operador *OR*) y **!** (*negación*).

llevando datos a estilo tidy

```
library(magrittr)
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:purrr':
##
##     set_names
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
# se realizan algunas modificaciones al set de datos, para transformarlo en 'tidy data'
## levamos a formato alargado
vuelos %<>% tidyr::pivot_longer(., cols = colnames(.)[-1],
                                names_to = 'aeropuerto',
                                values_to = 'lat/lon')

vuelos
```

```
## # A tibble: 2,099,520 x 3
##    `Flight code (altitude)` aeropuerto                    `lat/lon`
##    <chr>                    <chr>                         <chr>
```

```
##  1 04G (1044)               Lansdowne Airport              41.1304722/-80.61958~
##  2 04G (1044)               Moton Field Municipal Airport  <NA>
##  3 04G (1044)               Schaumburg Regional            <NA>
##  4 04G (1044)               Randall Airport                <NA>
##  5 04G (1044)               Jekyll Island Airport          <NA>
##  6 04G (1044)               Elizabethton Municipal Airport <NA>
##  7 04G (1044)               Williams County Airport        <NA>
##  8 04G (1044)               Finger Lakes Regional Airport  <NA>
##  9 04G (1044)               Shoestring Aviation Airfield   <NA>
## 10 04G (1044)               Jefferson County Intl          <NA>
## # ... with 2,099,510 more rows
```

```
# filtramos los valores que son NA
vuelos %<>% dplyr::filter(!is.na(`lat/lon`))

vuelos
```

```
## # A tibble: 1,007 x 3
##    `Flight code (altitude)` aeropuerto                    `lat/lon`
##    <chr>                    <chr>                         <chr>
##  1 04G (1044)               Lansdowne Airport             41.1304722/-80.6195833
##  2 06A (264)                Moton Field Municipal Airport 32.4605722/-85.6800278
##  3 06C (801)                Schaumburg Regional           41.9893408/-88.1012428
##  4 06N (523)                Randall Airport               41.431912/-74.3915611
##  5 09J (11)                 Jekyll Island Airport         31.0744722/-81.4277778
##  6 0A9 (1593)               Elizabethton Municipal Airpo~ 36.3712222/-82.1734167
##  7 0G6 (730)                Williams County Airport       41.4673056/-84.5067778
##  8 0G7 (492)                Finger Lakes Regional Airport 42.8835647/-76.7812318
##  9 0P2 (1000)               Shoestring Aviation Airfield  39.7948244/-76.6471914
## 10 0S9 (108)                Jefferson County Intl         48.0538086/-122.81064~
## # ... with 997 more rows
```

```
# separamos algunos valores que estan unidos
vuelos %<>% tidyr::separate(., col = `Flight code (altitude)`,
                            sep = ' ',
                            into = c('Flight code', 'altitude'))

vuelos
```

```
## # A tibble: 1,007 x 4
##    `Flight code` altitude aeropuerto                    `lat/lon`
##    <chr>         <chr>    <chr>                         <chr>
##  1 04G           (1044)   Lansdowne Airport             41.1304722/-80.6195833
##  2 06A           (264)    Moton Field Municipal Airport 32.4605722/-85.6800278
##  3 06C           (801)    Schaumburg Regional           41.9893408/-88.1012428
##  4 06N           (523)    Randall Airport               41.431912/-74.3915611
##  5 09J           (11)     Jekyll Island Airport         31.0744722/-81.4277778
##  6 0A9           (1593)   Elizabethton Municipal Airport 36.3712222/-82.1734167
##  7 0G6           (730)    Williams County Airport       41.4673056/-84.5067778
##  8 0G7           (492)    Finger Lakes Regional Airport 42.8835647/-76.7812318
##  9 0P2           (1000)   Shoestring Aviation Airfield  39.7948244/-76.6471914
## 10 0S9           (108)    Jefferson County Intl         48.0538086/-122.8106436
## # ... with 997 more rows
```

```
vuelos %<>% tidyr::separate(., col = `lat/lon`,
                            sep = '/',
```

```
                              into = c('lat', 'lon'))
vuelos
```

```
## # A tibble: 1,007 x 5
##    `Flight code` altitude aeropuerto                      lat        lon
##    <chr>         <chr>    <chr>                           <chr>      <chr>
##  1 04G           (1044)   Lansdowne Airport               41.1304722 -80.6195833
##  2 06A           (264)    Moton Field Municipal Airport   32.4605722 -85.6800278
##  3 06C           (801)    Schaumburg Regional             41.9893408 -88.1012428
##  4 06N           (523)    Randall Airport                 41.431912  -74.3915611
##  5 09J           (11)     Jekyll Island Airport           31.0744722 -81.4277778
##  6 0A9           (1593)   Elizabethton Municipal Airport  36.3712222 -82.1734167
##  7 0G6           (730)    Williams County Airport         41.4673056 -84.5067778
##  8 0G7           (492)    Finger Lakes Regional Airport   42.8835647 -76.7812318
##  9 0P2           (1000)   Shoestring Aviation Airfield    39.7948244 -76.6471914
## 10 0S9           (108)    Jefferson County Intl           48.0538086 -122.8106436
## # ... with 997 more rows
```

```
# modificamos el tipo de estos datos
vuelos$lat %<>% as.numeric()
vuelos$lon %<>% as.numeric()

vuelos
```

```
## # A tibble: 1,007 x 5
##    `Flight code` altitude aeropuerto                       lat    lon
##    <chr>         <chr>    <chr>                           <dbl>  <dbl>
##  1 04G           (1044)   Lansdowne Airport                41.1  -80.6
##  2 06A           (264)    Moton Field Municipal Airport    32.5  -85.7
##  3 06C           (801)    Schaumburg Regional              42.0  -88.1
##  4 06N           (523)    Randall Airport                  41.4  -74.4
##  5 09J           (11)     Jekyll Island Airport            31.1  -81.4
##  6 0A9           (1593)   Elizabethton Municipal Airport   36.4  -82.2
##  7 0G6           (730)    Williams County Airport          41.5  -84.5
##  8 0G7           (492)    Finger Lakes Regional Airport    42.9  -76.8
##  9 0P2           (1000)   Shoestring Aviation Airfield     39.8  -76.6
## 10 0S9           (108)    Jefferson County Intl            48.1 -123.
## # ... with 997 more rows
```

Ejercicio 2

Lleve los datos presentes en la variable de clase *tibble* donde alojó los datos de Gapminder a un formato de tipo tidy. **HAY QUE ACLARAR QUE SON DATOS DE EXPECTATIVA DE VIDAAAAA**

```
library(magrittr)
# se realizan algunas modificaciones al set de datos, para transformarlo en 'tidy data'
## levamos a formato alargado
gapminder_1 %<>% tidyr::pivot_longer(., cols = colnames(.)[-1],
                                     names_to = 'Continente/Pais',
                                     values_to = 'expectativaVida')

# separamos algunos valores que estan unidos
gapminder_1 %<>% tidyr::separate(., col = `Continente/Pais`,
                                 sep = '/',
                                 into = c('Continente', 'Pais'))
```

mas edicion

a b c

a b c

mas edicion: edicion de texto y union de tablas

```
# editamos strings con la libreria stringr.
#utilizamos expresiones regulares para definir dos caracteres a modificar; '(' y ')'.

vuelos$altitude %<>% stringr::str_replace_all(string = .,
                                              pattern = '\\(|\\)',
                                              replacement = '')
vuelos$altitude %<>% as.numeric()

vuelos
```

```
## # A tibble: 1,007 x 5
##    `Flight code` altitude aeropuerto                        lat    lon
##    <chr>            <dbl> <chr>                           <dbl>  <dbl>
##  1 04G               1044 Lansdowne Airport                41.1  -80.6
##  2 06A                264 Moton Field Municipal Airport    32.5  -85.7
##  3 06C                801 Schaumburg Regional              42.0  -88.1
##  4 06N                523 Randall Airport                  41.4  -74.4
##  5 09J                 11 Jekyll Island Airport            31.1  -81.4
##  6 0A9               1593 Elizabethton Municipal Airport   36.4  -82.2
##  7 0G6                730 Williams County Airport          41.5  -84.5
##  8 0G7                492 Finger Lakes Regional Airport    42.9  -76.8
##  9 0P2               1000 Shoestring Aviation Airfield     39.8  -76.6
## 10 0S9                108 Jefferson County Intl            48.1 -123.
## # ... with 997 more rows
```

```
# leo segunda tabla que tiene informacion referente a la ubicacion de los aeropuertos.
# nuevamente, se separan datos en una columna
aeropuertos = readr::read_tsv('airports_2.tsv') %>%
              tidyr::separate(data = .,
                              col = 'tzone',
                              sep = '/',
                              into = c('continente', 'ciudad'))
```

```
## Parsed with column specification:
## cols(
##   airport = col_character(),
##   tzone = col_character()
## )
```

```
aeropuertos
```

```
## # A tibble: 1,458 x 3
##    airport                          continente ciudad
##    <chr>                            <chr>      <chr>
##  1 Lansdowne Airport                America    New_York
##  2 Moton Field Municipal Airport    America    Chicago
##  3 Schaumburg Regional              America    Chicago
##  4 Randall Airport                  America    New_York
##  5 Jekyll Island Airport            America    New_York
##  6 Elizabethton Municipal Airport   America    New_York
##  7 Williams County Airport          America    New_York
##  8 Finger Lakes Regional Airport    America    New_York
##  9 Shoestring Aviation Airfield     America    New_York
## 10 Jefferson County Intl            America    Los_Angeles
## # ... with 1,448 more rows
```

```
# uno ambas tablas
vuelos %>%
  left_join(x = .,
            y = aeropuertos,
            by = c('aeropuerto' = 'airport')) -> vuelos_final

vuelos_final
```

```
## # A tibble: 1,053 x 7
##    `Flight code` altitude aeropuerto              lat    lon continente ciudad
##    <chr>            <dbl> <chr>                 <dbl>  <dbl> <chr>      <chr>
##  1 04G               1044 Lansdowne Airport      41.1  -80.6 America    New_York
##  2 06A                264 Moton Field Municipa~  32.5  -85.7 America    Chicago
##  3 06C                801 Schaumburg Regional    42.0  -88.1 America    Chicago
##  4 06N                523 Randall Airport        41.4  -74.4 America    New_York
##  5 09J                 11 Jekyll Island Airport  31.1  -81.4 America    New_York
##  6 0A9               1593 Elizabethton Municip~  36.4  -82.2 America    New_York
##  7 0G6                730 Williams County Airp~  41.5  -84.5 America    New_York
##  8 0G7                492 Finger Lakes Regiona~  42.9  -76.8 America    New_York
##  9 0P2               1000 Shoestring Aviation ~  39.8  -76.6 America    New_York
## 10 0S9                108 Jefferson County Intl  48.1 -123.  America    Los_Ang~
## # ... with 1,043 more rows
```

Ejercicio 3

Cargue la tabla alojada en el archivo 'gapminder_2.tsv'. Una la misma junto a los datos que viene trabajando para crear una tabla final.

```
# leo segunda tabla que tiene informacion referente a la ubicacion de los aeropuertos.
# nuevamente, se separan datos en una columna
gapminder_2 = readr::read_tsv('gapminder_2.tsv')
```

```
## Parsed with column specification:
## cols(
##   country = col_character(),
##   year = col_double(),
##   pop = col_double(),
##   gdpPercap = col_double()
## )
```

```
# uno ambas tablas
gapminder_1 %>%
  left_join(x = .,
            y = gapminder_2,
            by = c('Pais' = 'country')) -> gapminder_final
```

The first step in making this plot [un scatterplot] is to create a new dataset that reflects the mapping of x-position to A, y-position to C, and shape to D. x-position, y-position, and shape > are examples of aesthetics, things that we can perceive on the graphic.

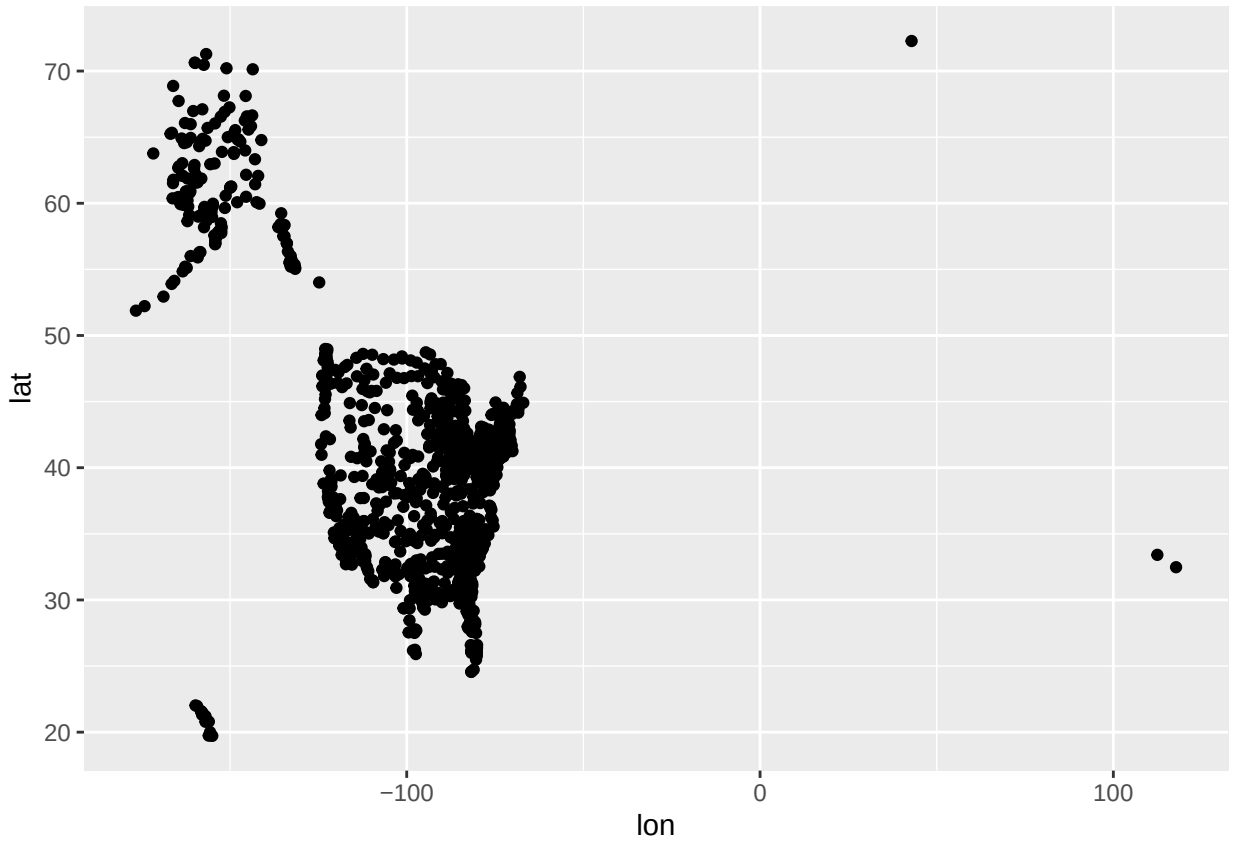
We can create many different types of plots using this same basic specification. For example, if we were to draw lines instead of points, we would get a line plot. If we used bars, we would get a bar plot. Bars, lines, and points are all examples of geometric objects

Faceting is a more general case of the techniques known as conditioning, trellising, and latticing, and produces small multiples showing different subsets of the data.

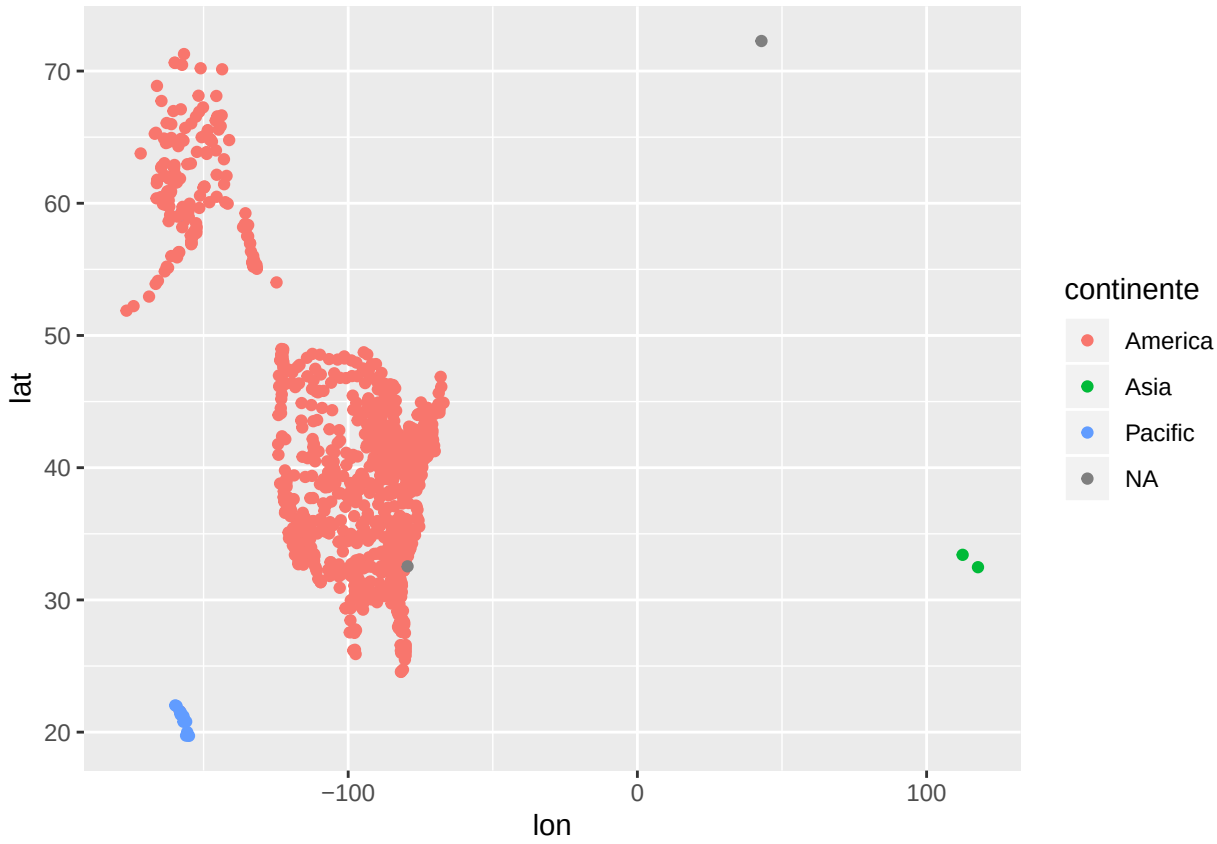
- In the examples above, we have seen some of the components that make up a plot: • data and aesthetic mappings, • geometric objects, • scales, and • facet specification. We have also touched on two other components: • statistical transformations, and • the coordinate system.

visualizando datos

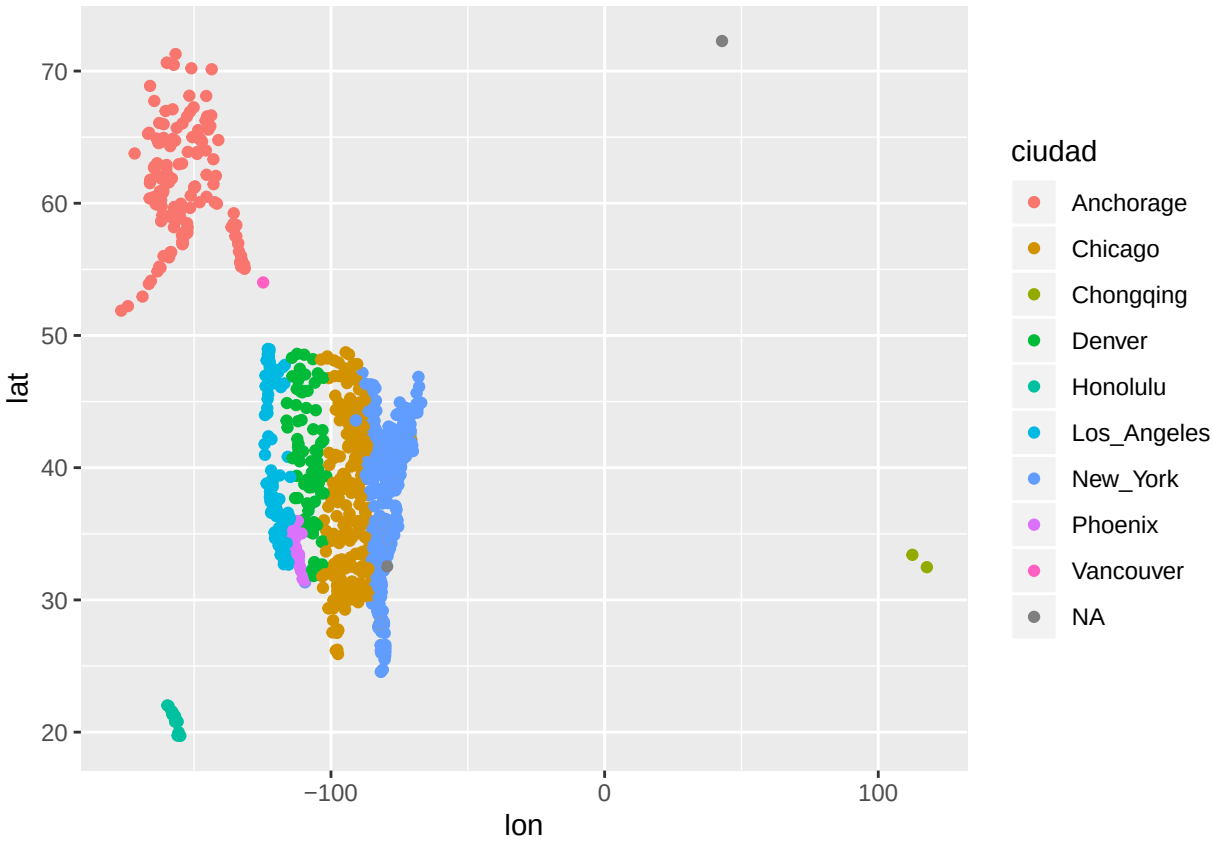
```
# realizamos un plot sencillo
vuelos_final %>%
  ggplot(data = .,
         mapping = aes(x = lon, y = lat)) +
  geom_point()
```




```
# podemos colorear segun los continentes o las ciudades
vuelos_final %>%
  ggplot(data = .,
         mapping = aes(x = lon, y = lat, color = continente)) +
  geom_point()
```



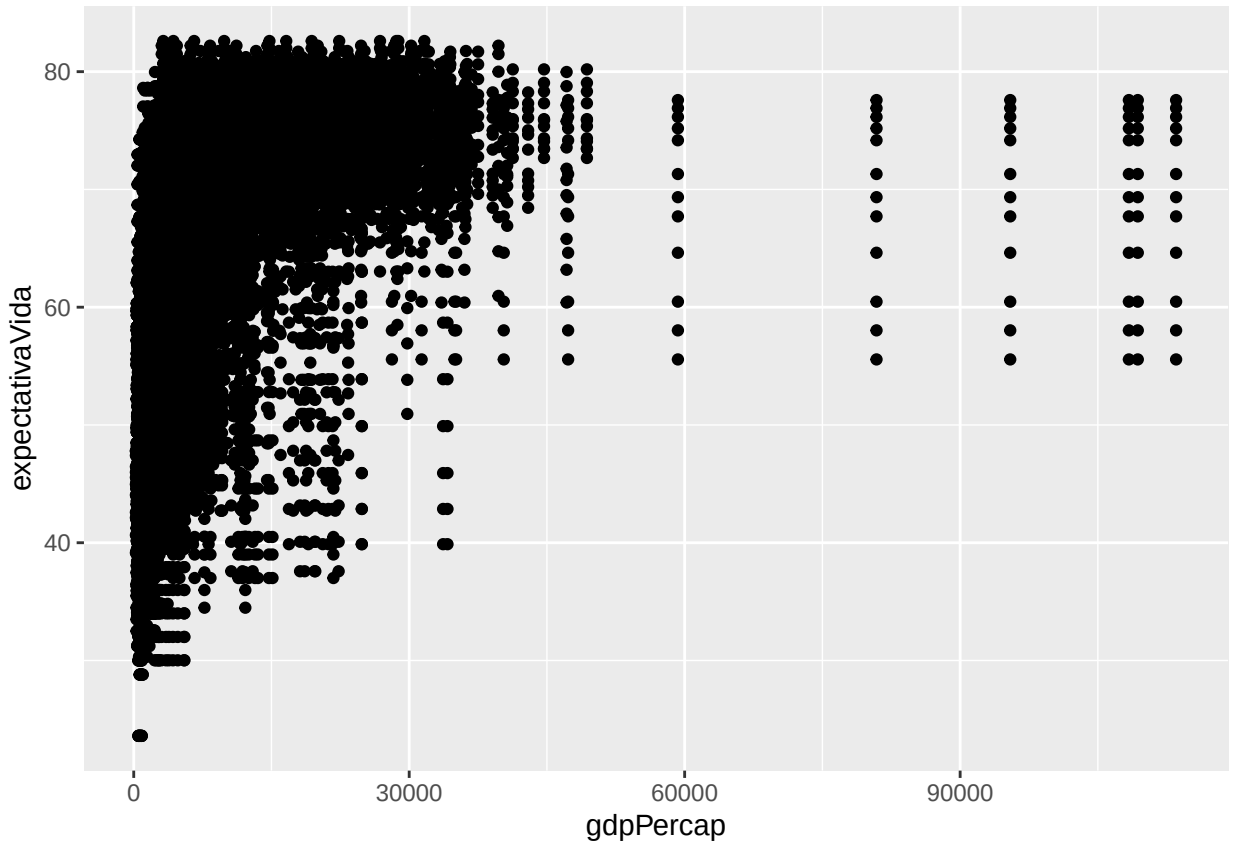

```
vuelos_final %>%
  ggplot(data = .,
         mapping = aes(x = lon, y = lat, color = ciudad)) +
  geom_point()
```



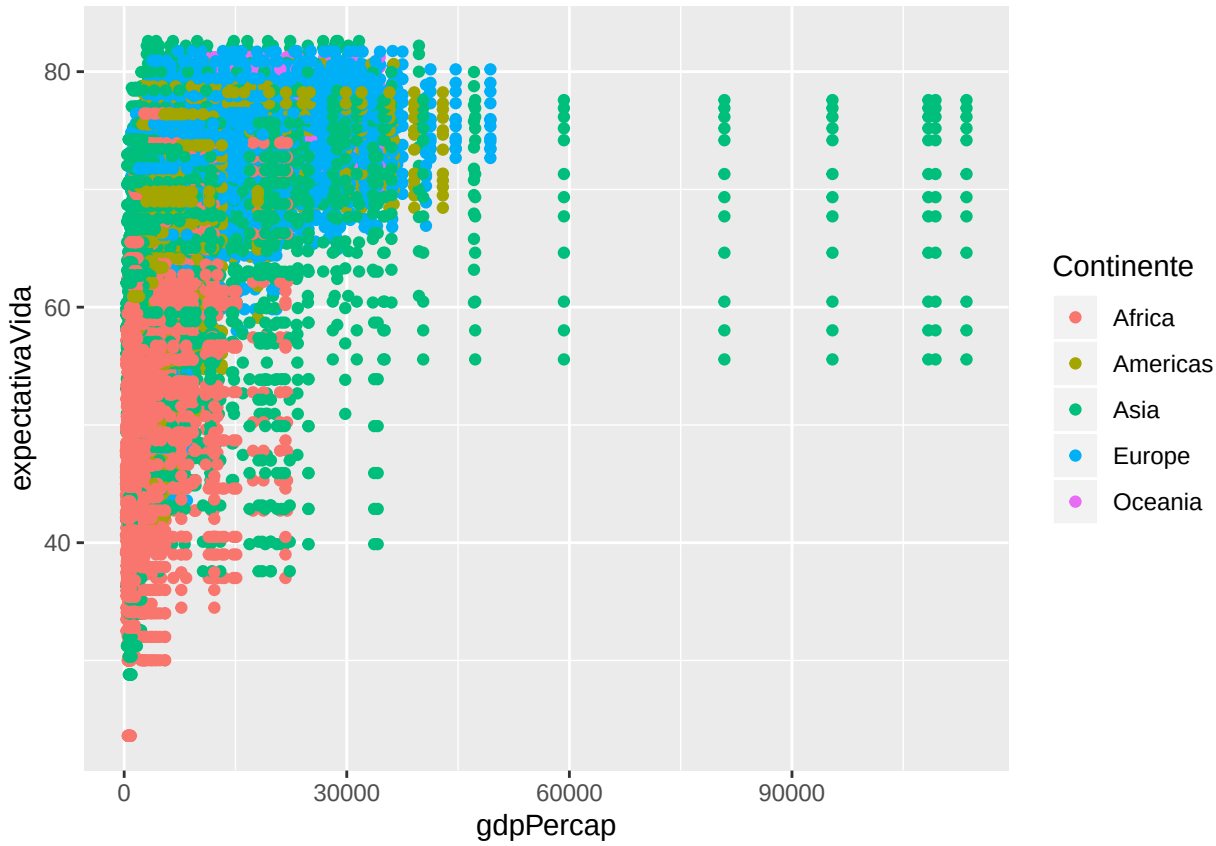
Ejercicio 4.1

a) Grafique la expectativa de vida en función del GDP per capita (). ***b)*** Modifique el código anterior para colorear en función del continente. ***c)*** Filtre para obtener, además, solo datos para África y Europa. ***d)*** Seleccione datos para África, y coloree en función del país.

```
# a)
gapminder_final %>%
  ggplot(data = .,
         mapping = aes(x = gdpPercap, y = expectativaVida)) +
  geom_point()
```

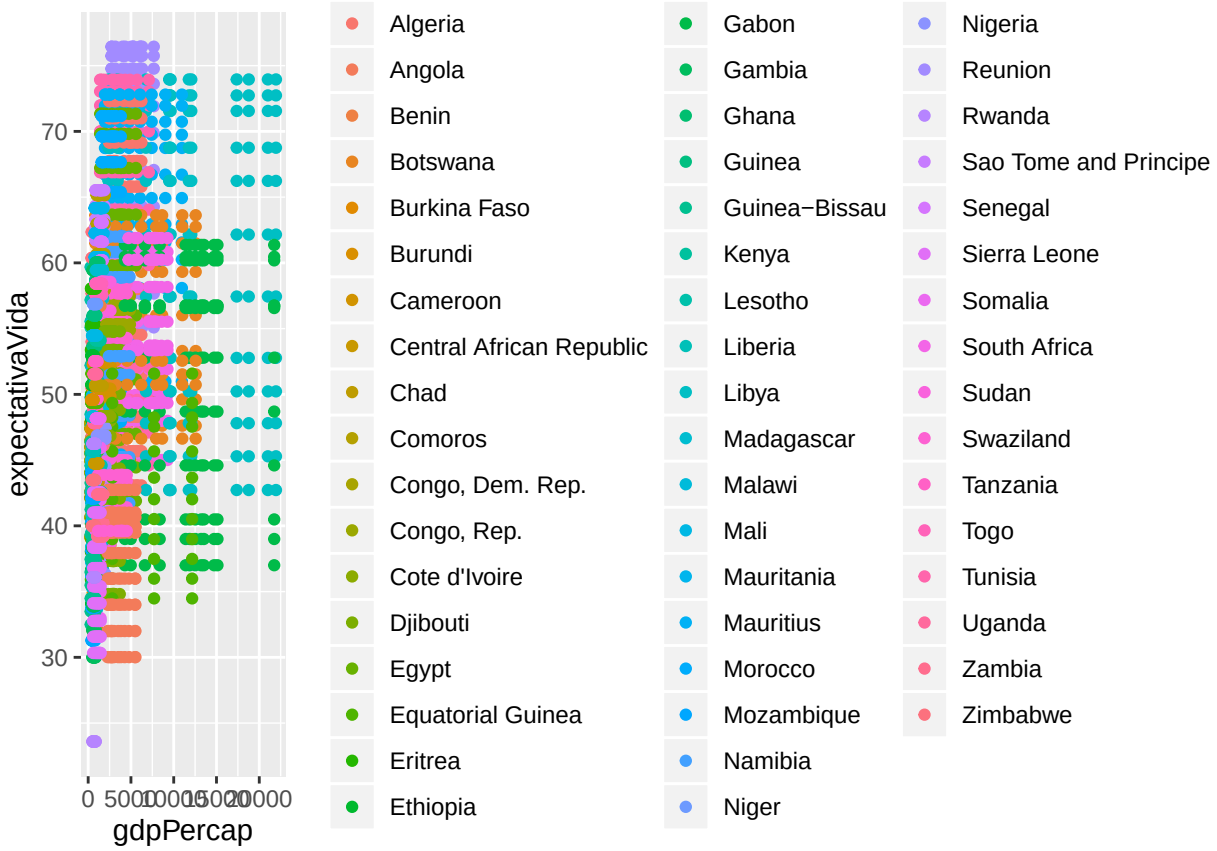



```
# b)
gapminder_final %>%
  ggplot(data = .,
         mapping = aes(x = gdpPercap, y = expectativaVida, color = Continente)) +
  geom_point()
```




```
# c)
gapminder_final %>%
  filter(., Continente == 'Africa' | Continente == 'Europa') %>%
  ggplot(data = .,
         mapping = aes(x = gdpPercap, y = expectativaVida, color = Continente)) +
  geom_point()
```

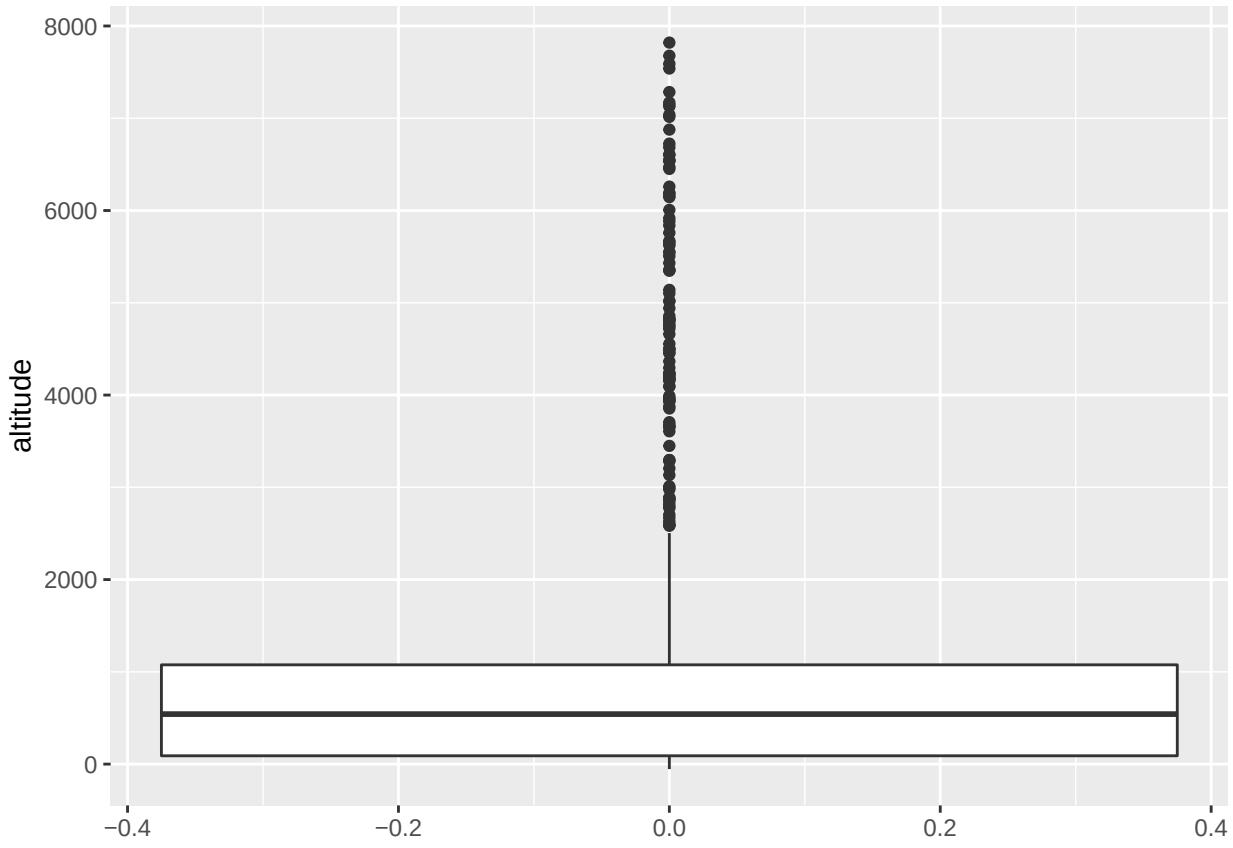

```
# d)
gapminder_final %>%
  filter(., Continente == 'Africa') %>%
  ggplot(data = .,
         mapping = aes(x = gdpPercap, y = expectativaVida, color = Pais)) +
  geom_point()
```



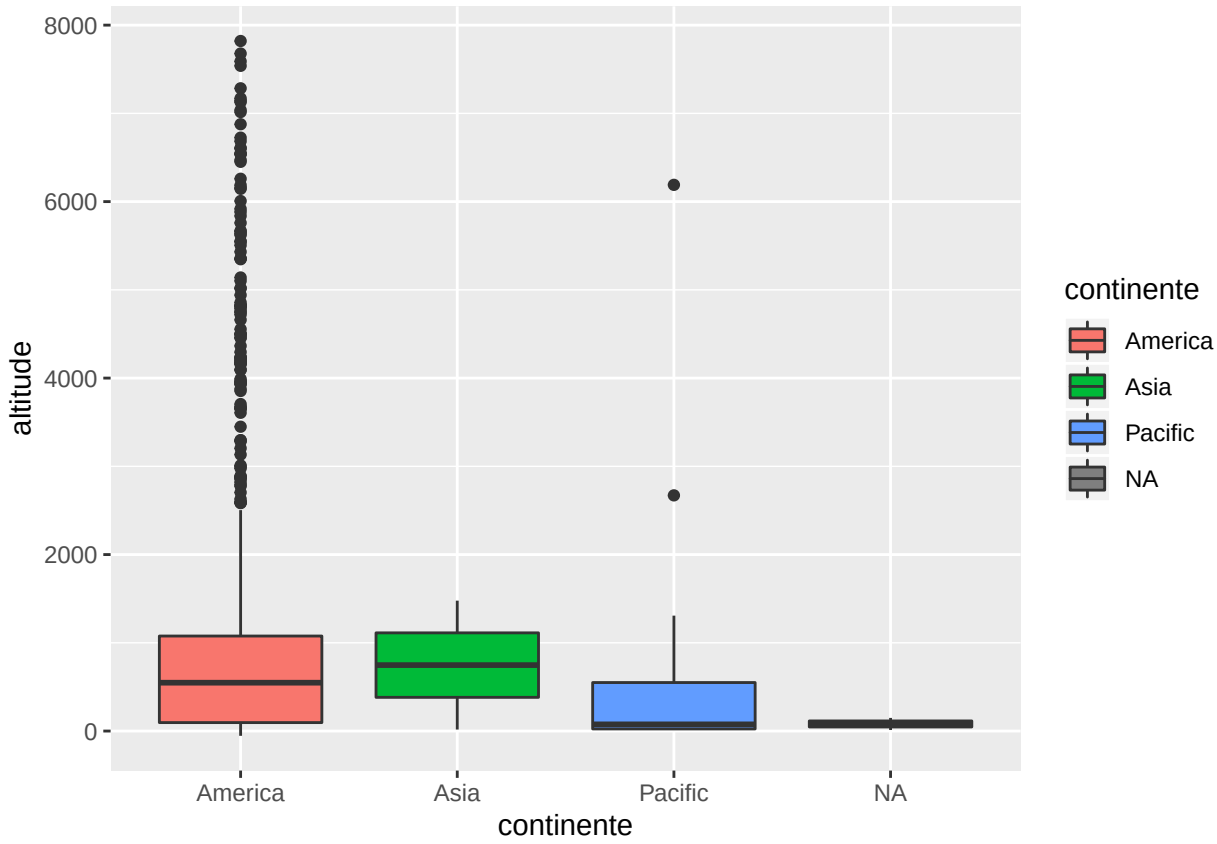
a b c

modificamos un poco mas (dplyr)

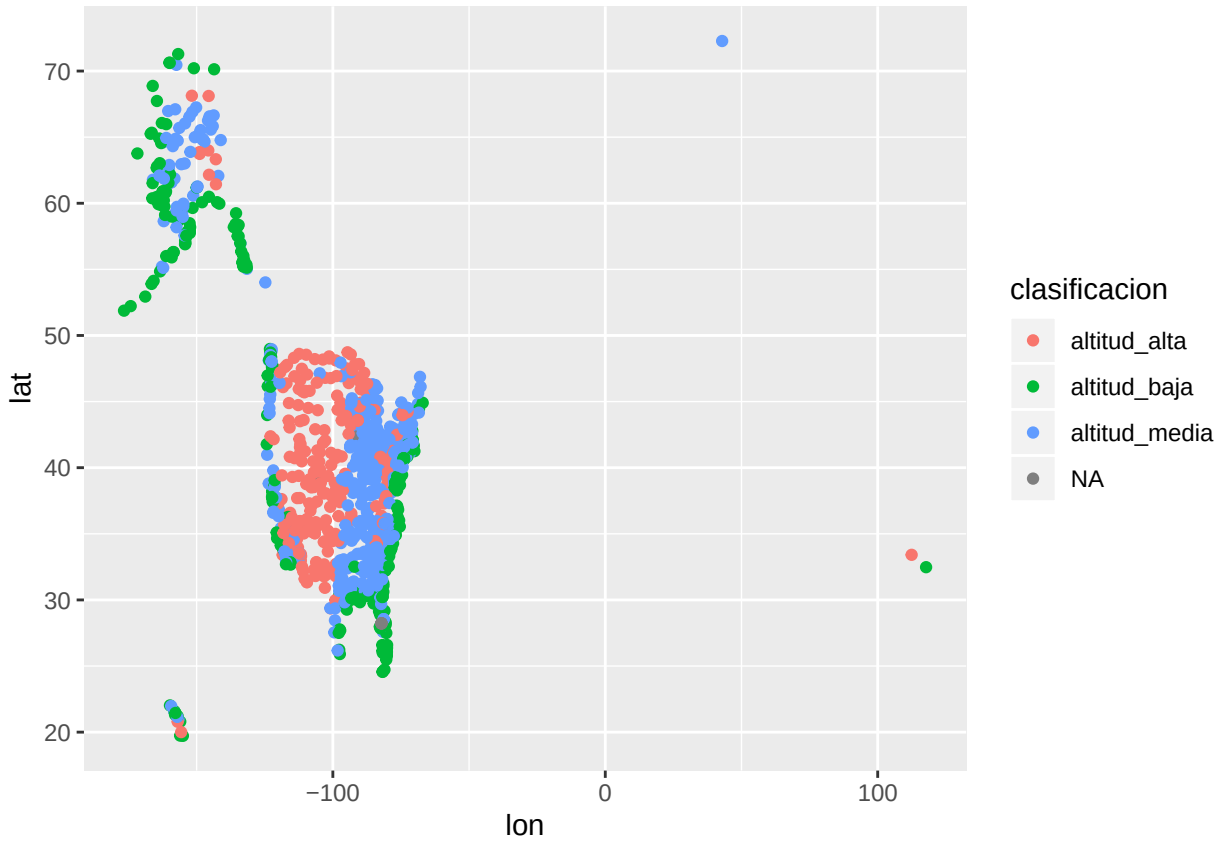
```
# ahora clasificaremos los vuelos segun tres tipos de altitudes: altas, medianas y bajas
# primero vamos a ver como se distribuyen estas altitudes
vuelos_final %>%
  ggplot(data = .,
         mapping = aes(y = altitude)) +
  geom_boxplot()
```



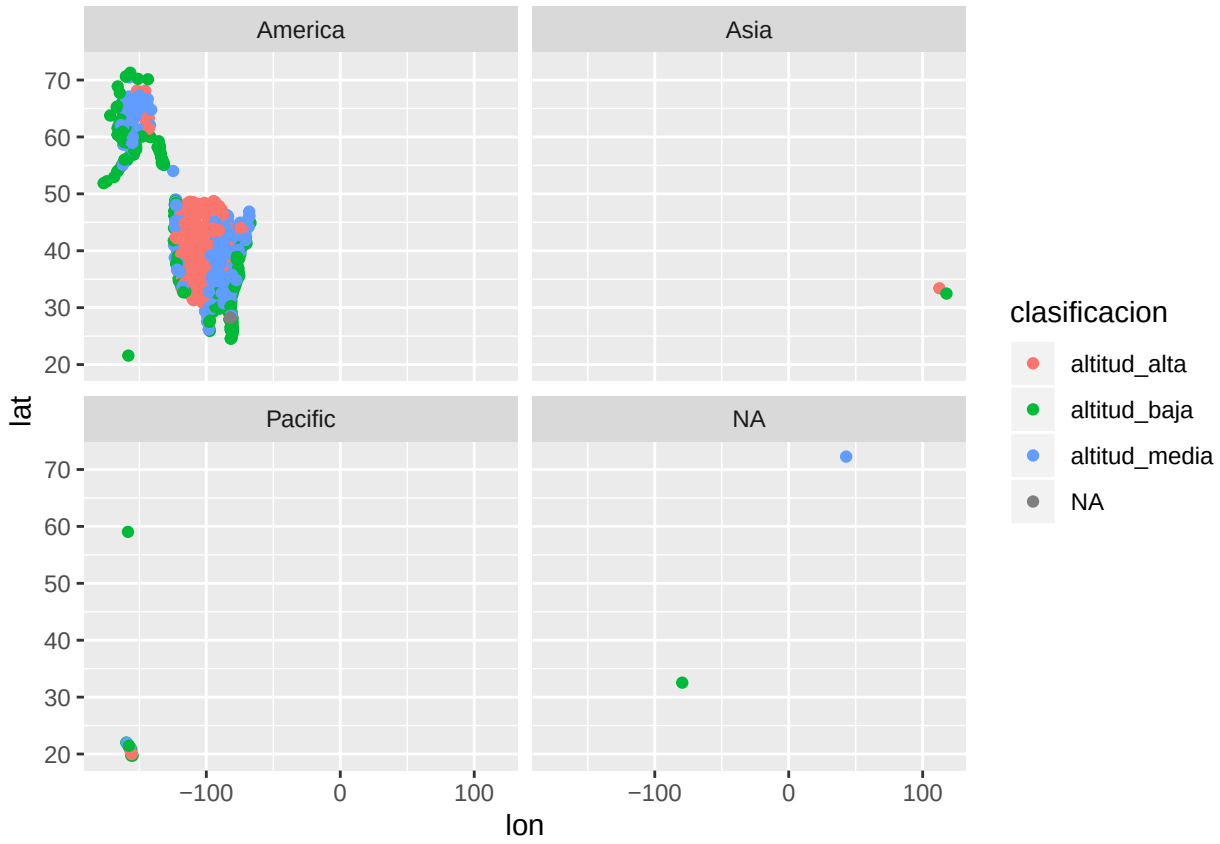

```
vuelos_final %>%
  ggplot(data = .,
         mapping = aes(y = altitude, x = continente, fill = continente)) +
  geom_boxplot()
```



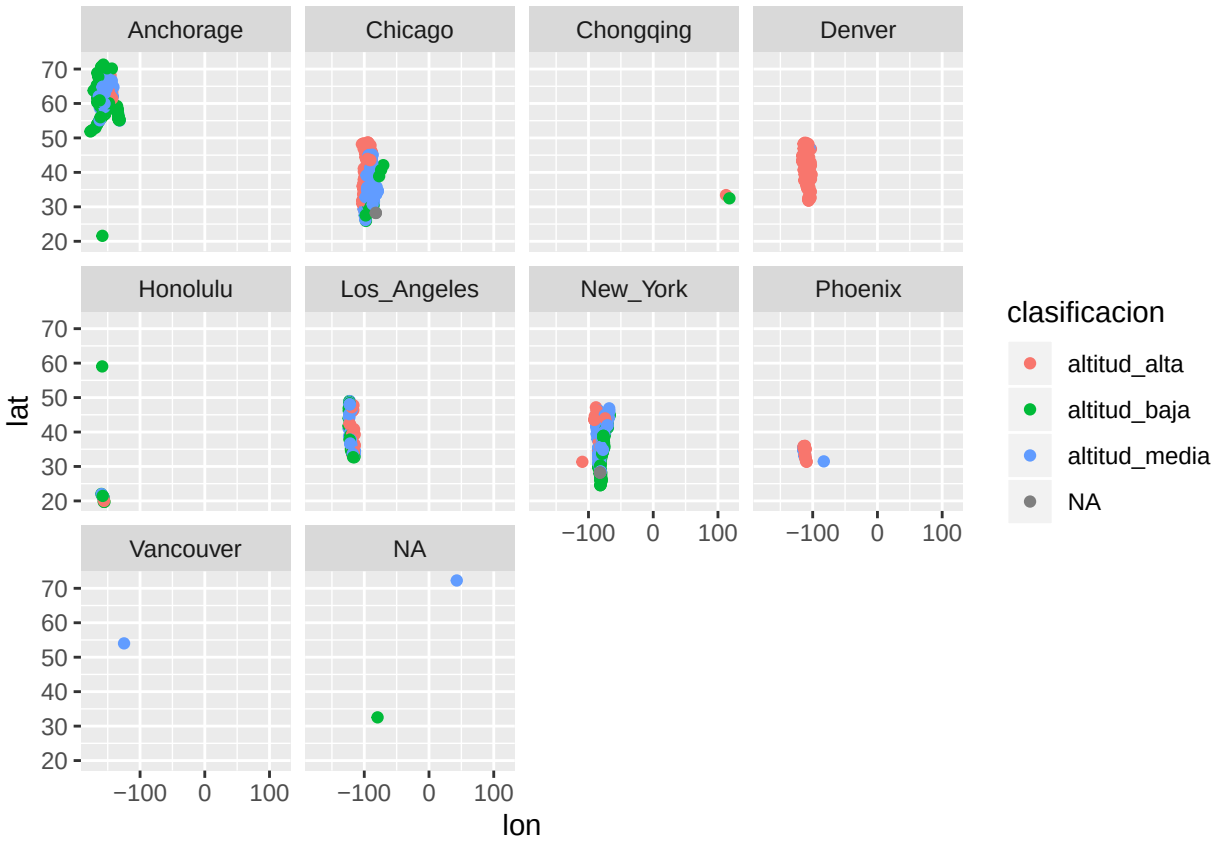

```
# pasamos ahora a hacer la clasificacion
## lo haremos en base a lo reportado por la funcion quantile()
quantiles_alt = quantile(vuelos_final$altitude)

vuelos_final %<>%
  dplyr::mutate(., clasificacion = case_when(altitude < quantiles_alt[2] ~ 'altitud_baja',
                                             altitude > quantiles_alt[2] &
                                             quantiles_alt[4] > altitude ~ 'altitud_media',
                                             altitude > quantiles_alt[4] &
                                             quantiles_alt[5] > altitude ~ 'altitud_alta')
                )

# ahora podemos visualizar esto de diferentes maneras
vuelos_final %>%
  ggplot(data = .,
         mapping = aes(x = lon, y = lat, color = clasificacion)) +
  geom_point()
```




```
vuelos_final %>%
  ggplot(data = .,
         mapping = aes(x = lon, y = lat, color = clasificacion)) +
  geom_point() +
  facet_wrap(~continente)
```

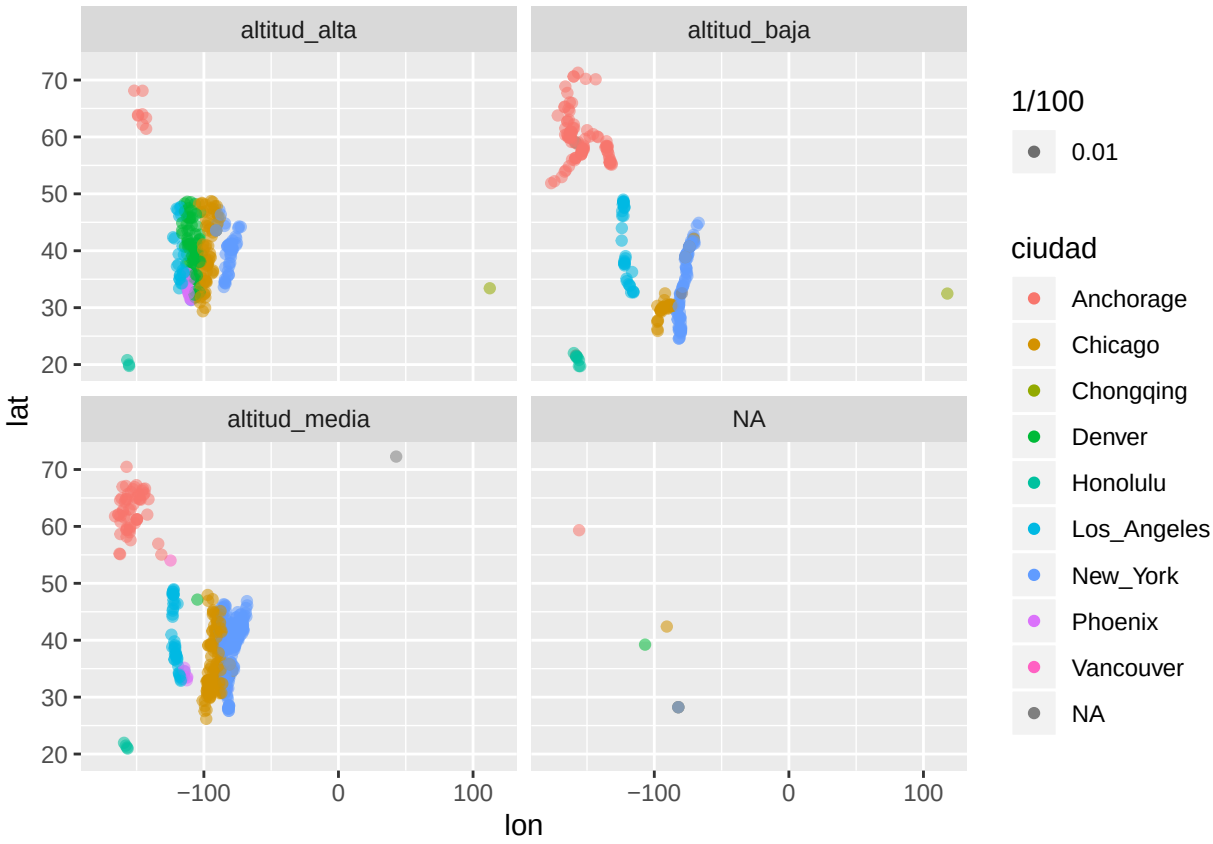



```
vuelos_final %>%
  ggplot(data = .,
         mapping = aes(x = lon, y = lat, color = clasificacion)) +
  geom_point() +
  facet_wrap(~ciudad)
```




```
# para ver mejor incluso podemos dar poca intensidad al coloreado
vuelos_final %>%
  ggplot(data = .,
         mapping = aes(x = lon, y = lat, color = clasificacion, alpha = 1/100)) +
  geom_point() +
  facet_wrap(~ciudad)
```


```
vuelos_final %>%
  ggplot(data = .,
         mapping = aes(x = lon, y = lat, color = ciudad, alpha = 1/100)) +
  geom_point() +
  facet_wrap(~clasificacion)
```




```
library(tibble)
as_tibble(iris)
```

Cargando datos: readr

```
library(readr)

aeropuertos = readr::read_tsv('airports.tsv')

aeropuertos
```

Ejercico 1

- Cargue en

```
# cargamos librerias
library(readr)
library(magrittr)
library(dplyr)

# cargamos los datasets
dataset_pollo = readr::read_tsv('KallistoNormalizedTPM_Chicken.txt')
dataset_raton = readr::read_tsv('KallistoNormalizedTPM_Mouse.txt')
dataset_rata = readr::read_tsv('KallistoNormalizedTPM_Rat.txt')
```

```r
# vemos que columnas son compartidas entre los datasets, a fin de crear un unico set de datos con el qu
columnas_compartidas = dplyr::intersect(x = colnames(dataset_pollo), y = colnames(dataset_raton)) %>%
  dplyr::intersect(x = ., y = colnames(dataset_rata))

# unimos los datasets
### esto hay que hacerlo con un loop capaz
dataset_pollo %<>% dplyr::select(columnas_compartidas)
dataset_raton %<>% dplyr::select(columnas_compartidas)
dataset_rata %<>% dplyr::select(columnas_compartidas)

# agregamos un tag para identificar de donde viene cada dataset
dataset_pollo %<>% dplyr::mutate(., set = 'pollo')
dataset_raton %<>% dplyr::mutate(., set = 'raton')
dataset_rata %<>% dplyr::mutate(., set = 'rata')

dataset_final = dplyr::bind_rows(dataset_pollo, dataset_raton, dataset_rata)
```

Filtrado de datos

```r
library(dplyr)
library(stringr)

aeropuertos %>%
  filter(lat > -100)

aeropuertos %>%
  filter(lat > -100 & lon > 100)

aeropuertos %>%
  filter(lat > -100 & lon > 100 & stringr::str_detect(string = tzone, 'Asia'))
```

Modificación y agregado de datos

```r
aeropuertos %>%
  mutate(., continent = stringr::str_split(tzone, '/') %>% purrr::map_chr(1))
```

Visualización de datos: ggplot2

```r
library(ggplot2)

aeropuertos %>%
  mutate(., continent = stringr::str_split(tzone, '/') %>% purrr::map_chr(1)) %>%
  ggplot(data = .,
         mapping = aes(x = lon, y = lat)) +
  geom_point()
```

```r
library(ggplot2)
library(reshape2)
library(purrr)

# visualizando todos los datos juntos
dataset_final %>%
  group_split(set) %>%
  purrr::map_dfr(., ~{
    .[1:3000,]
    }
  ) %>%
  melt() %>%
  as_tibble() %>%
  dplyr::rename(tejido = 'variable', TPM = 'value') %>%
  ggplot(data = .,
         mapping = aes(x = set, y = TPM, fill = tejido, color = tejido)
         ) +
  geom_point()
  #facet_wrap(~tejido)

# por set
  dataset_final %>%
  group_split(set) %>%
  purrr::map_dfr(., ~{
    .[1:3000,]
    }
  ) %>%
  melt() %>%
  as_tibble() %>%
  dplyr::rename(tejido = 'variable', TPM = 'value') %>%
  ggplot(data = .,
         mapping = aes(x = set, y = TPM, fill = tejido, color = tejido)
         ) +
  geom_point() +
  facet_wrap(~tejido)

  # realizando algunas modificaciones podemos dividir entre los tejidos en si
  library(stringr)

  dataset_final %>%
  group_split(set) %>%
  purrr::map_dfr(., ~{
    .[1:5000,]
    }
  ) %>%
  melt() %>%
  as_tibble() %>%
  dplyr::rename(tejido = 'variable', TPM = 'value') %>%
  mutate(set_tejido = str_split(tejido, '_') %>% purrr::map_chr(1)) %>%
  ggplot(data = .,
         mapping = aes(x = tejido, y = TPM, fill = set, color = set, group = set_tejido, alpha = 0.3)
         ) +
  geom_point() +
```

```r
theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
facet_wrap(~set_tejido, scales = 'free_x')

# nos focalizamos en algunos genes
dataset_final %>%
group_split(set) %>%
purrr::map_dfr(., ~{
  .[1:5000,]
  }
) %>% filter(Liver_LateEmbryo2 > 40000) %>% .$GeneID -> genes_interes

dataset_final %>%
group_split(set) %>%
purrr::map_dfr(., ~{
  .[1:5000,]
  }
) %>%
#dplyr::select(GeneID, set, Liver_EarlyEmbryo1, Liver_EarlyEmbryo2, Liver_LateEmbryo1, Liver_LateEmbr
melt() %>%
as_tibble() %>%
dplyr::rename(tejido = 'variable', TPM = 'value') %>%
#filter(GeneID %in% genes_interes) %>%
filter(TPM > 20000) %>%
mutate(set_tejido = str_split(tejido, '_') %>% purrr::map_chr(1)) %>%
ggplot(data = .,
       mapping = aes(x = tejido, y = TPM, fill = GeneID, color = GeneID, group = GeneID)
       ) +
geom_line() +
geom_point() +
theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
facet_wrap(~set_tejido+set, scale = 'free_x')
```

De una busqueda en Ensembl se ve que el gen **ENSRNOG00000002889** corresponde al gen *Afp*, un gen descrito como *'Predicted to have fatty acid binding activity and zinc ion binding activity. Involved in animal organ development and response to organic substance. (...)'*.

ENSRNOG00000002911, a su vez, corresponde al gen *Alb*, la albumina.

- Incluir el tema de hacer facets y eso

Algunos otros ejemplos

Visualización de filogenias: ggtree

Visualización de datos genómicos: BioCircos

Algo tipo lo de ver cosas genomicas