- **M1 MIDS & MFA**
- **Université Paris Cité**
- Année 2023-2024
- Course Homepage
- Moodle

We will use the following packages. If needed, we install them.

```
to_be_loaded <- c("tidyverse",
                  "patchwork",
                  "glue",
                  "ggforce",
                  "plotly",
                  "ggthemes",
                  "gapminder",
                  "ggrepel")

for (pck in to_be_loaded) {
  if (!require(pck, character.only = T)) {
    install.packages(pck, repos="http://cran.rstudio.com/")
    stopifnot(require(pck, character.only = T))
  }
}
```

Grammar of Graphics

We will use the *Grammar of Graphics* approach to visualization

The expression *Grammar of Graphics* was coined by Leiland Wilkinson to describe a principled approach to visualization in Data Analysis (EDA)

A plot is organized around data (a table with rows (observations) and columns (variables))

A *plot* is a *graphical object* that can be built *layer* by *layer*

Building a graphical object consists in *chaining* elementary operations

The acclaimed TED presentation by Hans Rosling illustrates the Grammar of Graphics approach

Visit https://www.youtube.com/embed/jbkSRLYSojo

Do It Yourself with `R`

We will reproduce the animated demonstration using

- `ggplot2`: an implementation of *grammar of graphics* in 'R
- `plotly`: a bridge between `R` and the javascript library `D3.js`
- Using `plotly`, opting for `html` ouput, brings the possibility of interactivity and animation

Install and load packages

```
require("gapminder")
```

Insist on the difference between *installing* and *loading* a package

- How do we get the list of installed packages?
- How do we get the list of loaded packages?
- Which objects are made available by a package?

Have a look at `gapminder` dataset

- A table has a *schema*: a list of named *columns*, each with a given type
- A table has a *content*: *rows*. Each row is a collection of items, corresponding to the columns
- `glimpse()` allows to see the schema and the first rows
- `head()` allows to see the first rows

solution

Dataframes

```
gapminder <- gapminder::gapminder

glimpse(gapminder)
```

```
Rows: 1,704
Columns: 6
$ country   <fct> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan", ~
$ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, ~
$ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, ~
$ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40.8~
$ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372, 12~
$ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134, ~
```

```
gapminder %>%
  glimpse()
```

```
Rows: 1,704
Columns: 6
$ country   <fct> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan", ~
$ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, ~
$ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, ~
$ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40.8~
$ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372, 12~
$ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134, ~
```

```
gapminder %>%
  head()
```

```
# A tibble: 6 x 6
  country     continent  year lifeExp      pop gdpPercap
  <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
1 Afghanistan Asia       1952    28.8  8425333      779.
2 Afghanistan Asia       1957    30.3  9240934      821.
3 Afghanistan Asia       1962    32.0 10267083      853.
4 Afghanistan Asia       1967    34.0 11537966      836.
5 Afghanistan Asia       1972    36.1 13079460      740.
6 Afghanistan Asia       1977    38.4 14880372      786.
```

Even an empty dataframe has a scheme:

```
gapminder %>%
  head(0) %>%
```

solution

The schema of a dataframe/tibble is the list of column names and classes. The content of a dataframe is made of the rows. A dataframe may have null content

```
gapminder %>%
  filter(FALSE) %>%
  glimpse()
```

```
Rows: 0
Columns: 6
$ country   <fct>
$ continent <fct>
$ year      <int>
$ lifeExp   <dbl>
$ pop       <int>
$ gdpPercap <dbl>
```

Get a feeling of the dataset

Pick two random rows for each continent using `slice_sample()`

solution

To pick a slice at random, we can use function `slice_sample`. We can even perform sampling within groups defined by the value of a column.

```
gapminder %>%
  slice_sample(n=2, by=continent)
```

```
# A tibble: 10 x 6
   country                continent  year lifeExp      pop gdpPercap
   <fct>                  <fct>     <int>   <dbl>    <int>     <dbl>
 1 Pakistan               Asia       1957    45.6 46679944      747.
 2 Singapore              Asia       1992    75.8  3235865    24770.
 3 Bosnia and Herzegovina Europe     1977    69.9  4086000     3528.
 4 Turkey                 Europe     1952    43.6 22235677     1969.
 5 Namibia                Africa     1997    58.9  1774766     3900.
 6 Guinea-Bissau          Africa     1992    43.3  1050938      746.
 7 Costa Rica             Americas   2007    78.8  4133884     9645.
 8 Uruguay                Americas   2002    75.3  3363085     7727.
 9 New Zealand            Oceania    1977    72.2  3164900    16234.
10 Australia              Oceania    1957    70.3  9712569    10950.
```

```
#< or equivalently
gapminder %>%
  group_by(continent) %>%
  slice_sample(n=2)
```

```
# A tibble: 10 x 6
# Groups:   continent [5]
   country     continent  year lifeExp      pop gdpPercap
   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
 1 Swaziland   Africa     1972    49.6   480105     3365.
 2 Lesotho     Africa     1967    48.5   996380      499.
 3 Ecuador     Americas   1977    61.3  7278866     6680.
 4 Mexico      Americas   1962    58.3 41121485     4582.
 5 Lebanon     Asia       1987    67.9  3089353     5377.
 6 Yemen, Rep. Asia       1992    55.6 13367997     1879.
 7 Spain       Europe     1972    73.1 34513161    10639.
 8 Croatia     Europe     1977    70.6  4318673    11305.
 9 Australia   Oceania    2002    80.4 19546792    30688.
10 Australia   Oceania    1967    71.1 11872264    14526.
```

What makes a table *tidy*?

Have a look at Data tidying in R for Data Science (2nd ed.)

Is the `gapminder` table redundant?

solution

`gapminder` is redundant: column `country` completely determines the content of column `continent`. In database parlance, we have a functional dependancy: `country` → `continent` whereas the *key* of the table is made of columns `country, year`. Column `gapminder` is not in Boyce-Codd Normal Form (BCNF), not even in Third Normal Form (3NF).

Gapminder tibble (extract)

Extract/filter a subset of rows using `dplyr::filter(...)`

solution

```
gapminder %>%
  filter(country=='France') %>%
  head()
```

```
# A tibble: 6 x 6
  country continent  year lifeExp      pop gdpPercap
  <fct>   <fct>     <int>   <dbl>    <int>     <dbl>
1 France  Europe     1952    67.4 42459667     7030.
2 France  Europe     1957    68.9 44310863     8663.
3 France  Europe     1962    70.5 47124000    10560.
4 France  Europe     1967    71.6 49569000    13000.
5 France  Europe     1972    72.4 51732000    16107.
6 France  Europe     1977    73.8 53165019    18293.
```

Note that equality testing is performed using `==` not `=` (which is used to implement assignment)

Filtering (selection σ from database theory) : Picking one year of data

There is simple way to filter rows satisfying some condition. It consists in mimicking indexation in a matrix, leaving the colum index empty, replacing the row index by a condition statement (a logical expression) also called a mask.

```
gapminder_2002 <- gapminder[gapminder$year==2002, ]
```

Have a look at `gapminder$year==2002`. What is the type/class of this expression?

This is possible in base `R` and very often convenient.

Nevertheless, this way of performing row filtering does not emphasize the connection between the dataframe and the condition. Any logical vector with the right length could be used as a mask. Moreover, this way of performing filtering is not very functional.

In the parlance of Relational Algebra, `filter` performs a *selection* of rows. Relational expression

$$\sigma_{\text{condition}}(\text{Table})$$

translates to

```
filter(Table, condition)
```

where condition is a boolean expression that can be evaluated on each row of Table. In `SQL`, the relational expression would translate into

```
SELECT *
FROM Table
WHERE condition
```

Check Package `dplyr` docs
The `posit` cheatsheet on `dplyr` is an unvaluable resource for table manipulation.

Use `dplyr::filter()` to perform row filtering

solution

```
# filter(gapminder, year==2002)

gapminder %>%
  filter(year==2002)
```

```
# A tibble: 142 x 6
   country     continent  year lifeExp       pop gdpPercap
   <fct>       <fct>     <int>   <dbl>     <int>     <dbl>
 1 Afghanistan Asia       2002    42.1  25268405      727.
 2 Albania     Europe     2002    75.7   3508512     4604.
 3 Algeria     Africa     2002    71.0  31287142     5288.
 4 Angola      Africa     2002    41.0  10866106     2773.
 5 Argentina   Americas   2002    74.3  38331121     8798.
 6 Australia   Oceania    2002    80.4  19546792    30688.
 7 Austria     Europe     2002    79.0   8148312    32418.
 8 Bahrain     Asia       2002    74.8    656397    23404.
 9 Bangladesh  Asia       2002    62.0 135656790     1136.
10 Belgium     Europe     2002    78.3  10311970    30486.
# i 132 more rows
```

Note that in stating the condition, we simply write `year==2002` even though `year` is not the name of an object in our current session. This is possible because `filter( )` uses *data masking*, `year` is meant to denote a column in `gapminder`.
The ability to use data masking is one of the great strengths of the `R` programming language.

Static plotting: First attempt

- Define a plot with respect to `gapminder_2002`

solution

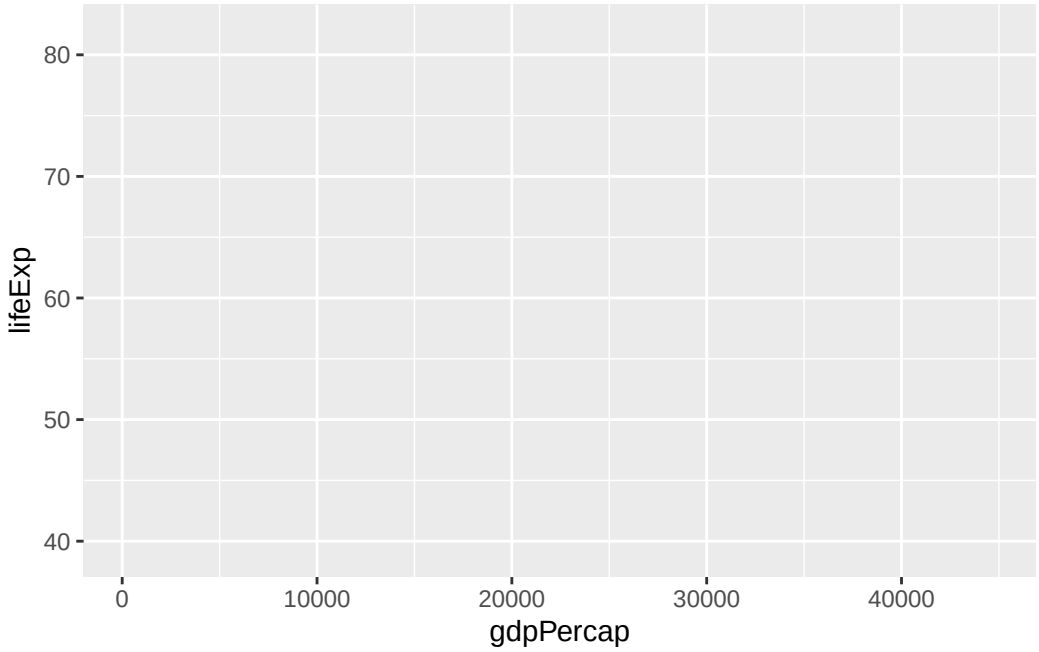
```
p <- gapminder_2002 %>%
  ggplot()
```

You should define a `ggplot` object with data layer `gapminder_2022` and call this object `p` for further reuse.

- Map variables `gdpPercap` and `lifeExp` to axes `x` and `y`

solution

```
p <- p +
  aes(x=gdpPercap, y=lifeExp)
p
```

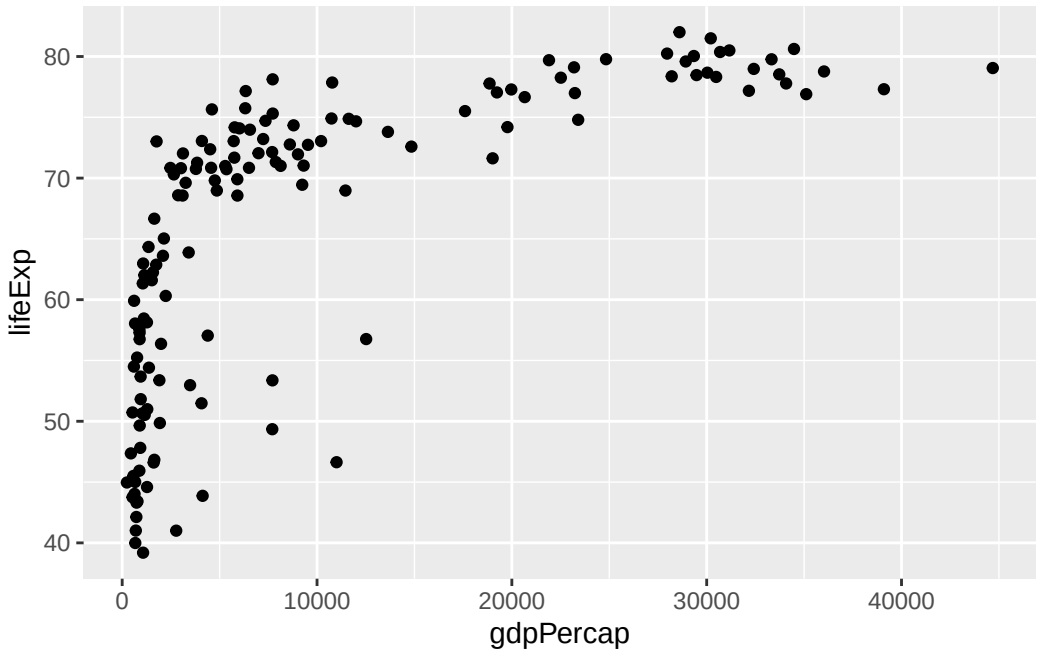


Use `ggplot` object `p` and add a global aesthetic mapping `gdpPercap` and `lifeExp` to axes `x` and `y` (using `+` from `ggplot2`) .

- For each row, draw a point at coordinates defined by the mapping

solution

```
p +
  geom_point()
```



You need to add a `geom_` layer to your `ggplot` object, in this case `geom_point()` will do.

We are building a graphical object (a `ggplot` object) around a data frame (`gapminder`)

We supply *aesthetic mappings* (`aes()`) that can be either global or bound to some *geometries* (`geom_point()`)or *statistics*

The global aesthetic mapping defines which columns are

- mapped to which axes,
- possibly mapped to colours, linetypes, shapes, ...

Geometries and Statistics describe the building blocks of graphics

What's missing here?

when comparing to the Gapminder demonstration, we can spot that

- colors are missing
- bubble sizes are all the same. They should reflect the population size of the country
- titles and legends are missing. This means the graphic object is useless.

We will add layers to the graphical object to complete the plot

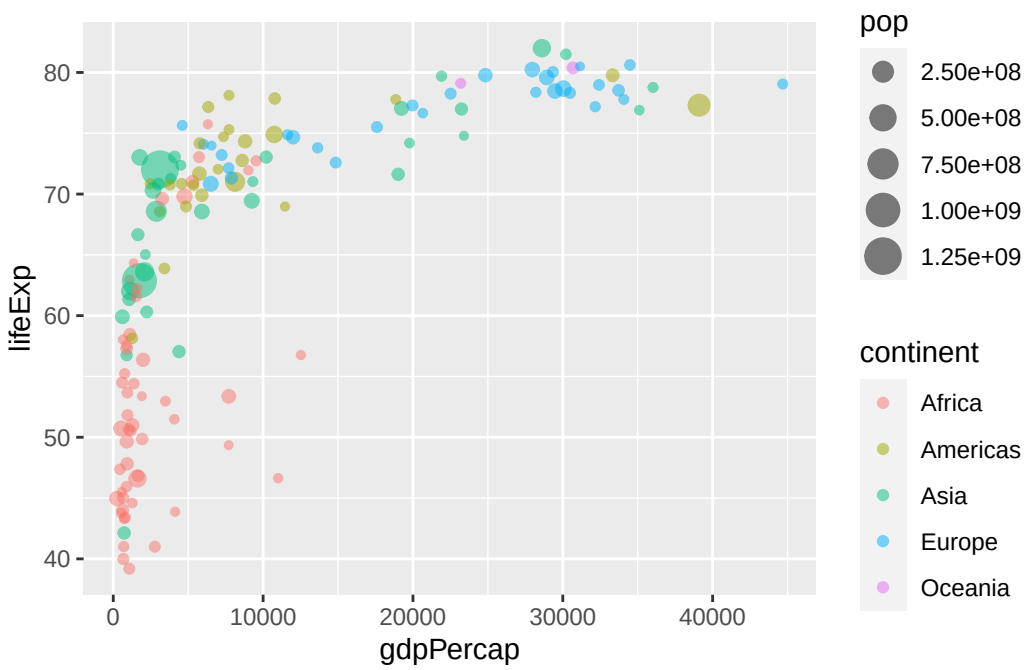
Second attempt: display more information

- Map `continent` to color (use `aes()`)
- Map `pop` to bubble size (use `aes()`)
- Make point transparent by tuning `alpha` (inside `geom_point()` avoid *overplotting*)

solution

```
p <- p +
  aes(color=continent, size=pop) +
  geom_point(alpha=.5)

p
```



solution

In this enrichment of the graphical object, *guides* have been automatically added for two aesthetics: `color` and `size`. Those two guides are deemed necessary since the reader has no way to guess the mapping from the five levels of `continent` to color (the color scale), and the reader needs help to connect population size and bubble size.
`ggplot2` provides us with helpers to fine tune guides.
The scalings on the `x` and `y` axis do not deserve guides: the ticks along the coordinate axes provide enough information.

Scaling

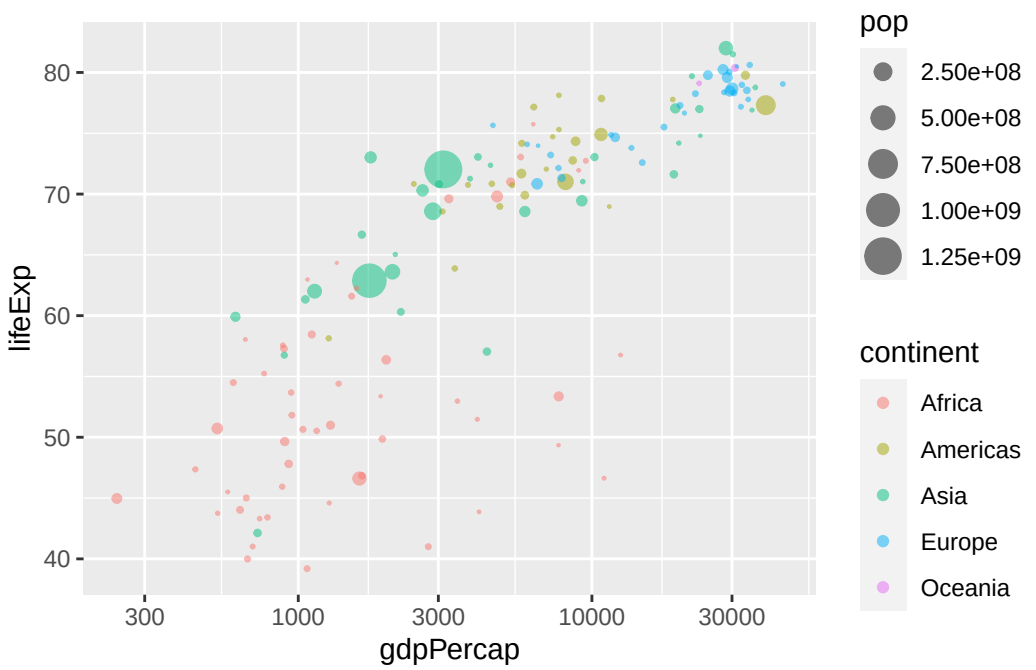
In order to pay tribute to Hans Rosling, we need to take care of two *scaling* issues:

- the gdp per capita axis should be *logarithmic* `scale_x_log10()`
- the *area* of the point should be proportional to the population `scale_size_area()`

solution

```
p <- p +
  scale_x_log10() +
  scale_size_area()

p
```



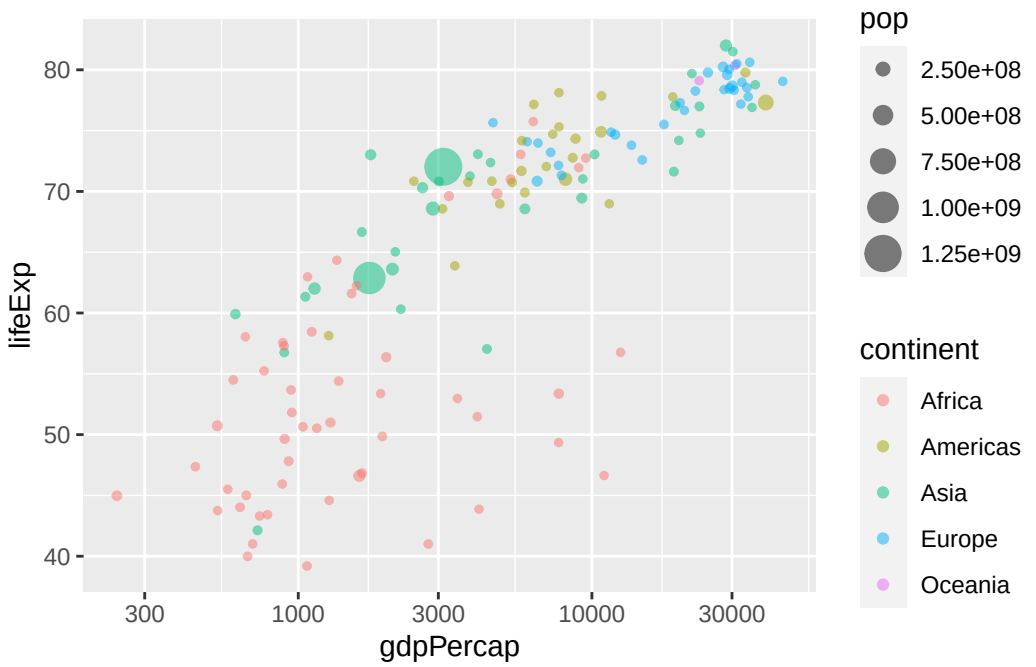
Motivate the proposed scalings.

- Why is it important to use logarithmic scaling for gdp per capita?
- When is it important to use logarithmic scaling on some axis (in other contexts)?
- Why is it important to specify `scale_size_area()` ?

solution

```
p +
  scale_radius()
```

```
Scale for size is already present.
Adding another scale for size, which will replace the existing scale.
```

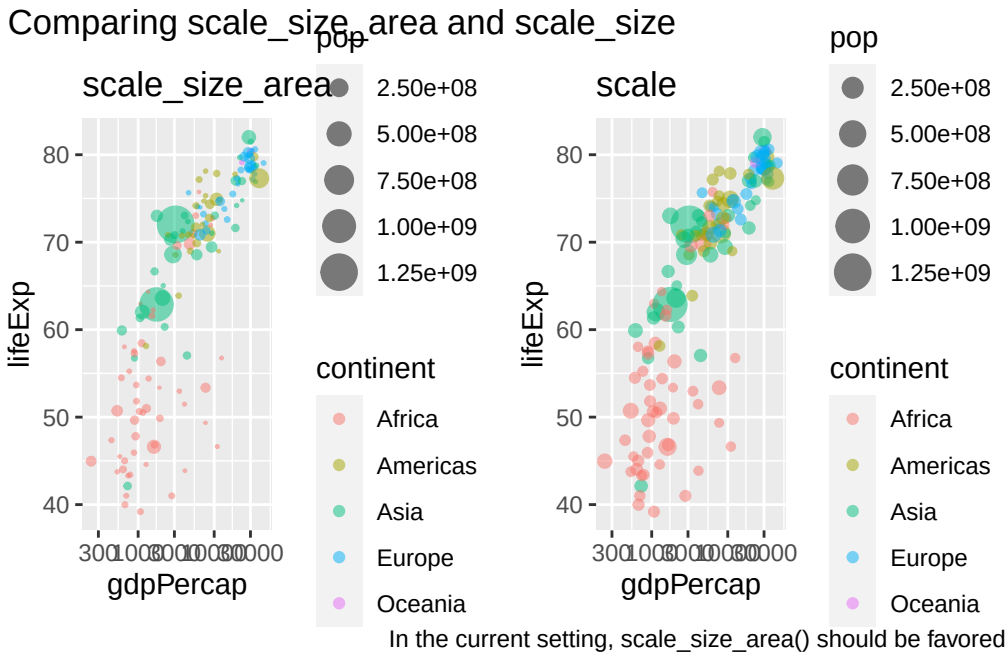


solution

```
ptchwrk <- (p + ggtitle("scale_size_area")) + (p + scale_size() + ggtitle("scale"))
```

```
Scale for size is already present.
Adding another scale for size, which will replace the existing scale.
```

```
ptchwrk + plot_annotation(
  title='Comparing scale_size_area and scale_size',
  caption='In the current setting, scale_size_area() should be favored'
)
```



In perspective

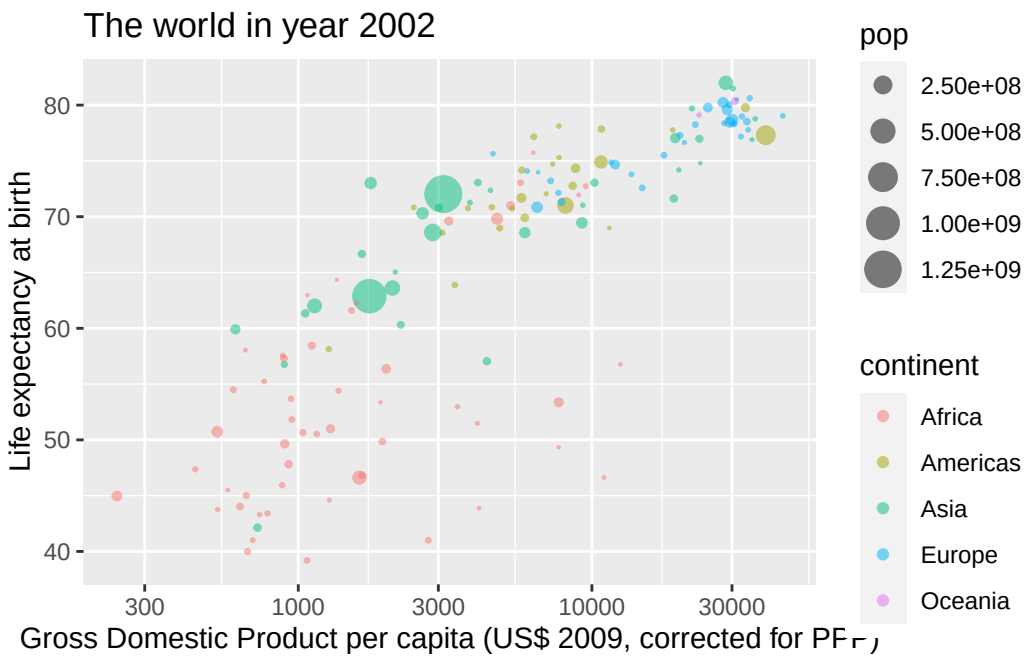
- Add a plot title
- Make axes titles
  - explicit
  - readable
- Use `labs(...)`

solution

```
yoi <- 2002

p <-  p +
  labs(
    title=glue('The world in year {yoi}'),
    x="Gross Domestic Product per capita (US$ 2009, corrected for PPP)",
    y="Life expectancy at birth"
  )

p
```



solution

We should also fine tune the guides: replace `pop` by `Population` and titlecase `continent`.

What should be the respective purposes of Title, Subtitle, Caption, … ?

Theming using `ggthemes` (or not)

- Theming

```
require("ggthemes")
```

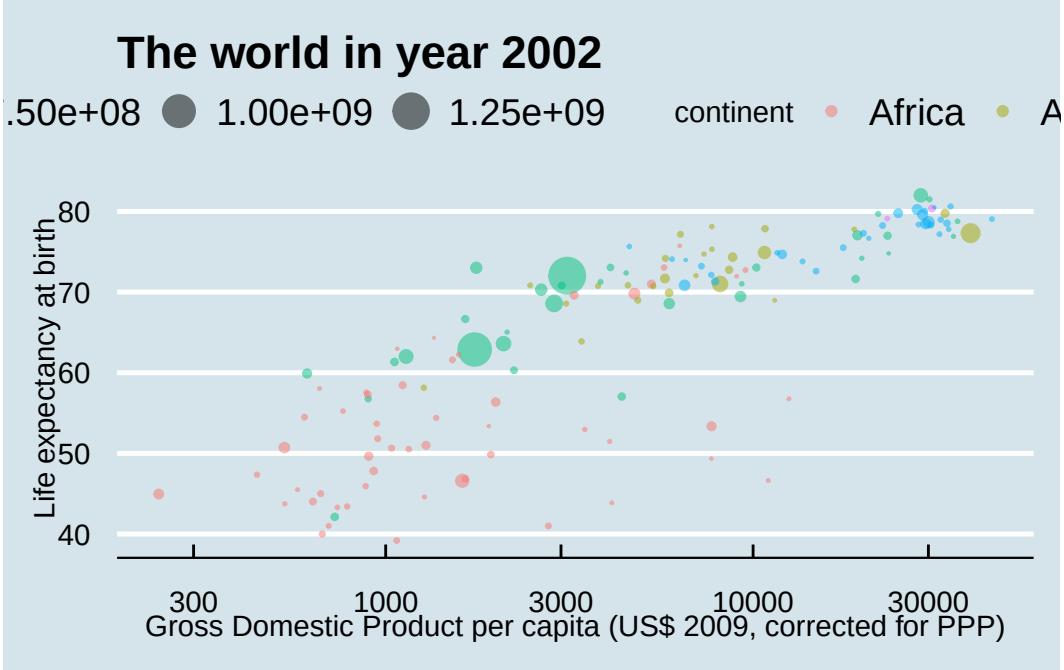
Look at the online help on `pacman::p_load()`, how does `pacman::p_load()` relate to `require()` and `library()`?

A theme defines the *look and feel* of plots

Within a single document, we should use only one theme

See Getting the theme for a gallery of available themes

```
p +
  theme_economist()
```



Tuning scales

Use `scale_color_manual(...)` to hand-tune the color aesthetic mapping.

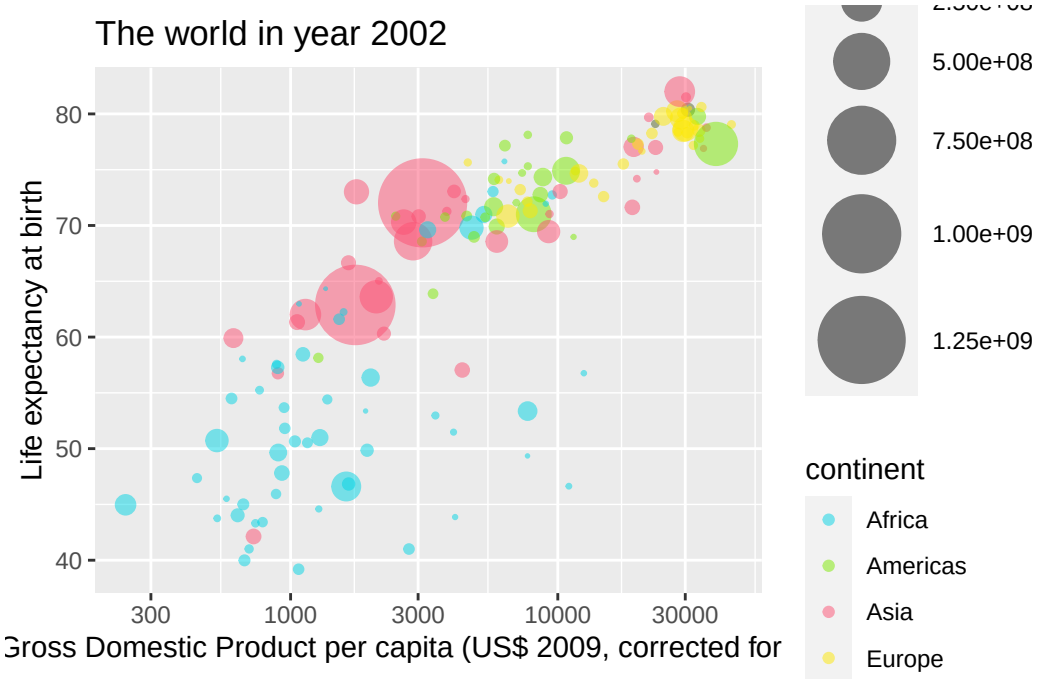
```
```{r}
#| label: theme_scale
neat_color_scale <-
      c("Africa" = "#01d4e5",
        "Americas" = "#7dea01" ,
        "Asia" = "#fc5173",
        "Europe" = "#fde803",
        "Oceania" = "#536227")
```
```

solution

```
p <- p +
  scale_size_area(max_size = 15) + #<<
  scale_color_manual(values = neat_color_scale) #<<
```

```
Scale for size is already present.
Adding another scale for size, which will replace the existing scale.
```

```
p
```



Choosing a color scale is a difficult task

`viridis` is often a good pick.

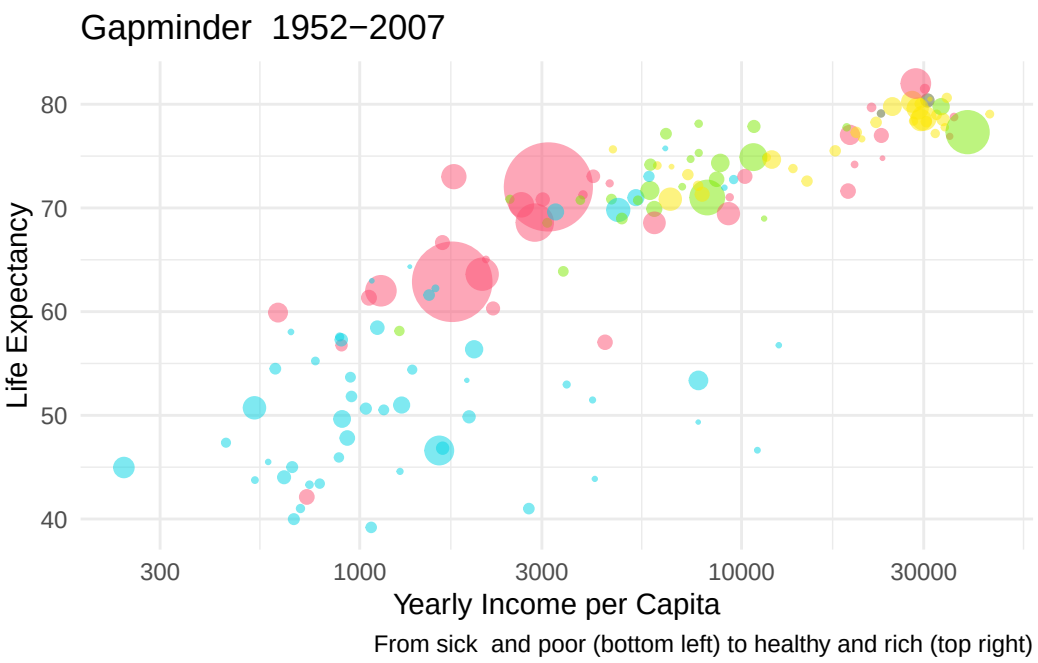
solution

Mimnimalist themes are often a good pick.

```
p <- p +
   scale_size_area(max_size = 15,
                   labels= scales::label_number(scale=1/1e6,
                                                suffix=" M")) +
   scale_color_manual(values = neat_color_scale) +
   theme_minimal() +
    labs(title= glue("Gapminder  {min(gapminder$year)}-{max(gapminder$year)}"),
         x = "Yearly Income per Capita",
         y = "Life Expectancy",
       caption="From sick  and poor (bottom left) to healthy and rich (top right)")
```

```
Scale for size is already present.
Adding another scale for size, which will replace the existing scale.
Scale for colour is already present.
Adding another scale for colour, which will replace the existing scale.
```

```
p + theme(legend.position = "none")
```



Zooming on a continent

```
zoom_continent <- 'Europe'  # choose another continent at your convenience
```

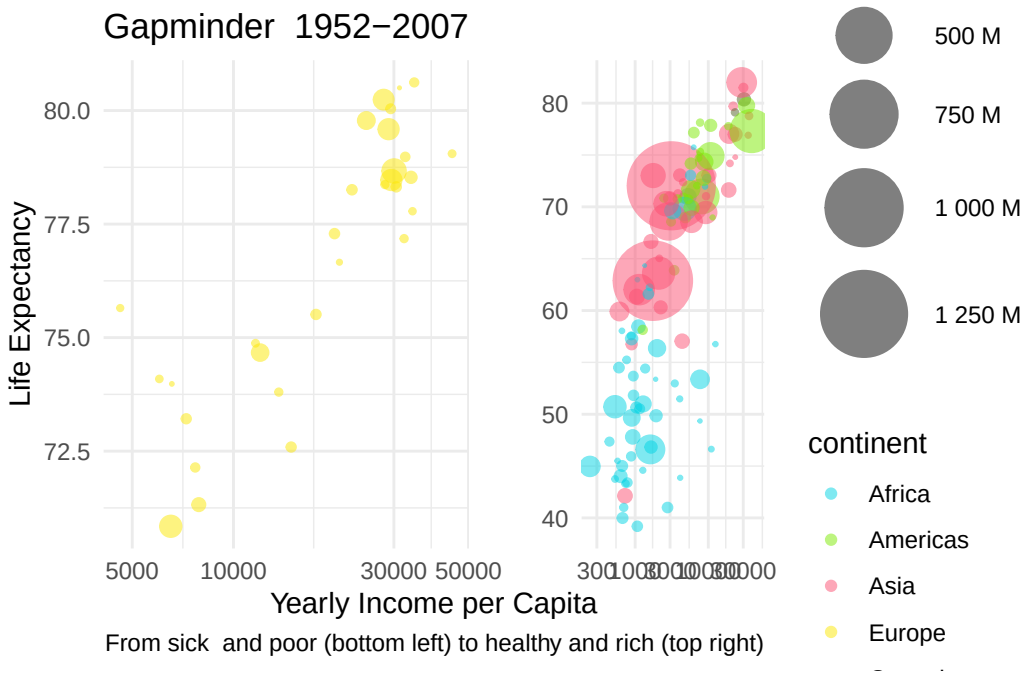
Use `facet_zoom()` from package `ggforce`

solution

```
require("ggforce") #<<

p_zoom_continent <- p +
  facet_zoom( #<<
    xy= continent==zoom_continent, #<<
    zoom.data= continent==zoom_continent #<<
    ) #<<

p_zoom_continent
```



Adding labels

solution

```
require(ggrepel) #<<

p +
   aes(label=country) + #<<
   ggrepel::geom_label_repel(max.overlaps = 5) + #<<
   scale_size_area(max_size = 15,
                   labels= scales::label_number(scale=1/1e6,
                                                suffix=" M")) +
   scale_color_manual(values = neat_color_scale) +
   theme_minimal() +
   theme(legend.position = "none") +
    labs(title= glue("Gapminder  {min(gapminder$year)}-{max(gapminder$year)}"),
         x = "Yearly Income per Capita",
         y = "Life Expectancy",
       caption="From sick  and poor (bottom left) to healthy and rich (top right)")
```

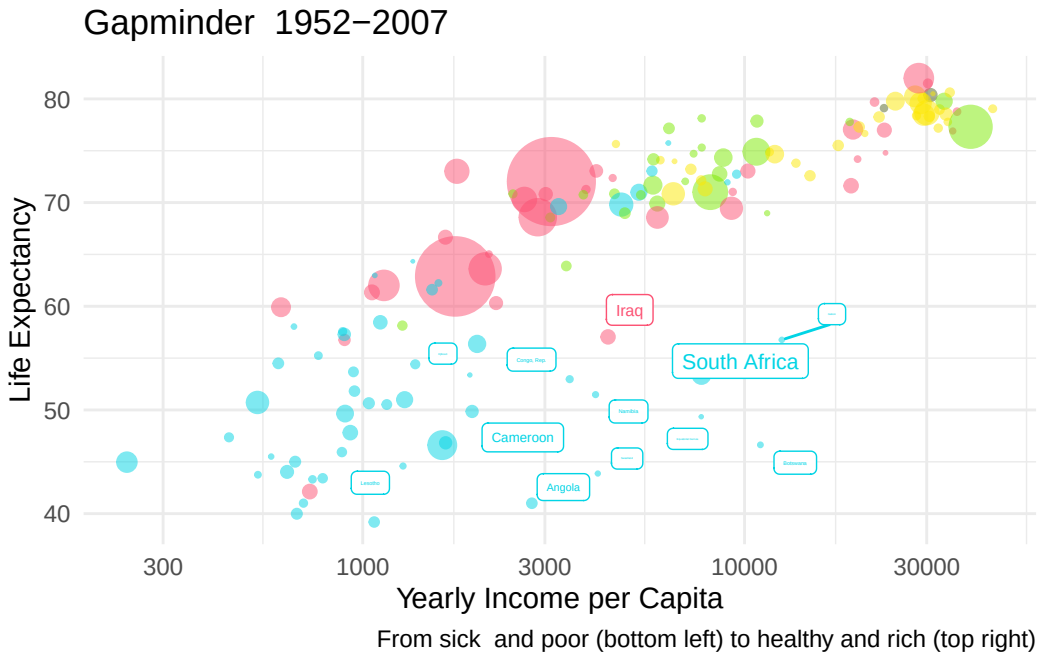


Figure 1: Gapminder 2002 layer by layer

Facetting

So far we have only presented one year of data (2002)

Rosling used an *animation* to display the flow of time

If we have to deliver a printable report, we cannot rely on animation, but we can rely on *facetting*

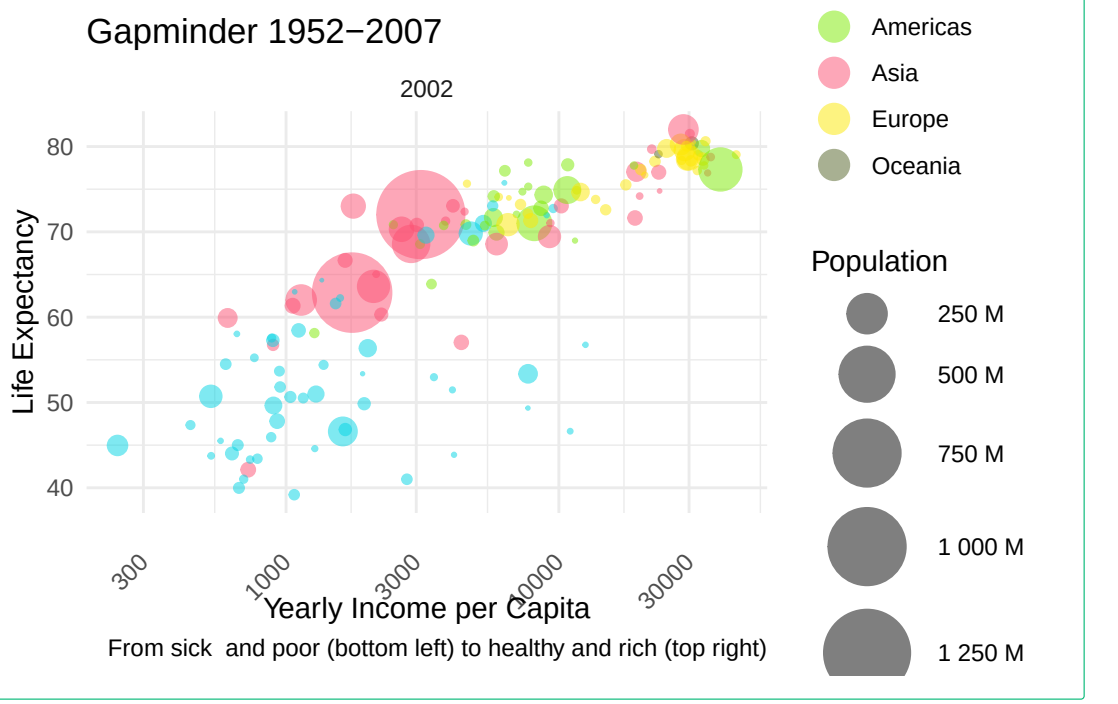
Facets are collections of small plots constructed in the same way on subsets of the data

We add a layer to the graphical object using `facet_wrap()`

solution

```
p <- p +
  aes(text=country) +
  guides(color = guide_legend(title = "Continent",
                              override.aes = list(size = 5),
                              order = 1),
         size = guide_legend(title = "Population",
                             order = 2)) +
  theme(axis.text.x = element_text(angle = 45, vjust = 0.5, hjust=1)) +
  facet_wrap(vars(year), ncol=6) +
  ggtitle("Gapminder 1952-2007")

p
```



As all rows in `gapminder_2002` are all related to `year` 2002, we need to rebuild the graphical object along the same lines (using the same *graphical pipeline*) but starting from the whole `gapminder` dataset.

Should we do this using *cut and paste*?

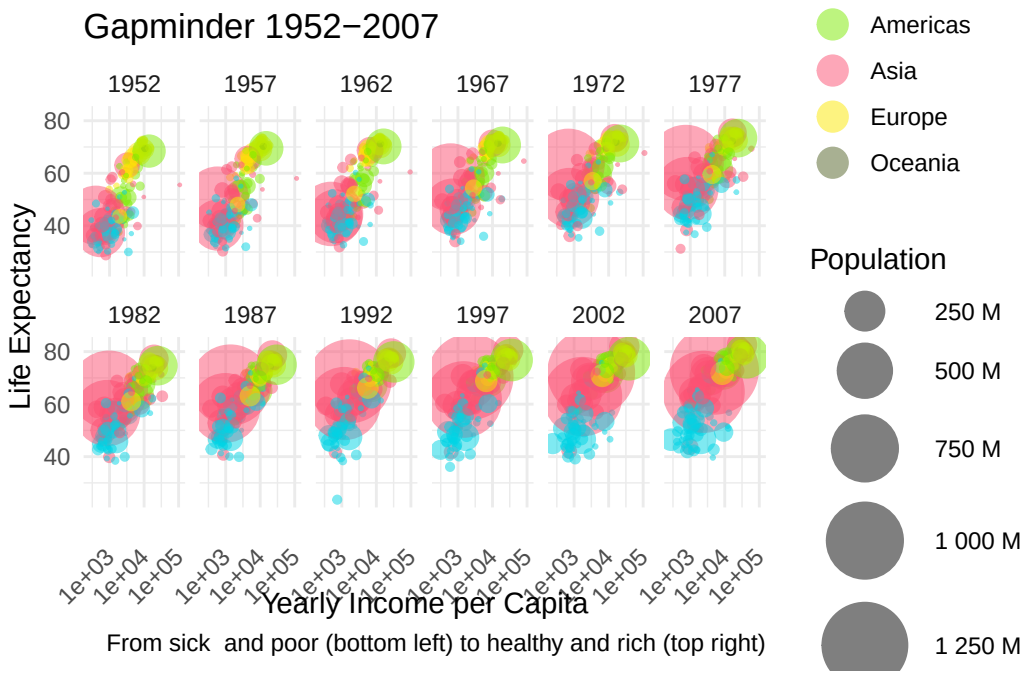
No

Don't Repeat Yoursel (DRY)

Abide to the DRY principle using operator `%+%`: the `ggplot2` object `p` can be fed with another dataframe and all you need is proper facetting.

solution

```
p %+% gapminder
```



Animate for free with `plotly`

Use `plotly::ggplotly()`

solution

```{r}
#| label: animate
#| eval: !expr knitr::is_html_output()
#| code-annotations: hover

q <- filter(gapminder, FALSE) %>%
   ggplot() +
   aes(x = gdpPercap) +
   aes(y = lifeExp) +
   aes(size = pop) +
   aes(text = country) +                    #
   aes(fill = continent) +
   # aes(frame = year) +                      #
  geom_point(alpha=.5, colour='black') +
  scale_x_log10() +
  scale_size_area(max_size = 15,
                  labels= scales::label_number(scale=1/1e6,
                                               suffix=" M")) +
  scale_fill_manual(values = neat_color_scale) +
  theme(legend.position = "none") +
  labs(title= glue("Gapminder  {min(gapminder$year)}-{max(gapminder$year)}"),
       x = "Yearly Income per Capita",
       y = "Life Expectancy",
       caption="From sick  and poor (bottom left) to healthy and rich (top right)")


(q %+% gapminder) %>%
  plotly::ggplotly(height = 500, width=750)
```

1. `text` will be used while *hovering*
2. `frame` is used by `plotly` to drive the animation. One `frame` per year

solution

```{r}
#| eval: !expr knitr::is_html_output()

(p %+% gapminder +
 facet_null() +
  theme_minimal() +
 aes(frame=year)) %>%
 plotly::ggplotly(height = 500, width=750)
```

More material

Visit Data visualization using ggplot2 and its extensions, UseR 2021 Tutorial