Session 3b: Data Visualization

R for Stata Users

Rony Rodrigo Maximiliano Rodriguez-Ramirez
The World Bank – DIME | WB Github

24 November 2020

Table of contents

Introduction

Introduction

Goals of this session

In this session, you'll learn how to use R to produce insightful, meaningful and (hopefully) nice-looking graphs. In particular, you'll use a package called `ggplot2` from the `tidyverse` meta-package.

Similarly to previous sessions, you can find some references at the end of this presentation that include a more comprehensive discussions on data visualization.

Introduction

Before we start

- Make sure the packages `ggplot2` and `plotly` are installed and loaded.
- Load the `whr_panel.csv` data set.

```
# Packages
library(tidyverse)
library(plotly)

# Load CSV data
# Replace with where your data is

FolderPath ← file.path("YOUR/FOLDER/PATH")

whr_panel ← read.csv(file.path(FolderPath,"whr_panel.csv"),
                header = T,
                stringsAsFactors = F)
```

Introduction

In our workflow there are usually two distinct uses for plots:

1. **Exploratory analysis**: Quickly visualize your data in an insightful way.
    - We'll do this using `{ggplot2}`, but in a more simplistic way which will allow us to quickly create some basic figures.
2. **Publication/Reporting**: Make pretty graphs for a presentation, a project report, or to just show your boss something other than the laziest graph you could come up with:
    - We'll do this using `ggplot2` with more customization. The idea is to create beautiful graphs.
    - `ggplot2`'s syntax is more complicated, but it's easier to make your graphs look good with it.

Tidy Data (Yes, again)

First, we need to remember the following:

- Is our data in a tidy format?

If it is not, we might need to first clean it and then plot it. It will really difficult to create good graphics when your data is tidy.

> {ggplot} and other R packages tend to set up their functions in a way that we need a tidy data. In a few cases, these packages will require a different data structure.

Exploratory Analysis

Base Plot

First, we're going to use base plot, i.e., using Base R default libraries. It is easy to use and can produce useful graphs with very few lines of code.

Exercise 1: Exploratory graph.

Let's plot the whr dataset that we constructed last week. We are going to use the function `plot()`. Before we plot it, let's create a vector called `vars` that contains the variables: economy_gdp_per_capita, happiness_score, health_life_expectancy, and freedom.

Base Plot

First, we're going to use base plot, i.e., using Base R default libraries. It is easy to use and can produce useful graphs with very few lines of code.

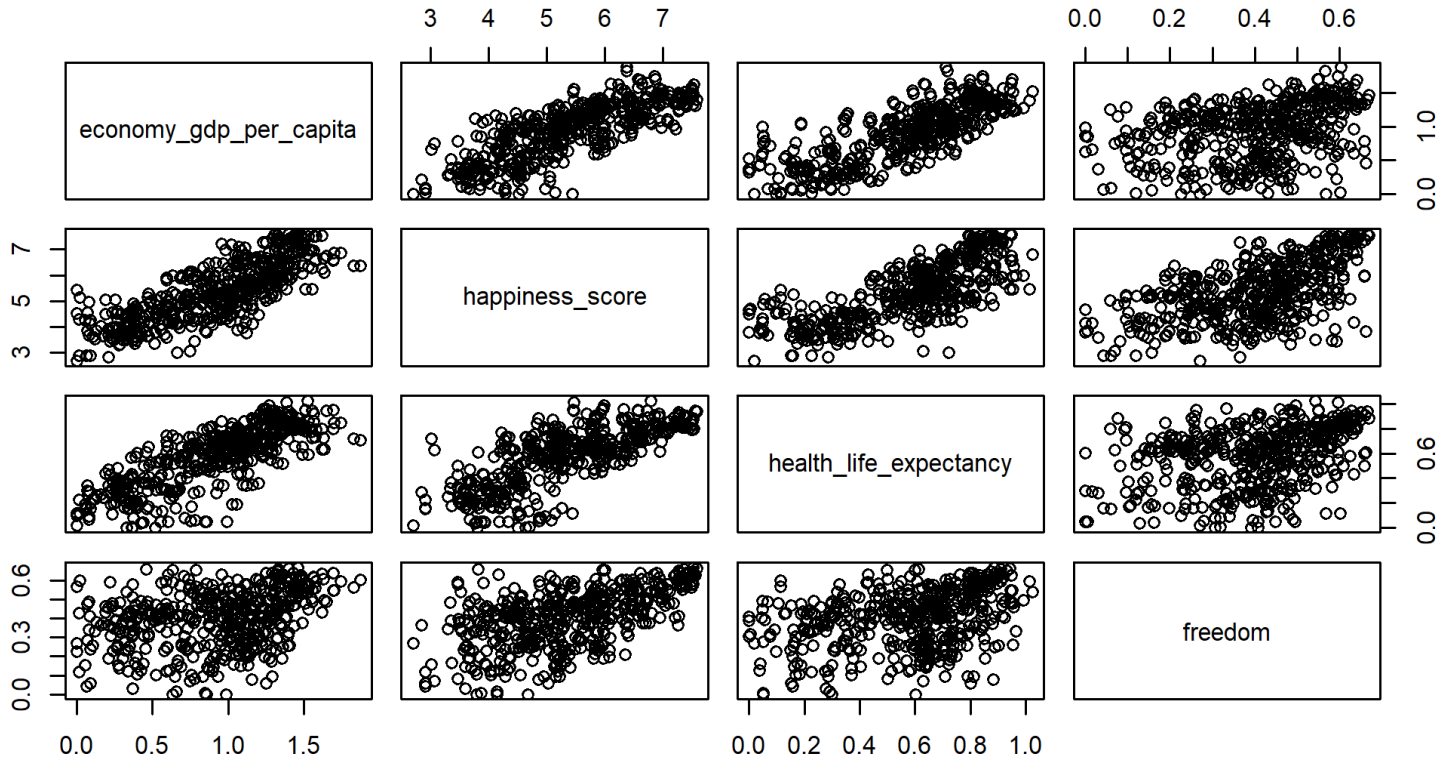
Exercise 1: Exploratory graph.

Let's plot the whr dataset that we constructed last week. We are going to use the function `plot()`. Before we plot it, let's create a vector called `vars` that contains the variables: economy_gdp_per_capita, happiness_score, health_life_expectancy, and freedom.

```
# Vector of variables

vars ← c("economy_gdp_per_capita", "happiness_score", "health_life_expectancy", "freedom")

# Create a subset with only those variables, let's called this subset whr_simp

whr_simp ← whr_panel %>%
  select(all_of(vars))
```

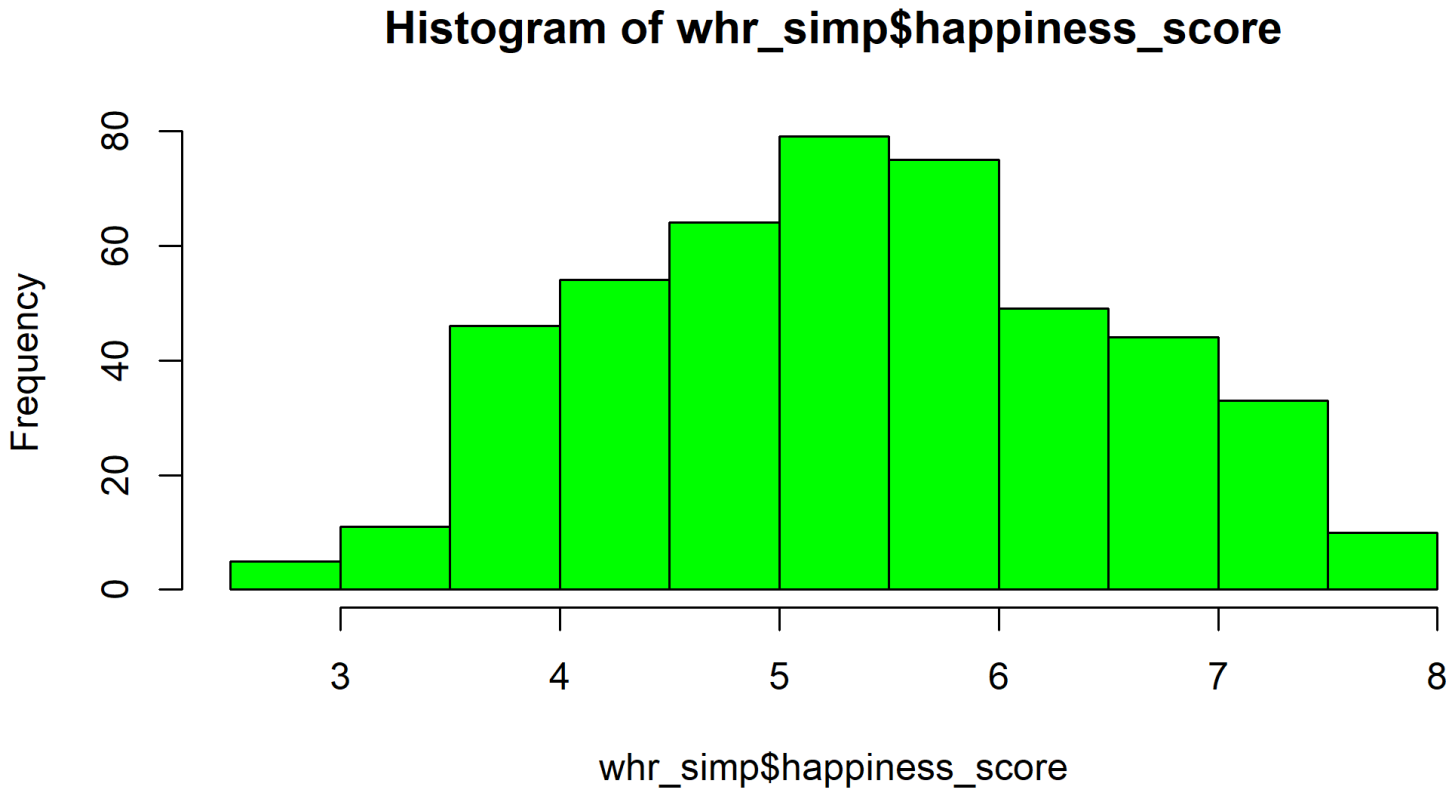
Base Plot

```
plot(whr_simp)
```



Base Plot: Histogram

```
hist(whr_simp$happiness_score, col = "green")
```



The beauty of {ggplot2}

Some advantages of `ggplot2`

1. Consistency with the **Grammar of Graphics**
    - This book is the foundation of several data viz applications: {ggplot2}, {polaris-tableau}, {vega-lite}.
2. Flexibility
3. Layering and theme customization (way better than Stata).
4. Community

It is a powerful and easy to use tool (once you understand its logic) that produces complex and multifaceted plots.

{ggplot2}'s structure

After we have load our dataset. Let's plot something basic. The structure of a basic ggplot is:

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNTION> +
  <THEME_FUNCTION>
```

{ggplot2}'s structure

After we have load our dataset. Let's plot something basic. The structure of a basic ggplot is:

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNTION> +
  <THEME_FUNCTION>
```

> There is an ongoing debate on how we should structure our ggplot. More about this later on.

{ggplot2}'s decomp

I will stick to Thomas Lin Pedersen's decomposition who is one of most prominent developers of the ggplot and gganimate package.

{ggplot2}'s structure

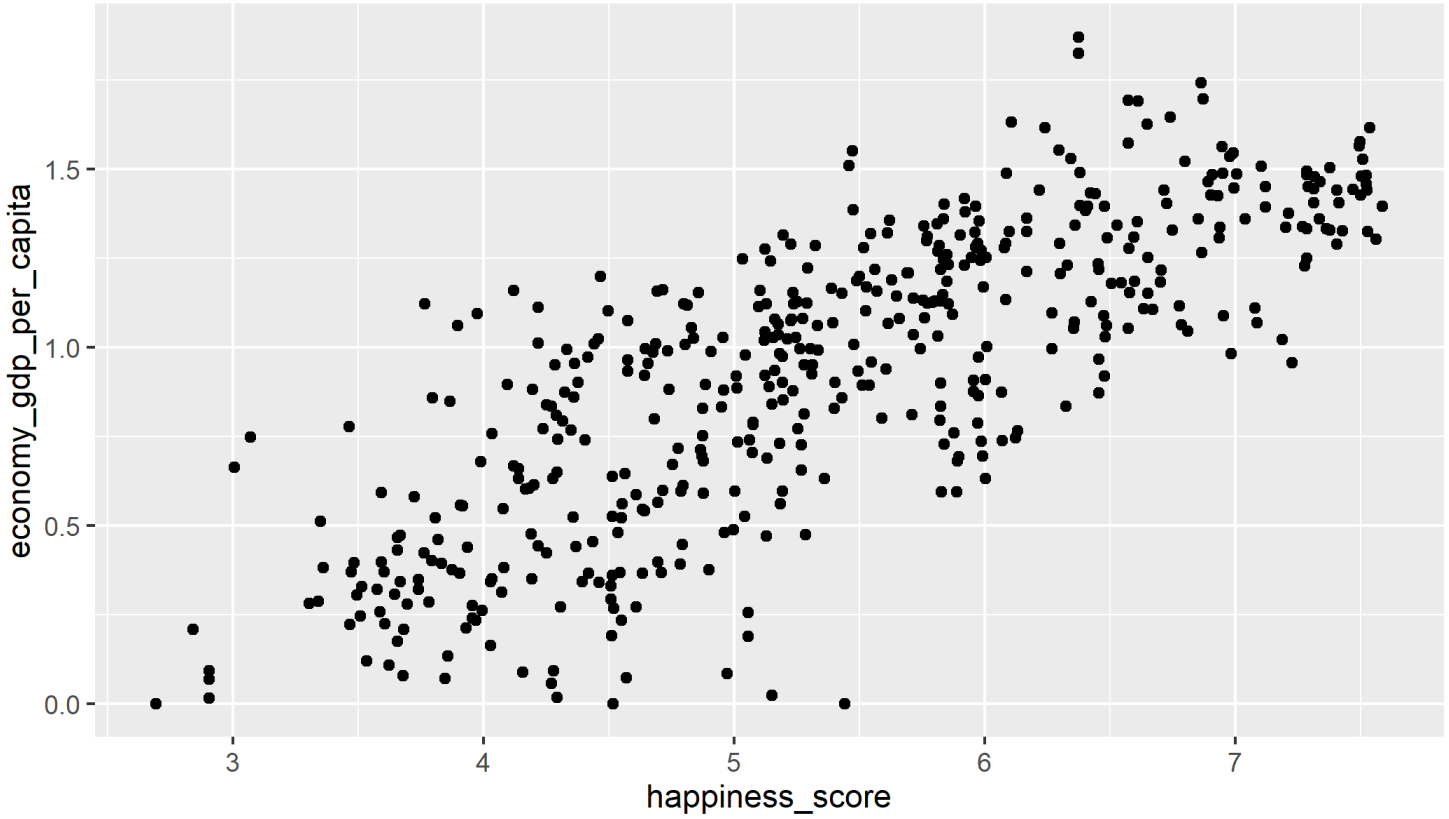
Therefore, we have the following:

1. `Data`: The raw data that you want to visualize
2. `Layers`: geom *and stat* → The geometric shapes and statistical summaries representing the data
3. `Aesthetics`: aes() → Aesthetic mappings of the geometric and statistical objects
4. `Scales` scale_ → Maps between the data and the aesthetic dimensions
5. `Coordinate system`: coord_ → Maps data into the plane of the data rectangle
6. `Facets`: facet_ → The arrangement of the data into a grid of plots
7. `Visual themes`: theme() and theme_ → The overall visual defaults of a plot

Exploratory Analysis

Ok, enough chit-chat about the gramar of graphics. Let's start making some plots.

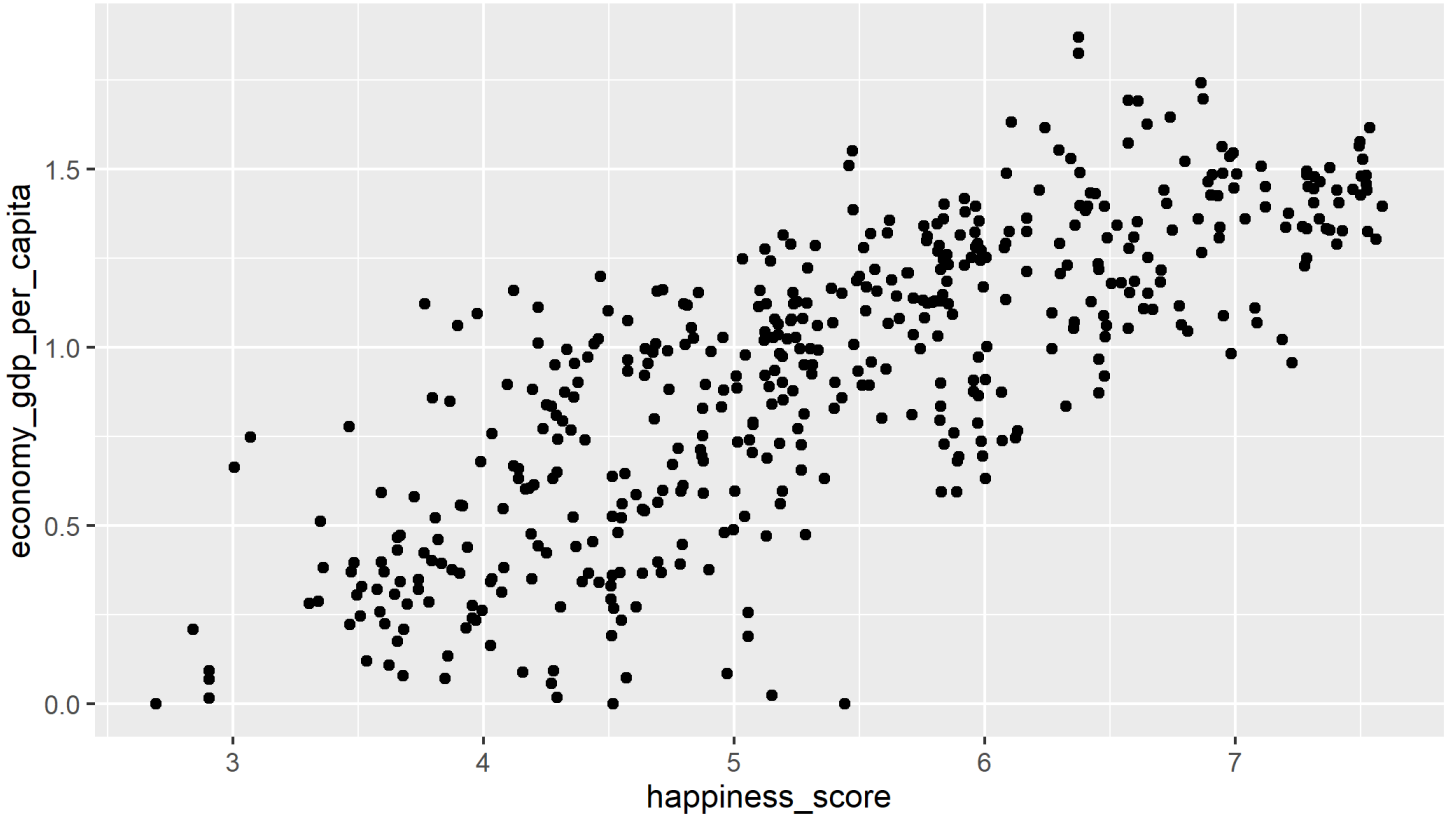
```
ggplot(data = whr_panel) +
  geom_point(mapping = aes(x = happiness_score, y = economy_gdp_per_capita))
```



Exploratory Analysis

Ok, enough chit-chat about the gramar of graphics. Let's start making some plots.

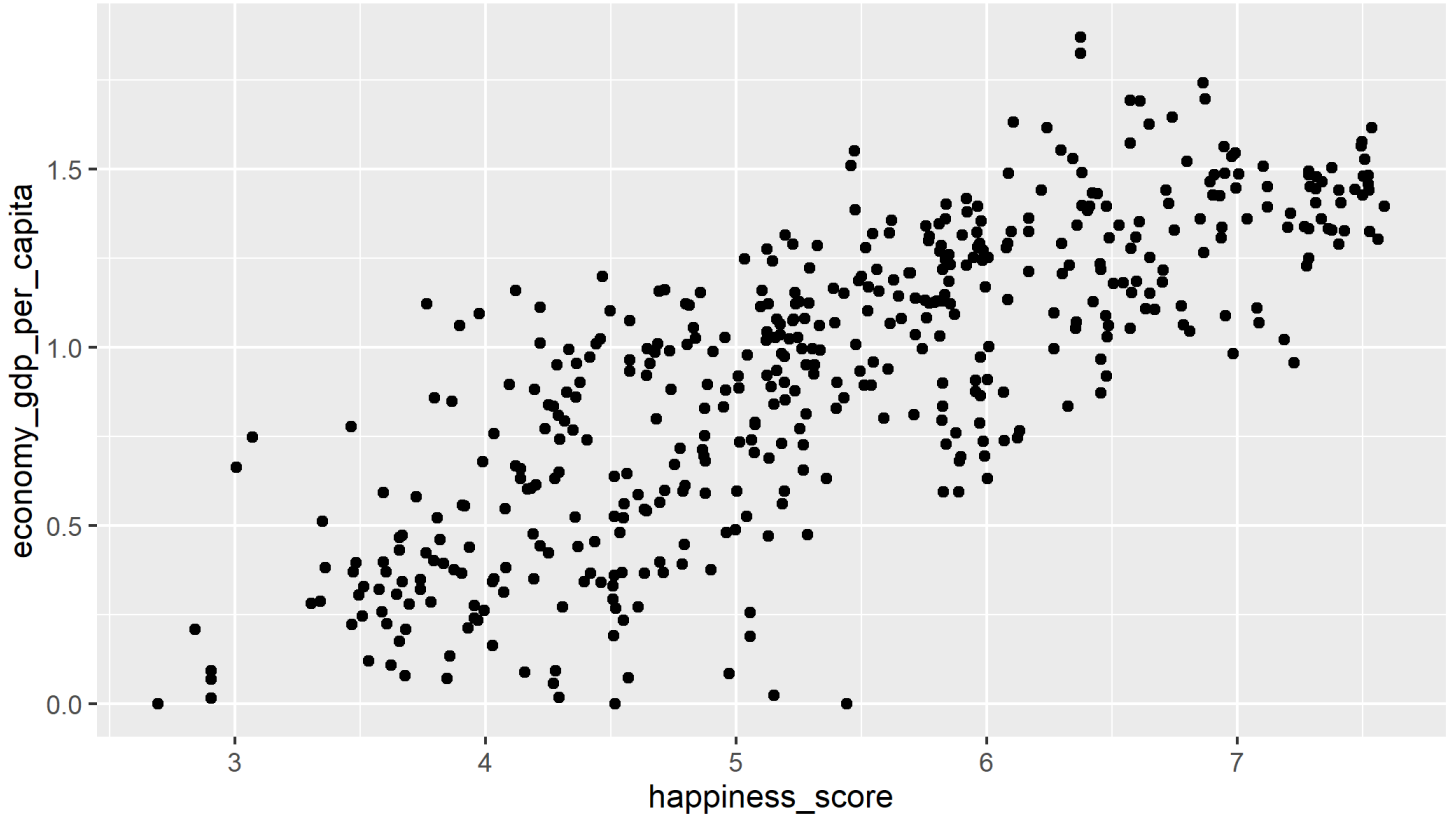
```
ggplot(data = whr_panel) +
  geom_point(mapping = aes(x = happiness_score, y = economy_gdp_per_capita))
```



Exploratory Analysis

We can also set up our mapping in the `ggplot()` function.

```
ggplot(data = whr_panel, aes(x = happiness_score, y = economy_gdp_per_capita)) +
  geom_point()
```



Exploratory Analysis

I prefer to use the second way of structuring our ggplot. First, setting our data and aesthetics, and then the geometries.

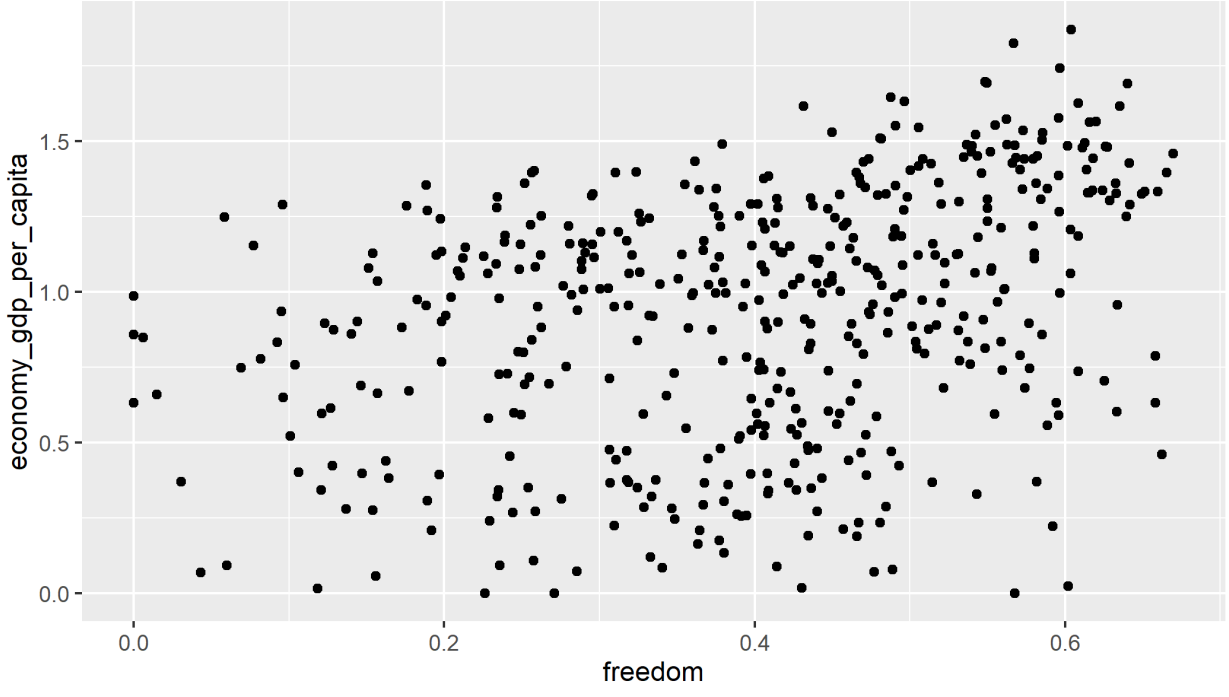
Exercise 2: Create a scatter plot with x = freedom and y = economy_gdp_per_capita.

Exploratory Analysis

I prefer to use the second way of structuring our ggplot. First, setting our data and aesthetics, and then the geometries.

Exercise 2: Create a scatter plot with x = freedom and y = economy_gdp_per_capita.

```
ggplot(data = whr_panel, aes(x = freedom, y = economy_gdp_per_capita)) +
  geom_point()
```



Exploratory Analysis

The most common `geoms` are:

- `geom_bar()`, `geom_col()`: bar charts.
- `geom_boxplot()`: box and whiskers plots.
- `geom_density()`: density estiamtes.
- `geom_jitter()`: jittered points.
- `geom_line()`: line plots.
- `geom_point()`: scatter plots.

> If you want to know more about layers, you can refer to this.

Exploratory Analysis

In summary, our basic plots should have the following:

```
ggplot(data = whr_panel,
       aes(x = happiness_score,
           y = economy_gdp_per_capita)) +
  geom_point()
```

The data we want to plot.

Exploratory Analysis

In summary, our basic plots should have the following:

```
ggplot(data = whr_panel,
       aes(x = happiness_score,
           y = economy_gdp_per_capita)) +
  geom_point()
```

Columns to use for `x` and `y`

Exploratory Analysis

In summary, our basic plots should have the following:

```
ggplot(data = whr_panel,
       aes(x = happiness_score,
           y = economy_gdp_per_capita)) +
  geom_point()
```

How the plot is going to be drawn.

Exploratory Analysis

We can also **map** colors.

```
ggplot(data = whr_panel,
       aes(x = happiness_score,
           y = economy_gdp_per_capita,
           color = region)) +
  geom_point()
```

Exploratory Analysis

In `{ggplot2}`, these settings are called **aesthetics**.

> "Aesthetics of the geometric and statistical objects".

We can set up:

- `position`: x, y, xmin, xmax, ymin, ymax, etc.
- `colors`: color and fill.
- `transparency`: alpha.
- `sizes`: size and width.
- `shapes`: shape and linetype.

Notice that it is important to know where we are setting our aesthetics. For example:

- `geom_point(aes(color = region))` to color points based on the variable `region`
- `geom_point(color = "red")` to color all points in the same color.

Exploratory Analysis

Let's modify our last plot.

Exploratory Analysis

Let's modify our last plot.

```
ggplot(data = whr_panel,
       aes(x = happiness_score,
           y = economy_gdp_per_capita)) +
  geom_point(color = "red")
```

Exploratory Analysis

Exercise 3: Map colors per year for the freedom and gdp plot we did before.

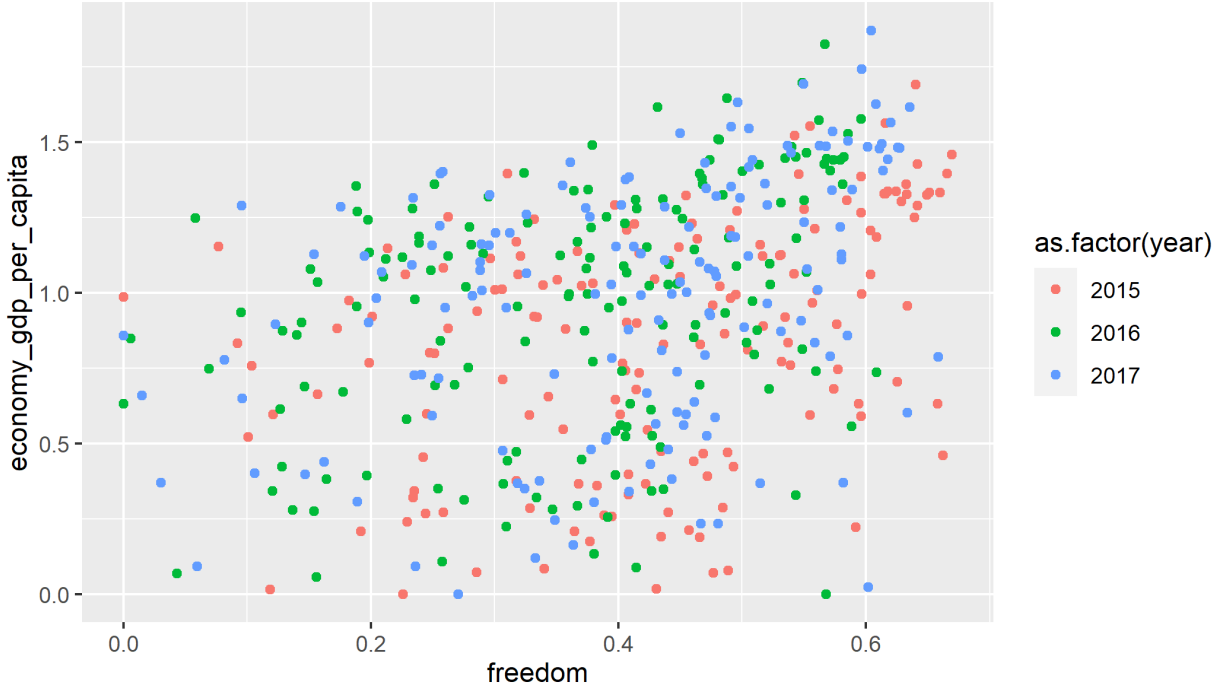
- Keep in mind the type of the variable year.

Exploratory Analysis

Exercise 3: Map colors per year for the freedom and gdp plot we did before.

- Keep in mind the type of the variable year.

```
ggplot(data = whr_panel,
       aes(x = freedom, y = economy_gdp_per_capita, color = as.factor(year))) +
  geom_point()
```



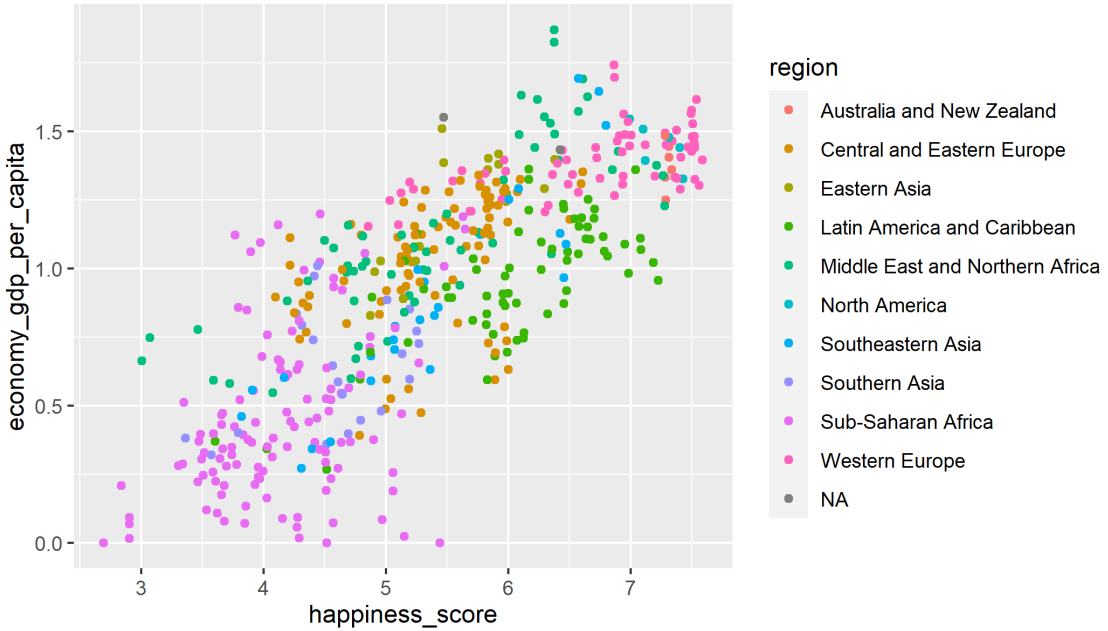
Exploratory Analysis

Let's modify our last plot.

Exploratory Analysis

Let's modify our last plot.

```
ggplot(data = whr_panel,
       aes(x = happiness_score,
           y = economy_gdp_per_capita)) +
  geom_point(aes(color = region))
```



Exploratory Analysis

Let's modify our last plot.

```
ggplot(data = whr_panel,
       aes(x = happiness_score,
           y = economy_gdp_per_capita)) +
  geom_point(aes(color = region))
```

> Notice that we can either set up our aesthetics at the global level our within each layer.

{ggplot2}'s settings

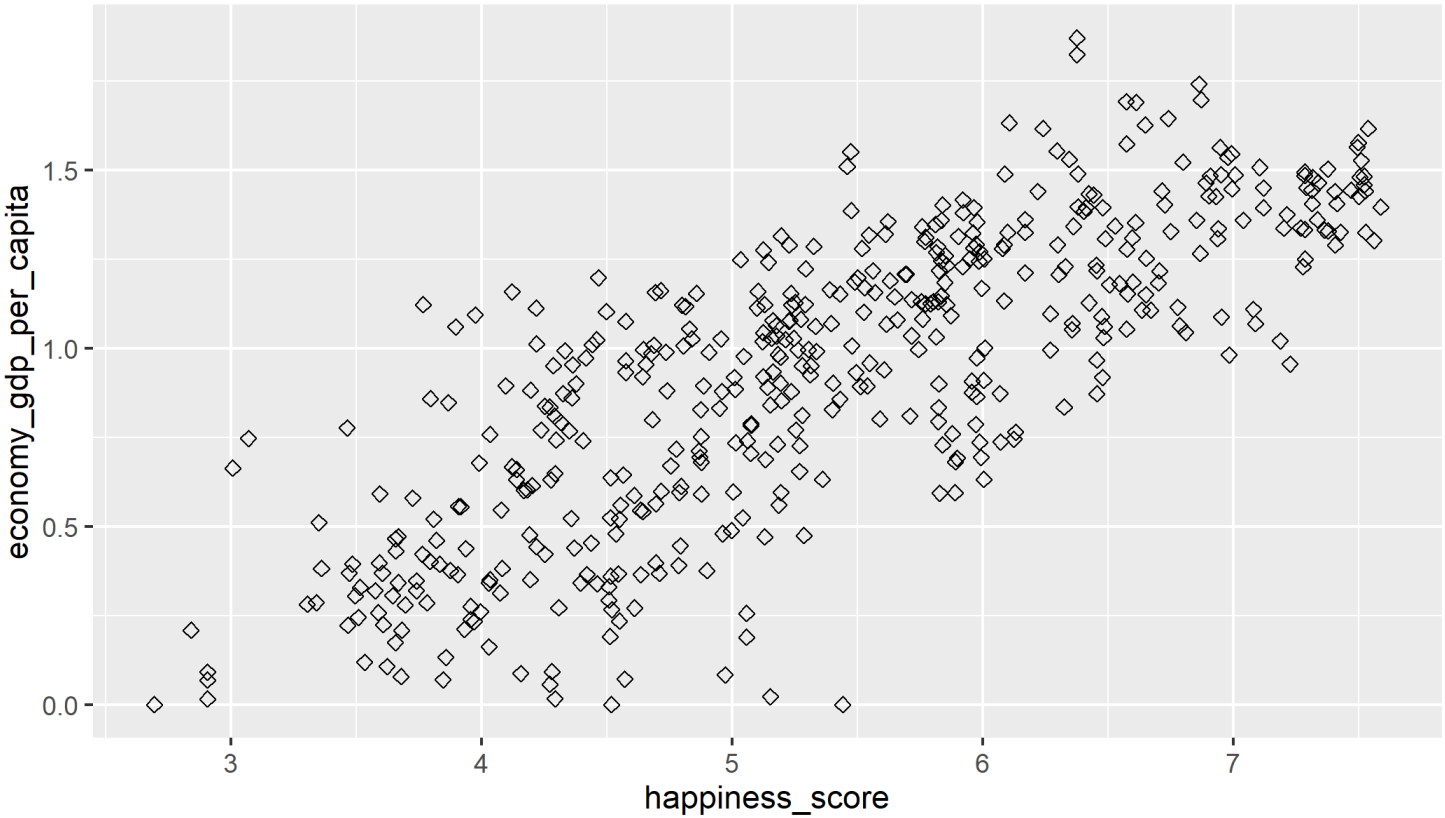
{ggplot2}'s settings

Now, let's try to modify our plots. In the following slides, we are going to:

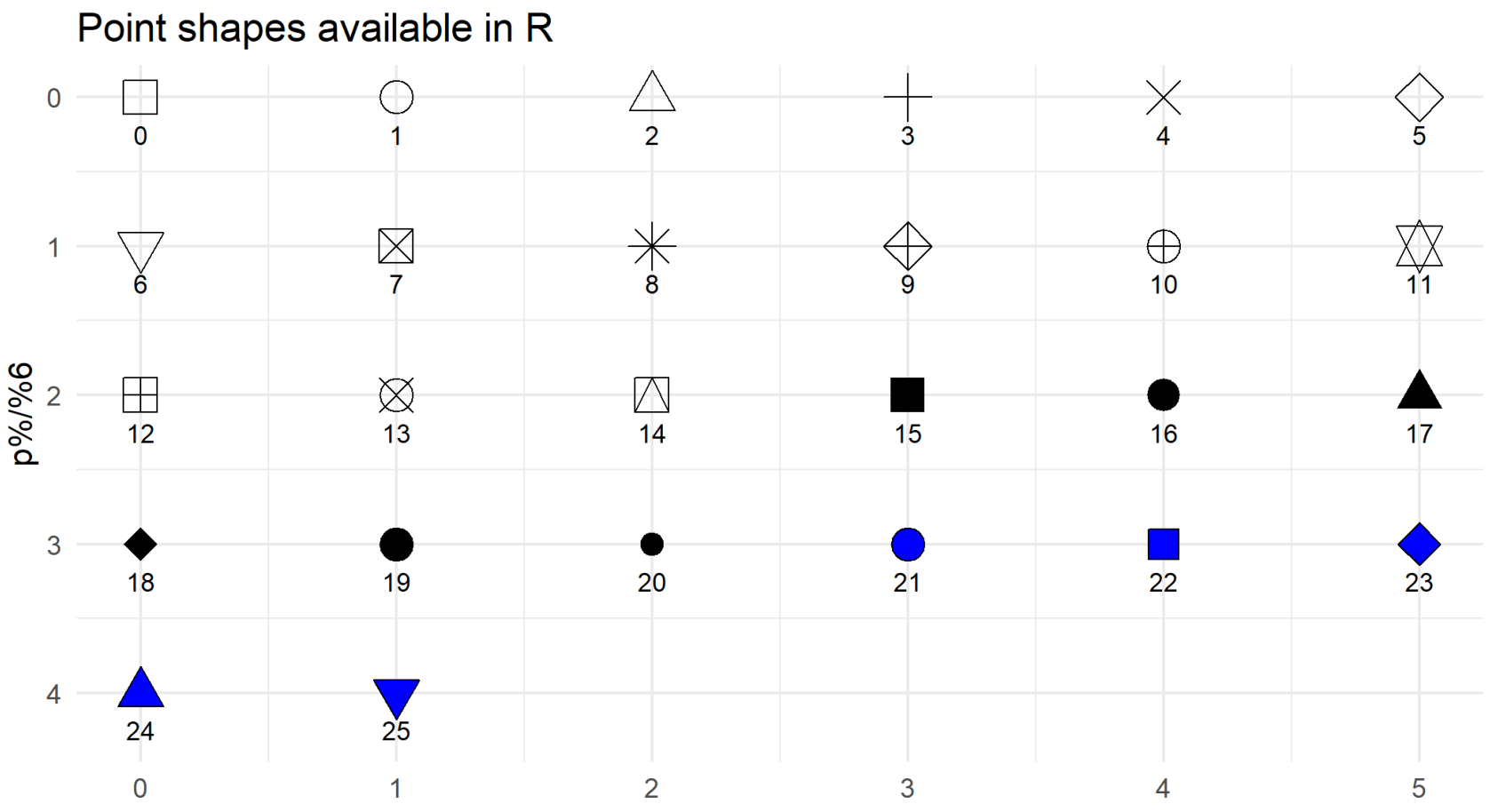
1. Change shapes.
2. Include more geoms.
3. Separate by regions.
4. Pipe and mutate before plotting.
5. Changing scales.
6. Modify our theme.

{ggplot2}: shapes


```
ggplot(data = whr_panel,
       aes(x = happiness_score,
           y = economy_gdp_per_capita)) +
  geom_point(shape = 5)
```



{ggplot2}: shapes



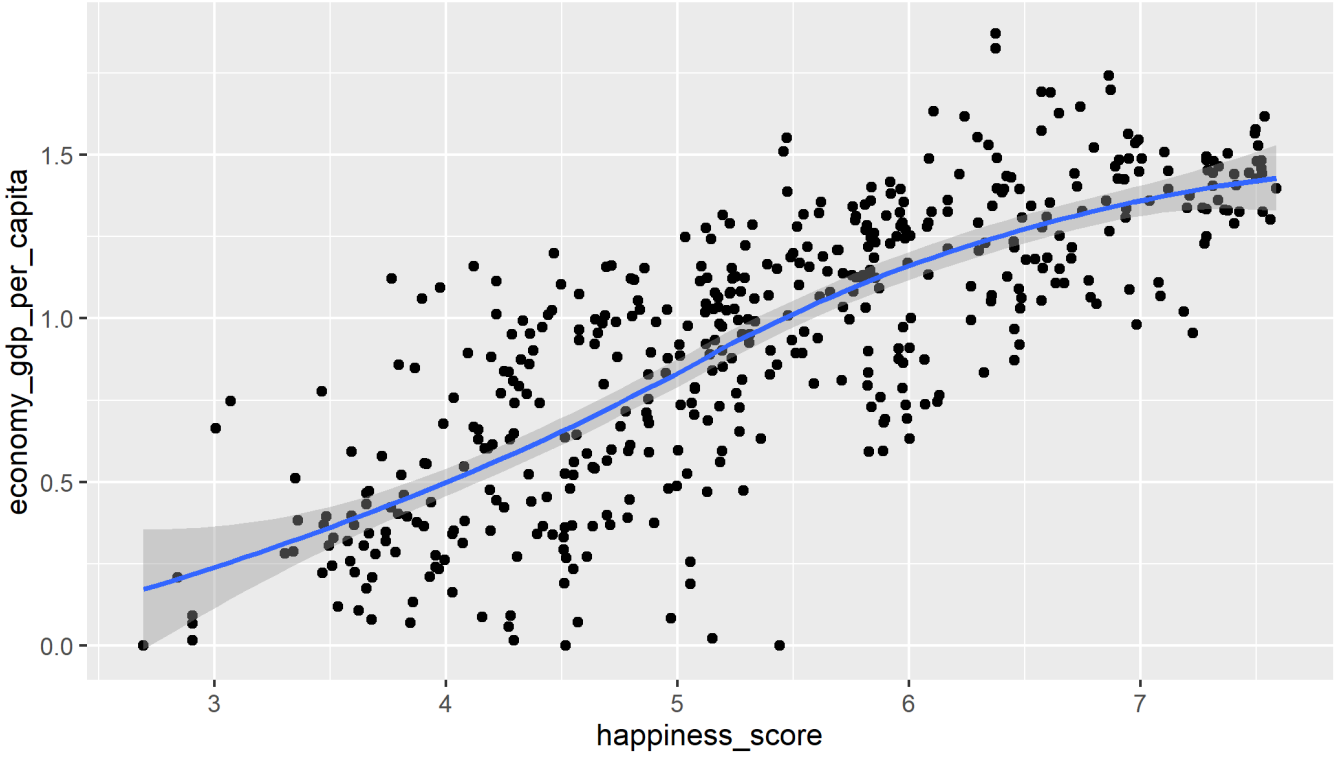
{ggplot2}: including more geoms

```
mean_happiness_score ← mean(whr_panel$happiness_score)

ggplot(data = whr_panel,
       aes(x = happiness_score,  y = economy_gdp_per_capita,
           color = region)) +
  geom_point() +
  geom_vline(xintercept = mean_happiness_score)
```

{ggplot2}: including more geoms

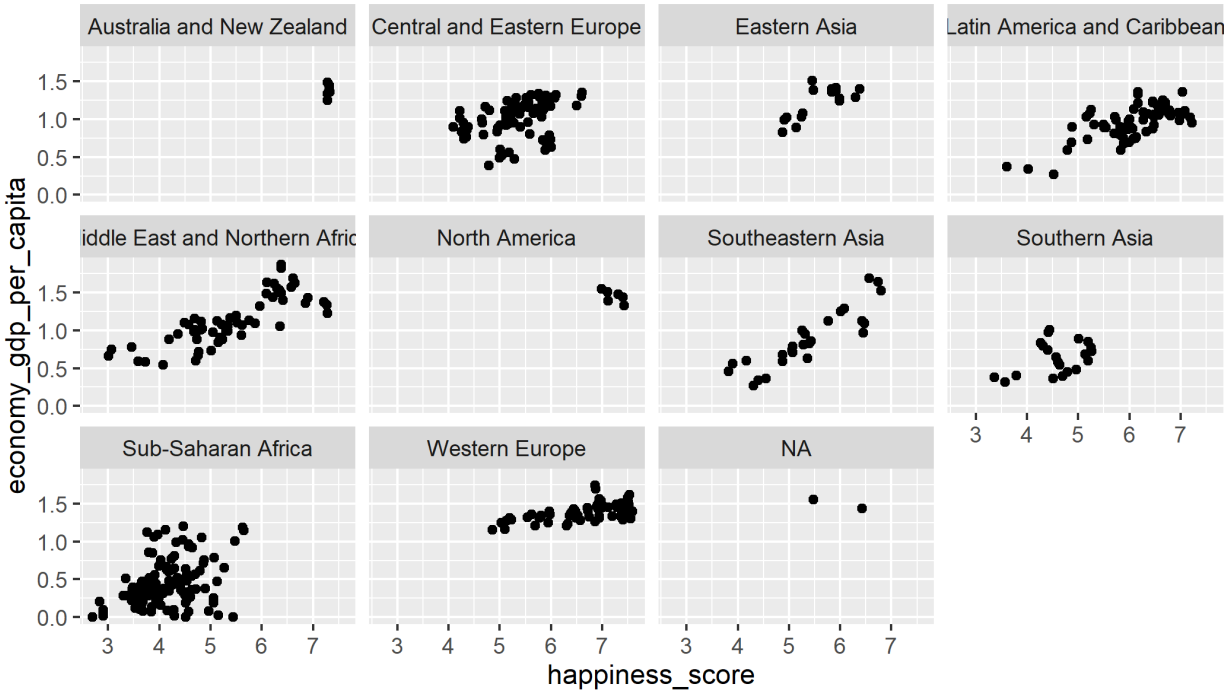
```
ggplot(data = whr_panel,
       aes(x = happiness_score,  y = economy_gdp_per_capita)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



{ggplot2}: Facets


```
ggplot(data = whr_panel,
       aes(x = happiness_score,  y = economy_gdp_per_capita)) +
  geom_point() +
  facet_wrap(~ region)
```



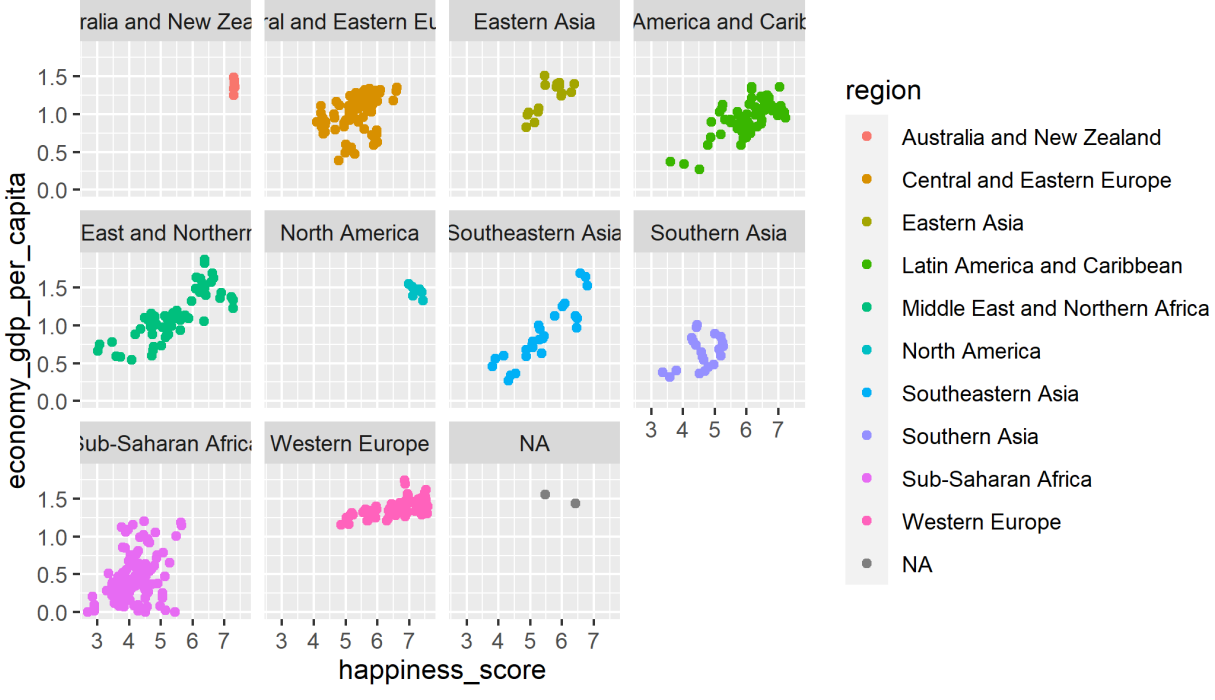
{ggplot2}: Colors and facets

Exercise 4: Use the last plot and add a color aesthetic per region.

{ggplot2}: Colors and facets

Exercise 4: Use the last plot and add a color aesthetic per region.

```
ggplot(data = whr_panel,
       aes(x = happiness_score,  y = economy_gdp_per_capita, color = region)) +
  geom_point() +
  facet_wrap(~ region)
```



{ggplot2}: Pipe and mutate before plotting

Let's imagine now, that we would like to transform a variable before plotting.

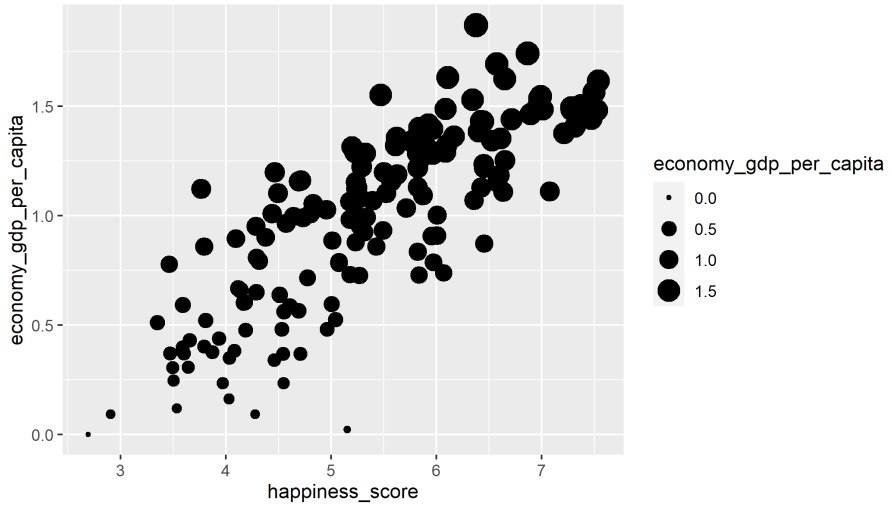
R Code | Plot

```
whr_panel %>%
  mutate(
    latam = ifelse(region == "Latin America and Caribbean", TRUE, FALSE)
  ) %>%
  ggplot(aes(x = happiness_score, y = economy_gdp_per_capita,
             color = latam)) +
  geom_point()
```

{ggplot2}: geom's sizes

We can also specify the size of a geom, either by a variable or just a number.

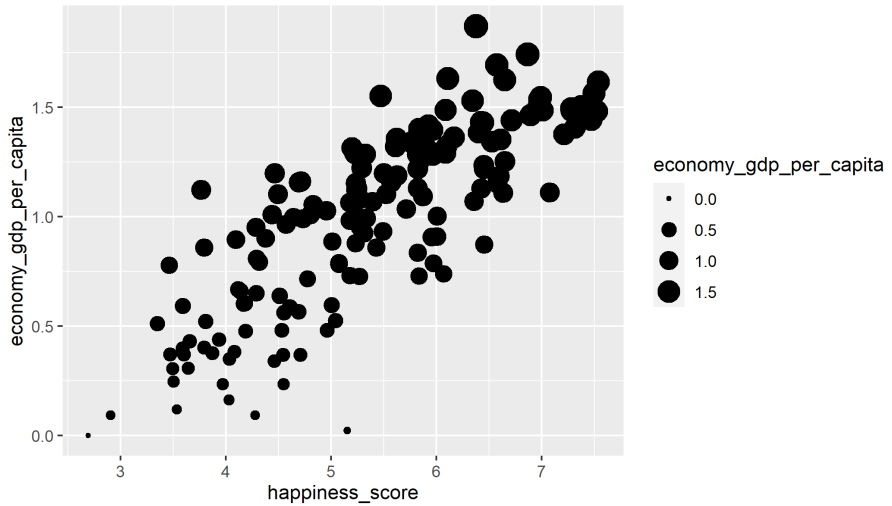
```
whr_panel %>%
  filter(year == 2017) %>%
  ggplot(aes(x = happiness_score,  y = economy_gdp_per_capita)) +
  geom_point(aes(size = economy_gdp_per_capita))
```



{ggplot2}: geom's sizes

We can also specify the size of a geom, either by a variable or just a number.

```
whr_panel %>%
  filter(year == 2017) %>%
  ggplot(aes(x = happiness_score,  y = economy_gdp_per_capita)) +
  geom_point(aes(size = economy_gdp_per_capita))
```

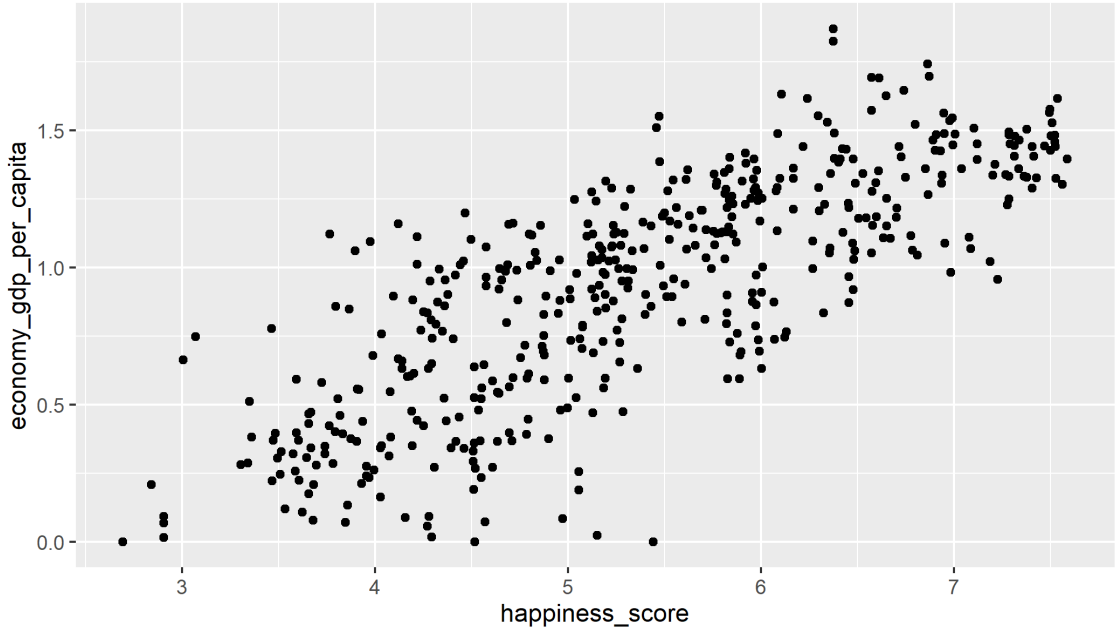


Why do you think we are adding the size to the geom_point()'s aes?

{ggplot2}: Changing scales

Linear Log

```
ggplot(data = whr_panel,
       aes(x = happiness_score,
           y = economy_gdp_per_capita)) +
  geom_point()
```



{ggplot2}: Themes

Let's go back to our plot with the `latam` dummy.

We are going to do the following to this plot:

1. Filter only for the year 2015.
2. Change our theme.
3. Add correct labels.
4. Add some annotations.
5. Modify our legends.

{ggplot2}: Labs

R Code | Plot

```
whr_panel %>%
  mutate(
    latam = ifelse(region == "Latin America and Caribbean", TRUE, FALSE)
  ) %>%
  filter(year == 2015) %>%
  ggplot(aes(x = happiness_score, y = economy_gdp_per_capita,
             color = latam)) +
  geom_point() +
  labs(
    x = "Happiness Score",
    y = "GDP per capita",
    title = "Happiness Score vs GDP per capita, 2015"
  )
```

{ggplot2}: Legends

R Code | Plot

```
whr_panel %>%
  mutate(
    latam = ifelse(region == "Latin America and Caribbean", TRUE, FALSE)
  ) %>%
  filter(year == 2015) %>%
  ggplot(aes(x = happiness_score, y = economy_gdp_per_capita,
             color = latam)) +
  geom_point() +
  scale_color_discrete(labels = c("No", "Yes")) +
  labs(
    x = "Happiness Score",
    y = "GDP per capita",
    color = "Country in Latin America\nand the Caribbean",
    title = "Happiness Score vs GDP per capita, 2015"
  )
```

{ggplot2}: Themes

R Code | Plot

```
whr_panel %>%
  mutate(
    latam = ifelse(region == "Latin America and Caribbean", TRUE, FALSE)
  ) %>%
  filter(year == 2015) %>%
  ggplot(aes(x = happiness_score, y = economy_gdp_per_capita,
             color = latam)) +
  geom_point() +
  scale_color_discrete(labels = c("No", "Yes")) +
  labs(
    x = "Happiness Score",
    y = "GDP per capita",
    color = "Country in Latin America\nand the Caribbean",
    title = "Happiness Score vs GDP per capita, 2015"
  ) +
  theme_minimal()
```

{ggplot2}: Themes

The `theme()` function allows you to modify each aspect of your plot. Some arguments are:

```
theme(
    # Legend title and text labels
    #:::::::::::::::::::::::::::::::::::::::::::::::::::::::::
    # Title font color size and face
    legend.title = element_text(color, size, face),
    # Title alignment. Number from 0 (left) to 1 (right)
    legend.title.align = NULL,
    # Text label font color size and face
    legend.text = element_text(color, size, face),
    # Text label alignment. Number from 0 (left) to 1 (right)
    legend.text.align = NULL,
)
```

More about these modification can be found here

Saving a plot

Saving a plot

Remember that in R we can always assign our functions to an object. In this case, we can assign our {ggplot2} code to an object called fig as follows.

```
fig ← whr_panel %>%
  mutate(
    latam = ifelse(region == "Latin America and Caribbean", TRUE, FALSE)
  ) %>%
  filter(year == 2015) %>%
  ggplot(aes(x = happiness_score, y = economy_gdp_per_capita,
             color = latam)) +
  geom_point() +
  scale_color_discrete(labels = c("No", "Yes")) +
  labs(
    x = "Happiness Score",
    y = "GDP per capita",
    color = "Country in Latin America\nand the Caribbean",
    title = "Happiness Score vs GDP per capita, 2015"
  ) +
  theme_minimal()
```

Therefore, if you want to plot it again, you can just type `fig` in the console.

Saving a plot

Exercise 5: Save our last plot.

We will use the `ggsave()` function. You can either include the function after your plot or, first, save the ggplot as an object and then save the plot.

The syntax is `ggsave(OBJECT, filename = FILEPATH, heigth = ... , width = ... , dpi = ... )`.

Saving a plot

Exercise 5: Save our last plot.

We will use the `ggsave()` function. You can either include the function after your plot or, first, save the ggplot as an object and then save the plot.

The syntax is `ggsave(OBJECT, filename = FILEPATH, heigth = ... , width = ... , dpi = ... )`.

How to do it?

```
ggsave(fig,
       filename = file.path(rawOutput, "fig.png"),
       dpi = 750, scale = 0.8,
       height = 8, width = 12)
```

If we want to save it as a pdf, we can include the argument `device = cairo_pdf`.

```
# Save Plot
ggsave(fig,
       filename = file.path(rawOutput, "fig.pdf"),
       device = cairo_pdf, scale = 0.8,
       height = 8, width = 12)
```

Interactive graphs

Interactive graphs

There are several packages to create interactive or dynamic data vizualizations with R. Here are a few:

- `leaflet` - R integration tp one of the most popular open-source libraries for interactive maps.
- `highcharter` - cool interactive graphs.
- `plotly` - interactive graphs with integration to ggplot.
- `gganimate` - ggplot GIFs.
- `DT` - Interactive table

These are generally, html widgets that can be incorporated in to an html document and websites.

Interactive graphs

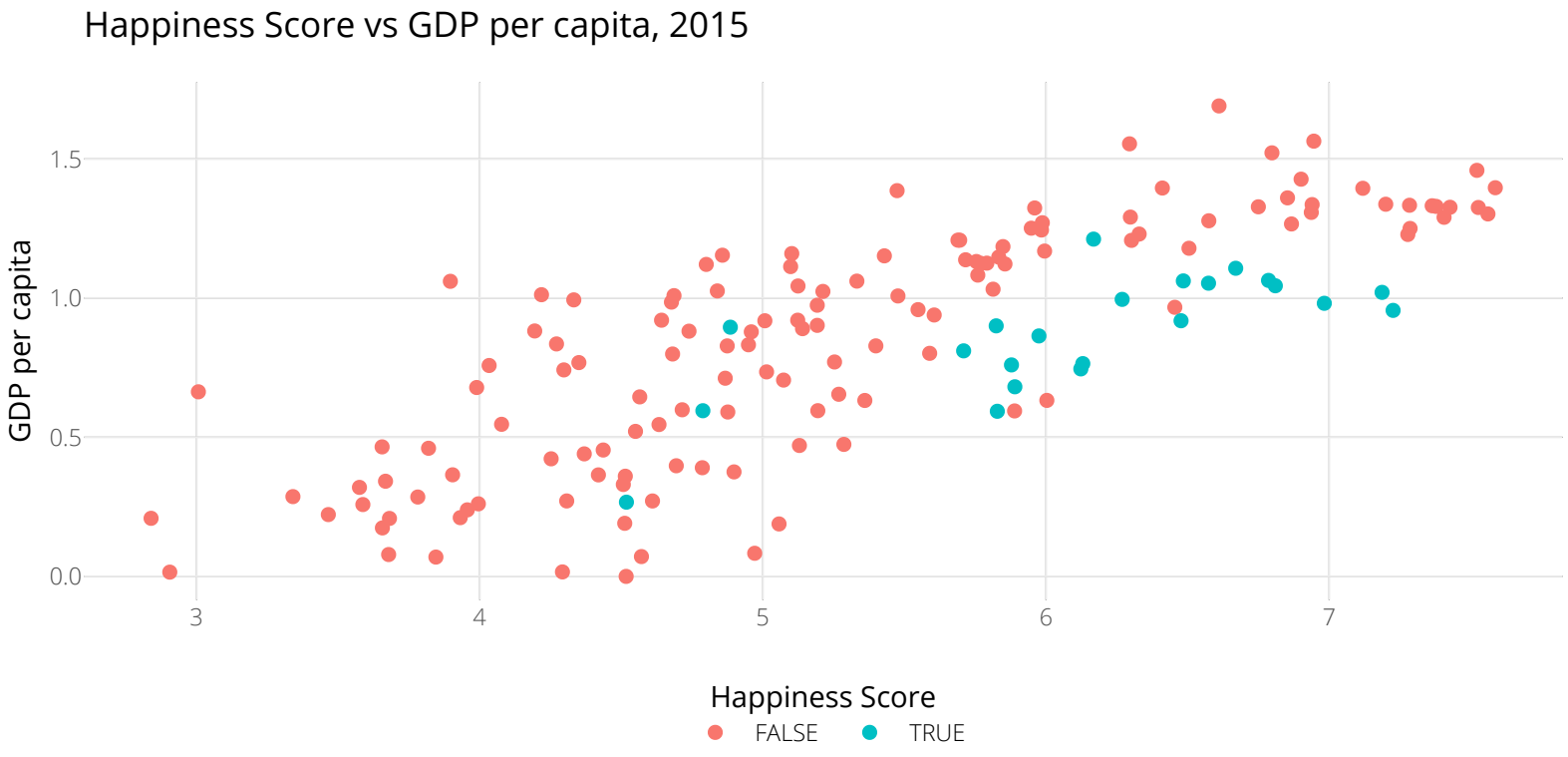
Now we'll use the ggplotly() function from the plotly package to create an interactive graph!

Exercise 6: Interactive graphs.

- Load the plotly package
- Pass that object with the last plot you created to the ggplotly() function

Interactive graphs

```r
# Load package
library(plotly)

# Use ggplotly to create an interactive plot
ggplotly(fig) %>%
  layout(legend = list(orientation = "h", x = 0.4, y = -0.2))
```



References and recommendations

References and recommendations

- **Websites**:
  - Interactive stuff : http://www.htmlwidgets.org/
  - The R Graph Gallery: https://www.r-graph-gallery.com/
  - Gpplot official site: http://ggplot2.tidyverse.org/
- **Online courses**:
  - Johns Hopkins Exploratory Data Analysis at Coursera: https://www.coursera.org/learn/exploratory-data-analysis
- **Books**:
  - The grammar of graphics by Leland Wilkinson.
  - Beautiful Evidence by Edward Tufte.
  - R Graphics cook book by Winston Chang
  - R for Data Science by Hadley Wickham andGarrett Grolemund

Thank you~